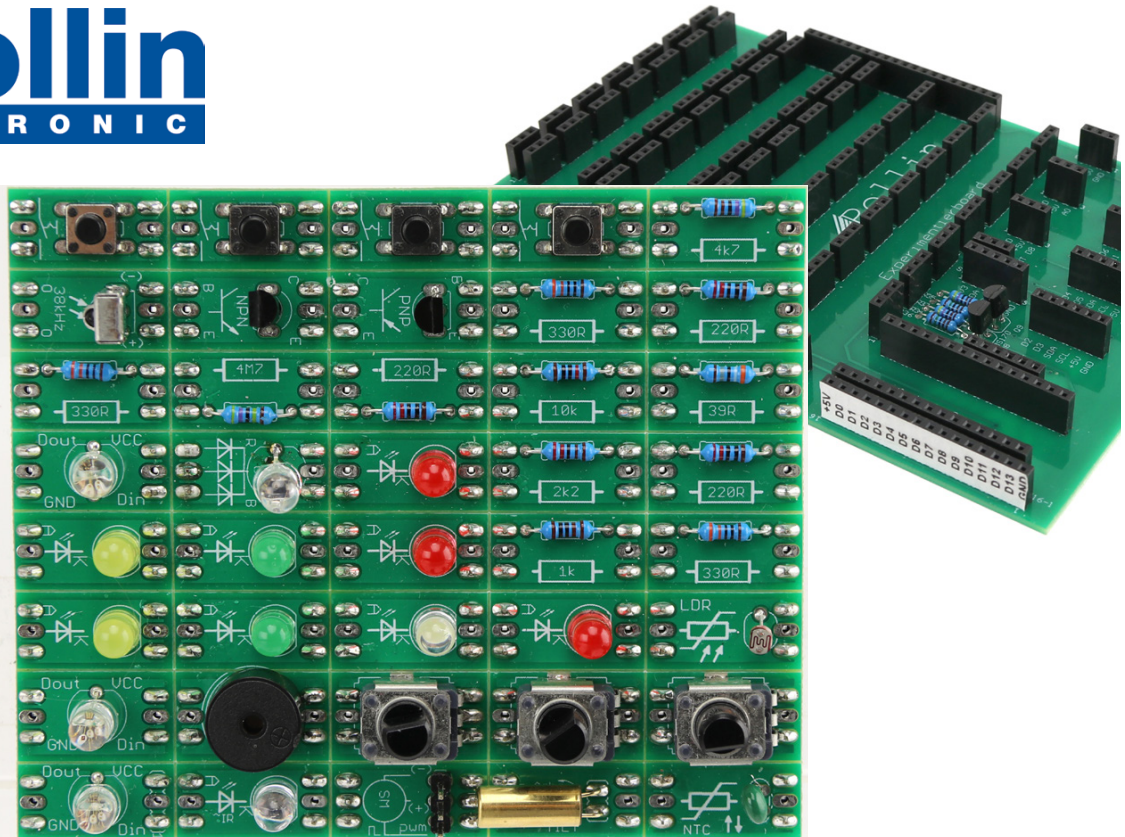


Der Schritt für Schritt Einstieg in Beispielen mit dem Pollin Starter Kit Arduino V1.0



Mit diesem Starter Kit gelingt der einfache Einstieg in die Arbeitsweise, Anwendung und Programmierung des Arduino :

- Durch überschaubare Beispiele, deren Funktion beschrieben und in Bildern für den einfachen Nachbau dargestellt werden;
- Durch den im Zip-File beigefügten Quellcode aller Beispiele;
- Durch Spannende Experimente mit WOW-Effekt;

Mit den Aufsteckplatinen können kleine Schaltungen einfach, sicher und schnell, entsprechend den Abbildungen, in den einzelnen Lektionen nachgebaut werden.

In einfachen C Beispielen werden diese programmiert. Dabei soll die Funktionsweise des Arduino Schritt für Schritt vermittelt werden. Zudem wird auch die Funktionsweise elektronischer Bauelemente erläutert.

Es werden nur grundlegende Programmierkenntnisse vermittelt. Ziel dieses Einsteigerkits ist deshalb nicht, einen C Programmierkurs damit zu ersetzen.

Es soll lediglich den Einstieg erleichtern, kleine Schaltungen aufzubauen und diese dann zu programmieren.

Der zweite Teil umfasst die Ansteuerung von Sensoren und Modulen, die als Artikel bei Pollin Electronic erworben werden können.

Inhaltsverzeichnis

1. Grundlagen und Historisches.....	7
1.1. Physikalische Größen.....	8
1.1.1. Spannung.....	8
1.1.1.1. Gleichspannung.....	8
1.1.1.2. getaktete Gleichspannung.....	9
1.1.1.3. Wechselspannung.....	9
1.1.2. Strom.....	10
1.1.3. Der Widerstand als physikalische Größe (Ohm).....	11
2. passive Bauteile.....	12
2.1. Der Widerstand.....	12
2.1.1. der ohmsche Widerstand.....	12
2.1.2. Das Potentiometer.....	13
2.2. Der Kondensator.....	14
2.2.1. Keramikkondensatoren.....	15
2.2.2. Folienkondensatoren.....	15
2.2.3. Elektrolytkondensatoren.....	15
2.3. Induktivitäten (Spulen).....	16
2.4.1. Drosseln, Trafo, Übertrager.....	16
2.4.2. Relais.....	17
3. aktive Bauteile.....	18
3.1. Die Diode.....	18
3.1.1. Gleichrichterdiode (Siliziumdiode).....	18
3.1.2. die Leuchtdiode (LED).....	19
3.1.2.1. LED-Balken.....	21
3.1.2.2. WS2811.....	21
3.1.2.3. 7-Segment-Display.....	21
3.1.2.4. 8x8 Matrix Display mit MAX7219 Ansteuerung (im Artikel 810692 enthalten). ..	22
3.1.3. Zenerdiode.....	22
3.1.4. Solarpaneel.....	22
3.1.5. IR-Sender, Empfänger.....	23
3.2. Der Transistor.....	24
3.3. Der Feld-Effekt-Transistor (FET) und MOSFET's.....	26
3.4. Transistoren und die Grundsaltungen:.....	27
3.4.1. NPN-Transistor als Schalter.....	27
3.4.2. PNP-Transistor als Schalter.....	27
3.4.3. der FET als Schalter.....	27
3.5. Peltierelement.....	28
4. Anzeigeeinheit - alphanumerisches LC-Display.....	29
5. Sensoren.....	30
5.1. FSR-Sensor (siehe Artikel 811270 und 811271).....	30
5.2. Temperatursensoren.....	30
5.2.1. Der NTC (Heissleiter Artikelnummer 220800).....	30
5.2.2. Digitale Temperatursensoren.....	30
5.2.3. Pt100 und PT1000 (Artikelnummer 180053 und 180052).....	31
5.2.4. PTC (Kaltleiter).....	31
5.3. Der LDR.....	31
5.4. Boden Feuchtesensor.....	32
5.5. Drucksensoren.....	32

5.6. Dehnungsmessstreifen.....	32
5.7. PIR-Sensor.....	32
5.8. IR-Empfänger.....	32
5.9. Herzschlag-Sensor KY-039.....	33
5.10. Infrarot Lichtschranke KY033LT.....	33
5.11. Feuchte und Temperatursensor DHT11:.....	33
5.12. Magnetfeld-Sensor.....	33
5.13. der kapazitive Feuchtesensor KFS140-D.....	33
5.14. Gassensoren für brennbare Gase Rauch und Luftqualität.....	34
5.15. Erschütterungssensor / Vibrationssensoren.....	34
5.16. LED Modul.....	35
6. Aktoren.....	35
6.1. REED Kontakt (z.B. Bestell-Nr.: 420165).....	35
6.2. Reed-Relais.....	35
6.3. Servomotor.....	35
6.4. Gleichstrommotor (Rotationsmotor).....	36
6.5. Schrittmotor.....	36
7. Module.....	37
7.1. Tastenfeld mit Touch:.....	37
7.2. RTC Modul PCF8563:.....	37
7.3. DCF77 Zeitzeichenempfänger.....	37
7.4. Beschleunigungsmodul.....	38
7.5. Ultraschall Modul –.....	38
7.6. Infrarot-Abstandsensor.....	38
7.7. Kraftsensor.....	38
7.8. Farbsensor.....	39
7.9. Luftdrucksensor.....	39
7.10. SD-Speicherkartenmodul.....	39
7.11. Relais Modul.....	39
7.12. Motorantriebsmodul 9110.....	39
7.13. Stromsensor - Strom-Wandlermodul.....	40
7.14. Einphasen AC Spannungswandler Modul.....	40
7.15. HMC5883 Kompassmodul.....	40
7.16. RFID MFRC522.....	40
7.17. WLAN-ESP8266-01.....	41
7.18. Bluetooth-Modul.....	41
7.19. CAN-Modul.....	41
7.1.20. Verstärkermodul.....	41
7.21. I/O Modul: Buserweiterung.....	42
7.22. Level-shifter.....	42
7.23. Wägezelle.....	42
7.24. Fingerprint-Modul.....	42
8. Der ATMEGA – ein viel verwendeter Mikrocontroller.....	43
8.1. Der ATMEGA328 im Arduino Nano und Arduino uno.....	43
9. Die ARDUINO IDE (Entwicklungsumgebung).....	49
9.1. Installation.....	49
9.1.2. Programmstart.....	50
9.1.3. Installation eines Treibers.....	51
9.1.3.1. bei Original Arduino Board:.....	51
9.1.3.2. bei Arduino kompatiblen Board:.....	51

9.2. Bibliotheken.....	51
9.2.1. Bibliothek einbinden.....	51
9.2.2. Beispieldatei der Bibliothek laden.....	52
9.3. Programmbeispiel.....	52
9.3.1. Programmbeispiel öffnen.....	52
9.3.2. Programmaufbau.....	52
9.3.3. Programm in den Arduino schreiben.....	53
9.3.4. Werkzeug: serieller Monitor.....	53
9.4. Voreinstellungen.....	54
9.5. Objektorientierte Programmierung.....	55
9.6. Der Arduino aus der Sicht der Arduino IDE.....	56
9.6.1. Digitale I/O Pins:.....	56
9.6.2. Interrupt Pins: 2 und 3.....	57
9.6.3. Synchrone Serielle Datenübertragung:.....	57
9.6.4. Die analogen Eingänge.....	57
9.6.5. Die Ports des Arduino Nano:.....	57
9.7. Bibliotheksfunktionen.....	59
9.8. Schleifen und Kontrollstrukturen:.....	60
9.8.1. Die if-Bedingung.....	60
9.8.2. Die while-Schleife.....	60
9.8.3. Die <i>for – Schleife</i>	61
9.8.4. Die switch-Anweisung:.....	61
9.8.5. Die ASCII-Tabelle.....	62
10. Grundlagen des binären Zahlensystems.....	63
10.1. Bitoperationen:.....	63
10.1.1. Bitweise UND-Verknüpfung mit &.....	63
10.1.2. Die Bitweise ODER-Funktion mit 	64
10.1.3. Die Bitweise NOT-Funktion mit ~.....	64
10.1.4. Die Bitweise XOR-Funktion mit ^.....	64
10.1.5. Schiebe (Shift oder Rotate) Operationen.....	65
10.2. Zuweisungsoperatoren.....	65
10.3. Die Operatoren bei booleschen Abfragen (if-Abfragen).....	66
11. Die Programmierung des Arduino in der Praxis.....	67
11.1. Einführendes Beispiel: blink.ino.....	68
11.1.1 Beispiel: Blinkende LED.....	68
Zur Schaltung:.....	68
Flussdiagramm für Befehlsabfolge.....	69
11.1.2. Beispiel 2: alternierende LEDs: blink2.ino.....	71
11.1.3. LED Licht mit Taster ein und ausschalten: Blink5.ino.....	75
11.1.4. Reaktionstester: blink6_zufall_reaktion.ino.....	77
11.1.5. Kopf oder Zahl.....	78
11.2. Der LDR Widerstand.....	80
11.2.1 Arduino als Helligkeitsanzeige.....	80
11.2.2.Helligkeitsanzeige mit mehreren Schaltschwellen:.....	82
11.2.3.Geheimfachwächter: Schublade.ino.....	83
11.3. Übungen mit Lauflicht.....	86
11.3.1. Schlag die Fliege.....	87
11.3.2. Erweiterung mit 2 Tastern (Tennis.ino).....	90
11.3.3. binärer Würfel.....	90
Was bedeutet LSB und MSB?.....	92

11.3.4. Lauflicht (Beispiel: Lauflicht.ino).....	93
11.3.5. FSR-Sensor (Artikel 811 164) steuert LED band (FSR_beispiel.ino).....	95
11.4. LED-Dimmen.....	97
11.4.1. einfaches Dimmen.....	98
11.4.2 dimmen mit PWM.....	100
11.6.3 Dimmen durch steuern der PWM mit Poti (PWM_dimmung_poti.ino).....	100
11.4.3. Wechselblinker am Andreaskreuz.....	101
11.4.5. Leuchtturm.....	102
die verschachtelte for – Schleife:.....	104
11.4.6. Kerzenlicht.ino.....	105
11.4.7. Stroboskop.....	106
11.5. Servo (Artikel 820 570) ansteuern.....	109
11.5.1. Servo mit Poti ansteuern.....	110
11.5.2. Teetimer.....	111
11.5.3. der NTC (NTC-10k Artikelnummer 220757).....	112
11.5.4. Temperaturmessung mit Servo anzeigen.....	114
11.6. RGB LED.....	115
11.6.1. Grundlegendes.....	115
11.6.2. Programmbeispiele für die analoge RGB-LED.....	116
11.6.3. Programmbeispiel für digitale RGB-LED.....	117
11.7. Tonausgabe:.....	118
11.7.1. Kurzeittimer (mit Taster bestückt).....	119
11.7.2. Martinshorn Martinshorn.ino.....	119
11.7.3. vorbeifahrendes Martinshorn (Dopplereffekt) Martinshorn_doppler.ino.....	119
11.7.4. US Polizeisirene: US_police.ino.....	119
11.7.5. Frostwarner-Simulation Frost_sim.ino.....	119
11.7.6. Ampelschaltung für Fußgänger mit Summer.....	122
Komplexere Beispiele.....	124
11.8. IR Empfang.....	124
11.8.1. Das Übertragungsprotokoll.....	124
11.8.2. Decodieren des IR-Signals.....	125
12. Diverse Module an den Arduino anschließen.....	127
12.1. PIR-Sensor (Artikelnummer 811274).....	127
12.2. Tastenfeld (Artikelnummer 421221).....	128
12.3. RFID (Artikelnummer 810678).....	129
12.4. SD Karte (Artikel 810359).....	130
12.5. Farben erkennen mit dem Farbsensor TCS3200 (Artikelnummer 810681).....	131
12.6. Beschleunigungssensor MPU5060 (Artikelnummer 811107).....	133
12.7. Herzschlagsensor (Artikelnummer 810917).....	135
12.8. Ultraschallsensor (Artikelnummer 810481).....	136
12.8.1. Durchgangshöhenmesser.....	136
12.8.2. Personenzähler.....	137
12.8.3. Einparkwarner.....	138
12.9. Infrarotsensor (Artikel 810480).....	140
12.10. Infrarotlichtschranke SEN-KY033LT (Artikel 811269).....	141
12.11. Temperaturmessung mit PT1000.....	143
12.12. MOSFET-Treiber (Artikelnummer 810329).....	146
12.13. Die Sieben Segment Anzeige.....	147
Aufbau und Funktionsweise.....	147
12.13.1. erstes Beispiel.....	148

12.13.2. Uhrzeit mit LED-Modul (Artikelnummer 810453).....	150
12.13.3 Countdown timer.....	152
12.13.3. Uhrzeit mit Digitalanzeige-Modul (Artikelnummer 810568).....	153
12.14. Luftdruckmesser (Artikel 810914).....	155
12.15. OLED Displaymodul (Artikel 811108).....	156
12.16. Gassensoren (Beispiel CO-Sensor 811162).....	157
12.17. DHT11 Feuchte und Temperatursensor (Artikel 810914).....	159
12.18. LC-Display (Artikelnummer 121738).....	160
12.18.1. Uhrzeit anzeigen.....	160
12.18.2. Uhrzeit Stellen mit Joystick (Artikelnummer 810922).....	161
12.18.3. Beispiel mit codiertem Drehschalter (Digitalencoder Artikelnummer 810923).....	162
12.18.4. Uhrzeit mit RTC Baustein (Artikelnummer 810515).....	163
12.19. Hallsensor (Artikelnummer 810913).....	164
12.20. Schrittmotor (Artikelnummer 310543).....	165
12.21. analoger Vibrationssensor (Artikelnummer 811421).....	166
12.22. Lüfteransteuerung.....	167
12.22.2. Drehzahlbestimmung mit Stroboskop (PWM_Motor_LED.ino).....	168
12.23. Wägezelle mit HX711 (Artikelnummer 811419).....	169
12.24. RGB-LED-Matrix (Artikelnummer 810664).....	170
12.24.1. Farbenspiel:.....	170
12.24.2. Würfelspiel (gleicher Aufbau wie 12.22.1.).....	171
12.24.3. Zeilen und Spalten ansteuern (gleicher Aufbau wie 12.22.1.).....	171
12.25. MAX712 LED Matrixanzeige (Artikelnummer 810692).....	172
12.26. Fingerabdruck-Sensor (Artikelnummer 180132).....	173
12.27. Der Interrupt Befehl.....	175
12.28. WiFi modul ESP12 (Artikelnummer 811152).....	177
12.29. Bluetooth-Modul (Artikelnummer 810928).....	181
12.29.1. Aufbau: Datenübertragung und schalten einer LED.....	181
12.29.2.1. Kommunikation Smartphone ← → Arduino.....	183
12.29.2.2. Installation und Konfiguration der Android - APP.....	184
12.29.2.3. Texte oder Werte übertragen:.....	185
12.30. Ansteuerung eines Grafik-Display(Artikelnummer 121829).....	186
12.31. Speedsensor (Artikelnummer 810878) mit Motor (820 580).....	187
12.32. Motorantriebsmodul L9110 (Artikelnummer 810 572).....	188
12.33. Joy-IT Soundrecorder (Artikelnummer 810679).....	190
12.34. Ein touchmodul am Arduino.....	192
12.34.1 DAYCOM Tastaturmodul TTP229 mit 16 Tasten (Artikelnummer 810271).....	192
12.34.2. Kapazitives Berührungsschalter Modul DAYPOWER TTP-223.....	193
12.35. Staubsensor-Modul GP2Y1014AU (Artikel-Nr.: 811028).....	194
12.37. Ansteuerung des Raspberry Pi Blink! Moduls von pimoroni.....	195

Erklärung der physikalischen Grundlagen und der angewandten Bauelemente

1. Grundlagen und Historisches

Spannung und Strom:

Da Strom nicht sichtbar ist, ist es schwer ihn zu beschreiben. Das funktioniert in der Physik am Besten mit Modellen oder mit dessen Auswirkung:

Da gibt es die Wärmewirkung: Heizdecken, Föhn, Bügeleisen, Waschmaschine, Geschirrspüler usw. Der Strom kann eine Bewegung erzeugen in Relais und in Motoren;

Es gibt auch noch die chemische Wirkung des Stromes. So z.B. bei der Galvanisierung oder des Verzinken von Metall. Es gibt Vermutungen, dass die Batterie von Bagdad schon in der Antike zur Vergoldung verwendet wurde. Als erster, der die elektrische Spannung erkannt hat, wird in den Geschichtsbüchern Alessandro Volta erwähnt. Zuerst hat er mit Froschschenkeln experimentiert. Dabei verwendete er die nach ihm benannte Voltasäule.

Er erkannte, dass zwischen zwei unterschiedlichen Metallplatten, eine elektrische Spannung entsteht. Die beiden Metalle (z.B. Kupfer und Zink) werden durch einen mit einem Elektrolyt getränkten Stoff, Leder oder Pappe voneinander getrennt. Das unedlere Metall (z.B. Zink) löst sich mit der Zeit auf, es oxidiert und zersetzt sich. Die Zinkatome geben Elektronen ab und wandern als Zinkionen (positiv geladen) in die Elektrolyt-Lösung. Die abgegebenen Elektronen bleiben dabei in der Zinkplatte zurück, dort entsteht also ein Elektronenüberschuss – das Zink ist nun negativ aufgeladen (= Minuspol). In der Kupferplatte findet dieser Prozess ebenso statt. Dabei entstehen allerdings weniger Elektronen als beim Zink. Im Vergleich zum Zink befinden sich im Kupfer wesentlich weniger Ionen, also ist Kupfer gegenüber dem Zink positiv aufgeladen (= Pluspol). Verbindet man nun Minuspol und Pluspol, fließen die überschüssigen Elektronen als elektrischer Strom von der Zinkplatte zu der Kupferplatte, bis ein Ausgleich stattgefunden hat. Dieser Stromfluss findet so lange statt, bis das Zink sich vollständig zersetzt hat. Nach diesem Prinzip funktionieren auch heute noch viele Batterien.

Aufgrund dieser Entdeckung von Volta, wird die Einheit der Spannung nach ihm benannt. Eine Steckdose liefert eine Spannung von 230V. Allerdings ist die Spannung nur von ihrer Wirkung vergleichbar. Denn im Gegensatz zur Voltasäule, kommt aus der Steckdose keine Gleichspannung, sondern eine Wechselspannung.

Eine der wichtigsten Wirkungen des Stromes ist der Magnetismus. Der erste der das Prinzip des Elektromotors erkannte war Michael Faraday im Jahr 1821. Am meisten bekannt dürfte er jedoch sein, weil das Prinzip des Faradayschen Käfigs nach ihm benannt ist.

1834 gab Moritz Hermann Jacobi in Potsdam bekannt, dass er „gegenwärtig einen Apparat anfertigen lasse, um Versuche über den mechanischen Effekt anzustellen, den man durch die elektromagnetische Erregung im weichen Eisen erlangen kann“. Der Motor hatte ein Gewicht von ca. 25 kg und lieferte eine Leistung von ca. 15W.

Den ersten wirklich real nutzbaren **Gleichstrommotor** entwarf Moritz Hermann von Jacobi im Jahr 1838. Am 13. September 1838 bewegte er auf der **Newa** in **St. Petersburg** ein **Boot**, das ein Motor mit einer Leistung von ca. 220 **W** angetrieben hat und mit ca. 2,5 km/h eine ca. 7,5 km lange Strecke zurücklegte. 1839 konnte er die mechanische Leistung seines Motors auf 1 kW erhöhen und erreichte mit dem Boot Geschwindigkeiten bis etwa 4 km/h.

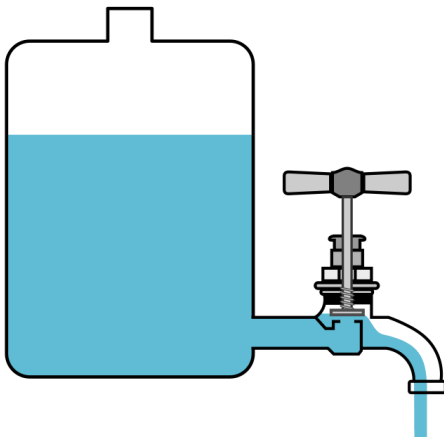
1.1. Physikalische Größen

1.1.1. Spannung

Beim Studium hat uns der Professor nahegelegt, immer in Modellen zu denken. Der ganze Bereich der Elektronik ist sehr komplex. Das beginnt schon bei den grundlegendsten Dingen, wie Spannung Strom und Widerstand.

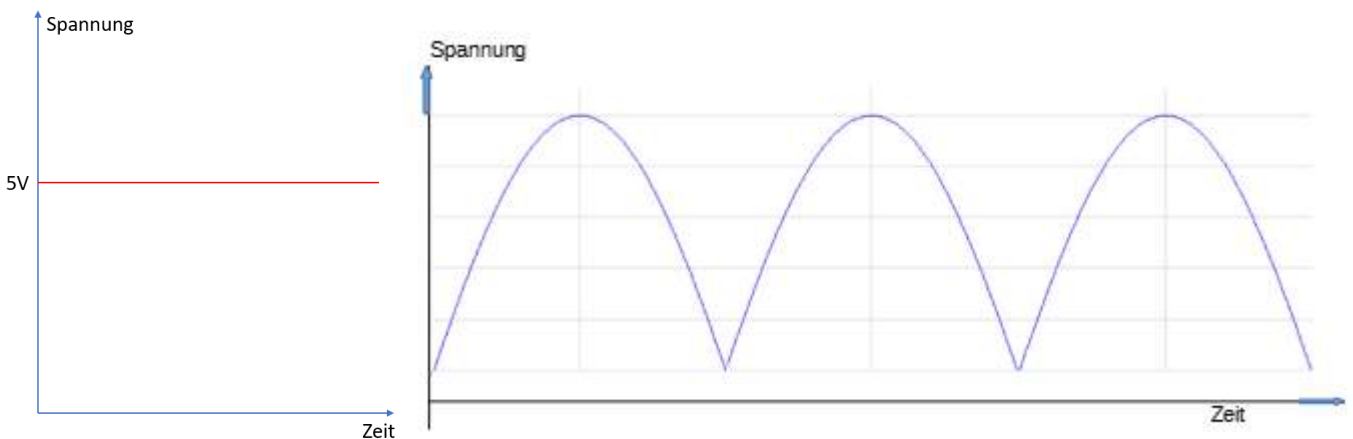
Die Spannung kann man sich vorstellen, wie ein Wasserbecken. Je höher der Pegelstand im Becken, desto höher die „Spannung“ der Batterie. Die Größe des Beckens ist vergleichbar mit der Kapazität der Batterie. Das Abflussrohr ist der Widerstand, der die Stromstärke bestimmt. Je größer der Durchmesser des Rohres, desto kleiner der Widerstand. Das bedeutet, desto mehr Wasser fließt aus dem Becken. Die Menge an Wasserdurchsatz ist quasi proportional dem Stromfluss.

Als physikalische Einheit wurde zum Gedenken an Alessandro Volta, das Volt [V] definiert.



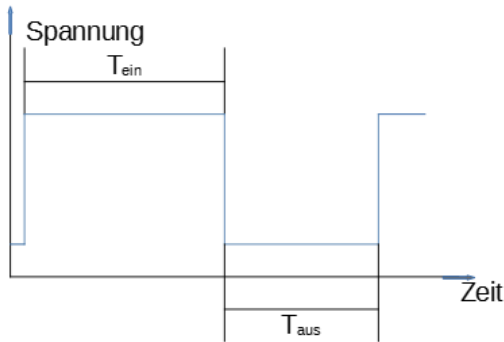
Welche Arten von Spannung gibt es

1.1.1.1. Gleichspannung



Das Wesen eines Gleichspannungssignals ist, dass der Strom immer in die gleiche Richtung fließt. Dies gilt auch für die pulsierende Gleichspannung aus einem billigen Steckernetzteil.

1.1.1.2. getaktete Gleichspannung



Mit dem Arduino kann man die Spannung ein und ausschalten. Dies kann man auch periodisch machen. Das Verhältnis von Einschalt- zu Ausschaltzeit bestimmt den Effektivwert der Spannung. So ist es möglich die Spannung von 0V ... 5V beliebig zu variieren. Dies nennt man Pulsweitenmodulation. So kann man eine LED in der Helligkeit verändern, ohne einen Widerstandswert zu verändern. Durch das ein und ausschalten kann man einen Summer anschließen und verschiedene Töne erzeugen.

dB(A) Kurve aufnehmen und schauen welcher, als der lauteste Ton empfunden wird.

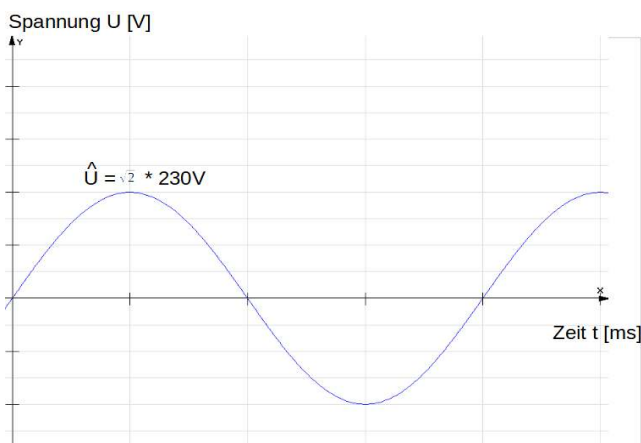
Das obige Signal kann auch durch Ein- und Ausschalten eines Pins am Arduino erzeugt werden. Zuerst muss der gewünschte Pin als Ausgang definiert werden. Er wird für die Zeit T_{ein} auf logisch High (eingeschaltet), dann erfolgt das Ausschalten des Pins für T_{aus} . Dieses ein und ausschalten wird mit dem Prozessor beliebig wiederholt und so ergibt sich das oben dargestellte Signal.

Die Frequenz des Signals ergibt sich aus dem Kehrwert der Summe von T_{ein} und T_{aus} . Das Verhältnis von $T_{\text{ein}} / T_{\text{aus}}$ nennt man Puls Pausenverhältnis. Dieses bestimmt den Effektivwert der Spannung. Wenn T_{aus} gegen Null geht, steigt der Wert bis zum Maximum der Ausgangsspannung des Prozessors von ca. 5V.

In der obigen Grafik ist zu erkennen wie mit zunehmender Ausschaltzeit der U_{eff} gegen Null geht. Der Effektivwert einer Wechselspannung entspricht dem Flächeninhalt des Signalverlaufs über der Zeitachse.

Diese Erkenntnis ist später von Bedeutung, wenn man mit einer erzeugten Rechteckspannung die Helligkeit einer Leuchtdiode oder die Drehzahl eines Gleichstrommotors verändern kann. Genauso kann man die Stellung eines Servomotors verändern. Aber dazu später mehr.

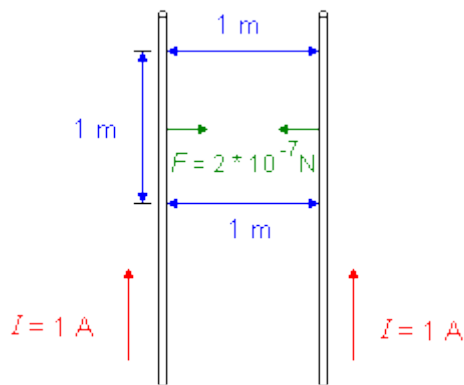
1.1.1.3. Wechselspannung



Die bekannteste Wechselspannung ist die Netzspannung an unserer Haushaltssteckdose. Diese verändert 50 mal in der Sekunde die Polarität. Sie ist ein Sinussignal. Der Stromfluss verändert periodisch seine Flussrichtung! Um die Spannungen, mit unterschiedlichen Kurvenformen, einigermaßen vergleichen zu können, benutzt man bei Wechselspannungen den sogenannten Effektivwert. Nur so ist es möglich die Wirkung der Spannungen vergleichen zu können. Dieser Wert ist unabhängig vom Kurvenverlauf des Zeitsignals.

Der Effektivwert ist der Wirkleistung proportional; der Effektivwert ist der Flächeninhalt unterhalb der Sinuskurve. Der Effektivwert kann sich z.B. zeigen, durch unterschiedliche Helligkeit einer Lampe oder LED-Leuchte.

1.1.2. Strom



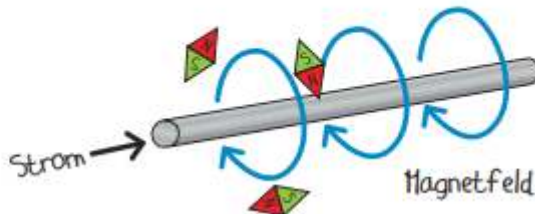
Die Einheit des elektrischen Stromes ist das Ampere [A] und ist folgendermaßen definiert:

Sind zwei elektrische Leitungen unendlich lang und vernachlässigbar dünn, haben einen Abstand von genau einem Meter, so fließt in beiden Leitern genau dann 1 A Strom, wenn zwischen den Leitern eine Kraft von $2 \cdot 10^{-7}$ N wirkt. Wer dies nachweisen will, braucht viel Geduld und eine teure Ausrüstung.

1898 wurde 1 A so festgelegt, dass bei einem Strom von 1 A in einer Sekunde 1,118 mg Silber in einer Silbernitratlösung abgeschieden werden soll.

Ampère wusste von den Versuchen von Christian Oersted, dass eine Magnethöhle in der Nähe einer elektrischen Leitung abgelenkt wird, sobald Strom durch einen Leiter fließt.

Gestreckter Draht:

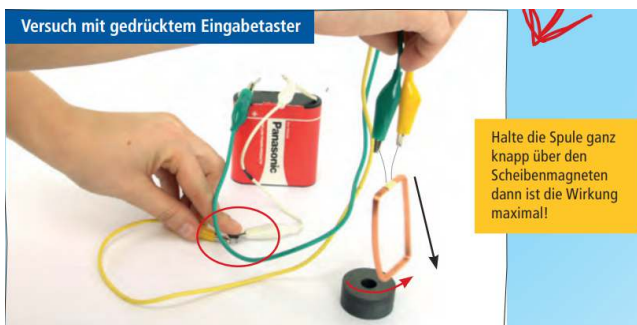


Im Jahr 1820 wiederholte Ampère diesen Versuch und bemerkte dabei, dass Oersted die Ablenkung des Magneten durch das Erdmagnetfeld unbeachtet liess. Ampère verbesserte seine Versuchsanordnung. Nun konnte er feststellen, dass sich die Magnethöhle immer senkrecht zum stromdurchflossenen Leiter ausrichtet.

Ampère vermutete, dass der Magnetismus, seine Ursache im elektrischen Strom hat. Der elektrische Strom erzeugt ein magnetisches Feld. Er konnte in seinen Versuchen nachweisen, dass zwei stromdurchflossene Leiter Kräfte aufeinander ausüben. Zwei parallel Leiter ziehen einander an, wenn der Strom in beiden Leitungen die gleiche Richtung hat. Es addieren sich die Felder, so funktioniert auch ein Elektromagnet. Ampère konstruierte ein Gerät zum Messen der Stromstärke, das **Galvanometer**.

Im Jahr 1822 beschäftigte sich Ampère mit der Kraft, die zwei nahe beieinander liegenden stromdurchflossene Leiter aufeinander ausüben. Er konnte zeigen, dass diese Kraft umgekehrt proportional ist zum Abstand der beiden Leitungen.

Ampère erklärte den Begriff der elektrischen Spannung und des elektrischen Stromes und setzte die technische Stromrichtung fest.



Ampère glaubte, dass das **Magnetfeld der Erde** durch starke elektrische Ströme entsteht, die in der Erdkruste von Osten nach Westen fließen.

Links im Bild ist ein Versuch aus der Zeitschrift *Faszination Elektronik* abgebildet.

<https://www.faszination-elektronik.de/wp-content/uploads/2016/10/Faszination-Elektronik-2014-1.pdf>. Dabei ist gezeigt, wie durch einfache

Mittel, die magnetische Wirkung des Stromes experimentell nachgewiesen werden kann.

1.1.3. Der Widerstand als physikalische Größe (Ohm)

der Widerstand bestimmt den Strom. Je kleiner der Widerstand desto größer der Strom. Damit ich den Strom zum Heizen verwenden kann, braucht es einen kleinen Widerstand. Deshalb gibt es bestimmte Widerstandsdrähte. z.B Wolfram in den Glühlampen; oder den Heizdraht im Kochfeld des Elektroherdes, oder eine Heizspirale in der Waschmaschine.

Die Einheit des Widerstandes wird als Ohm [Ω] definiert.

Georg Simon Ohm's Name ist in die Geschichte eingegangen, als er das Ohmsche Gesetz definierte. Dieses beschreibt die Abhängigkeit zwischen Spannung und Strom an einem Widerstand, bzw. in einem elektrischen Stromkreis.

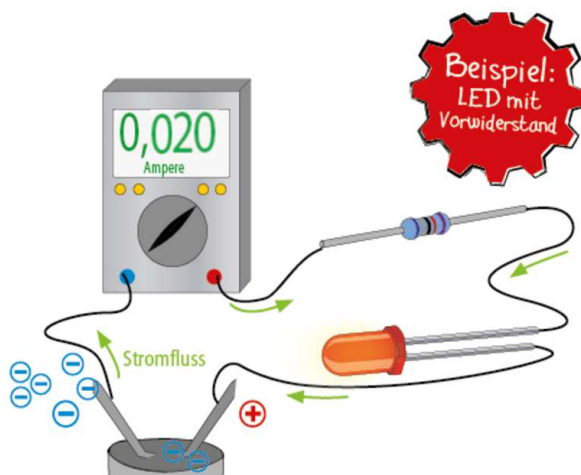
Dabei kann man sich vorstellen, wie Herr Ohm ein Bier aus einem Fass zapft. Je nachdem wie er den Hahn aufdreht oder zudreht, kann er den Bierfluss (Stromfluss) regulieren. Geöffneter Hahn bedeutet geringen Widerstand. Je weiter Herr Ohm den Hahn zudreht, desto größer wird der Widerstand.

Es gibt verschiedene Arten und unterschiedliche Bauformen von Widerständen. Die am meisten verwendeten sind Festwiderstände, um z.B. den Strom durch Leuchtdioden zu begrenzen, damit diese nicht kaputt gehen. Es gibt veränderbare Widerstände, sogenannte Potentiometer. Die Festwiderstände haben verschiedene Leistungsklassen und dadurch verschieden große Bauformen. SMD Widerstände werden direkt mit ihrer Oberfläche auf ein Trägermaterial (Leiterplatte) gelötet. Dann gibt es die älteren sogenannten bedrahteten oder THT Widerstände, die für Steckbretter oder Pollin Bausätze verwendet werden. Diese sind größer und leichter zu handhaben. Aber sie wurden zunehmend vom Markt verdrängt, weil die SMD Bauteile von Maschinen leichter zu bestücken sind und viel weniger Platz brauchen. Denn nur so war es möglich die Fernseher, Handys etc. immer kleiner und vor allem billiger fertigen zu können.

Bei den Potentiometer unterscheidet man Dreh-Potis und Schiebe-Potis. Zudem gibt es logarithmische und lineare Potis. Die logarithmischen verwendet man vorwiegend im Audiobereich, weil da kann man durch kleine Verstellung der Poti-Position, den Wert des Potis schnell ändern.

Mit der Größe des Widerstandes lässt sich die Helligkeit der LED steuern.

Ein Stromfluss ist allerdings nur möglich, so lange der Stromkreis geschlossen ist.



2. passive Bauteile

Generell ist der Begriff passive Bauteile so festgelegt, dass diese Bauteile in ihrer Anwendung keine Verstärkungswirkung und keine Steuerungsfunktion bewirken. Diese sind Widerstände, Kondensatoren und Induktivitäten (Spulen, Drosseln, Übertrager bzw. Transformatoren).
Wenden wir uns zunächst den wichtigsten Vertretern zu:

2.1. Der Widerstand

2.1.1. der ohmsche Widerstand

Im Kapitel 1.1.3. wurde gezeigt, wozu unter anderem, ein Widerstand benötigt wird. Der Strom durch eine Leuchtdiode darf einen bestimmten Wert nicht überschreiten. Deshalb wird dazu ein sogenannter Vorwiderstand benutzt. Die Werte der Widerstände werden mit Farbringen unterschieden, sofern sie der Bauform 0207 entsprechen. Die Bauform 0204 ist wesentlich kleiner und verträgt daher weniger Strom.

Es gibt eine Widerstandsuhr mit der Artikelnummer 220668. Damit kann der Wert eines Widerstandes einfach bestimmt werden.

Widerstandsringe:



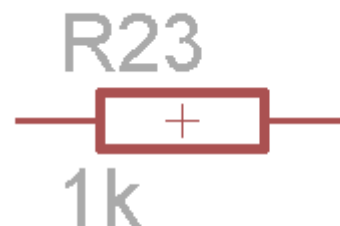
Links im Bild ist eine Farbcodetabelle dargestellt.

Damit ist es auch möglich die Werte von Widerständen zu bestimmen:

Die Widerstände haben eine unterschiedliche Anzahl von Ringen, je nach Toleranz der einzelnen Werte. Die Kohleschichtwiderstände haben 5% Toleranz und dazu reichen vier Ringe. Die Metallschichtwiderstände haben 1% Toleranz und besitzen fünf Farbringe.

Natürlich gibt es nicht jeden beliebigen Wert auch als Widerstand.

Das Schaltzeichen eines Widerstandes:



Im Schaltplan wird nicht nur angegeben, welchen Wert der Widerstand hat. Er bekommt auch eine Bezeichnung z.B. R23; R bedeutet Widerstand und mit der Nummer werden alle Bauteile durchgezählt. Damit jedes Bauteil in der Schaltung nur einmal vorkommt. Das ist wichtig für die Stückliste und später bei der Fehlersuche.

Der Wert entspricht somit : $1\ 0\ 0 * 10 = 1000R$ oder $1k$

Da es ein zu großer Aufwand wäre, für jeden in einer Schaltung berechneten Widerstandswert auch einen Widerstand zu fertigen, wurden sogenannte Widerstandsreihen festgelegt. Ähnlich wie bei Geldscheinen, wo es immer nur 1er 2er und 5er Werte gibt, sind auch die Widerstände logarithmisch unterteilt.

Der Unterschied zu den Geldwerten ist, dass die Anzahl der verfügbaren Widerstandswerte in einer Dekade in der sogenannten E-Reihe festgelegt ist. Eine Standard Reihe ist die E24 Reihe. Das bedeutet z.B. von 100 bis 1000 ist der Wertebereich logarithmisch in 24 Werte aufgeteilt: E24 bedeutet 24-te Wurzel aus 10;

$(\sqrt[24]{10})^m$ m ist dabei ein Wert von 0 ... 23. Daraus ergeben sich folgende Werte: 100R, 110R, 120R, 130R, 150R, 160R, 180R, 200R, 220R, 240R, 270R, 300R, 330R, 360R, 390R, 430R, 470R, 510R, 560R, 620R, 680R, 750R, 820R, 910, 1kOhm

Je größer der Wert der E-Reihe, desto mehr Widerstände gibt es.

Der 4-Band Code zeigt ein Beispiel für einen Kohleschichtwiderstand mit 5%. Heute verwendet man überwiegend Metallschicht-Widerstände mit 1 % Toleranz. In großen Stückzahlen sparen die Kohleschicht-Widerstände Geld. Die Anwendung ob Kohle- oder Metallschicht ist also auch eine Preisfrage. Diese Widerstände sind für kleine Leistungen bis zu einem halben Watt bestimmt.

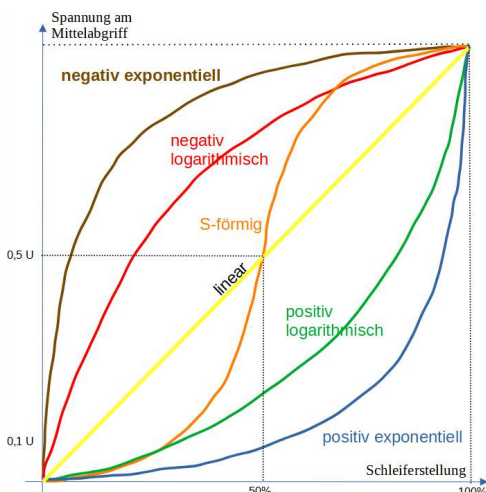
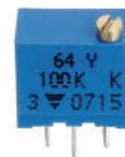
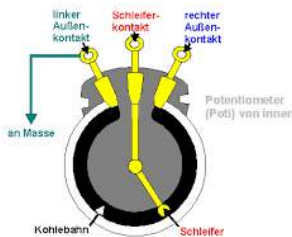
Bei SMD-Widerständen, die heutzutage standardmäßig in der Industrie verwendet werden, befindet sich auf dem Gehäuse ein Aufdruck, an dem sich der Widerstandswert ablesen lässt. Der Wert 103 bedeutet dann 10 und die 3 für drei mal 000, ergibt 10.000 Ohm also 10kOhm.

Bei höheren Leistungen mit 5W oder mehr, verwendet man Draht gewickelte Widerstände:



2.1.2. Das Potentiometer

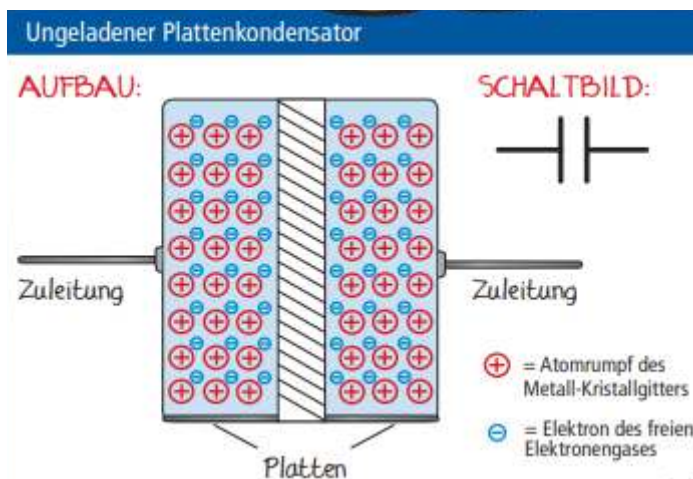
Eine Sonderform des Widerstandes, ist ein Potentiometer. Dabei hat das Poti, wie es auch genannt wird einen Nennwert, der dem Maximalwert entspricht. Durch Drehen an der Achse oder der Spindel, verändert sich der Wert an der Mittelanzapfung von 0 ... bis Nennwert. Es gibt viele unterschiedliche Bauformen, einige wenige sind nachfolgend abgebildet:



Ein weiterer wichtiger Punkt, an dem sich die Potis unterscheiden ist die Kennlinie:

Die meisten Potentiometer sind positiv logarithmisch. Das bedeutet, dass sich der Wert anfangs nur geringfügig ändert. Erst mit zunehmenden Drehwinkel ändert sich der Wert, dann aber immer mehr. Dies ist auch durchaus so gewollt, z.B. bei Lautstärkereglern, denn das menschliche Gehör hat vom Lautstärke Empfinden auch eine logarithmische Kurve. Eine große Verbreitung hat auch das lineare Potentiometer. Dieses ändert ihren Wert linear mit der Drehbewegung.

2.2. Der Kondensator



Der Kondensator besteht im Prinzip aus zwei parallelen Metallplatten. Damit lassen sich Ladungen speichern. Die Fähigkeit Ladungen zu speichern hängt in erster Linie von dem Material zwischen den beiden Platten ab, dem Dielektrikum. Aber der Kondensator dient in der Elektronik nicht nur dem Speichern von Energie. Er dient auch in Elektromotoren dazu eine künstliche Phase zu erzeugen, damit dieser sich beim Anlegen einer Netzspannung dreht. Der Kondensator diente früher im Radio und Fernseher dazu den Sender zu wählen. Er dient dazu Störungen auszufiltern.

Aber eine der wichtigsten Anwendungen ist in Kombination mit einer Spule einen Schwingkreis zu bauen. Diese war in Radioempfängern, Radargeräten, Fernsehempfängern, Funkgeräten wichtig, die Sende- und Empfangsfrequenz zu synchronisieren. Diese aufeinander abzustimmen, war die wichtigste Funktion.

So wie eine Feder und eine Masse in der Mechanik ein schwingendes System erzeugten, erzeugen in der Elektronik eine Spule und ein Kondensator eine Schwingfrequenz. Mit der Stärke der Feder und der Größe der Masse, kann man die Schwingfrequenz beeinflussen.

Genauso kann man mit Spule und Kondensator die Frequenz des Schwingkreises einstellen.

Eine Antenne ist nichts anderes als eine Spule die die Sendefrequenzen einfängt. Die Größe der Spule ist dabei Abhängig vom gewünschten Empfangsfrequenzbereich. Je kleiner die Frequenz, desto größer ist die Antenne. Deshalb war eine Langwellenantenne bei einem Sender früher so groß wie der Sendemast. Je größer die Frequenz desto kleiner die Antenne. Deshalb sind Satellitenschüsseln relativ klein weil die Wellenlänge des Sendesignals auch sehr sehr klein ist. Die Wellenlänge eines Signals wird bestimmt vom Dielektrikum in dem es sich ausbreitet und von der Frequenz. Je höher die Frequenz desto kleiner die Wellenlänge. Je höher das Dielektrikum desto kleiner die Wellenlänge. Deshalb wird Licht gebrochen, wenn es in Glas oder Wasser eintaucht. Zudem bestimmt die Lichtgeschwindigkeit noch den Wert der Wellenlänge.

Bauformen von Kondensatoren:

2.2.1. Keramikkondensatoren



Schaltbild:



Wertebereich üblicherweise einige pF ... nF; hier im Bild links 1nF; im Bild rechts ein SMD -typ

2.2.2. Folienkondensatoren



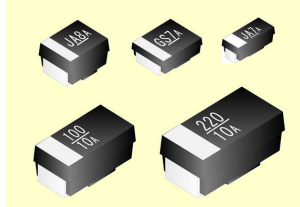
Schaltbild:



diese werden überwiegend unter extremeren Bedingungen besonders bei Impulsbelastungen eingesetzt. Der Typ rechts ist z.B. aus einer Dunstabzugshaube.

Keramik- und Folienkondensatoren sind sowohl für Betrieb an Gleichspannung wie an Wechselspannung geeignet.

2.2.3. Elektrolytkondensatoren



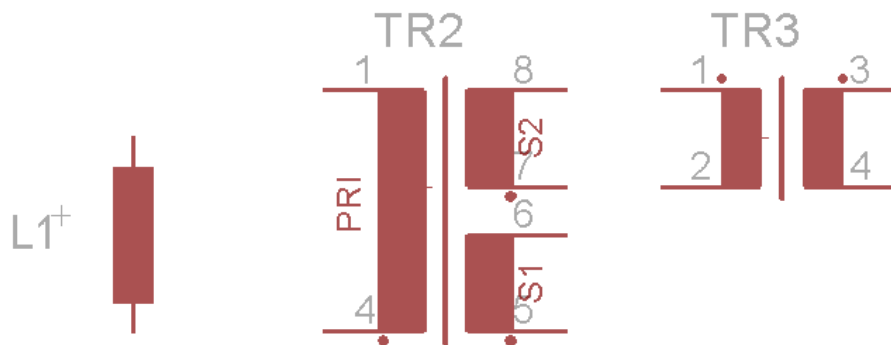
Diese werden überwiegend eingesetzt zum Speichern von Energie und zum Filtern/Stabilisieren von Signalen und Versorgungsspannungen. Die Bilder sind leider nicht Maßstab getreu. Der rechte Elko ist mit Schraubanschlüssen und hat einen Durchmesser von ca. 6 cm. Der Tantal-Elko ist der zweite von rechts und hat einen Durchmesser von weniger als 1cm.

Elko's dürfen nur an Gleichspannung betrieben werden. Deshalb ist es wichtig auf deren korrekten Anschluss zu achten. Verpolt angeschlossenen Elko's können explodieren und dabei großen Schaden anrichten!

2.3. Induktivitäten (Spulen)

2.4.1. Drosseln, Trafo, Übertrager

Schaltbilder:



L1 ist das Schaltsymbol für eine Drossel ohne Eisenkern; TR2 ist das Schaltsymbol für einen Transformator mit einer Primärwicklung und zwei Sekundärwicklungen für zwei galvanisch getrennte Spannungen. TR3 ist das Schaltsymbol für einen Trafo oder Übertrager. Der Übertrager ist ein Transformator für Signalspannungen, also für Spannungen mit wenig Strom auf der Sekundärseite. Der Punkt an jeder Spule signalisiert die Polung der Wicklungen des Übertragers. Beim Transformator ist dies nicht so entscheidend. Aber beim Übertrager ist es wichtig, zu wissen, mit welcher Phasenlage ein Signal übertragen wird. Manchmal ist es wichtig, dass das Eingangssignal in der gleichen Phasenlage übertragen wird. Manchmal kann es auch von nutzen sein, dass ein Signal in Gegenphase auf die Sekundärseite übertragen wird.

Eine Spule ist ähnlich wie ein Kondensator ein Energiespeicher. Allerdings speichert er nicht die elektrische Ladung wie ein Kondensator, sondern eine Spule hat die Fähigkeit die Energie im Magnetfeld zu speichern, welches durch einen Stromfluss aufgebaut wurde. Die Induktivität, wie diese Fähigkeit genannt wird hängt ebenfalls von einer physikalischen Materialkonstante, der magnetischen Leitfähigkeit, der Permeabilität ab.

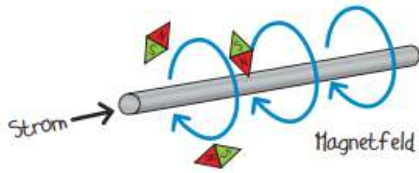
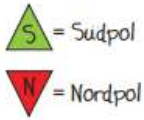
Spulen werden viel in Netzteilen zur Spannungsversorgung (interne Spannungserzeugung für Endgeräte) verwendet. Dazu dienen sie wie der Trafo/Übertrager zur Erzeugung oder der Transformation von Spannungen unterschiedlicher Größe, wie auch der Filterung von Störungen und auch der Anwendung in Schwingkreisen.



Links im Bild ist eine Luftspule von einem RFID System zu sehen. Dabei ist diese Spule gleichzeitig Sender und Empfänger. Sie erzeugt ein Magnetfeld und dieses wird von einer angeschlossenen Elektronik analysiert, wie es sich nach Abschalten der Spannung verhält. Im Empfangssystem wird die Energie des Magnetfeldes genutzt, die Elektronik zu versorgen und gleichzeitig mit zeitlich definierten Kurzschlüssen, Informationen zu übertragen.

2.4.2. Relais

Gestreckter Draht:



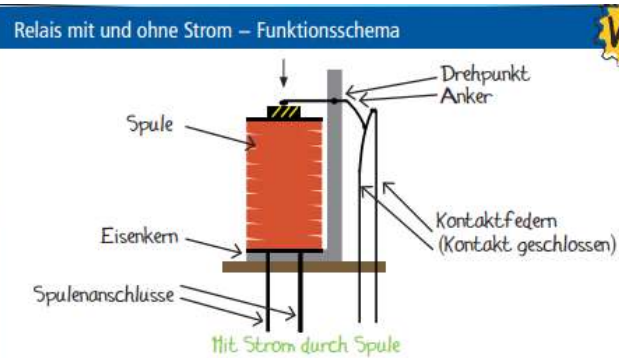
Spule:



Links im Bild ist zu sehen, wie eine Magnetenadel durch einen stromdurchflossenen Leiter abgelenkt wird. Wird der Einzelleiter zu einer Spule gewickelt, verstärkt sich der Effekt. Somit kann eine Spule, durch das erzeugte Magnetfeld eine Kraft auf Metallkörper in deren Umfeld ausüben.

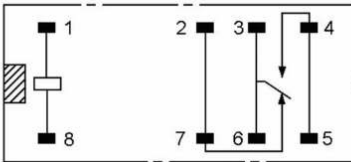
Eine besondere Form der Anwendung einer Spule ist in einem Relais.

Dabei wird das Magnetfeld benutzt, eine Kraft auf ein Stück Metall auszuüben. Ein Relais dient dazu, dass man mit einer kleinen Spannung und wenig Leistung, eine höhere Spannung und damit auch höhere Lasten und Ströme aus der Netzsteckdose schalten kann. So kann man ganz einfach durch eine Fernbedienung die Gartenpumpe oder das Licht ein und ausschalten. Links im Bild ist dargestellt, wie über einen Anker durch ein Magnetfeld zwei Kontakte geschlossen werden. Es gibt auch Relais, bei



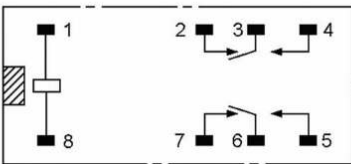
denen gleichzeitig, wenn ein Kontakt geschlossen wird, durch eine Hebelwirkung, ein anderer Kontakt geöffnet wird.

G2R1E-



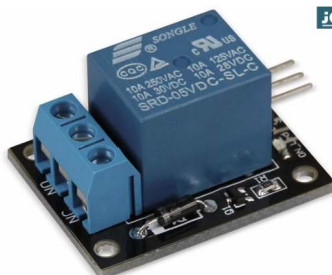
Hier links im Bild ist eine Innenbeschaltung von zwei Relaisstypen gezeigt. Dabei sind im oberen Bild die Kontakte parallel geschaltet. Somit kann im Vergleich mit dem darunter abgebildeten Relais, der doppelte Strom geschaltet werden.

G2R2-



Dafür kann mit dem links abgebildeten Relais zwei unterschiedliche Spannungspotentiale geschaltet werden. An Pin 1 und 8 wird die Spannung für das Schalten der Spule angeschlossen. Pin 2 und 3 und 7 und 6 sind im spannungslosen Zustand kurzgeschlossen. Im Datenblatt werden dann solche Pins

mit NC (normally closed = öffnender Kontakt) und die Pins 4 und 5 als NO (normally open = schließender Kontakt) bezeichnet. Denn mit Anlegen einer Spannung werden die Kontakte 3 und 4 sowie das Kontaktpaar 5 und 6 geschlossen.



Links ist ein Relais für einen Arduino, in der Mitte befindet sich ein KFZ Relais und rechts befindet sich ein Relais, wie es zu Steuerungsaufgaben in der Industrie verwendet wird.



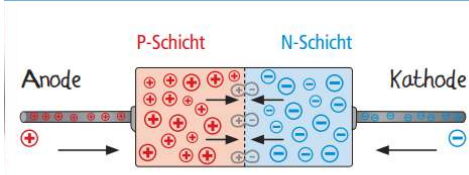
3.aktive Bauteile

3.1. Die Diode



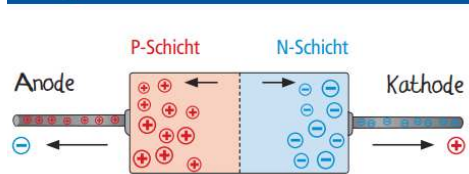
3.1.1. Gleichrichterdiode (Siliziumdiode)

Diode leitend



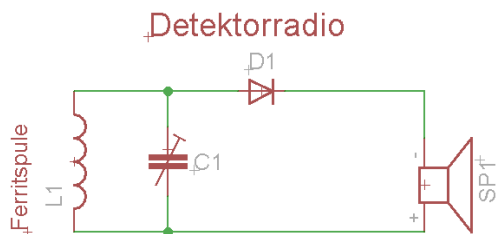
Die Diode besteht aus zwei nacheinander geschalteten, verschieden behandelten Zonen eines Siliziumkristalles. Durch gezielte Platzierung von unterschiedlichen Fremdatomen werden positive (+) und negative (-) Ladungsträger im Kristall erzeugt. Damit die Diode leitend wird, muss die Anode mit dem Pluspol und die Kathode mit dem Minuspol der Spannungsquelle verbunden werden. Weil sich gleichnamige Ladungen abstoßen, werden die positiven und die negativen Ladungen von den Batteriepolen auf die Trennfläche von N- und P-Schicht aufeinander zugetrieben, wo sie sich gegenseitig aufheben und so den Nachschub von weiteren positiven und negativen Ladungsträgern ermöglichen. So wird der Stromfluss am Laufen gehalten.

Diode sperrend



Polrt man die Diode andersherum, so saugen die Pole der Spannungsquelle die positiven und negativen Ladungsträger nach außen ab, also von der Trennschicht weg. Dadurch bildet sich um die Trennschicht herum eine isolierende Zone und der Stromfluss wird unterbrochen.

Dioden dienen in erster Linie zum „Gleichrichten von elektrischem Strom“. Dabei muss Gleichrichtung nicht nur bedeuten, dass in einem Netzteil die Spannung aus der Sekundärwicklung eines Transformators gleichgerichtet wird. Es kann sich auch bei Signalen um eine Gleichrichtung

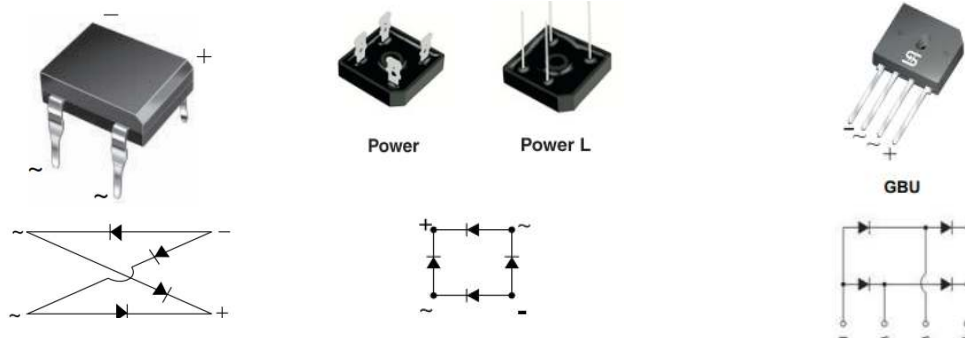


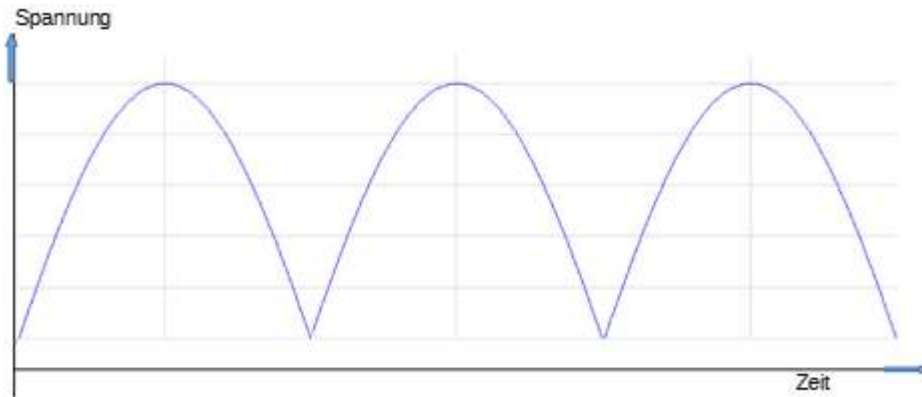
handeln, so wie in der einfachsten aller Radioschaltungen: der Detektorschaltung. Damit konnte man in den Anfängen des Radiozeitalters einen ganz einfachen Empfänger bauen. Man hatte eine Ferritantenne L1, einen Drehkondensator C1 zum Verändern der Schwingfrequenz, eine Diode D1 zum Ausfiltern des Wechselstromes und einen Kristallohrhörer SP1.

So konnte im Kurzwellen und Mittelwellenbereich Radio gehört werden. Der UKW Bereich konnte damit nicht abgedeckt werden, weil der Signalaufbau etwas komplizierter ist, als bei der Amplituden Modulation. Aber die ersten Radios waren ja nur ausgelegt für die Kurzwelle. So wäre am 31.Januar 1925 in Deutschland eine Rundfunkübertragung aus den USA zu hören gewesen.

Eine Sonderform des Gleichrichters ist der **Brückengleichrichter**; vier Gleichrichterdioden ergeben einen Brückengleichrichter;

Diese gibt es in unterschiedlichen Bauformen und Baugrößen, entsprechend der Anwendung und Leistung.





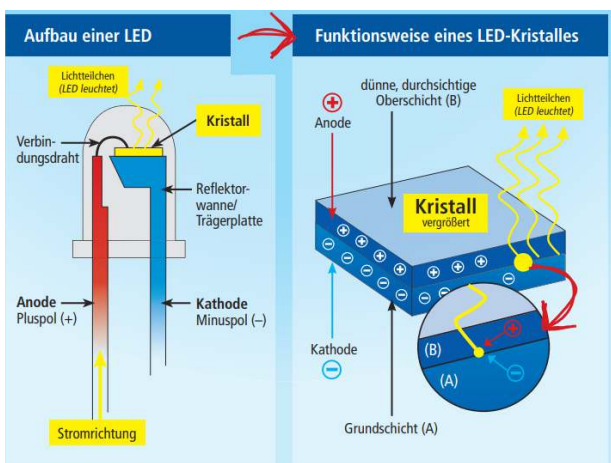
Im Bild oben ist der Signalverlauf dargestellt, wie er nach einem Brückengleichrichter zu messen ist. Bei einer Einweggleichrichtung (=Gleichrichter mit nur einer Diode) würde jede zweite Halbwellen wegfallen. Damit würde der Effektivwert erheblich niedriger sein. Mit einem Brückengleichrichter gewinnt man also mehr Leistung für seine elektronische Schaltung. Diese Schaltung wird in Netzteilen verwendet, die einen Netztrafo besitzen.

Bei einer Gleichrichterdiode aus Silizium fällt eine Spannung von 0,7V ab. Dies kommt davon, dass eine Diode auch einen Innenwiderstand hat. Bei einer Schottky ist der Spannungsabfall nur bei ca. 0.3V, weil der pn-Übergang aus einem speziellen Metall besteht. Allerdings hat dieses Metall wieder negative Eigenschaften in anderen Anwendungsbereichen, weshalb diese nicht überall verwendet werden kann.

Es gibt noch Kleinsignaldiode, die in der Signalverarbeitung z.B. bei Radios, Fernsehern, Messschaltungen u.a. verwendet werden.

Der Innenwiderstand einer Diode ändert sich mit abhängigkeit vom Strom der durch sie hindurch fließt. Mit zunehmendem Strom nimmt auch der Spannungsabfall zu. Jedoch steigt der Spannungsabfall nicht proportional wie beim ohmschen Widerstand.

3.1.2. die Leuchtdiode (LED)



Ein Diodentyp, der hier in diesem Kit besonders häufig verwendet wird, ist die Leuchtdiode (LED). Dieser Effekt wurde 1962 entdeckt. Dabei wird die Diode mit einer bestimmten Spannung „angeregt“. Dabei werden Photonen (Lichtteilchen) freigesetzt. Je nach Aufbau und Zusammensetzung der Kristallstruktur dieser Dioden, brauchen diese unterschiedliche Spannungen für deren Betrieb. So dass damit verschiedene Farben erzeugt werden können. Anfangs gab es nur

rot, gelbe, grüne und Infrarot LEDs, bis dann 1988 in Japan die blaue LED erfunden wurde. 1995 gelang es der Firma Nichia, eine LED aus Galliumnitrid herzustellen, die blaues Licht emittiert. Weisses Licht kann seitdem mit einer RGB-LED erzeugt werden. Heute sind die LED für die Beleuchtung aus einer blauen LED aufgebaut, die mit einer Phosphorschicht bedeckt ist, welche das bläuliche Licht in weißes umwandelt.

Artikelnummer: **121741**



LED's und deren Spannungswerte:

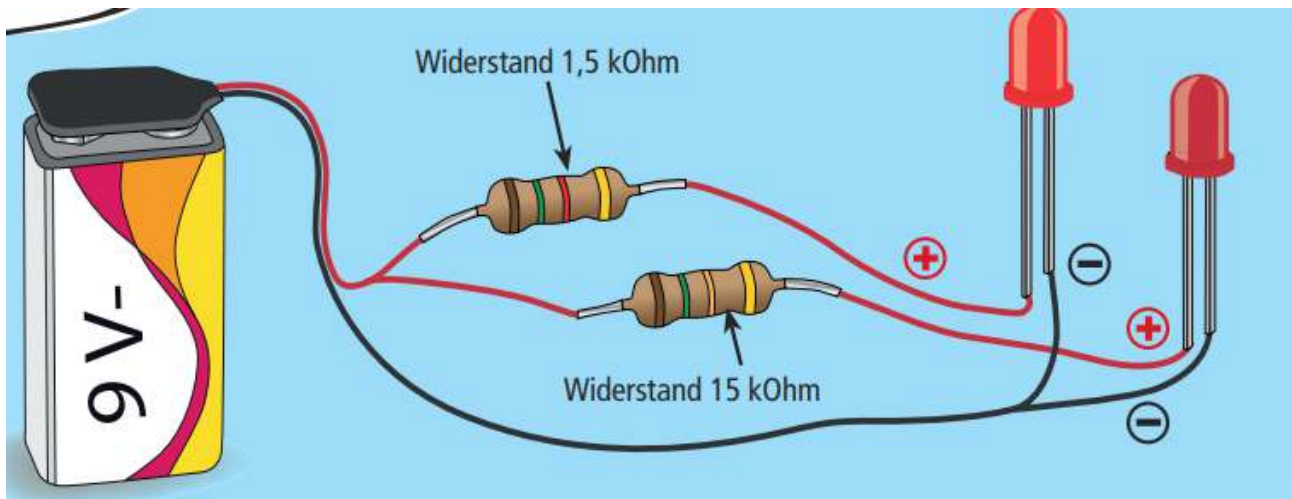
Infrarot-LED: 1,2–1,8 V, Typ. 1,3 V; Rot: 1,6–2,2 V; gelb, Grün: 1,9–2,5 V; blau, Weiß: 2,7–3,5 V;

UV-LED: 3,1–4,5V, Typ. 3,7V

Woher kommt es, dass die LEDs unterschiedliche Durchlassspannungen haben. Das Ohmsche Gesetz besagt, dass der Spannungsabfall durch den Strom und den Widerstand bestimmt wird ?!

Das liegt am differentiellen Widerstand der LEDs. Dieser ist Materialabhängig. Die LEDs und die anderen Halbleiter bestehen überwiegend aus Silizium Kristallen. Aber das Geheimnis der unterschiedlichen Funktion, der einzelnen Halbleiter liegt in deren atomaren Aufbau der Kristallstruktur.

Die blauen LEDs z.B. gibt es erst seit den 1990er Jahren. Obwohl die Funktionsweise in den 70er Jahren bereits bekannt war. Aber es war schwierig durch einen chemischen Prozess dieses Bauteile in Großserie und damit kostengünstig zu produzieren. Deshalb dauerte auch die Einführung der LED Fernseher fast 40 Jahre seit der Entwicklung der ersten LED im Jahre 1951. Sogar die ersten LED Ampeln gab es erst im Jahre 2008, obwohl rote, gelbe und grüne LED's längst verfügbar waren.



Wie berechnet man den Vorwiderstand?

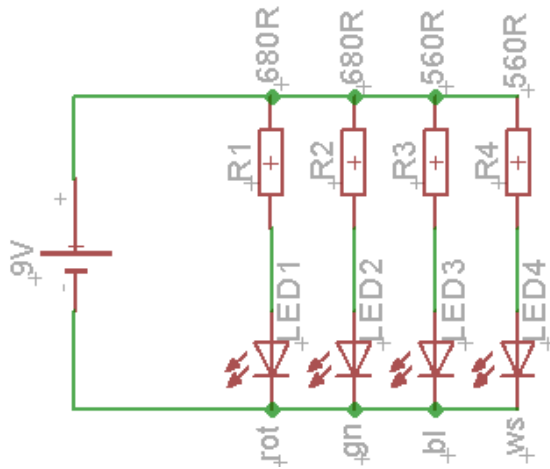
Wie bereits erwähnt, hat jeder LED Typ einen eigenen individuellen Aufbau. Die Kristallstruktur bestimmt die Farbe in der die LED leuchtet und damit auch die Spannung, und den Strom den die LED benötigt. Aus dem Spannungsabfall und den erforderlichen Strom kann leicht der Vorwiderstand bestimmt werden.

Zuerst legt man fest, welche Spannung als Versorgung vorhanden ist. Nehmen wir z.B. das obige Beispiel mit einer 9V Blockbatterie.

Diese Spannung teilt sich auf auf den Widerstand und die Diode. Also Subtrahiert man die Spannung die die LED benötigt, von der Versorgungsspannung. Nun kann berechnet werden,

welche Spannung am Widerstand abfällt. Durch die Diode und den Widerstand fließt derselbe Strom. In unserem Fall sind das bei jeder Diode 10mA also 0,010A. Der Widerstand ist also $(9V - V_{LED}) / 0,01 [\Omega]$.

Bei uns ist LED4=rot; LED3=grün, LED2=gelb; LED1=blau.



Durchflussspannung
+ von LED's

- rot: 2V
- gelb: 2,1V
- grün: 2,1V
- blau: 3,3V
- weiß: 3,6V

Berechne nun die Widerstände für die Verwendung mit dem Arduino bei 5V:

R1= Ω R2= Ω R3= Ω R4= Ω

Als Versuch kann man verschieden farbige LEDs verwenden und die Vorwiderstände gegenseitig austauschen.

Der Vorwiderstand bestimmt die Stromstärke und damit die Helligkeit der LED.

In unserem Fall ist der Strom so dimensioniert, dass die Widerstände beliebig getauscht werden können. Aber ACHTUNG, es gibt LEDs, die nur wenig mA vertragen und leicht kaputt gehen können.

3.1.2.1. LED-Balken

Es gibt den LED Balken. Darin sind 7 LEDs verbaut in einem Gehäuse und man konnte damit früher eine Aussteuerungsanzeige für den Kassettenrecorder bauen.

3.1.2.2. WS2811

Dies ist eine mehrfarbige LED, in die der Controller bereits eingebaut ist. Es ist dabei möglich mehrere hintereinander zu schalten und diese nur mit einer einzigen Signalleitung zu steuern.



Dabei ist es nicht notwendig Vorwiderstände anzubringen. Diese Funktion ist bereits in der LED integriert.

3.1.2.3. 7-Segment-Display

Dies ist eine Art Balkengrafik, mit der man früher auf einfache Weise Zahlen darstellen konnte. Diese wird auch heute noch verwendet, weil sie sehr billig und einfach anzusteuern ist. In der Abbildung rechts ist der Artikel 121542 dargestellt.



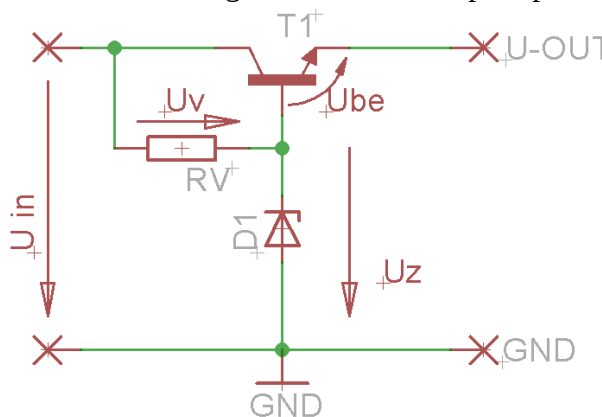
3.1.2.4. 8x8 Matrix Display mit MAX7219 Ansteuerung (im Artikel 810692 enthalten)

Der Baustein MAX7219 bildet die Schnittstelle zur Matrix Anzeige. Damit ist es möglich, mit einem Mikrocontroller 64 LED's der Matrix beliebig anzusteuern und so Buchstaben oder Zeichen damit darzustellen.



3.1.3. Zenerdiode

Eine weitere Variante und vielfach verbreitete Diode ist die Zener Diode. Der Zener-Effekt beschreibt eine Eigenschaft, die eine Diode hat, wenn man sie entgegen der Durchlassrichtung verwendet. Die Anode ist dabei mit Masse (GND) und die Kathode mit Plus verbunden. Dabei ist eine Zenerdiode mit genau definierter Sperrspannung erhältlich. Das bedeutet, man kann mit



Zenerdioden fast jede beliebige Sperrspannung erzeugen und somit findet diese ihre Verwendung in billigen Netzteilen.

Einfache Spannungsstabilisierschaltung:

Dabei ist die Ausgangsspannung

$$U_a = U_z - U_{be} \text{ also in etwa } U_z - 0,6V$$

Der Vorwiderstand ist abhängig vom Laststrom, weil dieser sich aus dem Zenerstrom und dem Basisstrom zusammensetzt.

Seit den 1980er Jahren gibt es noch die Laserdiode, die überwiegend Verwendung in CD-Playern, DVD-Laufwerken und Laserpointern findet.

Eine sehr spezielle Art ist die Tunnel diode. Diese arbeitet nach einem Effekt der Quantenmechanik im Bereich von 100GHz. Dies bedeutet bei Frequenzen, die ihre Amplitude 100 Milliarden Mal in einer Sekunde verändern!

Typisch für diese Diodenart ist, dass ein Teil der Kennlinie einen negativen differentiellen Innenwiderstand besitzt. Das bedeutet, dass trotz ansteigender Spannung der Strom sinkt.

3.1.4. Solarpaneel

Den umgekehrten Effekt einer LED findet man bei einer Solarzelle. Dabei werden die empfangenen Photonen in elektrischen Strom umgewandelt. Da leider nur ein geringer Teil des Sonnenlichtes in Strom umgewandelt wird, haben Solarzellen einen sehr niedrigen Wirkungsgrad.

Artikel-Nr.:590038

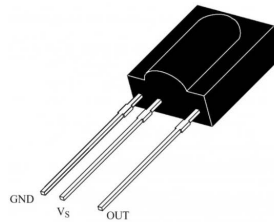


In den 1980er Jahren lag der Wirkungsgrad bei 10% und der theoretisch vorhergesagte zu erreichende Wirkungsgrad bei 17%. Mittlerweile gibt es neuere Technologien, die einen Wirkungsgrad von bis zu 30% oder mehr erreichen sollen.

3.1.5. IR-Sender, Empfänger

Bei den Fernsehern wurden als erstes eine Ultraschallfernbedienung eingesetzt. Ab den 1970er Jahren kamen die Infrarotfernbedienungen zur Anwendung, die wesentlich kleiner und mit mehr Funktionalitäten ausgestattet waren.

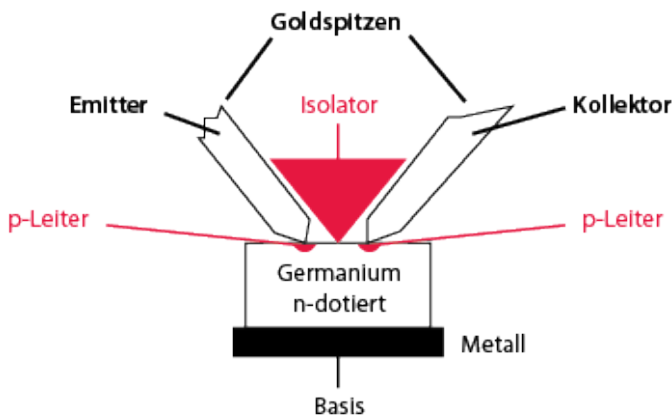
Im unteren linken Bild ist ein Sortiment an Sendedioden dargestellt. In den Bildern rechts davon sind Empfänger dazu dargestellt.



3.2. Der Transistor

Am Heiligabend 1947 wurde uns in den Bell Laboratorien in den USA der Heiland der Unterhaltungsindustrie geboren - der Transistor. Keine Erfindung in der Neuzeit hat die Technologie mehr geprägt, als die Erfindung des Transistors. Es gäbe keine Computer, keine Raumfahrt, keine Smartphones ohne die Nachfahren des Transistors. Nur so war es möglich Schaltungen, die früher ganze Häuser ausfüllten, so weit zu integrieren und zu verkleinern, dass sie heutzutage am Handgelenk getragen werden können.

Das Wort Transistor setzt sich zusammen aus **Transfer** (übertragen) und **Resistor** (Widerstand). Ein Transistor, kann entweder als Schalter, oder aber auch als Verstärker verwendet werden. In Stereoanlagen wird er als Verstärker verwendet. Man kann daraus aber auch eine Blinkschaltung bauen.

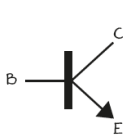


Im Bild links ist der schematische Aufbau des Transistors aus dem Jahre 1947 dargestellt. Dieser ist ein NPN-Transistor.

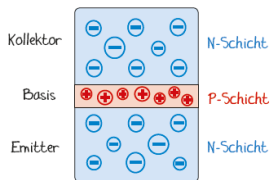
Bipolare Transistoren (NPN und PNP) arbeiten nach dem Prinzip, dass ein kleiner Steuerstrom (bzw. Steuerspannung) einen größeren Laststrom an- und ausschalten kann.

3.2.1. Vielfach verwendet wird der NPN-Transistor

Schaltbild



Aufbau



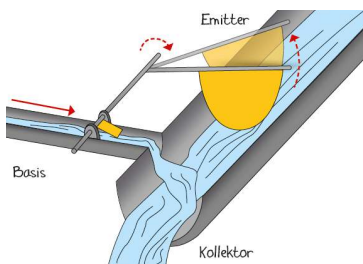
Bauform



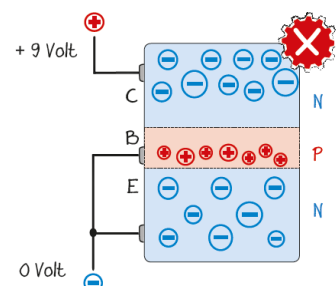
Der Steueranschluss heißt Basis „B“, die Laststromanschlüsse werden Emitter „E“ (Entsender) und Kollektor „C“ (Aufsammler) genannt. Der Name NPN-Transistor leitet sich von der Schichtfolge seines Aufbaus: N-Schicht; P-Schicht, N-Schicht. N-Schicht bedeutet, es ist ein Elektronenüberschuss in dieser Schicht vorhanden.

P-Schicht bedeutet, es ist ein Elektronenmangel in dieser Schicht vorhanden.

3.2.2. Wasser Analogon zur Erklärung der Transistorfunktion



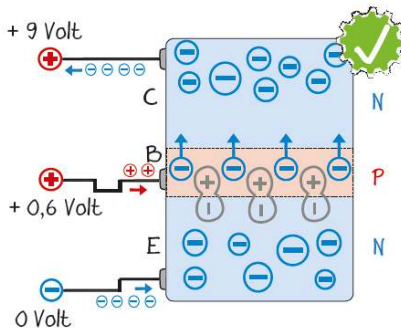
Der kleine Wasserfluss im Basiskanal drückt auf die kleine Klappe, die über ein Gestänge die große Klappe im Emitterkanal hebt und so den Hauptfluss vom Emitter zum Kollektor mehr oder weniger freigibt.



3.2.3. Der Transistor im Sperrzustand:

Beide Übergänge von Basis nach Emmitter und von Basis nach Kollektor sind gesperrt, weil B und E auf gleicher Spannung liegen (B-E Diode leitet erst ab 0,6 V!) und die B-C Diode ist sowieso in Sperrrichtung gepolt.

3.2.4. Der Transistor leitet:

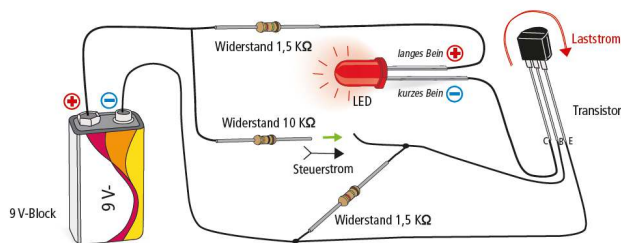


Legt man zwischen Basis und Emmitter mindestens 0,6 Volt Spannung an, so wird die Sperrschicht zwischen Basis und Emmitter abgebaut, weil positive und negative Ladungsträger aufeinander zugetrieben werden und sich gegenseitig aufheben. Vom Emmitter aus können nun viele negative Ladungsträger in die Basisschicht gelangen.

Weil diese aber nur ein paar tausendstel Millimeter dick ist, werden sie sofort von der positiven Kollektorspannung abgesaugt.

Der Steuerstrom, der von der Basis zum Emmitter fließt, ist sehr viel kleiner als der Strom vom Emmitter zum Kollektor.

3.2.5. Der NPN-Transistor schaltet eine LED:



Sobald der 10 K-Widerstand den Basisanschluss berührt fließt ein kleiner Steuerstrom und schaltet den Laststrom durch die LED über den Kollektoranschluss an.

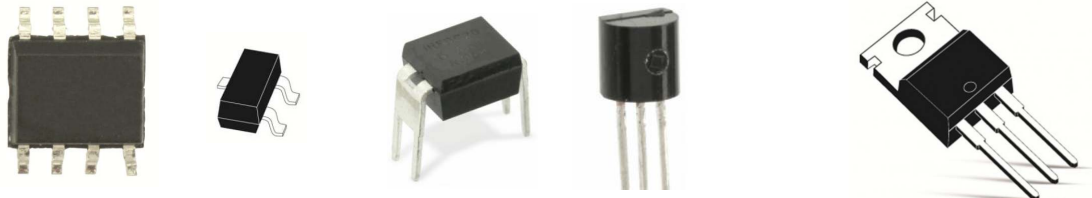
3.3. Der Feld-Effekt-Transistor (FET) und MOSFET's

der Feldeffekttransistor ist eine Weiterentwicklung des Transistors aus dem Jahre 1947. Der bipolare Transistor, wie er genau benannt wird, steuert mit Hilfe des Basisstroms den „Hauptstrom“ durch den Transistor.

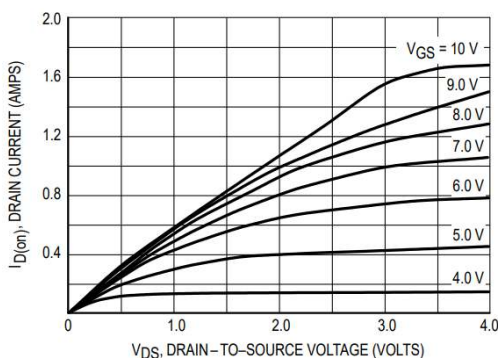
Das Patent des Feldeffekttransistors basiert aus den 1920er Jahren und ist von Julius Edgar Lilienfeld.

Allerdings konnten erst in den 1960er Jahren erste Muster dieses Transistortyps hergestellt werden. Da diese in Plasmatechnologie hergestellt werden. Diese Technik war notwendig, um mehrere Transistoren auf einem Trägermaterial aufzubringen und diese zu verbinden.

Der Feldeffekttransistor hat die Eigenschaft, dass nicht durch einen Strom, sondern nur durch anlegen einer Spannung der Transistor in den leitenden, oder in den sperrenden Zustand versetzt wird. Das bedeutet, man kann völlig leistungslos und mit viel weniger Verlusten eine Last schalten. Es gibt auch keinen mechanischen Verschleiß, durch Abnutzung von Kontakten, wie beim Relais. Allerdings sind FET's etwas empfindlicher in der Handhabung, weil sie durch elektrostatische Aufladung sehr leicht zerstört oder geschädigt werden können.

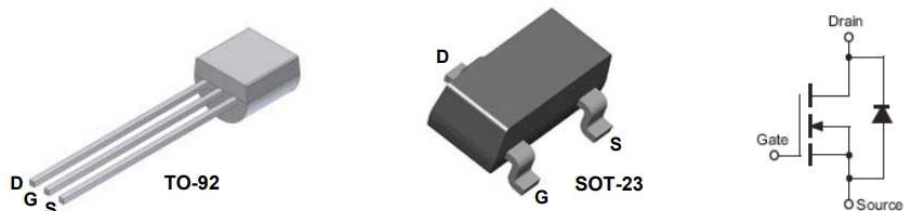


Typ:	IRF8736	BSS138	IRFD9024	BS170	STF16NF06
Artikel-Nr.:	131104	131099	131127	130268	130777



Links im Bild ist ein Diagramm aus dem Datenblatt des BS170 dargestellt. Darin ist zu sehen, wie sich mit der Änderung Spannung U_{GS} der Kanalwiderstand und somit der Drainstrom I_D verändert. Dies ist einer charakteristischen Eigenschaften eines FET's, dass mit Änderung der Spannung U_{GS} der Kanalwiderstand variiert werden kann. So ist es möglich den Strom I_D völlig leistungslos nur über die Spannung zu steuern. Eine häufige Anwendung des FET ist als variabler Widerstand und eine andere als Lastschalter. Dadurch, dass es möglich

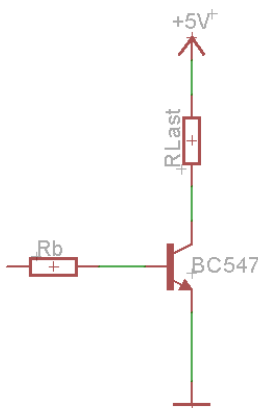
ist, den Kanalwiderstand sehr gering zu halten, kann mit einem FET ein wesentlich höherer Laststrom als bei einem bipolaren Transistor geschaltet werden. Aber in diesem Skript beschränken wir uns auf Ströme im [mA] – Bereich. Dafür ausreichend ist der BS170.



Oben dargestellt sind zum einen die erhältlichen Gehäusebauformen und rechts das Ersatzschaltbild des MOS-FET's. Der MOS-FET ist eine Weiterentwicklung des FET's. MOSFET's haben noch höhere Eingangswiderstände und kleinere Kanalwiderstände als FET's. Die dargestellte Diode (Bodydiode) zwischen Source und Drain kann auch als Schutzdiode an induktiven Lasten dienen.

3.4. Transistoren und die Grundsaltungen:

3.4.1. NPN-Transistor als Schalter



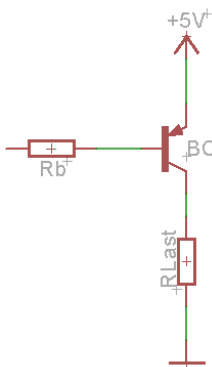
Damit der Transistor den vollen benötigten Strom schalten kann, muss der Basisstrom einen bestimmten Wert haben. Wie groß dieser sein soll, hängt vom verwendeten Transistor ab. Dazu ist der Wert für den Verstärkungsfaktor h_{FE} aus dem Datenblatt zu entnehmen. Der Basisstrom errechnet sich so: $I_B = I_C / h_{FE}$

Der Basisvorwiderstand errechnet sich dann: $R_b = (U_v - U_{BE}) / I_B$

wobei U_v die Versorgungsspannung ist (in unserem Fall 5V) und $U_{BE} = 0,7V$

Beim NPN-Transistor muss ein „High“-Pegel (5V) des Prozessors am Basiswiderstand anliegen, damit der volle Laststrom fließen kann.

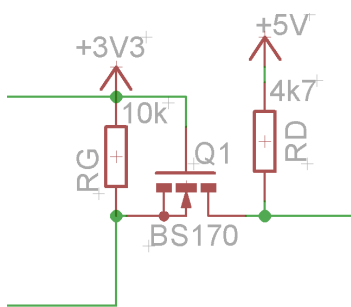
3.4.2. PNP-Transistor als Schalter



Im Gegensatz zum NPN-Transistor schaltet der PNP-Transistor dann „ein“, wenn der Basiswiderstand auf GND geschaltet wird. Der Arduino muss somit auf „Low“ (0V) geschaltet sein, damit der Transistor durchgesteuert ist.

Es gelten für den PNP-Transistor die selben Formeln wie beim NPN-Transistor.

3.4.3. der FET als Schalter



Im Gegensatz zu den bipolaren Transistoren unterscheiden sich FET's im besonderen dadurch, dass kein „Basisstrom“ fließt. Das Gate, wie es bei den Feldeffekttransistoren genannt wird, wird nur über die anliegende Spannung gesteuert. Der Einganswiderstand ist so groß, dass kein Strom für die Steuerung notwendig ist. Der Widerstand R_D begrenzt nur den Strom zum Drain-Anschluss. Zu groß sollten die Widerstände trotzdem nicht sein, weil der FET durch dessen Aufbau höhere Kapazitäten besitzt. Ein höherer Widerstand bedeutet dann auch

eine höhere Schaltzeit und damit eine niedrigere Schalthäufigkeit. Im linken Bild ist die Schaltung aus einer Pegelanpassung um Signal von 5V nach 3,3V und umgekehrt zu übertragen.

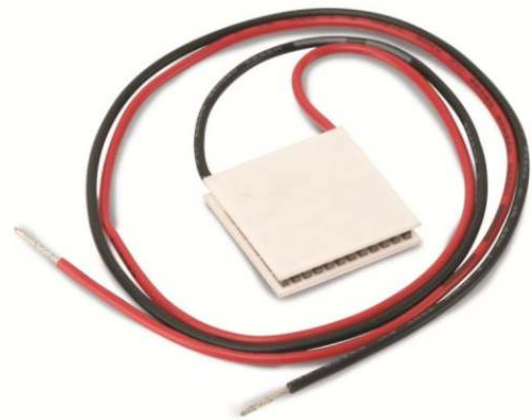
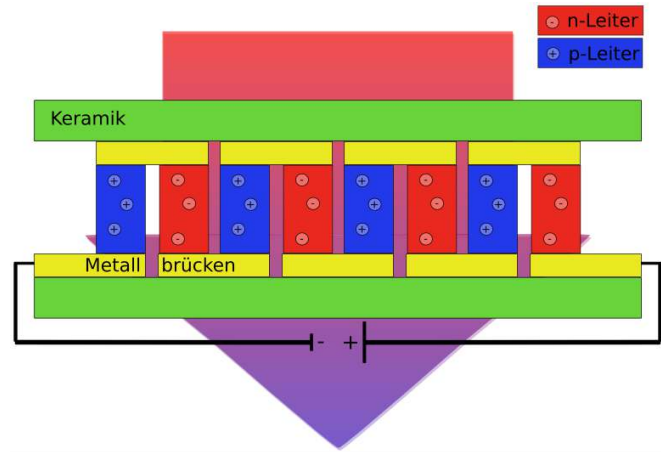
3.5. Peltierelement

1834 entdeckte der Franzose Jean Peltier, dass beim Anlegen einer Spannung an zwei unterschiedlichen Metallen, die miteinander verbunden sind, dass die eine Seite gekühlt, während die andere gleichzeitig erwärmt wird. Da der Wirkungsgrad sehr schlecht ist eignet sich diese Erfindung eher zum Kühlen von einer Flasche Bier, als für den Einsatz in einem Kühlschrank.

Die Umkehrung des Peltiereffekts nennt man den Seebeck - Effekt. Dabei wird durch eine Temperaturdifferenz an den beiden Platten des Peiltier-Elements eine Spannung erzeugt. Wegen der Zuverlässigkeit und weil keine beweglichen Teile verbaut sind, findet dieses Bauteil oft Anwendung z.B. in der Raumfahrt. Für die Stromversorgung von Satelliten oder Raumfahrzeugen.

Grundlage des Phänomens ist ein Diffusionsstrom von Ladungsträgern der entsteht, wenn ein Temperaturgradient an einen elektrischen Leiter angelegt und damit eine elektrische Spannung erzeugt wird. In Halbleitern ist dieser Effekt besonders stark. Ein Halbleiter Thermoelement besteht dabei meist aus einer Kombination von n- und p-Typ-Halbleitern.

In der Abbildung rechts, ist das Peltierelement 127/021 mit der Artikelnummer: **180060** dargestellt.



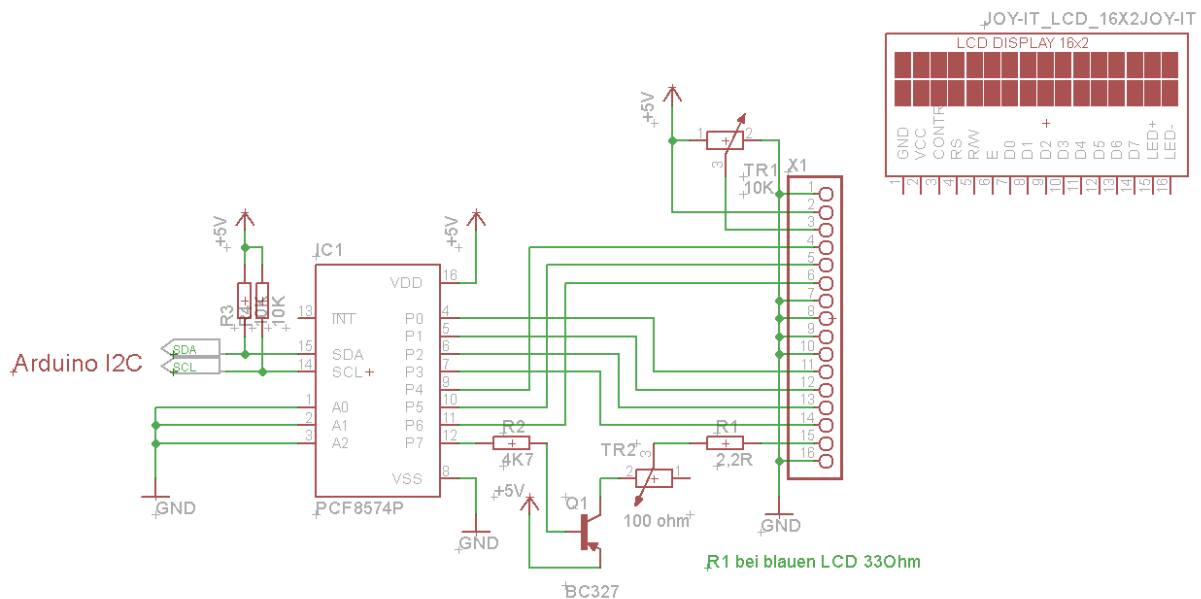
4. Anzeigeeinheit - alphanumerisches LC-Display



Bestellnummer: 121714

Im Gegensatz zu einem LED -display benötigt ein LCD Display weniger Strom. Die Eigenschaften flüssiger Kristalle wurde bereits Ende des 19.Jh. entdeckt und daran geforscht. Aber erst ab dem Jahre 1983 ließen sich *Super-Twisted Nematic* STN-LCD, wie das rechts dargestellt Display

auch bezeichnet wird, fertigen. Das Prinzip ist theoretisch relativ einfach: Durch lange Flüssigkristallstränge wird die Polarisation geändert. So ist je nach Feld, die dünne Schicht lichtdurchlässig oder nicht. Angesteuert werden die Displays über Daten- und Steuerleitungen.



Im Bild oben ist dargestellt, wie eine Ansteuerung eines Standard Displays mit 16 Zeichen und 2 Zeilen aussehen kann. Die Schaltung um das IC PCF8574P ist dazu da, die Ansteuerung mit dem Arduino zu vereinfachen. Damit erspart man sich Zeit für die Verdrahtung und vor allem spart es die Verwendung von I/O-Pins. Denn für die Ansteuerung braucht es nur Die Pins A4 und A5 des Arduino. Das Display ist ein wenig teurer, weil die Zusatzplatine aufgelötet ist.

Das LC-Display wie es rechts neben der Ansteuerschaltung abgebildet ist braucht eine Spannungsversorgung; ein Poti TR1 zur Kontrasteinstellung; Datenleitungen D0 ... D7 um ein Datenbyte zu übertragen, oder zu lesen; eine Hintergrundbeleuchtung (LED+ und LED-) und eben die Steuersignale:

RegisterSelect: Bei High.Pegel wird Datenregister und bei 0 das Befehlsregister ausgewählt

ReadWrite: Sollen Daten gelesen oder geschrieben werden

Enable: Damit wird dem display mitgeteilt, dass die anliegenden Daten jetzt gültig sind

5. Sensoren

Ein Sensor ist ein elektronisches Bauelement, der physikalische Messgrößen wie Temperatur, Luftdruck, Licht, Feucht, Wärmestrahlung usw. in elektrische Spannungen umzuwandeln hilft. Dadurch wird es möglich, diese Messgrößen mit einem Mikrorechner zu verarbeiten.

5.1. FSR-Sensor (siehe Artikel 811270 und 811271)

Der Force sensitive Sensor ist ein Kraftsensor, der seinen Widerstandswert durch ausüben einer Druckkraft senkrecht auf die Folie ändert. Dabei erniedrigt sich sein Widerstandswert bei



Belastung. Es gibt diese für verschiedene Gewichtskräfte. Anwendung finden diese Sensoren z.B. in der Medizin, wenn ein Patient den Fuß nur zu einem gewissen Betrag belasten darf. Leider ist dieser

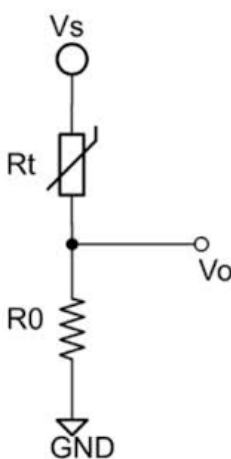
Sensor sehr ungenau. Ausserdem ist seine Kennlinie nicht linear. Bei geringer Belastung ändert sich der Widerstandswert sehr schnell. Um jedoch den Nennwert zu erreichen, muss die Krafteinwirkung überproportional steigen. Deshalb ist dieser Sensor zum Abwiegen ungeeignet.

5.2. Temperatursensoren

5.2.1. Der NTC (Heissleiter Artikelnummer 220800)

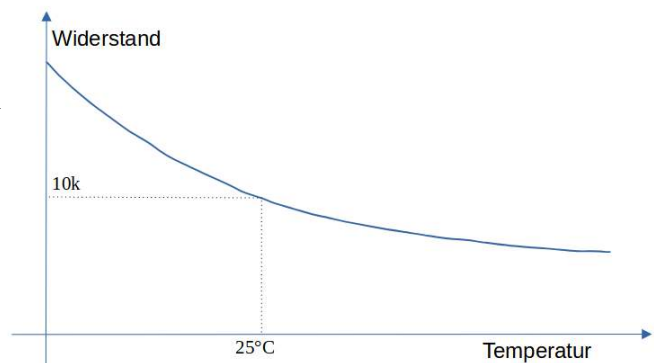


Der NTC ist ein Widerstandssensor, der einen negativen Temperatur Koeffizienten hat. Das bedeutet mit zunehmender Temperatur erniedrigt sich sein Widerstandswert. Dabei hat er bei Raumtemperatur seinen Nennwert.



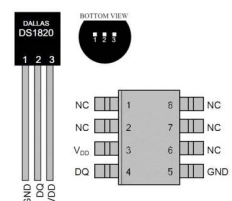
Rechts in der Abbildung der Kennlinie erkennt man, dass der Sensor einen nichtlinearen Verlauf hat. Da er aber sehr preiswert ist findet er große Verbreitung.

Die Linearisierung der Kennlinie erfolgt entweder mit einer Zusatzschaltung oder aber meist über eine Tabelle per Software.



5.2.2. Digitale Temperatursensoren

Will man sich mit diesem Thema nicht herumärgern, sondern lieber einen Euro mehr investieren, bedient man sich eines digitalen Temperatursensors. Dieser kann über eine spezielle Schnittstelle, per Software, angesteuert werden. Dazu gibt es im Internet auch schon eine Bibliothek und man kann so den Sensor einfach anwenden.



Temperaturmessung mit dem DS18B20 mit der Bestell-Nr.: **180101**



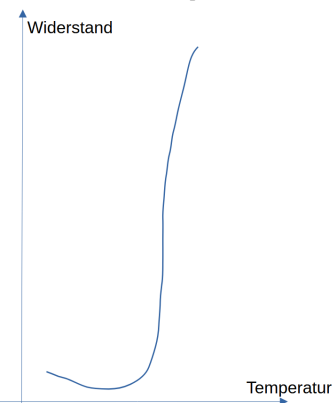
Bestell-Nr.: **810912**

Damit kann man Temperaturen von -55°C ... 125°C , bei einer Auflösung von $\pm 0.5^{\circ}\text{C}$ messen. Die ist für unsere und viele andere Zwecke ausreichend. Der Vorteil ist, es ist nur eine Datenleitung erforderlich, deshalb wenig Verdrahtungsaufwand. Zudem braucht es keine Zusatzbeschaltung. Der Sensor kann verwendet werden, wie er zu kaufen ist. Es gibt für den Arduino viele Tipps im Internet. Es gibt sogar die Möglichkeit einen Auslöser für eine Temperaturüber- bzw. Unterschreitung festzulegen. Man muss den Sensor nicht permanent abfragen. Es reicht den Status abzufragen, so erspart man sich die Temperatúrauswertung.

5.2.3. Pt100 und PT1000 (Artikelnummer 180053 und 180052)

Große Verbreitung bei Temperatursensoren haben sogenannte PT100 und PT1000 Sensoren. PT100 bedeutet dass der Widerstand bei 0°C der Widerstand 100R ist. Dazu hat der PT1000 einen Nennwiderstand von 1k bei 0°C . Diese Sensoren besitzen einen Platin-Temperaturfühler. Sie sind sehr zuverlässig, sehr genau und haben eine annähernd lineare Kennlinie. Sie sind etwas teuer, aber können bei hohen Temperaturen bis 900°C eingesetzt werden. Am besten nutzt man dazu den Bausatz Messwandler Bestell-Nr.:810272. Damit wird der Messwert des Sensors in den Mesbereich des Arduino von zwischen 0 und 5V verstärkt. Denn die Änderung der Temperatur bewirkt nur sehr geringe und mit der 10Bit Auflösung des Arduino kaum messbare Spannungsänderungen. Auch für den PT1000 gibt es einen Messwandler mit der Bestell-Nr.: **810144**.

5.2.4. PTC (Kaltleiter)

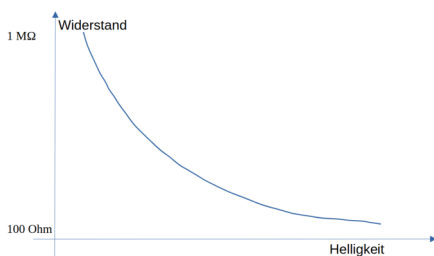


Im Bild links ist schematisch dargestellt, wie sich der Widerstand mit zunehmender Temperatur ändert.

Der PTC findet seine Verwendung sowohl als Sensor und auch als Sicherung; Beim PTC erhöht sich nur der Widerstandswert mit zunehmendem Strom und kann den Stromanstieg auf diese Weise begrenzen.

Auch eine alte Glühlampe ist ein Heißleiter. Der Widerstand einer Glühlampe ist im ausgeschalteten Zustand sehr klein. Daher kann beim Einschalten ein höherer Strom fließen. Dies war der Grund, warum eine Glühlampe gerade beim Einschalten und weniger im Betrieb ausgefallen ist.

5.3. Der LDR



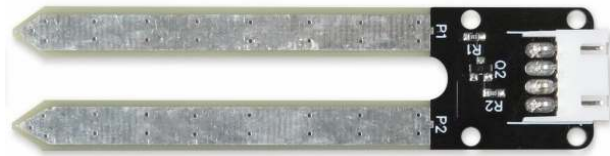
Um die Helligkeit in einem Raum zu messen, um z.B. eine Anzeige einer Uhr zu dimmen, wird der Light Dependent Resistor verwendet.



Bestell-Nr.: **810467**

Rechts ist ein LDR abgebildet und links ist die Kennlinie qualitativ dargestellt. Da ist gut zu erkennen, dass mit zunehmender Helligkeit der Widerstandswert sehr schnell, sehr klein wird.

5.4. Boden Feuchtesensor



Bestell-Nr.: **811119**

Der Feuchtesensor ist im Prinzip sehr einfach aufgebaut. Es ist nur eine Kupferbeschichtete Platte.

Diese sind quasi zwei Elektroden. Wenn man diese in die Erde steckt und den Widerstand misst, wird man feststellen, dass sich der Wert mit zunehmender Feuchte erniedrigt. Das Ausgangssignal ist deshalb ein analoger Spannungswert, der von einem Transistor verstärkt wird.

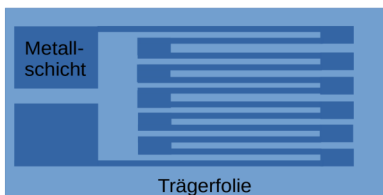
5.5. Drucksensoren

piezoresistive Drucksensoren sind quasi die Umkehrung des Piezosummers. Wenn man Spannung an den Piezosummer anlegt erzeugt dieser eine Schwingung. Wenn man auf ein Piezoelement eine Kraft ausübt, erzeugt dieses eine Spannung. Diese ist allerdings sehr gering und nur durch eine zusätzliche Auswerteelektronik messbar.

Der DSP310 von Infineon ist dabei die Spitze der Entwicklung. Der Sensor ist zwar sehr klein, beinhaltet aber die analoge Verstärkereinheit und auch schon eine kleine Rechneinheit mit Interface für einen Mikrocontroller. Damit ist es möglich, den Höhenunterschied im Haus zu messen. Der Sensor ist sogar so empfindlich, dass er die Druckänderung messen kann, wenn sich ein Fenster öffnet.

Die Auflösung beträgt ± 0.002 hPa. Das bedeutet er erkennt eine Höhenänderung von 2 cm. Deshalb wird er z.B. in Drohnen oder in Smartwatches verwendet.

5.6. Dehnungsmessstreifen



Der Dehnungsmessstreifen ist im Prinzip eine Kupferfolie, die auf eine Folie gedruckt ist. Er verändert seinen Wert, wenn sich seine Länge ändert. Dieser ist sehr präzise und wird daher in elektronischen Waagen verwendet. (z.B. im Artikel **811418**)

5.7. PIR-Sensor



Auf dem linken Foto ist ein PIR-Sensor der Firma Joy-it dargestellt. Der Sensor auf diesem Modul ist ein Pyroelektrischer Infrarot-Sensor, welcher im Detektionsbereich der Fresnel-Linse eine Bewegung erkennt und am Sensor-Ausgabepin "SIG" für die Dauer von 3 Sekunden ein "HIGH"-Signal ausgibt. Danach geht das Signal wieder in den "LOW"-Zustand über. Das verlötete Potentiometer dient zur stufenlosen Reichweiteinstellung. Mithilfe diesem Modul und einem Mikrocontroller wie Raspberry Pi oder Arduino kann er u.a. für Sicherheitssysteme oder Automation eingesetzt werden.

Bestell-Nr.: **811274**

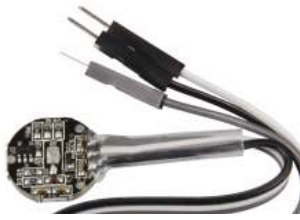
5.8. IR-Empfänger

Damit werden RC5 Signale von Fernbedienungen decodiert.

Bestell-Nr.: **121086**



5.9. Herzschlag-Sensor KY-039



Wird ein Finger zwischen der Infrarot-Leuchtdiode und dem Foto-Transistor gehalten, so kann am Signalausgang der Puls detektiert werden.

Bestell-Nr.: **810917**

5.10. Infrarot Lichtschranke KY033LT

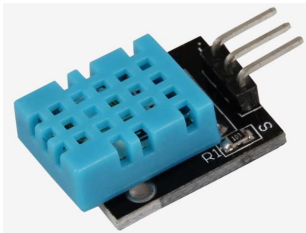


Dieser Sensor sendet und empfängt Infrarot. Dabei kann er erkennen, ob er eine reflektierende Fläche (der Sensor empfängt sein eigenes Signal) oder eine absorbierende Fläche (der Sensor empfängt kein Signal) vor sich hat. Dabei kann man die Empfindlichkeit des Sensors mit Hilfe eines Potentiometers einstellen.

Roboter zum Beispiel können mit Hilfe von zwei dieser Sensoren eine automatisierte Line Tracker Funktion ausführen. Sobald eine Linie an einer Seite verlassen wird erkennt dies der entsprechende Sensor und der Roboter weiss, in welche Richtung er nun steuern muss.

Bestell-Nr.: **811269**

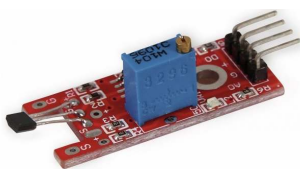
5.11. Feuchte und Temperatursensor DHT11:



Dieser Sensor hat einen speziellen Pin, der ein Digital-Signal liefert, so dass die Daten vom Prozessor eingelesen und ausgewertet werden können. Dafür gibt es eine Bibliothek, die zur Verfügung steht, so muss man sich um diesen komplizierten Prozess nicht selber kümmern. Der Sensor ist leider etwas ungenau. Die bessere Alternative wäre der DHT12 mit der Bestellnummer DHT 22: 811093

Bestell-Nr.: **810914**

5.12. Magnetfeld-Sensor



Das Magnetfeld wird vom Sensor gemessen und als analoger Spannungswert ausgegeben, dabei kann die Empfindlichkeit des Sensors durch einen Potentiometer geregelt werden. Allerdings ist der Sensor nicht besonders empfindlich, so dass ein stärkerer Magnet verwendet werden muss. Ein magnetisiertes Stück Metall reicht leider nicht.

Bestell-Nr.: **810915**

5.13. der kapazitive Feuchtesensor KFS140-D

Der KFS140-D ist ein kapazitiver Feuchtesensor mit sehr guten Leistungsdaten. Hervorzuheben ist der weite Anwendungsbereich, die geringe Hysterese sowie die lineare Kennlinie. Das eingesetzte Hochleistungs-Polymer ist beständig gegen Betauung und viele chemische Einflüsse und garantiert eine hervorragende Langzeitstabilität. Der Sensor besitzt ein günstiges Preis-Leistungsverhältnis und eignet sich dadurch auch für Anwendungen in der Lüftungs- und Klimatechnik. Durch die

optimalen Leistungsdaten ist der Sensor aber auch ideal für anspruchsvolle Aufgabenstellungen in der industriellen Messtechnik geeignet.

Technische Daten:



- Messprinzip: Kapazitiver Polymer Feuchtesensor
- Feuchte: 0 ... 100 % relative Feuchte
- Kapazität: 150 pF ± 50 pF (bei 23 °C und 30 % RH)
- Steigung: 0,25 pF / % RH
- Frequenzbereich: 1 ... 100 kHz
- Max. Auswertespannung: < 12 Vpp
- Signalform: Wechselspannung (ohne Gleichspannungsanteil)

Bestell-Nr.: **180103**

5.14. Gassensoren für brennbare Gase Rauch und Luftqualität.



Bestellnr.: **811159**

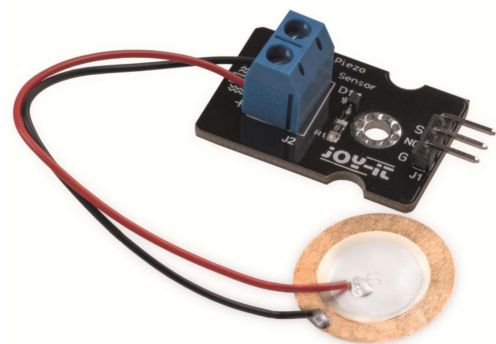
Messbereich: 10 - 1000 ppm

- Messbare Substanzen: Benzol (C₆H₆), Ammoniak (NH₃), Sulfid, Rauch, andere Luftverunreinigungen
- Einsatzbereiche: Erkennen von Gaslecks, für Gasalarm, Robotik, Mikrocontrollerprojekte
- Kompatibel mit: Raspberry Pi (mit AD-Wandler), Arduino, etc.
- Besonderheiten: hohe Empfindlichkeit, Erkennung einer weiten Spanne an Konzentrationen
- Analoge Ausgabe: Auswertung der Messwerte vom Mikrocontroller
- Digitale Ausgabe: Schwellenwerteinstellung möglich
- Pins: VCC: Stromversorgung 5V, GND Masseanschluss, AOUT: Analoger Output, DOUT: Digitaler Output
- Erfassungsspannen: Ammoniak (NH₃), Alkohol: 10 - 300 ppm Benzol: 10 - 1000 ppm
- Heizspannung: 5,0 V ± 0,2 V
- Heizwiderstand: 31 Ω ± 3 Ω bei Raumtemperatur
- Sensitivität: R_s (in der Luft) / R_s (100ppm HH₃) ≥ 5

5.15. Erschütterungssensor / Vibrationsensoren



Bestell-Nr.: **180119**



Bestell-Nr.: **811421**

Vibrationssensoren werden z.B. bei der Erkennung einer Bewegung, oder Erschütterung verwendet. Um zu erkennen ob gerade eine Tür zugeschlagen oder geöffnet wurde, im Fahrzeug werden Vibrationssensoren bei Dieselmotoren als Klopfensensoren verwendet. Es ist auch möglich ein Spiel zu machen, bei dem erkannt wird, ob ein Nagel mit dem Hammer getroffen wurde, oder ob daneben geschlagen wurde.

5.16. LED Modul



Bestell-Nr.: **810568**



Bestell-Nr.: **810453**

LED – Module sind günstige Alternativen zu einem LCD Modul, um Zahlenwerte oder die Uhrzeit anzuzeigen. Ein besonderer Vorteil liegt dabei in der besseren Lesbarkeit bei Dunkelheit.

6. Aktoren

6.1. REED Kontakt (z.B. Bestell-Nr.: 420165)

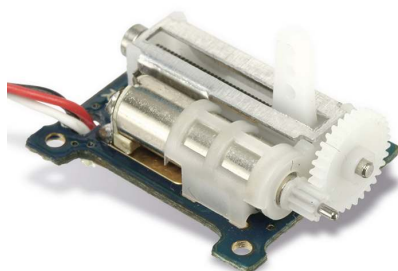


Der REED Kontakt ändert seinen Schaltzustand. Er öffnet oder schließt sich beim Anlegen eines Magnetfeldes. Diese wurden früher verwendet als Anzeige für Manipulationsversuche an Wasseruhren, Gaszähler etc. Aber auch bei Alarmanlagen als Fenster- oder Türalarmgeber findet dieser Kontakt Verwendung. Dazu wird am Fenster- oder Türrahmen ein Magnet befestigt. Wenn das Fenster geöffnet wird, entfernt sich der Kontakt vom Magnetfeld und der Schalter öffnet.

6.2. Reed-Relais

Das besondere an REED-Relais ist die kleine Bauform. Dabei ist im Prinzip, der Reed Kontakt nur von einer Spule umwickelt. Fließt Strom durch die Spule, wird ein Magnetfeld erzeugt, das den REED-Kontakt - je nach Ausführung- schließt, bzw öffnet.

6.3. Servomotor



Der links abgebildete Servomotor hat ein Nylongetriebe, damit ist es möglich Schalter zu betätigen. Das besondere an einem Servomotor gegenüber einem



Gleichstrommotor Bestell-Nr.: **820081**

Bestell-Nr.: **820230**

ist, dass der Servo nur einen gewissen Winkel drehen kann.

6.4. Gleichstrommotor (Rotationsmotor)



Diese Art der Motoren wurde früher im Modellbau eingesetzt. Die Drehzahl wurde bestimmt durch die angelegte Spannung. Die Drehrichtung des Motors hängt von der Polarität der angelegten Spannung ab.

Bestell-Nr.: **310722**

6.5. Schrittmotor



Der Schrittmotor ist eine besondere Art eines Motors. Er benötigt zur Ansteuerung ein bestimmtes Feld, bzw. eine bestimmte Abfolge der Feldänderung. Dafür werden auch extra Ansteuerbausteine benötigt. Das besondere ist, dass dieser Motor sich besonders genau steuern lässt und besonders bei Robotern in der Industrie und bei 3D Druckern Verwendung findet.

Bestellnummer: **310543**

7.Module

Module sind Baugruppen, die auf einer fertig aufgebauten kleinen Leiterplatte geliefert werden. Die Kommunikation mit einem Mikrocontroller geschieht über einfache analoge oder digitale Ausgänge, bei komplexeren Modulen ist entweder eine serielle asynchrone Schnittstelle (UART) oder eine synchrone Schnittstelle (SPI, I2C-Bus) enthalten. Bei der Synchronen Datenübertragung erfolgt zur Datenübertragung eine Impulsübertragung, damit kann mit jedem Impuls synchron ein Datenbit abgefragt werden. Die als I2C-Bus bezeichnete Schnittstelle wurde von PHILIPS entwickelt und patentiert. So wurden Mitbewerber gezwungen ein ähnliches System, den SPI-Bus, auf den Markt zu bringen. Der I2C Bus hatte jedoch den Vorteil, dass mehrere Bausteine damit adressiert werden können. Es sind also immer nur zwei Leitungen zur Kommunikation mit mehreren Bausteinen erforderlich, was Platz auf der Platine und auch Anschlüsse an den IC's und Mikrocontrollern einspart.

7.1. Tastenfeld mit Touch:



Bei dem links abgebildeten Tastaturmodul handelt es sich um eine Touch Tastatur Matrix. Das besondere ist, dass es hier möglich ist, allein durch zwei Signale des Mikrocontrollers den Zustand aller 16 Tasten des Moduls abzufragen. Die Tasten sind nur als Fläche auf der Leiterplatte ausgeführt. Es gibt keine mechanisch zu betätigenden Tasten. Durch das Blut in den Händen des Benutzers wird das elektrische Feld an den jeweiligen Taster-Flächen verändert.

Bestell-Nr.: **810271**

7.2. RTC Modul PCF8563:



Mit dem Uhren Baustein PCF8463 und der auf der Platine aufsteckbaren Batterie ist es möglich die Uhrzeit immer parat zu haben, ohne dass der Mikrocontroller und der Programmierer sich kümmern müssen. Dabei wird sogar das Datum und das Schaltjahr berücksichtigt. Nur für die Umstellung auf Sommerzeit benötigt man ein DCF77 Modul. Dieses Modul ist auch für 5V geeignet, obwohl 3V3 an der Pinbezeichnung steht.

Bestell-Nr.: **810515**

7.3. DCF77 Zeitzeichenempfänger

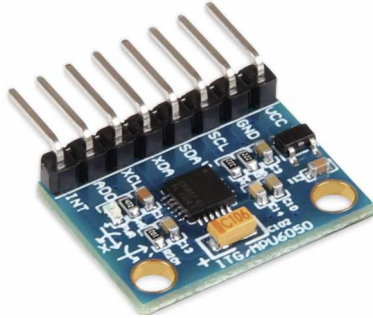


Das DCF77 Modul dekodiert das Zeitzeichensignal der PTB in Braunschweig. Der Sender ist in Frankfurt-Mainflingen und sendet im Langwellenbereich bei 77kHz ein Signal, das in ganz Europa empfangbar ist, und auch noch kostenlos. Der Baustein benötigt lediglich eine Spannung von

Bestell-Nr.: **810054**

5V um ein decodiertes Signal zu liefern. Leider ist die Auswertung nicht ganz so einfach, weil Störungen z.B durch WLAN Router, oder Elektromotoren die Signalqualität beeinträchtigen.

7.4. Beschleunigungsmodul



Das Prinzip dieses Sensors beruht auf einer Kapazitätsmessung:

Wirkt eine Kraft auf den Sensor, so verschiebt sich eine Achse. Auf dieser Achse sind mehrere Platten angebracht. Dadurch ändert sich die Kapazität zu den feststehenden Platten. So lässt sich die Beschleunigung in jeder Richtung des Raumes messen.



Bestell-Nr.: **811107**

7.5. Ultraschall Modul –



Das Ultraschallmodul sendet ein Impulspaket mit ca. 40kHz Trägerfrequenz. Gestartet wird ein Impulspaket dadurch, dass am Pin: „TRIG“ ein kurzer Impuls geschaltet wird. Nach der Laufzeit des Ultraschallsignals, erscheint am Pin:“ECHO“ ein Signal. Aus der Zeitdifferenz des Signals ECHO und TRIG (Schallgeschwindigkeit $\sim 340\text{m/s}$) kann die Entfernung des Objektes berechnet werden .

Bestell-Nr.: **810481**

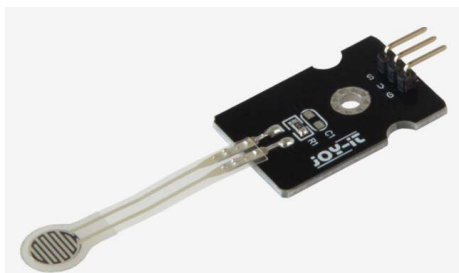
7.6. Infrarot-Abstandsensord

Beim Infrarot Abstandsensord wird ein Moduliertes Signal (Impulspakete der HIGH-Pegel unterschiedliche Werte haben) von einer Sendediode abgestrahlt. Aus der Größe des empfangenen Signals können Rückschlüsse auf die Entfernung gezogen werden. Dabei sollte das Objekt nicht schwarz sein. Leider ist dieser Sensor nicht linear. Das bedeutet ein Objekt das die doppelte Entfernung hat, liefert ein deutlich kleineres Signal als nur die Hälfte des Erwarteten.



Bestell-Nr.: **810480**

7.7. Kraftsensor



Der Kraftsensor unterliegt hohen Toleranzschwankungen. Deshalb ist er nur quantitativ zur Auswertung der Gewichtskraft geeignet. Er verringert seinen Widerstandswert mit zunehmender Kraft. Es gibt diese Widerstände z.B: für 500g, 2kg und 10kg. Für exakte Messungen des Gewichtes kommt nur eine Wägezelle in Frage. siehe auch 5.1.

Im Bild: SEN-pressure02 für 2kg mit der Bestell-Nr.: **811270**

7.8. Farbsensor

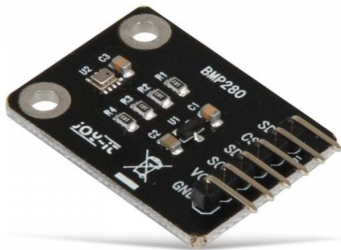


Der Farbsensor sendet nacheinander drei Farben rot, blau, grün. Aus den reflektierten Signalen lässt sich die Farbe berechnen. Den jede Farbe absorbiert die Rot- Grün- und Blautöne unterschiedlich.

Es gibt auch schon digitale Sensoren, deren Farbwert über ein Busprotokoll vom Mikrocontroller abgefragt werden kann.

Bestell-Nr.: **810681**

7.9. Luftdrucksensor



Der Luftdrucksensor besitzt eine Widerstandsbrücke, mit einem Piezo resistiven Sensor. Dieser ändert mit schwankendem Luftdruck seinen Widerstandswert. Allerdings besitzt dieser Sensor bereits eine Auswerteelektronik und ein SPI-Interface zum Auslesen der Daten vom Arduino.

Bestell-Nr.: **810918**

7.10. SD-Speicherkartenmodul



Damit können SD Karten ausgelesen werden. Es können Messwerte von Sensoren abgespeichert werden, oder WAV Dateien gelesen und abgespielt werden.

Bestell-Nr.: **810359**

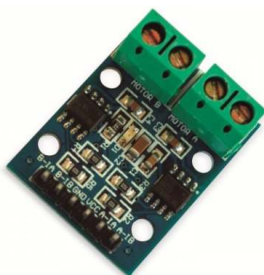
7.11. Relais Modul



Um größere Lasten, die höhere Spannungen oder höhere Ströme benötigen, mit einem Mikrocontroller zu schalten, verwendet man ein Relais. Dies ist relativ kostengünstig, braucht aber immer eine Ansteuerspannung, damit der Schaltstatus erhalten bleibt. Durch die Spule entstehen auch Wärmeverluste.

Bestell-Nr.: **340951**

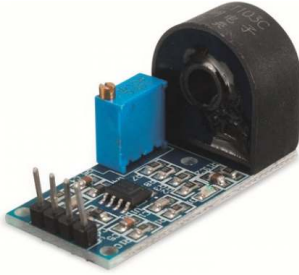
7.12. Motorantriebsmodul 9110



Dabei sind zwei FET IC's als Brücke geschaltet. Es kann jeder Ausgang auf Versorgung oder auf Masse geschaltet werden. Damit kann die Drehrichtung des Gleichstrommotors geändert werden. Oder wenn beide Ausgänge auf Masse geschaltet werden, wird der Motor gebremst.

Bestell-Nr.: **810572**

7.13. Stromsensor - Strom-Wandlermodul



Das stromführende Kabel ist durch die Öffnung des Sensors durchzuführen. Dadurch wird wie bei einem Trafo, auf der Sekundärseite eine Spannung induziert. Diese wird durch das Magnetfeld des stromführenden Kabels erzeugt. Leider funktioniert dieser Stromwandler nur mit Wechselstrom. Das Ausgangssignal ist eine Spannung, die den Stromverlauf zeitlich nachbildet.

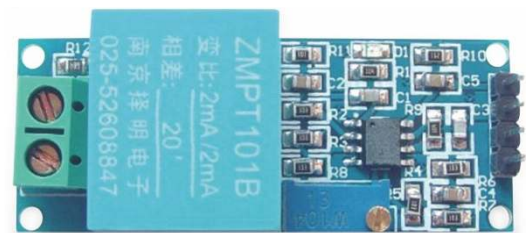
Bestell-Nr.: 810575

7.14. Einphasen AC Spannungswandler Modul

Damit ist es möglich Wechselspannung zu messen. Allerdings wird auch Wechselspannung von diesem Modul ausgegeben.

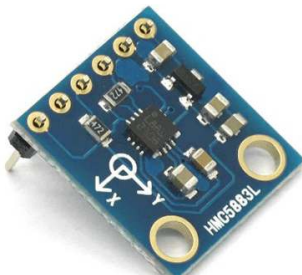
Kenndaten für das Modul:

- Nennstrom: 2 mA
- Betriebsspannung: 5- 30 V-
- Eingangsspannungsbereich: 1- 1000 V~
- Isolationsspannung: 4000 V
- Ausgangsspannung: 1 / 2 Vcc
- Eingang: Schraubklemmen
- Maße(LxB): 50x19 mm



Bestell-Nr.: 810581

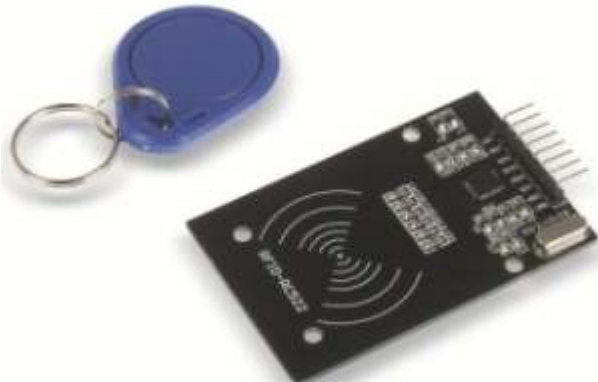
7.15. HMC5883 Kompassmodul



In diesem IC sind drei Widerstandsmessbrücken integriert. In jeder Messbrücke befindet sich ein magnetfeldabhängiger Widerstand. Dadurch ist es möglich, das Erdmagnetfeld in drei Achsen zu vermessen. Die Signale der Messbrücken, werden verstärkt und im IC digital verarbeitet. So ist es möglich die Daten über einen Bus mit zwei Leitungen abzufragen.

Leider nicht mehr im Bestand

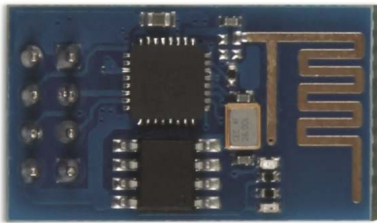
7.16. RFID MFRC522



Die Nahfeldkommunikation funktioniert nach dem Induktionsprinzip. Im Empfänger TAG befindet sich eine Antenne. Wird diese kurz geschlossen, so wird das Magnetfeld gedämpft. Ähnlich wie bei einem Trafo, dem die Sekundärseite kurzgeschlossen wird. Diese Änderung der Signalamplitude des Senders wird gemessen. So kann der TAG ein moduliertes Signal erzeugen. Wie beim Morsen, kann so der TAG dem Sender seine Nummer und andere Informationen mitteilen.

Bestell-Nr.: 810678

7.17. WLAN-ESP8266-01

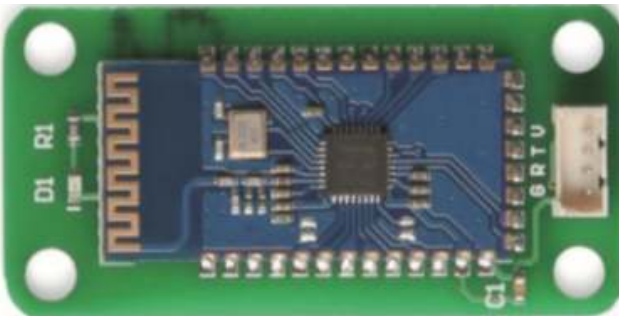


Mit diesem kleinen Board, lässt sich der Arduino mit der großen Welt des Internets verbinden. Er kann die Feldstärke der empfangenen Router anzeigen.

Er kann einen Arduino zu einer Webseite machen, die am PC angezeigt werden kann.

Bestell-Nr.: **810670**

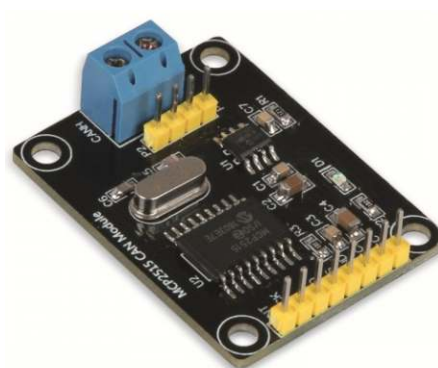
7.18. Bluetooth-Modul



Mit dem Bluetooth-Modul ist es möglich, dass der Arduino eine Verbindung zu einem Smartphone aufbauen kann. Damit können Daten oder Befehle einer APP vom Arduino abgefragt bzw. ausgeführt werden.

Bestell-Nr.: **810928**

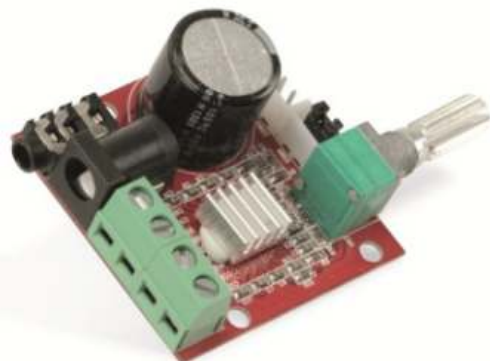
7.19. CAN-Modul



Damit ist es möglich, über die Diagnosebuchse am Auto einige Daten abfragen.

Bestell-Nr.: **810908**

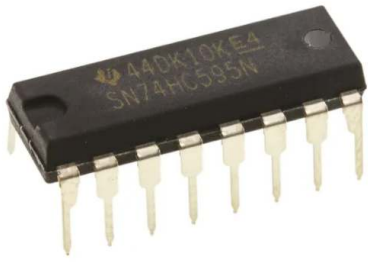
7.1.20. Verstärkermodul



Damit kann z.B. ein vom Mikrocontroller erzeugtes PWM Signal verstärkt werden. Es ist nur eine Spannung von größer 9V nötig und ein Lautsprecher.

Bestell-Nr.: **630561**

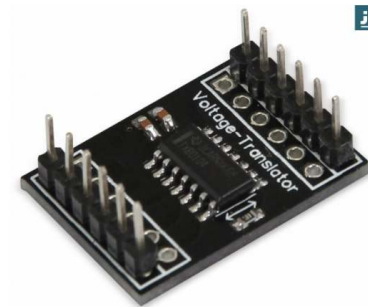
7.21. I/O Modul: Buserweiterung



74hc595 (Bestell-Nr.: **101267**)

Mit diesem IC ist es möglich ein Lauflicht anzusteuern, ohne weitere IO-Pins am Arduino verschwenden zu müssen.

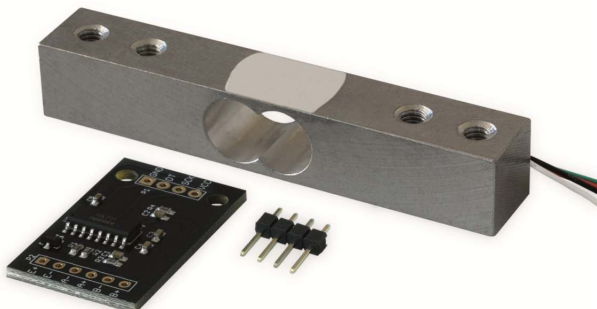
7.22. Level-shifter



Manchmal ist es notwendig einen Pegelwandler zu benutzen, weil Sensoren nur mit 3,3V statt mit 5V betrieben werden können. So kommt ein Level-shifter zum Einsatz, um diesen Sensor an einem Arduino trotzdem betreiben zu können. Dabei werden die Datensignale vom und zum Arduino von 5V auf 3,3V und umgekehrt gewandelt.

Bestell-Nr.: **810924**

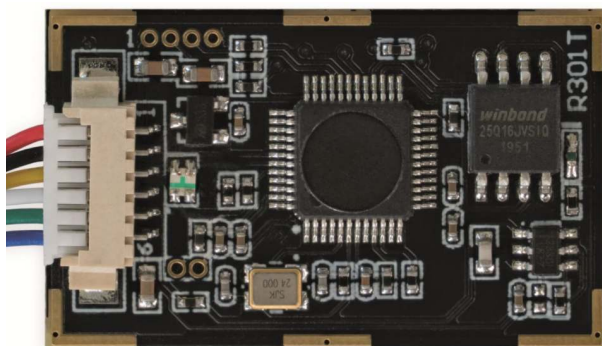
7.23. Wägezelle



Eine Wägezelle besteht aus zwei Dehnungsmessstreifen. Diese werden an die Verstärkerplatine angelötet. Die Auswertung der Daten erfolgt durch einen Mikrocontroller über die I2C-Schnittstelle. Die Spannungsversorgung beträgt 5V. Wägezellen gibt es in verschiedenen Gewichtsabstufungen. Hier kommt die 10kg Variante zum Einsatz.

Bestell-Nr.: **811419**

7.24. Fingerprint-Modul



Dieses Sensormodul ist durch den integrierten Chip in der Lage, Bilder zu sammeln und Algorithmen zu berechnen. Eine weitere bemerkenswerte Funktion des Sensors ist, dass er Fingerabdrücke unter verschiedenen Bedingungen erkennen kann, wie z.B. Feuchtigkeit, Lichtbeschaffenheit oder Veränderungen der Haut. Dies bietet ein sehr breites Spektrum an Anwendungsmöglichkeiten, unter anderem zur Sicherung von Schlössern und Türen. Der Chip kann Daten über UART an den angeschlossenen Controller senden.

Bestell-Nr.: **180132**

8. Der ATMEGA – ein viel verwendeter Mikrocontroller

8.1. Der ATMEGA328 im Arduino Nano und Arduino uno

Die exakte Beschreibung über die internen Abläufe eines Mikrocontrollers würde den Umfang dieses Dokumentes sprengen. Deshalb möchte ich mich hier auf das wesentliche Beschränken. Auch möchte ich nicht die Details haarklein erklären. Sondern ich möchte hier nur durch Modelle versuchen, den Funktionsablauf so weit zu behandeln, dass es einigermaßen verständlich wird, was man braucht um den Prozessor zu programmieren. Denn Programmieren bedeutet, dem Prozessor nicht nur Befehle zu geben, welche Daten er wie zu verarbeiten hat, ähnlich wie bei einem PC. Der Prozessor im PC hat Peripheriekomponenten, die den Bildschirm ansteuern, die Daten auf einer Festplatte speichern, die Daten aus dem USB Stick lesen, die die Kommunikation mit dem USB Geräten übernehmen.

Beim Mikrocontroller ist die Peripherie schon integriert. Da ist es die Aufgabe des Programmierers, dass die Kommunikation mit der Außenwelt von ihm erledigt wird. Bevor ein Programm ausgeführt werden kann, muss zuerst festgelegt werden, welche Module braucht man, und wie sollen sie ihre Arbeit erledigen. Der Mikrocontroller führt jeden Befehl des Programmierers exakt und blitzschnell aus. Aber er denkt nicht darüber nach. Das ist so, wenn ich einem Fahrer sage, wenn die Ampel auf grün schaltet, erst dann fahre los. Der Prozessor macht das auch, aber er schaut nicht, ob das Garagentor auch schon geöffnet ist. Das ist die Aufgabe des Programmierers.

Das rechte Bild ist einem Datenblatt von ATMEL entnommen.

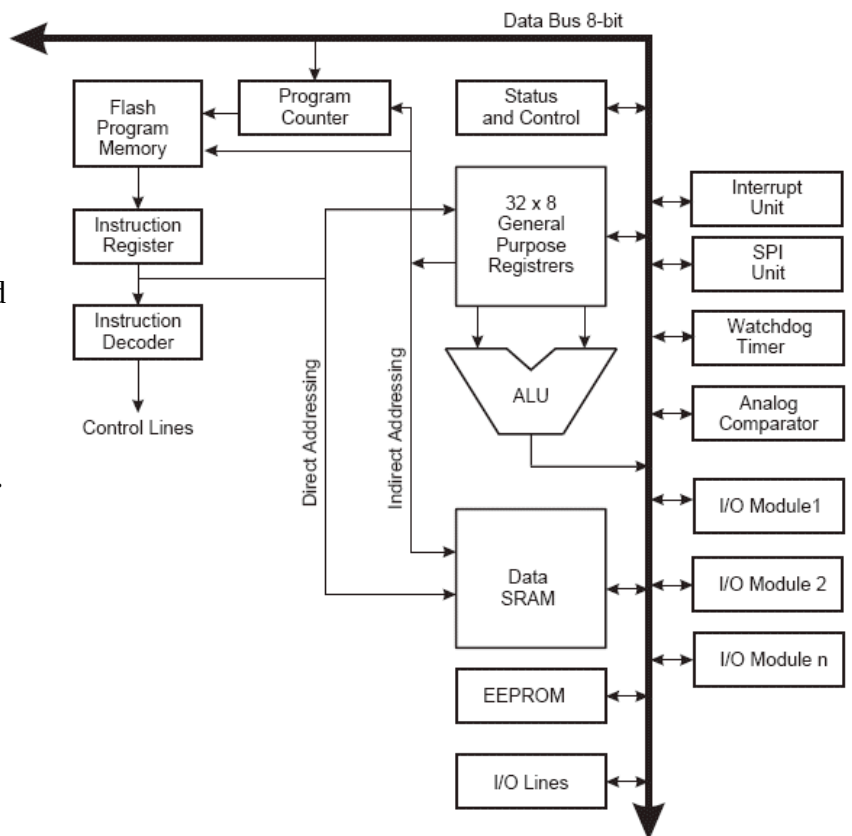
Es zeigt den schematischen Aufbau eines Mikrocontrollers.

Man kann sich das ganze vorstellen wie in einer Fabrikhalle.

Im Raum „Flash Program Memory“ ist der Auftragseingang. Dort hin wird das Programm gespeichert, das der Mikrocontroller ausführen soll. Man kann sich das vorstellen, als wenn jeder Auftrag eine eigene Schublade bekommt, in der dieser abgelegt wird. Im „Instruction Register“ werden die Aufträge (Befehle) nacheinander abgeholt, gespeichert und ausgeführt.

Der „Instruction Decoder“ zerlegt die Aufträge in einzelne Schritte. Der Computer hat seine eigene Sprache, die nichts mit der Sprache zu tun hat, wie sie hier verwendet wird. Deshalb braucht er quasi einen Übersetzer.

Der 8 Bit Datenbus kann man sich vorstellen wie eine Art Elektrokarren mit acht Ablagefächern auf der Ladefläche. Dieser fährt immer im Kreis und beliefert jede Abteilung. Dies ist auch die einzige Verbindung zwischen den Abteilungen. Wenn ein Befehl abgeholt wird ist er eine zusammengesetzte Zahl. Diese muss zuerst aufgeschlüsselt werden, So wie oftmals ein Bestellcode, in dem verschlüsselt ist, welcher Artikel das ist, welcher Produktgruppe er angehört, und wo im Lager er sich befindet. Beim Mikrocontroller ist es ähnlich, er muss wissen welcher Befehl auszuführen ist,



mit welchen Variablen, und wo diese abgelegt sind.

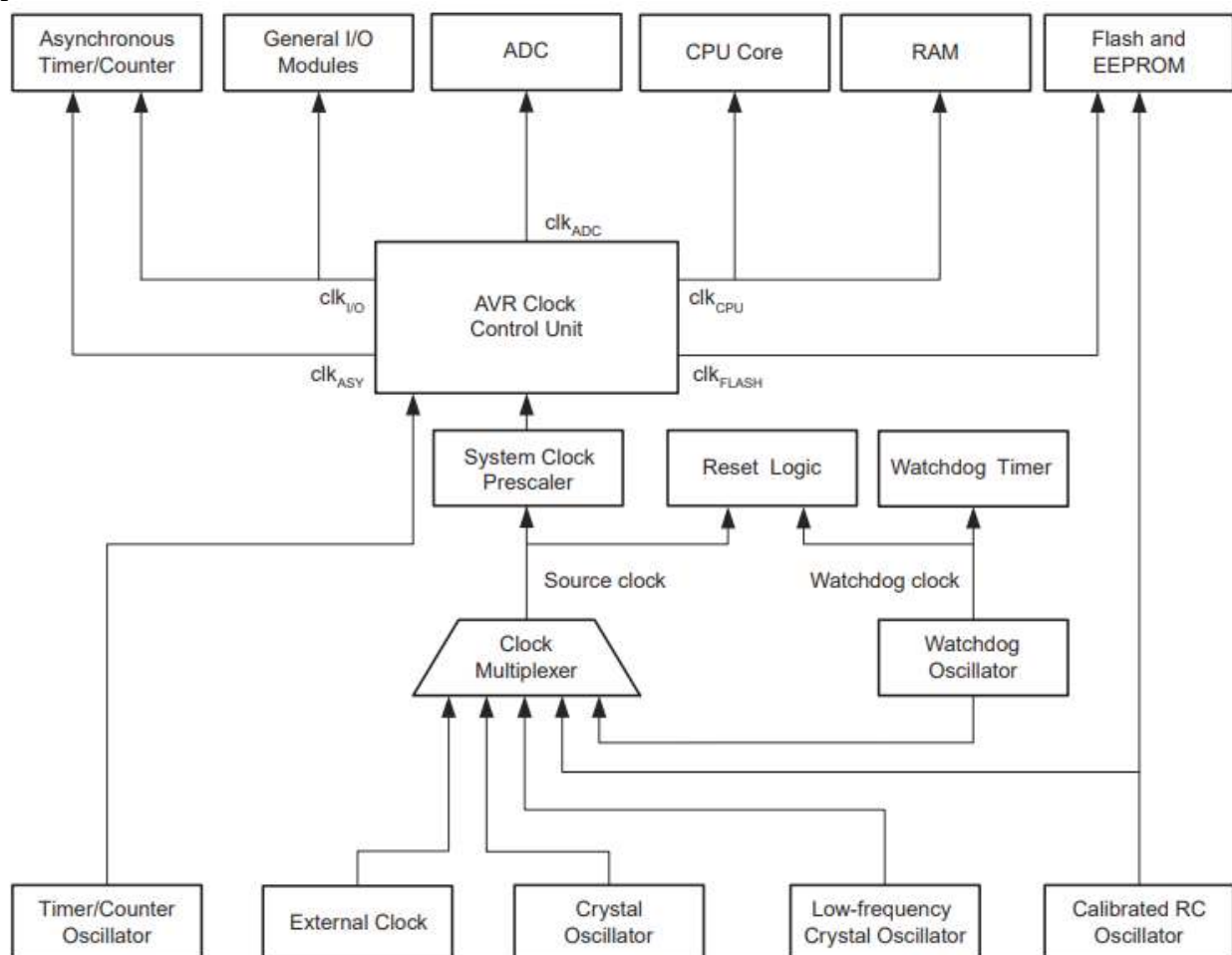
Ein ganz wichtiges Detail wie die meisten Mikrocontroller aufgebaut sind ist der Speicher. Es gibt nämlich eine strikte Trennung zwischen Programmcode und Variablenwerten. Variablenwerte werden im RAM abgelegt. Der Programmcode im Flash Programm Memory. Wenn man aus Platzgründen, weil der RAM-Speicher begrenzt ist, Variable oder Konstanten im Programmspeicher abgelegt, gibt es dann spezielle Befehle, um während der Programmlaufzeit, auch auf diese Werte zugreifen zu können. Dabei können diese nur gelesen und nicht verändert werden!

Die Werte können nur im Programmcode geändert werden. Beim Schreiben des Programms in den Arduino werden sie in dessen Flashspeicher abgelegt.

So können individuelle Texte abgespeichert werden, oder Wertetabellen zum Linearisieren von Temperatursensoren.

Ein spezieller Speicher ist das EEPROM. Es ist elektrisch löschar und beschreibbar. In einem RAM gehen die Werte verloren, wenn der Arduino spannungslos wird. Aus dem Flash-Speicher kann während der Programmlaufzeit nur gelesen werden. Um Variablen mit Konfigurationsdaten für den nächsten Programmstart zwischen zu speichern, wird das EEPROM benutzt. Beim EEPROM ist zu beachten, dass ein Schreibzugriff eine gewisse Zeit dauert und auch eine bestimmte Höhe der Spannung am Prozessor anliegen muss. Sonst könnte es passieren, dass Daten nicht oder nicht vollständig abgespeichert werden.

Das Speichern ist ein komplexer Prozess, um den sich der Programmierer allerdings nicht kümmern braucht. Beim Arduino gibt es eine vorprogrammierte Funktionen, in Bibliotheken, die sowohl das Speichern als auch das Lesen übernehmen.



Das Herzstück für die Funktion eines Mikrocontrollers ist der Oszillator. Meistens wird ein kleiner Quarz außen an den Controller angeschlossen.

Die Oszillatorfrequenz wird im ganzen Mikrocontroller verteilt. Fast jede Komponente braucht das Zeitsignal um richtig arbeiten zu können und um die Daten zum richtigen Zeitpunkt, wenn der Controller sie erwartet, auch zur Verfügung zu stellen.

Der Arduino besitzt z.B. mehrere Timer, die ein Register besitzen, um den Takt weiter herunter zu teilen, um die Zeiten verlängern zu können. Außerdem haben die Register unterschiedliche Größen. Es gibt welche mit der doppelten Größe. Diese haben dann bei gleicher Taktrate die 254 fache Länge, was die einzustellende Zeit anbelangt.

Das nächste sind die seriellen Schnittstellen, wenn da die Daten nicht zum richtigen Zeitpunkt und in der gültigen Zeitdauer anliegen, so sendet der Prozessor nur fehlerhafte Daten oder er liest die Daten falsch.

Der AD-Wandler braucht eine korrekte Frequenz um die gültigen Daten jedes Eingangs zum richtigen Zeitpunkt und in der richtigen Dauer an den Datenwandler anzulegen.

Der Zugriff auf den Speicher erfolgt auch nach genau definierten Zeitabläufen.

Die Funktion des Watchdog timers ist, zu überwachen, dass die Rechenabteilung nicht zu lange an einer Aufgabe hängt. Es kann sein, dass der Programmierer einen Fehler in seinen Anweisungen gemacht hat und Rechenabteilung diese Aufgabe nicht erledigen kann. Dann kann man eine Zeit vorgeben, wenn die Rechenabteilung kein Ergebnis liefert, dass der Wachhund bellt und der Elektrokarrenfahrer die Rechenabteilung anweist, von vorne zu beginnen. Der Mikrocontroller führt somit einen Reset durch, der vom Watchdog veranlasst wurde.

Weiter gibt es noch die I/O Module. Diese kann man sich wie Garagentorwächter vorstellen. Sie bekommen die Aufgabe, das Garagentor zu öffnen, zu schließen, einen LKW einfahren zu lassen, oder ausfahren zu lassen zu beladen oder zu entladen. Manche werden auch nur als Fenster benutzt, um zu sehen, wie voll oder leer der LKW ist, der gerade an der Einfahrt steht.

Bei den ersten Mikrocontrollern und bis in die 90er Jahre hinein, waren die meisten Mikrocontroller mit einer ALU ausgestattet, die nur ein einziges Register hatte, das zur Addition und Subtraktion und zum Links- oder Rechts-shift verwendet wurde. Ansonsten waren nur Speicheroperationen und logische Operationen möglich. Eine Multiplikation oder Division musste vom Programmierer in eine Reihe von Additionen oder Subtraktionen in Programmcode umgesetzt werden. So als gäbe es nur einen PC in einer Firma. Das Register mit dem bei den älteren Mikrocontrollern gerechnet wurde / wird heißt Akkumulator.

Nur in diesem Register sind Operationen möglich, weil nur dieses Register an ein Statusregister angeschlossen ist, das nach den Rechenoperationen oder logischen Operationen angezeigt, ob ein Übertrag entstanden ist, ob das Ergebnis 0 ist usw.

Um eine Variable zu bearbeiten, musste diese zuerst in den Akkumulator geladen werden, bevor sie verändert werden konnten. Nach der Berechnung, oder der Durchführung von logischen Operationen wurde sie wieder in ihrem Speicherplatz abgelegt. Dann erst konnte die nächste Variable bearbeitet werden.

Mittlerweile haben auch die XMEGA Controller zumindest einen Hardware Multiplizierer. Eine Division ist aber bei diesen Mikrocontrollern nicht eingebaut.

Eine ganz wichtige Einrichtung bei den ATMEGA's ist deshalb das General Purpose Register. Dies kann man sich vorstellen wie 32 Akkumulatoren. Man kann also direkt Variablen in diese Register speichern und bearbeiten. Dabei führt die ALU (Arithmetic Logical Unit) die Operationen zwischen den Registern aus und speichert den Wert wieder in einem der Register ab.

So kann man salopp sagen, mittlerweile gibt es in der „Firma: Arduino“ 32 Rechenabteilungen.

In der Abbildung auf Seite von Kapitel 8.1. sind noch weitere Register mit speziellen Funktionen abgebildet:

Der **Program Counter** (PC) ist auch ein ganz wichtiges Register. Darin wird abgelegt welche Befehlszeile gerade ausgeführt wird. Kommt dabei etwas durcheinander, so kann es zu Fehlfunktionen oder Programmabstürzen kommen.

Der Stack Pointer ist dabei eine mögliche Fehlerquelle. Wenn zum Beispiel ein Interrupt verursacht wird, springt der Programmzähler in einen ganz anderen Speicherbereich. Daher muss der augenblickliche Wert und oftmals auch Variablen zwischengespeichert werden, damit sie nicht überschrieben werden. Dazu dient der Stackpointer. Dies ist nichts anderes als ein Stapel von Registern. Wenn diese beschrieben werden, muss man sich die Reihenfolge merken, mit der die Daten abgelegt wurden. Der Stackpointer ist quasi nur ein Regal, auf das Dinge abgelegt werden. Welche Daten das sind ist dem Stackpointer egal, und auch in welcher Reihenfolge sie abgespeichert und wieder abgeholt werden. Wenn die Reihenfolge vom Abspeichern und vom Abholen der Werte, nicht gleich sind, werden die Inhalte von Variablen vertauscht. Oftmals wird dies sogar gewollt, nämlich, wenn der Programmierer Variablen in ihrer Größe nach sortieren will. Dann kann dieses Verhalten genutzt werden, um die Werte zweier Variablen zu tauschen.

Status und Kontrollregister: Im Status Register werden sogenannte Flags gesetzt. Diese kann man sich wirklich als eine Flaggen vorstellen, die von der ALU gehisst werden, nach einer mathematischen Operation.

Ihre Aufgabe ist es, das Auftreten bestimmter Ereignisse zu signalisieren, die während einer Rechenoperation auftreten können.

Die Bedeutung der einzelnen Flaggen (Bits) im Statusregister

I T H S V N Z C

Carry (C)

Das Carry Lag hält fest, ob es bei der letzten Berechnung einen Über- oder Unterlauf gab.

Zero (Z)

Das Zero Flag hält fest, ob das Ergebnis der letzten 8-Bit Berechnung 0 war (Z=1) oder nicht.

Negative (N)

Spiegelt den Zustand des höchstwertigen Bits (Bit 7) der letzten 8-Bit-Berechnung wieder. In 2-Komplement Arithmetik bedeutet ein gesetztes Bit 7 eine negative Zahl, das Bit kann also dazu genutzt werden um festzustellen ob das Ergebnis einer Berechnung im Sinne einer 2-Komplement Arithmetik positiv oder negativ ist. Auf das Rechnen im Binärsystem wird später noch detaillierter eingegangen.

Überlauf (V)

Dieses Bit wird gesetzt, wenn bei einer Berechnung mit 2-Komplement Arithmetik ein Überlauf (Unterlauf) stattgefunden hat. Dies entspricht einem Überlauf von Bit 6 ins Bit 7.

Der Übertrag, der bei der Addition/Subtraktion von Bit 6 auf Bit 7 auftritt, zeigt daher – wenn er vorhanden ist – an, dass es sich hier um einen Überlauf (Overflow) des Zahlenbereichs handelt und das Ergebnis falsch ist. Das ist allerdings nicht der Fall, wenn auch der Übertrag von Bit 7 nach Bit 8 (Carry) aufgetreten ist. Daher ist das Overflow-Flag die XOR-Verknüpfung aus den Übertrag von Bit 6 nach Bit 7 und dem Carry.

Sign (S)

Das Sign-Bit ergibt sich aus der Antivalenz der Flags N und V, also $S = N \text{ XOR } V$. Mit Hilfe des Signed-Flags können vorzeichenbehaftete Werte miteinander verglichen werden. Ist nach einem Vergleich zweier Register $S=1$, so ist der Wert des ersten Registers kleiner dem zweiten (in der Signed-Darstellung). Damit entspricht das Signed-Flag gewissermaßen dem Carry-Flag für Signed-Werte. Es wird hauptsächlich für 'Signed' Tests benötigt. Daher auch der Name.

Hilfs-Carry (H)

Das Half Carry Flag hat die gleiche Aufgabe wie das Carry Flag, nur beschäftigt es sich mit einem Überlauf von Bit 3 nach Bit 4, also dem Übertrag zwischen dem oberen und unteren Nibble. Wie beim Carry-Flag gilt, dass das Flag nicht durch Inkrementieren bzw. Dekrementieren ausgelöst werden kann. Das Haupteinsatzgebiet ist der Bereich der BCD Arithmetik, bei der jeweils 4 Bits eine Stelle einer Dezimalzahl repräsentieren.

Transfer (T)

Das T-Flag ist kein Statusbit im eigentlichen Sinne. Es steht dem Programmierer als 1-Bit-Speicher zur Verfügung. Der Zugriff erfolgt über die Befehle Bit Load (BLD), Bit Store (BST), Set (SET) und Clear (CLT) und wird sonst von keinen anderen Befehlen beeinflusst. Damit können Bits von einer Stelle schnell an eine andere kopiert, gesetzt oder getestet werden.

Interrupt (GI)

Das global Interrupt enable Flag fällt hier etwas aus dem Rahmen; es hat nichts mit Berechnungen zu tun, sondern steuert ob Interrupts im Controller zugelassen sind oder nicht. Interrupts sind sehr nützlich. Sie erleichtern die Arbeit beim Programmieren sehr. So kann man einen Interrupt abfragen, ob eine bestimmte Zeit vergangen ist. Oder es wird signalisiert, wenn an einem Eingang ein bestimmtes Signal anliegt. Wenn es keine Möglichkeit der Interrupts gäbe, müsste man immer die Zeit abfragen, wie viel schon vergangen ist. Oder immer einen Eingang um zu sehen ob sich der Signalzustand verändert hat. Oder man müsste fragen, ob an einer Schnittstelle bestimmte Daten angekommen sind. Dies wäre alles sehr aufwändig und würde dem Controller sinnlose Zeit kosten. Der einzige Nachteil ist, dass das Flag für alle Interrupts gilt. Aber in der Regel benutzt man eh nur einen, so dass der Programmierer weiß, welcher Interrupt ausgelöst wurde und so auf das Ereignis reagieren kann. Ansonsten müsste man eben jede Interruptroutine abfragen, was aber weniger aufwendig ist, als ohne Interrupt zu arbeiten. Diese Aufgabe wird von der Interrupt Unit erledigt und unterstützt so den Programmierer. Die analog Komparator Unit ist zuständig permanent einen Wert abzufragen. Der Programmierer legt nur fest welcher Wert (welcher Anschlusspin) abgefragt wird, und die Höhe des Schwellwertes. Wenn dieser erreicht ist, so schlägt der Operator Alarm und gibt dem Elektrokarren Bescheid, dass der Fahrer die entsprechende Flagge setzt.

Multiplexen der Ein- und Ausgänge:

Es gibt noch Register, die festlegen welcher Anschluss des Controllers welche Funktion hat. Ein Pin kann mehrere Funktionen haben. So kann jeder ein Ein- oder Ausgang sein. Zusätzlich haben manche Pins die Fähigkeit, die Zeit messen, oder ein zeitgesteuertes Signal ausgeben; oder irgendwelche Spannungswerte ablesen und diese an die Rechenabteilung weitergeben. Die SPI-Unit ist eine synchrone serielle Schnittstelle, die ein Display ansteuern kann, oder Messwerte von einem Sensor-Modul einlesen; dabei hat dieses serielle Schnittstelle den Vorteil, dass mehrere Sender oder Empfänger angeschlossen sein können. Denn zusätzlich zu jeder Fuhre an Daten, wird jeder Datencontainer mit einer Adresse versehen, so dass jedes Modul weiss, welches angesprochen wird. Bei der seriellen Schnittstelle gibt es nur einen möglichen Adressaten!

Es gibt die ALU. Sie ist das Herzstück des Mikrocontrollers. Das ist die Rechenabteilung. Hier wird nur addiert und subtrahiert. Aber eben eine Million Rechnungen pro Sekunde!

Damit der Rechner auch ausgelastet ist, gibt es ein internes Paket-Verteilsystem der Daten. Die Daten müssen dabei mit jedem Paukenschlag (Clockimpuls) weitergeben werden. Da gibt es genau

definierte Abläufe, wer welche Daten wo abholen und sofort die erhaltenen ablegen muss. Nur so ist eine reibungslose Funktion der Abläufe gewährleistet.

Der Analog-Digital-Converter:

Ein Element des Arduino wurde noch nicht besprochen. Das ist der ADC (Analog Digital Converter). Was bedeutet das?

Ein Computer kann nur Zahlen verarbeiten. Also muss es eine Möglichkeit geben, eine Spannung an einem Eingang des Controllers in eine Zahl umzuwandeln, zu konvertieren.

Um eine Spannung mit dem Controller zu messen, wird diese an einen Kondensator geleitet.

Dann wird dieser Kondensator vom Pin abgeklemmt, damit die Spannung nicht mehr beeinflusst werden kann. Nun wird die unbekannte Spannung am Kondensator mit einer anderen verglichen.

Dabei wird die bekannte Spannung so lange erhöht bis beide Werte gleich sind. Dies geschieht durch zuschalten von Widerständen, von denen 255 in Reihe geschaltet sind. Der Computer zählt genau mit, wie oft die Spannung erhöht wurde in dem immer ein Widerstand dazugeschaltet wurde und kann somit der Spannung einen Wert geben. Aber da erkennt man schon die Problematik an dem ganzen. Es gibt immer einen Messfehler. Dieser ist zwar merklich gering, aber vorhanden.

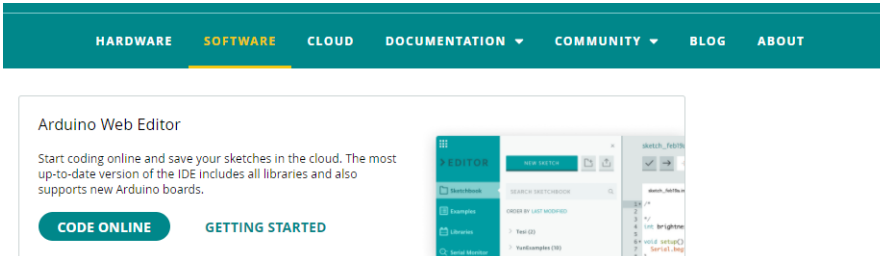
Denn Der Computer hört erst zu zählen auf, wenn die Spannung größer ist, als die zu messende Spannung. Es ist zwar in 99% der Fälle die Genauigkeit ausreichend, aber rein theoretisch ist immer ein Messfehler, ein sogenannter Quantisierungsfehler vorhanden. Denn die Widerstände unterliegen Toleranzen und auch der Komparator. Es gibt keine idealen Bauteile. Dies ist auch der Grund, warum es einige Hi-Fi Enthusiasten gibt, die sich keine Musik von einer CD, sondern nur von einer LP anhören. Diese Menschen behaupten, sie würden den Quantisierungsfehler hören. Beim Abspielen einer CD ist der Vorgang umgekehrt, da wird aus einem digitalen Wert ein analoger erzeugt. Die Spannungen werden ebenfalls mit Widerständen erzeugt, die in Serie geschaltet sind.

9. Die ARDUINO IDE (Entwicklungsumgebung)

Wie bei Wikipedia nachzulesen ist, soll der Name Arduino von einer Bar abgeleitet worden sein, in der sich einige der Gründer dieses Projektes getroffen haben.

9.1. Installation

von arduino.cc unter SOFTWARE herunterladen und installieren.



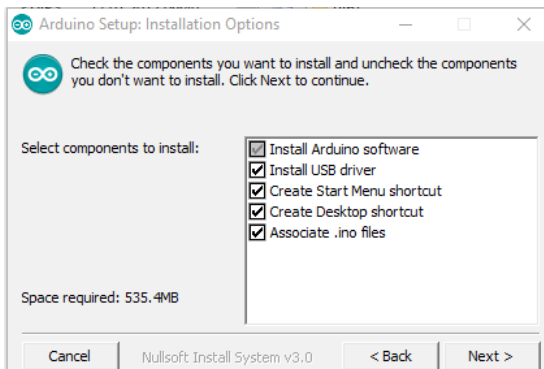
Downloads



Bei den Download OPTIONS die gewünschte auswählen. Für einen Windows 10 Rechner wählen Sie die Option Windows Win7 and newer. Nun erscheint das nachfolgende Fenster:



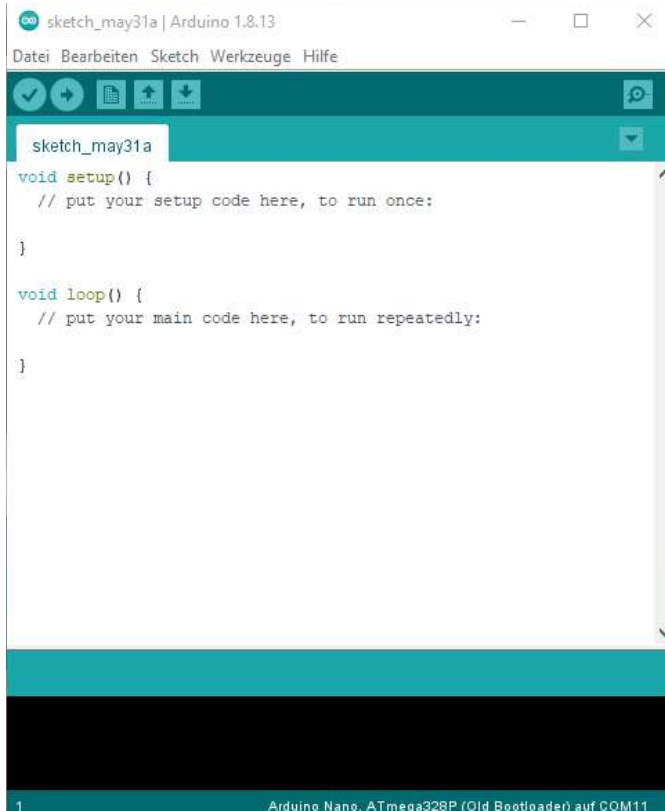
Nach dem Auswählen der gewünschten Option z.B: **JUST DOWNLOAD**, beginnt der Download der Datei. Diese wird dann im Downloadverzeichnis von Windows abgespeichert. Dann in das Downloadverzeichnis wechseln und das exe file aufrufen.



Zuerst muss man zustimmen, dass das Programm auch auf den PC zugreifen darf.

Dann die Auswahl so lassen, dass die Treiber und die Software komplett installiert wird

9.1.2. Programmstart



Nach dem Start der Arduino IDE über den Icon auf dem Desktop oder über die Windows-Startleiste, wird der zuletzt bearbeitete Sketch geöffnet. Mit Datei → Neu oder <STRG>+ <N> wird ein neuer Sketch geöffnet. Auf dem linken Bild ist zu sehen, wie eine neue Datei aussieht. Die wichtigsten Funktionen sind **setup()** zur Initialisierung der Variablen und der benötigten Hardware und **loop()**.

Im Gegensatz zu setup(), werden in der **loop**, wie der Name sagt, die Befehle laufend abgearbeitet. Im Setup() werden die Befehle nur einmal ausgeführt. Deshalb befinden sich dort eher selten Funktionen oder gar ganze Programme.

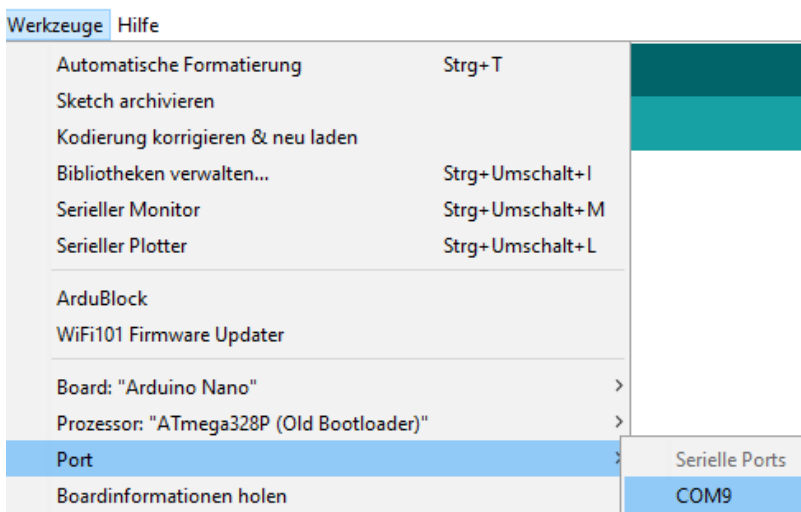
Natürlich kann es in ganz seltenen Fällen vorkommen, dass ein Programm, nach dem Einschalten nur einmal ablaufen soll. Deshalb ist dann der gesamte Programmcode im Setup() geschrieben.

Unten im Fenster wird angezeigt ob und welcher Arduino an einem COM-Port angeschlossen ist. Falls dort keiner angezeigt wird, kann es sein, dass er nicht angesteckt ist, oder nicht erkannt wurde. Jetzt ist es notwendig, nach dem anschließen des arduino Boards, das richtige Board, den Prozessor und den Port auswählen:

Board: Werkzeuge → Board : → „Arduino Nano“;

Prozessor: Werkzeuge → Prozessor: → „ATmega 328P (old bootloader)“ ;

Port: Werkzeuge → Port: → „COM ... “



Bei „Original Arduino-boards“ wird nach COM9 in runden Klammern, der Name des Boards, mit angezeigt. Bei billigeren Nachbauten, wird dies nicht angezeigt, weil bei diesen ein billiger Schnittstellen-baustein CH340 und nicht ein ATMEL32U4 verbaut ist.

Wird kein COM-Port angezeigt, ist vermutlich kein Treiber installiert:

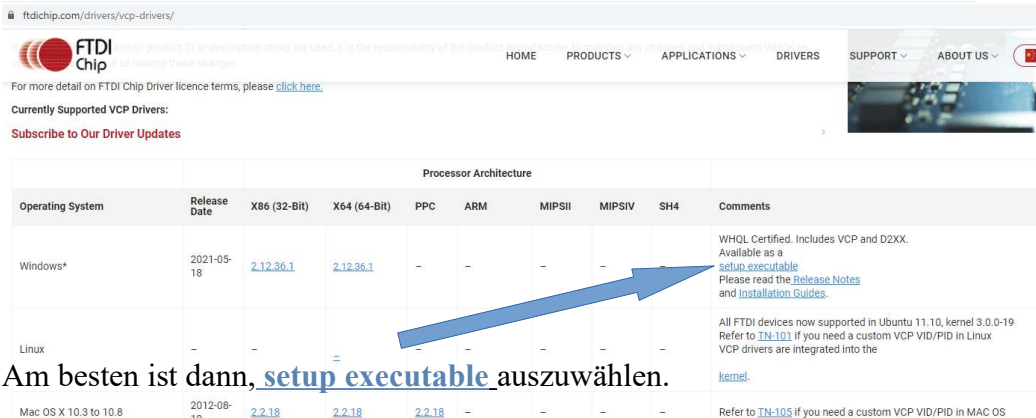
Die Nummer des COM-Ports

hängt vom angeschlossenen Board und davon ab, welche Portnummer vom Betriebssystem vergeben wird. Hier ist es COM9.

9.1.3. Installation eines Treibers

9.1.3.1. bei Original Arduino Board:

Falls es Probleme geben sollte, dass der Treiber für den Arduino nicht aktuell ist, kann dieser bei FTDI heruntergeladen werden:



Operating System	Release Date	Processor Architecture							Comments
		X86 (32-Bit)	X64 (64-Bit)	PPC	ARM	MIPSII	MIPSIV	SH4	
Windows*	2021-05-18	2.12.36.1	2.12.36.1	-	-	-	-	-	WHQL Certified. Includes VCP and D2XX. Available as a setup executable . Please read the Release Notes and Installation Guides .
Linux	-	-	-	-	-	-	-	-	All FTDI devices now supported in Ubuntu 11.10, kernel 3.0.0-19. Refer to TN-101 if you need a custom VCP VID/PID in Linux. VCP drivers are integrated into the kernel .
Mac OS X 10.3 to 10.8	2012-08-10	2.2.18	2.2.18	2.2.18	-	-	-	-	Refer to TN-105 if you need a custom VCP VID/PID in MAC OS

Am besten ist dann, [setup_executable](#) auszuwählen.

9.1.3.2. bei Arduino kompatiblen Board:

Falls Sie einen Arduino Nano, z.B. von Joy.it verwenden, dann ist dort ein CH340 Baustein verbaut. Dann müssten Sie im Internet nach diesem Treiber (CH341SER WIN USB Treiber) suchen. Dieser wird installiert, indem die „.exe“ Datei gestartet wird.

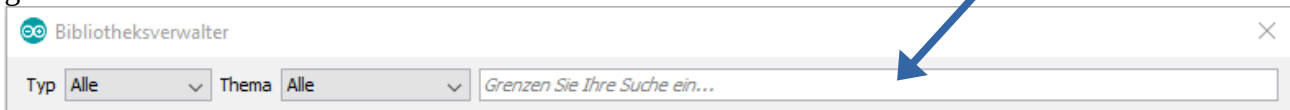
9.2. Bibliotheken

9.2.1. Bibliothek einbinden

Eine Bibliothek ist eine Art vorgefertigter Programmteil. Wenn eine Bibliothek eingebunden ist, dann kann der Programmierer auf weitere Funktionen zurückgreifen. Er kann Sensoren oder angesteckte Module nutzen, ohne sich um deren Funktion kümmern zu müssen. Er muss lediglich beachten, wie die Funktion aufzurufen ist; ob evtl. Daten zu übergeben sind; in welcher Form diese bereitgestellt werden müssen. Zu beachten ist, dass Rückgabewerte zu bearbeiten oder auszuwerten sind, um auf Fehler, Messwerte, oder Statusmeldungen verarbeiten oder ausgeben zu können.

Um Bibliotheken verwenden zu können ist es notwendig, diese zuerst **einzubinden**. Standardmäßig sind schon sehr viele Bibliotheken mit installiert, oder über den Bibliotheksverwalter einzubinden: Dazu „Werkzeuge“ → „Bibliotheken verwalten“ anklicken.

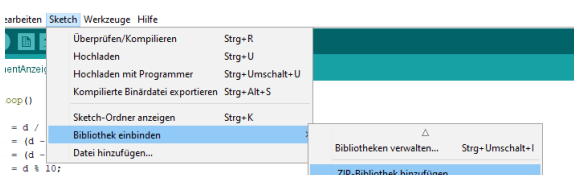
Daraufhin öffnet sich der Bibliotheksverwalter und dann ist es möglich im **Eingabefeld** nach einer gewünschten Bibliothek zu suchen:



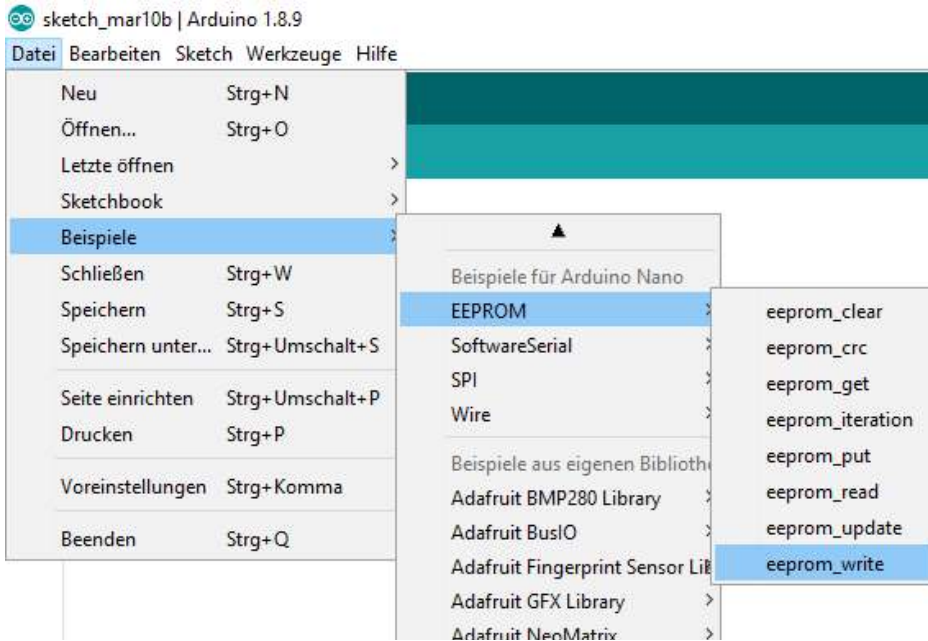
Ist diese jedoch dort nicht zu finden, dann bleibt nur, danach im Internet zu suchen. Meist ist z.B. bei Github eine passende zu finden und kann als *.zip-Datei heruntergeladen werden.

Mit der Funktion **Sketch** → **Bibliothek einbinden** → **.zip Bibliothek hinzufügen**, wird die

Bibliothek ins „Library-Verzeichnis“ der arduino IDE entpackt und gleichzeitig die dazugehörigen Beispiel-Projekte in die IDE eingebunden.



9.2.2. Beispieldatei der Bibliothek laden



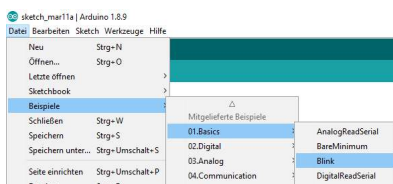
Im linken Bild ist zu sehen, wie eine Beispieldatei aus der Bibliothek für das EEPROM aufgerufen wird. Mit Datei → Beispiele → EEPROM können die gewünschten Dateien mit dem Programmcode geladen werden.

Im linken Bild, ist über dem blau markierten Feld „Beispiele“, der Begriff „Sketchbook“ zu lesen. Dahinter verbergen sich die vom

Benutzer bearbeiteten Programmdateien. Der Begriff Sketch beschreibt im Falle von Arduino, die Programme, die erstellt oder vom Benutzer bearbeitet wurden.

9.3. Programmbeispiel

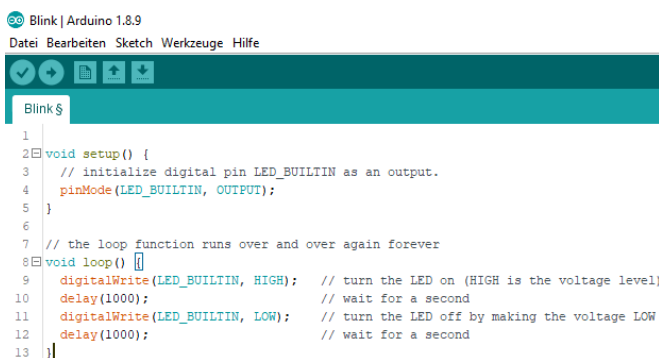
9.3.1. Programmbeispiel öffnen



Links im Bild ist gezeigt, wie ein erstes, einfaches Programm geöffnet wird:

Datei → Beispiele → 01.Basics → Blink

9.3.2. Programmaufbau



Nach dem Öffnen des Beispiel Blink, ist zu sehen, wie ein Arduino-Programmcode aufgebaut ist:

void setup() { }

void bedeutet, dass diese Funktion keine Variablen zurückgibt. Diese Funktion wird nur beim Start des Programms ein einziges Mal durchlaufen. Das bedeutet die Befehle zwischen den beiden { } Klammern werden einmal ausgeführt. In unserem Fall wird also

der Pin LED_BUILTIN als Ausgang definiert. Dies geschieht mit dem Befehl pinMode(Parameter 1, Parameter 2); Parameter 1 beschreibt den Pinnamen. Mit Parameter 2 wird die Funktion des Pins festgelegt:

OUTPUT: Pin ist ein Ausgang und kann Schaltfunktionen übernehmen;

INPUT: Pin ist ein Eingang; das bedeutet er besitzt einen großen Eingangswiderstand und belastet die Schaltung nicht, an die er angeschlossen ist.

INPUT_PULLUP: damit kann z.B. ein Taster eingelesen werden. Automatisch wird dann der Eingang mit einem Widerstand an 5V Spannungsversorgung gelegt.

LED_BUILTIN ist kein Begriff den der Arduino kennt. Dieser ist in einer gesonderten Headerdatei bereits vordefiniert. Diese „Definitionsdatei“ wird automatisch von der Arduino IDE bei der Einstellung des Boards und des Prozessortyps eingebunden. Der Pin 13 ist auf dem Board fest mit der LED verbunden. Dies gilt es immer zu beachten, wenn der PIN13 eine andere Funktion als die des Ausgangspins übernehmen soll.

Die Funktion loop() ist wie der Name schon sagt eine Schleife, deren Befehle zwischen den { } Klammern permanent durchlaufen werden.

Der Befehl digitalWrite(Parameter 1, Parameter 2) ändert den Schaltzustand des PIN. Nachdem dieser im Setup als Ausgang definiert wurde, muss ihm noch mitgeteilt werden, welchen Zustand er einnehmen soll. Im Parameter 1 wird der Funktion der Pin übergeben, der geschaltet werden soll. Parameter 2 legt fest welchen Zustand der Pin dabei haben soll: eingeschaltet, also HIGH (5V) oder abgeschaltet auf LOW (0V).

Der Befehl delay(1000) ist eine Wartezeit, bei der der Prozessor wartet, bis der nächste Befehl ausgeführt wird. Im obigen Beispiel bedeutet dies, dass die LED 1 Sekunde eingeschaltet wird und eine Sekunde aus bleibt. Der Wert dem der Delay-Funktion übergeben werden kann ist ein Wert mit 32 Bit. Er kann also von 1 ... 2^{32} ms, also 4294967296 ms in Worten: ca. 40 Tage warten.

9.3.3. Programm in den Arduino schreiben

Zuerst sollte das Programm kompiliert werden. Das bedeutet, der lesbare Programmtext wird übersetzt in eine für den Computer verständliche Befehlsabfolge. Da dabei Informationen verschiedene Programme miteinander verknüpft werden, kann es manchmal zu Fehlern kommen. Die häufigsten Fehler jedoch schleichen sich durch den Programmierer ein. Es ist auf Groß- und Kleinschreibung zu achten. Jeder Befehl wird durch ein Semikolon abgeschlossen. Jeder Klammer die geöffnet wurde, muss auch wieder geschlossen werden.

Durch drücken auf  wird das erstellte Programm übersetzt und auf syntaktische Fehler geprüft.



Durch drücken auf  diesen Pfeil, wird das Programm übersetzt und nach dem compilieren gleich auf den arduino geflasht, falls es fehlerfrei ist.

9.3.4. Werkzeug: serieller Monitor

Ein sehr gutes und wichtiges Werkzeug für den Programmierer ist der serielle Monitor. Der Vorteil ist, dass damit der Arduino dem Programmierer, Werte übergeben kann. Entweder Messwerte, oder Textnachrichten, mit denen dem Programmierer Statusmeldungen mitgeteilt werden können. Damit wird dann ersichtlich, welchen Programmteil der Prozessor gerade abarbeitet. Dies ist besonders bei größeren Programmen sehr von Vorteil. Der Nachteil ist, dass die Pins 0 und 1 des Prozessors damit für die Kommunikation reserviert sind und nicht mehr anderweitig zur Verfügung stehen.

Die serielle Schnittstelle ist die einfachste Art Daten zwischen zwei Geräten austauschen. Es werden nur drei Leitungen benötigt: Senden: TX, Empfangen: RX und die Bezugsmasse GND. Das Übertragungsprotokoll ist relativ einfach gehalten. Der Pegel liegt auf HIGH. Dann Schaltet zu Beginn der Datenübertragung der Pegel auf LOW (Startbit). Nun folgen 8 Datenbits und am Ende bleibt der Pegel eine definierte Zeit HIGH, abhängig von der Anzahl der Stopp-Bits. Nach

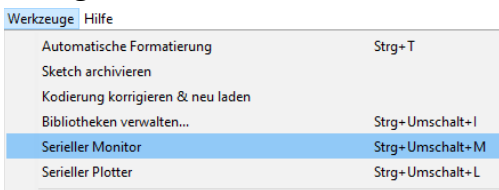
beliebiger Zeit kann das nächste Byte übertragen werden. Da die Abstände der Bytes variieren können, wird diese Form der Datenübertragung auch als asynchrone Datenübertragung bezeichnet. Die Zeiten für die jeweilige Länge der Bits wird von der Datenrate festgelegt. Standardmäßig wird häufig 9600 Baud verwendet. Dies ist auch die Standardeinstellung in der Arduino IDE. Wenn eine schnellere Datenübertragung gewünscht ist, wird oftmals dann 115200 Baud verwendet. Werte dazwischen sind eher selten. Manchmal wird bei der Datenübertragung noch das Parity-Bit verwendet. Dabei werden die Werte der einzelnen Bits addiert und dann ist die Summe entweder geradzahlig (even) oder ungeradzahlig (odd). Je nachdem wird dann das Bit gesetzt oder eben nicht. In diesen Beispielen wird auf das Parity-Bit verzichtet.

Um die Serielle Schnittstelle für den Arduino verwenden zu können, muss diese aktiviert und definiert werden.

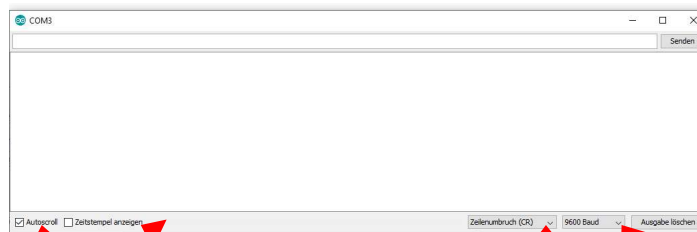
Dies geschieht mit dem Befehl: `Serial.begin(speed, config);`

speed ist in unserem Fall entweder 9600 oder 115200

config ist SERIAL_8N1. Da dies aber der Standardwert ist, wird dies immer weggelassen.



Links im Bild ist gezeigt, wie der serielle Monitor aufgerufen wird.



Genutzter **COM-Port**
Daten die zum Arduino gesendet werden sollen
 Button zum **Senden** der Daten
 Feld für die **empfangenen Daten**

Autoscroll
 damit immer die aktuellen Daten sichtbar sind, wenn der dargestellte Bereich überschritten wird.

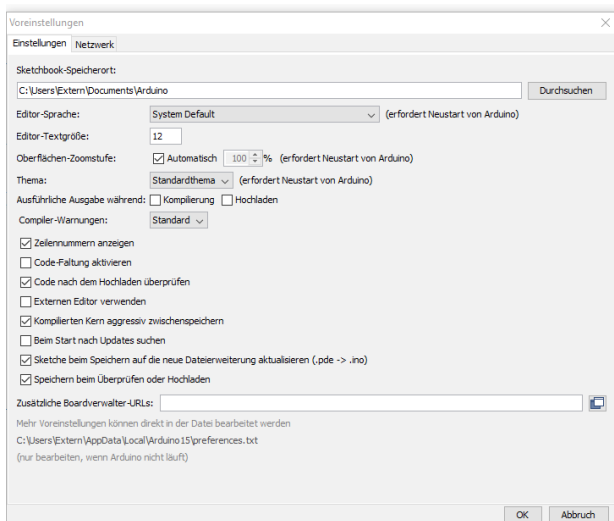
Zeitstempel
 um die Zeitdifferenz zwischen Botschaften messen zu können;

Button zum **Löschen** der Daten im Empfangsfeld

Baudrate des Arduino auswählen

Zeilenbruch: Bei Empfang von CR erfolgt automatisch ein Zeilenbruch im Empfangsfenster

9.4. Voreinstellungen



Unter **Datei** → **Voreinstellungen** erscheint das nebenstehende Fenster.

Dabei ist es u.U. hilfreich die **Zeilennummern** anzuzeigen, um sich besser bei längerem Code zurechtzufinden;

Ferner kann man bei den **Compiler-Warnungen** je nach dem wie erfahren man ist, unterschiedliche Stufen einstellen. Zu Anfang ist es besser auf **Standard** zu stellen, denn bei zu vielen Meldungen besteht nur die Gefahr dass ein unerfahrener Programmierer nur zu sehr verunsichert wird.

Den **Code nach dem Hochladen überprüfen** hilft, dass kein Fehler unerkannt bleibt, wenn das Programm in das Flash geschrieben wird.

9.5. Objektorientierte Programmierung

Es gibt noch viele Begriffe, die eigentlich erläutert werden sollten. Wir wollen uns jedoch hier auf praktische Beispiele und die Anwendung von Modulen und Sensoren beschränken. Es ist einfacher nur **Objekte** und deren **Instanzen** anzuwenden, als sie zu erstellen, zu programmieren, oder **Methoden** und **Klassen** in Bibliotheken für Programmierer nutzbar zu machen.

Dieses Skript soll nur eine Hilfe für den Einstieg in die Programmierung und dabei besonders in die Nutzung und Anwendung von Sensoren und Modulen sein. Es soll und kann kein Studium ersetzen. Bei der objektorientierten Programmierung geht es darum, vorgefertigte Bibliotheken einfach in das eigene Programm einzubinden. Dabei ist es dann relativ einfach auf die Funktionen und Variablen in der Bibliothek zuzugreifen. Man spricht dabei auch von Vererbung. Im Programm wird eine Variable als Objekt definiert. In der nächsten Zeile ist es dann schon möglich diese Funktion mit dem selber gewählten Namen zu nutzen und auf die Werte zuzugreifen und die Funktionen für eigene Zwecke zu nutzen. In Kapitel 9.7. wird dann dieses Thema an Hand der Nutzung von Libraries ein wenig näher erläutert.

```
ir_beispiel0 $
1 #include <IRremote.h> // Bibliothek irrecv
2 #define RECV_PIN 2
3
4 #define LED1 6
5 #define LED2 5
6 #define LED3 3
7
8 int val=85;
9
10 IRrecv irrecv(RECV_PIN);
11 decode_results results;
12
13 void setup()
14 {
15   Serial.begin(9600);
16   pinMode(LED1, OUTPUT);
17   pinMode(LED2, OUTPUT);
18   pinMode(LED3, OUTPUT);
19   irrecv.enableIRIn();
20   irrecv.blink13(true);
21 }
22 void loop()
23 {
24   if (irrecv.decode(&results))
25   {
26     Serial.println(results.value, HEX);
27     delay(100);
28     ///////////////////////////////////////////////////
29     if(results.value==0xFF30CF)
30     {
31       Serial.println("1");
32       digitalWrite(LED1, HIGH);
33     }
34   }
35   ///////////////////////////////////////////////////
36   else if(results.value==0xFF18E7)
```

Links im Bild ist ein Auszug eines Programmbeispiels: ir_beispiel0.ino.

Dieses Programm analysiert die Daten, die von einem Infrarotempfänger ausgegeben werden.

In Zeile 10 wird irrecv als Objekt von **IRrecv** abgeleitet. RECV_PIN ist dabei der Pin des Arduino, bei dem die Daten des Infrarotdecoders empfangen werden. Das ist Pin D2 des Arduino, wie aus Zeile 2 ersichtlich ist.

In Zeile 11 wird ein Objekt erzeugt, in dem die Ergebnisse des decodierten Signals abgespeichert werden.

In Zeile 19 wird der Empfänger gestartet. Mit Zeile 20 wird die LED auf dem Board initialisiert, dass diese den Empfang gültiger Daten signalisiert.

In Zeile 24 wird die Variable results als Adresse übergeben. Daher das **&** vor der Variable.

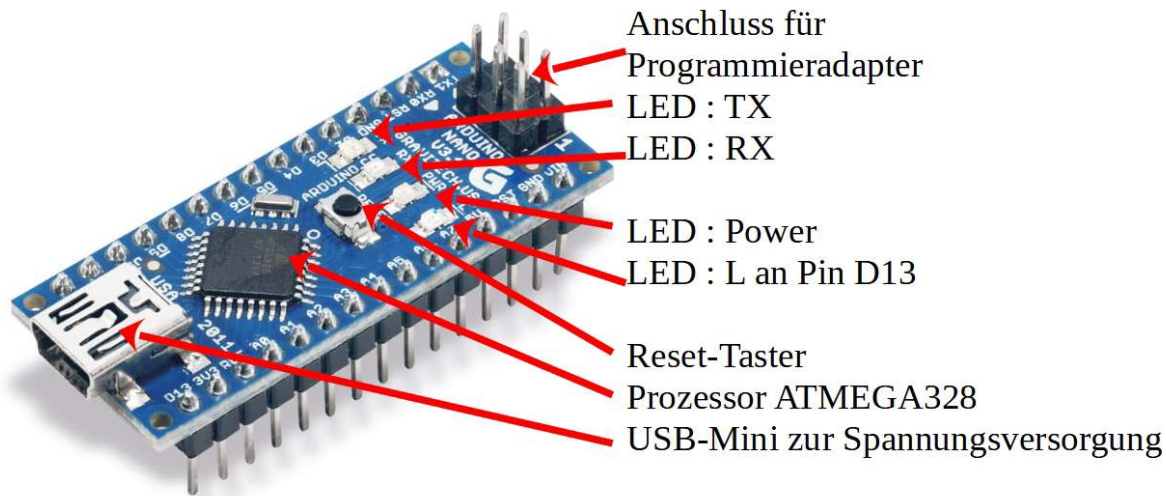
Um eine Funktion in einem Objekt aufzurufen, oder einen Wert einer Variablen abzufragen, wird immer folgende Syntax verwendet:

OBJECT (PUNKT) Funktion:
irrecv.blink13()

oder

OBJECT (PUNKT) Variable:
results.value = Wert

9.6. Der Arduino aus der Sicht der Arduino IDE



Artikelnummer 810330

9.6.1. Digitale I/O Pins:

Der Arduino Nano hat **digitale Pins**, die HIGH (5V) oder LOW (0V) geschaltet werden können: **digital Pin 0 ... digital Pin 13**. diese werden angesprochen:

z.B. `#define LED 13` // der digital Pin 13 kann nun mit LED angesprochen werden;

```
digitalWrite(LED, HIGH); // damit leuchtet die LED  
digitalWrite(LED, LOW); // so wird die LED ausgeschaltet
```

Die **digitalen Pins 0 und 1** haben noch eine weitere Funktionalität, nämlich D0 = RxD (empfangene Daten) und D1 ist TxD (Sende Daten auf diesem Pin). Diese sind die beiden Pins der sogenannten UART Schnittstelle. Universelle Asynchron Receive Transmit. Asynchron deshalb, weil zu beliebigen Zeiten gesendet und empfangen werden kann. Wobei RxD bedeutet receive data und TxD bedeutet transmit data. Das bedeutet, beide Pins können benutzt werden zur Kommunikation mit dem PC. Mit diesen Pins wird das Programm, vom PC in den internen Speicher übertragen. Denn im Hintergrund ist in den Arduino ein sogenannter BOOTLOADER installiert. Dies macht den Unterschied zu einem ATMEGA328 Mikrocontroller aus, den man üblicherweise kaufen kann. Auf dem Board ist nämlich noch ein USB-Baustein an diese beiden Pins angeschlossen, damit der PC über die USB-Schnittstelle mit dem Mikrocontroller kommunizieren kann. Denn ohne diesen Baustein könnte der Controller nicht an den PC angeschlossen werden.

Unter den digitalen Pins gibt es die sogenannten **PWM Ausgänge**. Diese werden mit dem Befehl `analogWrite(Pinnummer, val)` abgesprochen. Als **Pinnummer** sind folgende Ausgänge wählbar: **3, 5, 6, 9, 10 und 11**;

PWM bedeutet, daß dieser Pin ein Rechtecksignal konstanter Frequenz ausgeben und dessen Puls - Pause Verhältnis über den Wert **val** im Bereich von 0 ... 255 verändert werden kann.

0 bedeutet aus und 255 bedeutet dabei ganz eingeschaltet und dazwischen sind eben 254 Werte möglich. 127 bedeutet dabei dass das Puls Pausen-Verhältnis 50:50 ist. Damit lässt sich z.B. die Helligkeit einer LED stufenlos steuern.

9.6.2. Interrupt Pins: 2 und 3

Interrupt bedeutet Unterbrechung des laufenden Programms. Diese beiden Pins können so konfiguriert werden, dass bei einem Signalwechsel, ein bestimmtes Programm aufgerufen wird. Ohne dass dabei der Pin permanent abgefragt werden muss.

Mit dem Befehl `attachInterrupt()`; wird einer der beiden Pins als Interruptpin konfiguriert

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)
```

pin ist der Name des gewünschten Pin: 2 oder 3

ISR ist der Name der Interruptroutine; das bedeutet das Programm, das bei auslösen des Interruptes ausgeführt werden soll.

Mode : die Werte für mode können folgende vier „Werte“ bekommen

LOW, wenn der Pin auf LOW ist,

CHANGE, wenn der Pin den Wert ändert

RISING, wenn der Pin von LOW auf HIGH wechselt,

FALLING wenn der Pin von HIGH auf LOW wechselt.

9.6.3. Synchrone Serielle Datenübertragung:

Die Synchrone Datenübertragung erfolgt über die Pins: **A4 (SDA) und A5 (SCL)**: Von der Firma PHILIPS wurde Anfang der 1980er Jahre der I2C Bus entwickelt. I2C bedeutet Inter-Integrated-Circuit. Er dient dazu zwischen elektronischen Bausteinen eine Kommunikation aufzubauen. Dazu gibt es einen „Master“, meist der Mikrocontroller und einen oder mehrere „Slave“- Bausteine. Dabei gibt der Controller den Takt vor und die Slave Bausteine geben Daten im Rhythmus des Taktes aus. oder lesen sie ein. Dadurch, dass nicht nur Datenbytes, sondern auch Adressen übertragen werden, können auch IC's des selben Typs mehrmals in den Bus integriert und differenziert angesprochen werden. Theoretisch könnten bis zu 128 Teilnehmer mit diesen zwei Leitungen angesprochen werden.

Als Pendant zum I2C Bus wurde von Motorola 1987 der **SPI Bus** entwickelt. Dieser ist nicht ganz so perfekt. Zum einen braucht der Controller für den Datenaustausch mit dem Slave Baustein mehr Pins (Clock, MOSI = Master Out Slave In; MISO = Master In Slave Out; und SS = Slave Select) Das ist etwas umständlich, weil die Adressierung nur über das SS-Signal als für jeden Baustein eigenes erforderlich ist. Also pro Baustein ist ein weiterer Pin am Prozessor nötig.
D10 (SS) D11(MOSI) D12(MISO) D13 (CLOCK)

9.6.4. Die analogen Eingänge

Jetzt wären nur noch die **Analog Eingänge** zu erwähnen. Die Pins **A0 ... A7** sind dazu da, anliegende Spannungswerte in dezimale Zahlen umzuwandeln.

9.6.5. Die Ports des Arduino Nano:

PORTD D0 ... D7

PORTB D8 ... D13 (B0 ... B5)

PORTC A0... A5 (C0 ... C5)

nicht verwendbar: C6 (= RESET-Pin) und B6 (Xosc1) bzw. B7 (Xosc2) An diesen beiden Pins ist der Quarz-Oszillator angeschlossen;

Die Initialisierung ist der erste und wichtigste Schritt, ein Programm zu beginnen. Es wird dabei in der Funktion `setup()` festgelegt:

- welcher Pin wird benötigt ? (Ausgang oder Eingang, mit oder ohne Pull-up Widerstand)

pinMode(Pin, Funktion)

PIN: 0 ... 13 und analog PIN_A0 ... PINA7

Funktion: INPUT, INPUT_PULLUP, OUTPUT, wobei A0 ... A7 als Input definiert sind

variable=**digitalRead**(); // einlesen, ob am Pin der Status low [0] oder high [1] ist;

digitalWrite(variable); // einen Pin auf variable=1 / 0; oder variable=LOW / HIGH

Es ist vielleicht etwas verwirrend manche Pins mit 0 und andere wieder mit PIN_A0 anzusprechen.

So kann zur einheitlichen Anwendung der Pins auch folgende defines am Programmanfang stehen:

#define A0 PIN_A0 // nun kann der Analogport A0 mit A0 angesprochen werden

#define D1 1 // so kann ab hier der Pin 1 des Arduino mit D1 angesprochen werden

- welche Funktionen werden am häufigsten benötigt:

ADC: variable = **analogRead**(A0 ... A7); // variable ist als unsigned int definiert

PWM: **analogWrite**(Pin, Wert); // um Spannungen von 0 [0V] ... 255 [5V] zu erzeugen;

wobei als Pins dürfen nur: 3, 5, 6, 9, 10 und 11 verwendet werden!

Wert ist dabei eine Variable von 8Bit also 0 ... 255

tone(Pin, Frequenz, Dauer) ; // gibt ein Rechtecksignal der Frequenz und einer bestimmten Dauer aus;

Pin: jeder Pin kann als Ausgabe des Signals benutzt werden und muss vorher auch nicht als Ausgang konfiguriert sein. Dies übernimmt die Funktion tone()

Frequenz: welche Frequenz [Hz] **unsigned int** also theoretisch von 0 ... 65535.

Dauer: wie viele [ms] lang, soll der Ton spielen **long** 0 ... 2³² also umgerechnet 40 Tage;

noTone(Pin); // damit lässt sich der Ton wieder abstellen;

Denn wenn tone(Pin, Frequenz); aufgerufen wurde, also kein Wert für Dauer übergeben wird, dann wird das Rechtecksignal dauerhaft erzeugt.

9.7. Bibliotheksfunktionen

Die vielleicht am häufigsten genutzte Funktion ist Serial(), weil diese die Kommunikation mit dem PC übernimmt. So können über das Arduino IDE-Werkzeug „Serieller Monitor“ Variablenwerte oder Textbotschaften zum PC übertragen werden.

Serial.begin(variable1, variable2); // damit wird die serielle Schnittstelle aktiviert;
variable1: erlaubte Baudraten: 4800, **9600**, 19200, 38400, 57600, 76800, **115200**

Es gäbe die Möglichkeit eine zweite Variable mit anzugeben, die prüft, ob ein Übertragungsfehler aufgetreten ist. Diese wird hier aber nicht benutzt.

variable2: Konfiguration Anzahl der Daten-Bits, Anzahl Stopp-Bits, even oder odd
Standard SERIAL_8N1 → 8 Datenbits, **keine Paritätsprüfung 1** Stopp-Bit
mögliche Werte: SERIAL_5N1, SERIAL_6N1, SERIAL_7N1, SERIAL_8N1,
SERIAL_5N2, SERIAL_6N2, SERIAL_7N2, SERIAL_8N2,

even: SERIAL_5E1, SERIAL_6E1, SERIAL_7E1, SERIAL_8E1,
SERIAL_5E2, SERIAL_6E2, SERIAL_7E2, SERIAL_8E2,

odd: SERIAL_5O1, SERIAL_6O1, SERIAL_7O1, SERIAL_8O1,
SERIAL_5O2, SERIAL_6O2, SERIAL_7O2, SERIAL_8O2

wird an Stelle variable2 kein Parameter übergeben, so gilt SERIAL_8N1

Serial.begin(9600);

variable=**Serial.available()**; // wenn der Rückgabewert größer ist als 0, dann sind Daten im Empfangsregister
die Funktion Serial.available() wird dazu aber vielmehr in einer if-Abfrage eingebettet:

```
if( Serial.available() > 0 ) //wenn die Bedingung zwischen den runden Klammern erfüllt ist, dann wird die  
    { variable = Serial.read(); } // Funktion in den geschweiften Klammern ausgeführt, also die Daten in  
    der Variablen „variable“ abgespeichert.
```

Serial.write(variable); // diese Funktion gibt eine variable als binären Wert über die Schnittstelle aus.
variable = **Serial.write(„String“);** // diese Funktion gibt einen Text über die Schnittstelle aus und gibt die
Anzahl der übertragenen Zeichen in die integer variable zurück;

Serial.write(49); // gibt ein Zeichen aus

Serial.write('\n'); // erzeugt eine neue Zeile

Serial.write("Arduino\n\r"); // gibt Textstring aus mit Carriage Return und Line Feed

byte buffer[] = {'A', 'r', 'd', 'u', 'i', 'n', 'o'}; // damit kann ein array mit namen buffer definiert werden

Serial.write(buffer, 7); // write an array // dieses array buffer kann auch ausgegeben werden

Da Serial.write() nur Binärzeichen ausgibt, wurde die Funktion Serial.print() eingeführt. Damit werden lesbare ASCII Zeichen über die serielle Schnittstelle zum PC gesandt.

Serial.write(41); → das Zeichen: 1 wurde zum PC gesendet, weil gemäß der **ASCII-Tabelle** der Dezimalzahl 41 das Zeichen 1 entspricht.

ABER:

Serial.print(41); → der PC erhält die Zeichenfolge: 41

Was ist eine **ASCII-Tabelle**?

Dies soll im Abschnitt 9.8.5. näher erläutert werden.

Serial.print(variable,**parameter**);

Serial.println(variable,**parameter**); diese Funktion gibt zusätzlich zu der Variablen auch noch am Ende ein CR = /r und LF = /n also ein carriage return gefolgt von einem line feed aus also beginnt die nachfolgende Ausgabe in einer neuen Zeile; variable ist der Wert z.B. 65; Serial.print(65); gibt ein ‚A‘ aus, weil kein Parameter bedeutet, es wird der ASCII-Wert ausgegeben;

Serial.print(65, **OCT**); würde 101 ausgeben, also den Octal-Wert für 65;

Serial.print(65, **BIN**); 0100 0001 wäre der Wert in binärer Darstellung

Serial.print(65, **DEC**); so würde wirklich 65 ausgegeben, also der Dezimalwert

Serial.print(65, **HEX**); 41 wäre der Hexadezimale Zahlenwert

Serial.print(3.14, **0**); ergibt 3 also keine Ziffer hinter dem Dezimalpunkt

Serial.print(3.14, **1**); ergibt 3.1 also die Ziffer und eine Stelle hinter dem Dezimalpunkt

Serial.print(3.14, **2**); ergibt 3.14 also die Ziffer und zwei Stellen hinter dem Dezimalpunkt

Serial.print(65); **Serial.print**("\\t"); // damit werden zwischen den Ziffern Tabulatoren gesetzt

Serial.println("Zeichenfolge"); Hier wird die Zeichenfolge als ASCII Zeichen ausgegeben, gefolgt mit einer neuen Zeile; dies wäre identisch mit **Serial.print**("Zeichenfolge \\r \\n");

Serial.readBytes(buffer, length); liest Anzahl length in ein array of Byte oder Char

9.8. Schleifen und Kontrollstrukturen:

9.8.1. Die if-Bedingung

```
if( Bedingung )  
{Befehlsabfolge}  
else  
{Befehlsabfolge}
```

Die wohl am häufigsten verwendete Abfrage ist die **if-Bedingung**; ist die Bedingung in der Klammer erfüllt, dann wird die Befehlsabfolge in der nachfolgenden geschweiften Klammer abgearbeitet. Das anschließende **else** wird dann ausgeführt wenn die Bedingung nicht erfüllt ist. Dabei ist zu beachten, dass **else** nur optional verwendet werden kann.

9.8.2. Die while-Schleife

Bei der **do ... while** Schleife wird die Befehlsfolge einmal durchlaufen und erst dann die Abbruch-Bedingung abgefragt!

Falls sie erfüllt ist geht der Schleifendurchlauf wieder von vorne los. Die Abbruchbedingung wird also erst während des Programmablaufs erzeugt. Es ist also von vorne herein oftmals nicht abzusehen, wie oft die Schleife durchlaufen wird.

Manchmal kann, besonders beim Test eines Programms der Befehl **while(1)**; notwendig sein. Dieser Befehl bewirkt, dass das Programm an dieser Stelle stehen bleibt, weil die Bedingung in der runden Klammer immer erfüllt ist. So kann dann ein Programm schrittweise getestet werden, je nachdem, wo sich der „Haltepunkt“ im Programm befindet. Das **do** und die geschweiften Klammern entfallen dabei, weil keine Befehle mehr abgearbeitet werden.

9.8.3. Die for – Schleife

Bei der **for-Schleife** ist von vorne herein zwingend erforderlich zu wissen, wie oft die Schleife durchlaufen wird. Die for-Schleife ist so aufgebaut:

for (x=Startwert ; Abbruchbedingung ; Schrittweite) { Befehlsabfolge }
der Startwert ist meistens 0, kann aber größer sein, wenn heruntergezählt wird;
die Abbruchbedingung ist der Vergleich der Laufvariablen, mit einem festen Zahlenwert
die Schrittweite ist oftmals 1, muss aber nicht immer so sein;

1. Beispiel for(x=10; x < 50; x += 5) //Startwert 10, Änderung von x um 5 bei jedem Schleifendurchlauf bis der Wert von x die 50 überschreitet. Die Schleife wird bei diesem Beispiel nur 8 mal durchlaufen für x= 10, 15, 20, 25, 30, 35, 40, 45; wenn x=50 dann wird der Durchlauf abgebrochen, weil die Abbruchbedingung erfüllt ist.

2. Beispiel for(x=10; x > 0; x--) //Startwert 10, Änderung von x um -1 bei jedem Schleifendurchlauf bis der Wert von x die 0 erreicht. Bei 0 erfolgt der Abbruch und wird nicht mehr ausgeführt; Hier wird die Schleife 10x durchlaufen und zwar für x=10, 9, 8, 7, 6, 5, 4, 3, 2, 1; bei x=0 ist die Abbruchbedingung erfüllt und somit wird die Schleife verlassen.

3. Beispiel for(x=1; x <= 10; x++) //Startwert x=1, Änderung von x um +1 bei jedem Schleifendurchlauf bis der Wert von x die 10 erreicht. Bei 10 erfolgt der Abbruch und wird noch mit dem Wert x=10 ausgeführt;
Die Schleife wird auch hier wie in Beispiel2 genau 10 mal durchlaufen.

9.8.4. Die switch-Anweisung:

Um viele Fälle unterscheiden zu können, kann natürlich immer die if-Anweisung verwendet werden. Wenn die Fälle ähnlich aufgebaut sind, ist es übersichtlicher, die switch ... case – Anweisung zu verwenden. Dabei wird eine Variable mit mehreren Sprungzielen verglichen:

```
switch(Variable)    // die Variable ist vom Typ integer (Zahl) oder ein ASCII-Zeichen (char)
{
    case Sprungziel1: { Befehlsabfolge }
                    break;
    case Sprungziel2: { Befehlsabfolge }
                    break;
    case Sprungziel3: { Befehlsabfolge }
                    break;
    case Sprungzielx: { Befehlsabfolge }
                    break;
    default:{ Befehlsabfolge }
            break;
} // ende von Switch case
```

//die Variable kann z.B. eine Farbe sein, oder die Augen beim Würfel, oder Menüpunkte zur Auswahl

Ist die Variable mit einem Wert eines Sprungziels identisch, wird das Programm dort weiter ausgeführt und die Befehlsfolge abgearbeitet, bis zu break. Dann wird das Programm nach der geschweiften Klammer (

} // ende von Switch case) fortgesetzt. Hinter default **kann** eine Befehlsfolge eingegeben werden, wenn keine der vorher aufgeführten Bedingungen erfüllt wurde.

9.8.5. Die ASCII-Tabelle

ASCII bedeutet American Standard Code for Information Interchange, was so viel bedeutet wie amerikanischer Standard Code für den Informationsaustausch. Entstanden ist dieser Code in den 1960er Jahren und wurde eingeführt, um den Datentransfer zwischen Fernschreibern zu vereinheitlichen. Damit sollten Geräte verschiedener Hersteller miteinander kommunizieren können. So sollte vermieden werden, wenn Gerät A ein A sendet, dass Gerät B dann nicht ein C ausdrückt. Ursprünglich gab es nur 32 Steuerzeichen und 96 druckbare Zeichen. Dies reicht bis ins Jahr 1982. Dann wurde dieser Standard erweitert, denn Umlaute konnten mit der alten 7Bit Codierung nicht mehr dargestellt werden. Die Varianten wurden Codepage genannt. Für Deutschland gilt seit Einführung von MS-DOS, dem ersten Betriebssystem von Microsoft für PC's die Codepage 850 als Standard. Seitdem gibt es weitere Varianten, die wesentlich komplexer aufgebaut sind. Der neue Standard wurde 1992 eingeführt und heißt UTF-8.

Jedem Zeichen wurde zunächst ein Bitmuster aus 7 Bit zugeordnet. Damit können 128 Zeichen dargestellt werden. Das für ASCII nicht benutzte achte Bit konnte für die Fehlerkorrektur verwendet werden.

Mittlerweile wurde der Code jedoch auf 8 Bit erweitert. Die Zeichen 0...127 haben immer noch ihre Gültigkeit. Aber ab Zeichen 128 ist es auch abhängig, welcher Tastaturcode installiert ist, um Zeichen darstellen zu können. Denn Zeichen 128 sollte € sein, was aber nicht bei jedem Computer so angezeigt wird.

DEC	HEX	BIN	Symbol	DEC	HEX	BIN	Symbol
32	20	100000	Leerzeichen	93	5D	1011101]
33	21	100001	!	94	5E	1011110	^
34	22	100010	"	95	5F	1011111	_
35	23	100011	#	96	60	1100000	~
36	24	100100	\$	97	61	1100001	a
37	25	100101	%	98	62	1100010	b
38	26	100110	&	99	63	1100011	c
39	27	100111	'	100	64	1100100	d
40	28	101000	(101	65	1100101	e
41	29	101001)	102	66	1100110	f
42	2A	101010	*	103	67	1100111	g
43	2B	101011	+	104	68	1101000	h
44	2C	101100	,	105	69	1101001	i
45	2D	101101	-	106	6A	1101010	j
46	2E	101110	.	107	6B	1101011	k
47	2F	101111	/	108	6C	1101100	l
48	30	110000	0	109	6D	1101101	m
49	31	110001	1	110	6E	1101110	n
50	32	110010	2	111	6F	1101111	o
51	33	110011	3	112	70	1110000	p
52	34	110100	4	113	71	1110001	q
53	35	110101	5	114	72	1110010	r
54	36	110110	6	115	73	1110011	s
55	37	110111	7	116	74	1110100	t
56	38	111000	8	117	75	1110101	u
57	39	111001	9	118	76	1110110	v
58	3A	111010	:	119	77	1110111	w
59	3B	111011	;	120	78	1111000	x
60	3C	111100	<	121	79	1111001	y
61	3D	111101	=	122	7A	1111010	z
62	3E	111110	>	123	7B	1111011	{
63	3F	111111	?	124	7C	1111100	
64	40	1000000	@	125	7D	1111101	}
65	41	1000001	A	126	7E	1111110	~
66	42	1000010	B	127	7F	1111111	-

Standardschriftzeichen

DEC	HEX	BIN	Symbol	Beschreibung
0	0	0	NUL	null char
1	1	1	SOH	Start der Überschrift
2	2	10	STX	Textbeginn
3	3	11	ETX	Textende
4	4	100	EOT	Ende der Übertragung
5	5	101	ENQ	Enquiry zeigt an, dass eine Rückmeldung gewünscht ist
6	6	110	ACK	Quittierung
7	7	111	BEL	Bell
8	8	1000	BS	Back Space
9	9	1001	HT	horizontale Tab
10	0A	1010	LF	Zeilenvorschub
11	0B	1011	VT	vertikale Tab
12	0C	1100	FF	Form Feed
13	0D	1101	CR	Carriage Return
14	0E	1110	SO	Shift Out / X-On
15	0F	1111	SI	Shift In / X-Off
16	10	10000	DLE	Datenleitung entweichen
17	11	10001	DC1	Gerätekontrolle 1 (oft. XON)
18	12	10010	DC2	Gerätekontrolle 2
19	13	10011	DC3	XOFF
20	14	10100	DC4	Gerätekontrolle 4
21	15	10101	NAK	Negative Quittierung
22	16	10110	SYN	Synchron Idle
23	17	10111	ETB	Ende Senden Block
24	18	11000	CAN	stornieren
25	19	11001	EM	Ende des Medium
26	1A	11010	SUB	Substitute
27	1B	11011	ESC	Flucht
28	1C	11100	FS	Dateitrenn
29	1D	11101	GS	Gruppentrenn
30	1E	11110	RS	Rekord Separator
31	1F	11111	US	Einheit Separator
32	20	100000	SP	Leerzeichen

Steuerzeichen

DEC	HEX	BIN	Symbol	DEC	HEX	BIN	Symbol
128	80	10000000	€	193	C1	11000001	À
129	81	10000001		194	C2	11000010	Á
130	82	10000010	‚	195	C3	11000011	Â
131	83	10000011	ƒ	196	C4	11000100	Ã
132	84	10000100	„	197	C5	11000101	Ä
133	85	10000101	…	198	C6	11000110	Å
134	86	10000110	†	199	C7	11000111	Ç
135	87	10000111	‡	200	C8	11001000	È
136	88	10001000	•	201	C9	11001001	É
137	89	10001001	‰	202	CA	11001010	Ê
138	8A	10001010	Š	203	CB	11001011	Ë
139	8B	10001011	€	204	CC	11001100	Ì
140	8C	10001100	œ	205	CD	11001101	Í
141	8D	10001101		206	CE	11001110	Î
142	8E	10001110	Ž	207	CF	11001111	Ï
143	8F	10001111		208	DD	11010000	Ð
144	90	10010000		209	D1	11010001	Ñ
145	91	10010001	‚	210	D2	11010010	Ò
146	92	10010010	‚	211	D3	11010011	Ó
147	93	10010011	„	212	D4	11010100	Ô
148	94	10010100	„	213	D5	11010101	Õ
149	95	10010101	•	214	D6	11010110	Ö
150	96	10010110	•	215	D7	11010111	×
151	97	10010111	—	216	D8	11011000	Ø
152	98	10011000	—	217	D9	11011001	Ù
153	99	10011001	™	218	DA	11011010	Ú
154	9A	10011010	š	219	DB	11011011	Û
155	9B	10011011	š	220	DC	11011100	Ü
156	9C	10011100	œ	221	DD	11011101	Ý
157	9D	10011101		222	DE	11011110	Þ
158	9E	10011110	ž	223	DF	11011111	ß
159	9F	10011111	ž	224	E0	11100000	à
160	A0	10100000		225	E1	11100001	á
161	A1	10100001	ı	226	E2	11100010	â
162	A2	10100010	€	227	E3	11100011	ã
163	A3	10100011	€	228	E4	11100100	ä
164	A4	10100100	€	229	E5	11100101	å
165	A5	10100101	¥	230	E6	11100110	æ
166	A6	10100110	ı	231	E7	11100111	ç
167	A7	10100111	š	232	E8	11101000	è
168	A8	10101000	—	233	E9	11101001	é
169	A9	10101001	©	234	EA	11101010	ê
170	AA	10101010	•	235	EB	11101011	ë
171	AB	10101011	•	236	EC	11101100	ì
172	AC	10101100	—	237	ED	11101101	í
173	AD	10101101	—	238	EE	11101110	î
174	AE	10101110	•	239	EF	11101111	ï
175	AF	10101111	•	240	FO	11110000	ð
176	B0	10110000	•	241	F1	11110001	ñ
177	B1	10110001	±	242	F2	11110010	ò
178	B2	10110010	±	243	F3	11110011	ó
179	B3	10110011	±	244	F4	11110100	ô
180	B4	10110100	—	245	F5	11110101	õ
181	B5	10110101	µ	246	F6	11110110	ö
182	B6	10110110	¶	247	F7	11110111	÷
183	B7	10110111	•	248	F8	11111000	ø
184	B8	10111000	•	249	F9	11111001	ù
185	B9	10111001	•	250	FA	11111010	ú
186	BA	10111010	•	251	FB	11111011	û
187	BB	10111011	•	252	FC	11111100	ü
188	BC	10111100	•	253	FD	11111101	ý
189	BD	10111101	•	254	FE	11111110	þ
190	BE	10111110	•	255	FF	11111111	ÿ
191	BF	10111111	•				
192	CO	11000000	À				

erweiterter Zeichensatz

10. Grundlagen des binären Zahlensystems

Um das binäre Zahlensystem besser verstehen zu können, betrachtet man zuerst das vertraute Dezimalsystem aus einem anderen Blickwinkel. Die Zahl **492** setzt sich folgendermaßen zusammen: $4 * 10^2 + 9 * 10^1 + 2 * 10^0$ bedeutet $4 * (10^2 = 100) + 9 * (10^1 = 10) + 2 * (10^0 = 1)$.

Diese Erkenntnis auf das hexadezimal und binäre Zahlensystem angewandt bedeutet:

$0b\ 11001101 = 1 * 2^7 + 1 * 2^6 + 0 * 2^5 + 0 * 2^4 + 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$;
als Ergebnis dieser Berechnung ergibt sich als Dezimalzahl folgender Wert: 205

Mit 8 Bit lassen sich nur ganzzahlige Werte von 0 .. 254 oder -128 ... 127 darstellen.

Bei größeren Zahlen (Integer) fasst man zwei Byte zusammen:

Nun lassen sich entsprechend Zahlen von -32768 ... 32767 als vorzeichenbehaftete Zahlen darstellen, oder als vorzeichenlose Zahl 0 ... 65535.

Zusatzbemerkung: im hexadezimalen Zahlensystem wird gezählt 0 ... 9, A, B, C, D, E, F (0...15).
Im hexadezimalen Zahlensystem bedeutet dann:

$0xAFD0 = A \sim 10 * 16^3 + F \sim 15 * 16^2 + D \sim 13 * 16^1 + 0 * 16^0$ dies ergibt 45008 als Dezimalwert.
Um der Arduino IDE mitzuteilen, welches Zahlensystem wir gerade betrachten, wird die quasi mit einer „Vorsilbe“ signalisiert:

0x oder **0X** für hexadezimale Zahlen (z.B. **0xAF** oder **0XAF**);

0b oder **B** für binäre Zahlen (z.B: **0b00110101** oder **B00110101**)

10.1. Bitoperationen:

Manchmal ist es erforderlich zu wissen ob ein bestimmtes Bit gesetzt oder gelöscht wurde. Um dies schnell zu erkennen verwendet man Operatoren zur Bitmanipulation.

10.1.1. Bitweise UND-Verknüpfung mit &

Die UND-Verknüpfung ist wie eine Multiplikation in der „dezimalen“ - Welt

```
0 & 0 == 0
0 & 1 == 0
1 & 0 == 0
1 & 1 == 1
```

so ergibt sich als bitweise UND-Verknüpfung für ein Datenbyte (8Bit):

```
0b 1 0 0 1 0 0 1 0
& 0b 0 1 0 1 0 1 0 1
```

Ergebnis 0b 0 0 0 1 0 0 0 0

Anwendung findet diese mathematische Operation oftmals im Programm, wenn zu prüfen ist, ob ein bestimmtes Bit gesetzt ist. Das zu prüfende Bit, wird in einer Maske gesetzt und mit dem zu prüfenden Bit mit einer &-Verknüpfung verglichen. Wenn das Ergebnis 0 ist, so ist das Bit nicht gesetzt. Ist das Ergebnis != 0, so war das Bit gesetzt.

10.1.2. Die Bitweise ODER-Funktion mit |

Eine ODER-Funktion ist vergleichbar mit einer Addition in der „dezimalen“ Welt.

| wird erzeugt mit <alt> + 124) hier sind die Ziffern im aktivierten Nummernblock zu verwenden.

```
0 | 0 == 0
0 | 1 == 1
1 | 0 == 1
1 | 1 == 1
```

Verwendet werden diese Operationen, um z.B. ein Bit zu setzen:

```
PORTA = 0b 0 0 0 0 1 0 0 1
Maske  = 0b 0 1 0 0 0 0 0 0
```

Ergebnis = PORTA | Maske Ergebnis=0b 0 1 0 0 1 0 0 1

Wären nun an PortA LED's angeschlossen, so könnte man mit der obigen Operation eine LED an PORTA:Bit6 einschalten.

10.1.3. Die Bitweise NOT-Funktion mit ~

Das Ergebnis dieser Operation wird auch als 2er Komplement bezeichnet. Dabei wird bitweise jedes Bit in den entgegengesetzten Zustand gesetzt. 0 wird zu 1 und eine 1 wird zu 0.

a = 0b 1 0 0 1 0 1 1 0

~a=0b 0 1 1 0 1 0 0 1

So lässt sich auf eine einfach Weise die LED, die beim vorherigen Beispiel auf 1 gesetzt wurde, wieder abschalten:

```
Maske = 0b 0 1 0 0 0 0 0 0
```

```
PORTA = 0b 0 0 0 0 1 0 0 1
```

```
~Maske = 0b 1 0 1 1 1 1 1 1
```

Ergebnis = PORTA & ~Maske Ergebnis=0b 0 0 0 0 1 0 0 1

10.1.4. Die Bitweise XOR-Funktion mit ^

Die nun verwendete Funktion dürfte vielen neu sein. Es handelt sich um eine exklusiv Oder Verknüpfung. Diese ist ähnlich der Oder Verknüpfung, allerdings mit einer exklusiven Ausnahme.

Um eine Bitweise Funktion besser verstehen zu können ist die Wahrheitstabelle ein gutes Hilfsmittel. Eine Wahrheitstabelle zeigt das Ergebnis der möglichen Verknüpfungen die entstehen können:

```
0 ^ 0 == 0
```

Der Unterschied zur Oder-Verknüpfung ist die untere Zeile in der Tabelle. Zwei unterschiedliche Werte ergeben eine 1 und zwei gleiche Werte ergeben als Ergebnis eine 0!!


```
0 ^ 1 == 1
1 ^ 0 == 1
1 ^ 1 == 0
```

wie im Beispiel der Not-Verknüpfung kann mit einer XOR-Verknüpfung der Status sehr einfach von 0 auf 1 und umgekehrt gesetzt werden:

```
y = x ^ 1;
```

oder bei einem Byte:

```
x = 8;          // Binärzahl: 1000
y = 11;         // Binärzahl: 1011
z = x ^ y;      // Binärzahl: 0111,
```

10.1.5. Schiebe (Shift oder Rotate) Operationen

Eine Schiebe-Operation ist das anhängen einer 0 bei einer Dezimalzahl oder das Streichen der selben. Bei shift left wird eine Stelle erzeugt. Die Binärzahl verdoppelt somit ihren Wert. Eine shift right Operation verhält sich entgegengesetzt. Vorteil ist, dass damit der Befehl mit einem Rechenzyklus ausgeführt wird. Eine Multiplikation oder eine Division dauert bei einem Mikrocontroller sehr viel länger, weil nur wenige Mikrocontroller eine Multiplikation oder eine Division implementiert haben. Dies wird meist nur über Software realisiert.

```
1 << 0 == 1
1 << 1 == 2
1 << 2 == 4
1 << 3 == 8

1 << 8 == 256
1 << 9 == 512
1 << 10 == 1024
```

```
a = 0b01100000;
```

```
a = a » 1;
```

nun wird a zu 0b00110000 // wird anschließend a nochmal geschoben

```
a = a » 5
```

nun wird a zu 0b00000001 // dann gehen Bits verloren

10.2. Zuweisungsoperatoren

`x=3` einfache Werte z.B. 3 können so einer Variablen x zugeordnet werden

`x+=3` der Wert 3 wird zu einer Variablen x addiert.

`x-=3` der Wert 3 wird von einer Variablen x subtrahiert.

`x*=3` die Variable x wird mit 3 multipliziert.

`x/=3` die Variable x wird mit 3 dividiert.

`x &= y` ein Wert y wird mit der Variablen x ver-undet

$x \mid= y$ ein Wert y wird mit der Variablen x ver-odert

$x \wedge= y$ ein Wert y wird mit der Variablen x exklusiv ver-odert

ABER beachte die Zeichenfolge:

$x = \sim x$!!! die Zahl $x = 0111$ wird Bitweise negiert zu 1000

10.3. Die Operatoren bei booleschen Abfragen (if-Abfragen)

Logisches NICHT $!$ aus einer Bedingung die wahr ist, wird eine die false ist

logisches UND: $\&\&$ beide Bedingungen müssen erfüllt sein

logisches ODER: \parallel nur eine der Bedingungen braucht erfüllt sein

logische Negierung $!=$ bedeutet ungleich

Bedingung auf Gleichheit $==$ wenn eine Variable gleich der anderen ist

Vergleich $>=$ bedeutet größer oder gleich

Vergleich $<=$ bedeutet kleiner oder gleich

Vergleich $>$ bedeutet größer als

Vergleich $<$ bedeutet kleiner als

Alles zusammenfassendes Beispiel

```
y = (x >> n) & 1;     // n = { 0..15 }    y wird 0 or 1.  
x &= ~(1 << n);     // das n. bit of x wird 0. Alle anderen bleiben unverändert.  
x |= (1 << n);       // setzt das n. bit von x auf 1. Alle anderen bleiben unverändert.  
x ^= (1 << n);       // toggelt das n. bit von x. Alle anderen bleiben unverändert.  
x = ~x;             // jedes einzelne Bit in x wird negiert
```

Aufgabe für Profis:

was bewirkt diese Funktion:

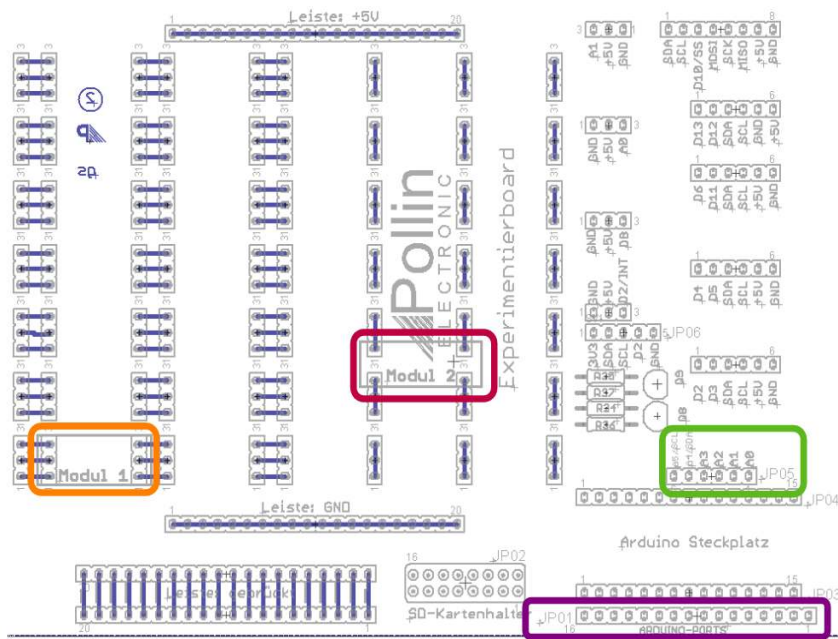
```
(x > 0) && (x & (x-1) == 0);
```

bei geraden Zahlen bleibt kein Rest; bei ungeraden positiven Zahlen bleibt die 1 als Rest übrig!

Was bewirkt:

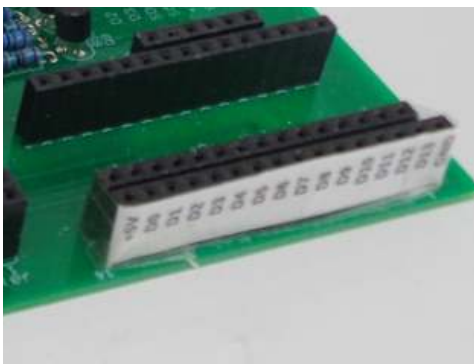
```
x &= (1 << (n+1)) - 1  
probier es aus!
```

11. Die Programmierung des Arduino in der Praxis



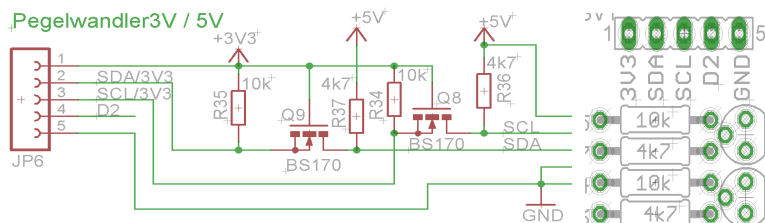
Im Bild links ist der Bestückungsplan der Hauptplatine vom Arduino-Kit dargestellt. Besonders zu beachten ist, wie die 3-poligen Buchsenstecker miteinander verbunden sind. Ist das Modul wie **Modul1** gesteckt, kann an den Buchsenleisten links und rechts kontaktiert werden. **Modul 2** kann nur an den Ecken kontaktiert werden. Dafür hat diese Variante der Kontaktierung, die Möglichkeit, zwei Leitungen an einen

Kontakt anzuschließen. Jeweils oberhalb, bzw. unterhalb des Moduls.



Unterhalb des Steckplatzes für den Arduino (JP03, JP04) ist eine Buchsenleiste **JP01** platziert. Dort wurden alle Portpins D0 ... D13, dazu GND und 5V herausgeführt. Links im Foto ist zu erkennen, wie diese beschriftet sind. Dies soll eine Hilfe sein, beim Verdrahten die Pin's besser zu finden. Über dem Arduino-Steckplatz ist eine weitere Buchsenleiste **JP05**. An dieser können die analogen Pins A0 ... A5 abgegriffen werden. Die Beschriftung dieser Pins befindet sich auf der Leiterplatte. Über dem Arduino-Steckplatz sind viele andere Buchsenleisten auf der Platine platziert. Damit wurde versucht,

Steckplätze für bestimmte Sensoren und Module zur Verfügung zu stellen. So sollte der Verdrahtungsaufwand verringert werden. Aber vorsicht beim Anstecken der Module, ob wirklich die Pinbelegung **und** die Spannung zusammen passt.



Es sind zwei MOSFETs auf der Platine Q8 und Q9. Diese sollten dazu dienen, eine Pegelanpassung der SPI-Signale SDA und SCL vom Prozessor zu einem Sensor oder einem anderen Modul zu gewährleisten.

Die Signale und die Spannungsversorgung 3,3V und GND können an der darüberliegenden Buchsenleiste JP06 abgenommen oder angesteckt werden. Leider funktioniert dies nicht bei jedem Modul. Denn manche Module haben bereits einen Widerstand an den Signalleitungen. So dass dadurch die Spannungsverhältnisse beeinflusst werden und das Modul dadurch nicht angesprochen werden kann.

11.1. Einführendes Beispiel: [blink.ino](#)

11.1.1 Beispiel: Blinkende LED

Zuerst den Pin am Arduino auswählen, den man verwenden will, um die LED anzusteuern.

Zur Schaltung:

Die Farbe der LED bestimmt den Wert von deren Vorwiderstand:

Beispiel rote LED:

Spannungsabfall an der LED: 2V bei 10mA Stromfluss (Werte aus Datenblatt)

Versorgungsspannung: 5V

Spannungsabfall am Widerstand $5V - 2V = 3V$

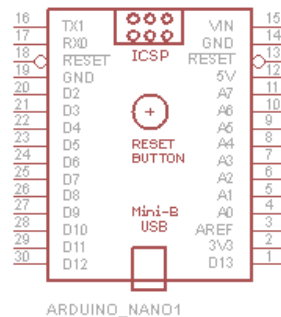
Widerstand = Spannung / Strom;

$$R = 3V / 0.01A$$

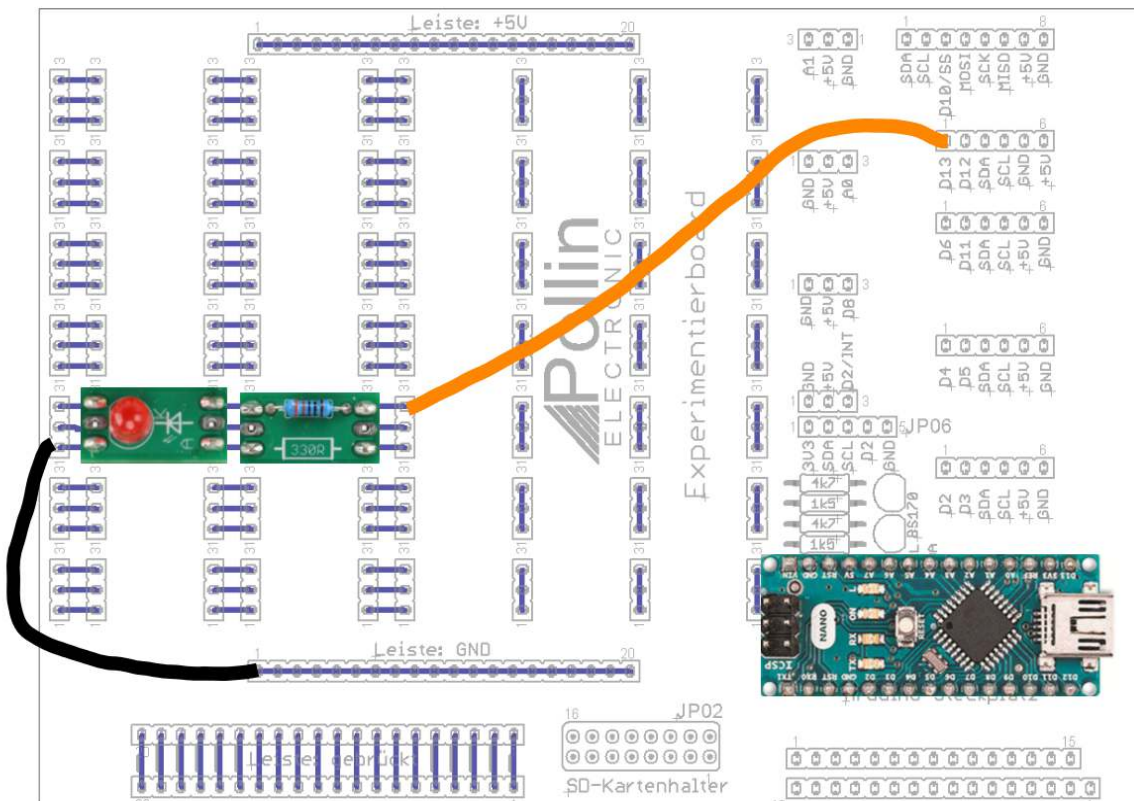
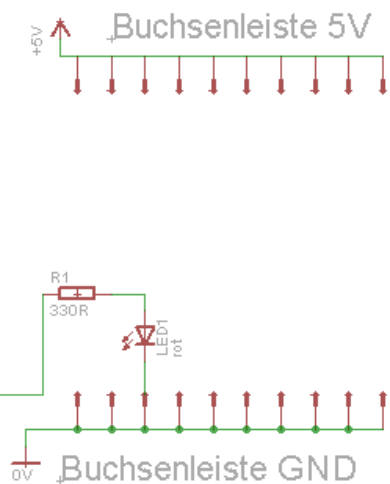
$$R = 300R;$$

$R=330R$ oder $270R$ wären mögliche Standardwerte.

Bei $270R$ wäre der Strom $11mA$, was auch möglich ist.



ARDUINO_NANO1



Flussdiagramm für Befehlsabfolge



```

Blink §
#define warten 1000 // wird in millisekunden angegeben
#define LED 13 // beim uno sind die Pins von 0 .. 13 durchnummeriert
// analogpins A0 ... A7 geneueres fidest du in
// c:\Program Files (x86)\Arduino\hardware\arduino\avr\variants\standard\pins_arduino.h

void setup() {
  // hier werden initialisierungen der Hardware und der Variablen vorgenommen
  pinMode(LED, OUTPUT); // hier wird der LED Port als Ausgang definiert
  digitalWrite(LED, LOW); // LED ausschalten
}

// im Gegensatz zu setup() wird loop, wieder der Name schon sagt, permanent durchlaufen
void loop() {
  digitalWrite(LED, HIGH); // HIGH ist in der Digitaltechnik als 5V definiert
  delay(warten); // warte 1 s
  digitalWrite(LED, LOW); // schalte LED aus
  delay(warten); // warte eine Sekunde
}

```

Programmzeile: **#define** warten 1000 // Wartezeit in [ms] (ms = tausendstel Sekunden)

Mit **#define** <name> wird eine Konstante <name> definiert. Dadurch, dass alle **#defines** am Programmanfang stehen, lassen sie sich leicht finden, um sie leichter ändern zu können. Der weitere Vorteil derartiger defines ist, es ist bequemer nur einen Wert am Programmanfang zu ändern. Würde man ohne diese Konstante arbeiten und man wollte den Wert ändern müsste man das ganze Programm durchsuchen, wo dieser Wert sonst noch verwendet wird. Ferner wird der Wert in den Code integriert und braucht damit keinen RAM-Speicher, wie eine Variable.

// die zwei Schrägstriche signalisieren dem Programm, hier befindet sich ein Kommentar.

#define LED 13 die 13 ist dabei der Pin, der den Wert LED bekommt. Erlaubte Werte sind dabei 0 ... 13, als Wert einen der Pins D0 ... D13 zu wählen. Auf dem Arduino ist eine LED bereits fest mit dem Port 13 verbunden.

pinMode(LED, OUTPUT); // Der Befehl **pinMode** legt fest, dass der Arduino, diesen als LED definierten Port als Ausgang schalten muss. Standardmäßig ist jeder Pin ein Eingang. Deshalb muss der Arduino wissen, wenn ein Pin ein Signal ausgeben muss. Dann gibt es eben die Option INPUT und INPUT_PULLUP. Mit der Option INPUT_PULLUP wird der Port zusätzlich mit einem ca. 20kOhm großen Widerstand auf 5V verbunden.

delay(warten); Mit der Funktion **delay()** wird eine Wartezeit für den Prozessor eingeführt. In dieser Zeit reagiert der Arduino auf keinerlei Eingaben. Ohne diese Wartezeit, würde der Arduino die Befehle so schnell abarbeiten, dass der Beobachter gar nicht mitbekommen würde, dass die LED angeschaltet wurde. Es gibt aber andere Beispiele, bei denen eine künstliche Verzögerung notwendig ist. Beim Einlesen von Tasten, um diese zu entprellen.

delayMicroseconds (warten); hat wesentlich kürzere Schaltzeiten. Wie der Name schon sagt sind es nicht tausendstel Sekunden wie bei delay() sondern millionstel Sekunden.

Variable=10; // 100 oder 1000

verändere den Wert der Variable warten :10000 oder 1000000 oder 10000000 und beobachte was passiert.

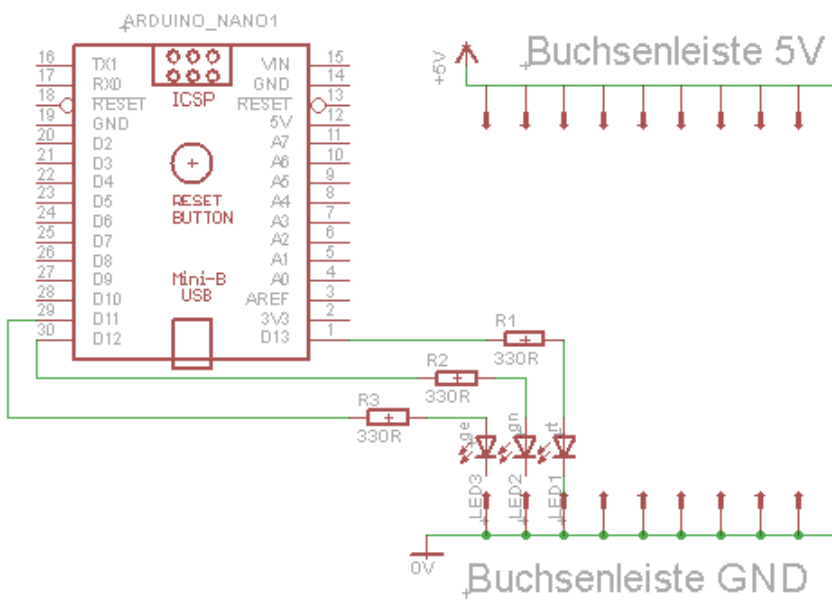
Die Funktion **Setup()**

Diese Funktion wird nach dem Programmstart nur einmal durchlaufen. Deshalb stehen in ihr die Befehle zur Initialisierung des Arduino. So z.B. Variable, die am Anfang einmal einen Startwert bekommen. Oder eben pinMode();

Die Funktion **loop()** ist das eigentliche Programm des Arduino. Alle Befehle, die innerhalb der geschweiften Klammer enthalten sind, werden nacheinander abgearbeitet und das wie die Bezeichnung loop sagt in einer immerwährenden Schleife, so lange bis der Arduino von der Spannung genommen wird.

Lasse nun die LED's gleichzeitig blinken. Dabei brauchts Du nur im Beispiel blink1.ino zwei weitere Ausgänge für die LED's definieren. Die Funktion pinMode kopieren und den Pin abändern. Genauso die Befehle unter HIGH und LOW.

11.1.2. Beispiel 2: alternierende LEDs: [blink2.ino](#)

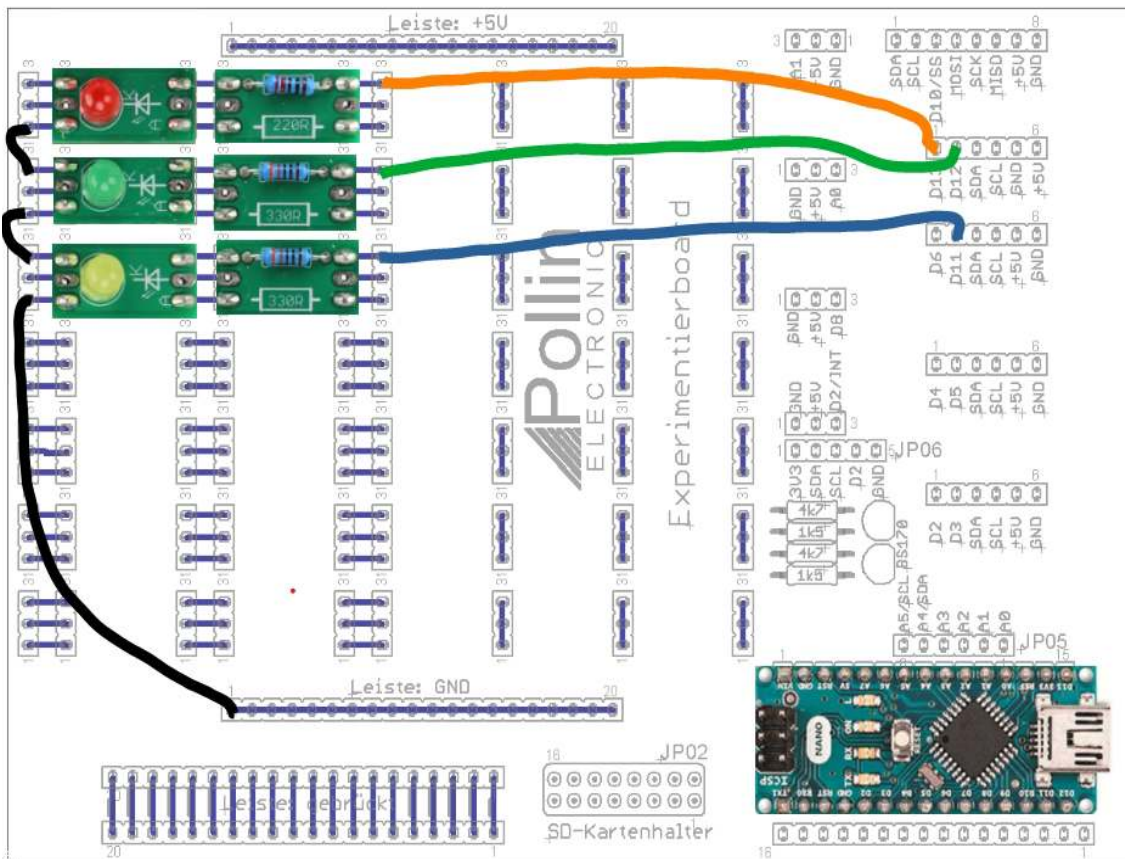


Wo ist der Fehler in der nebenstehenden Abbildung?

Es ist die Kathode der LED3 und LED2. Diese Anschlüsse sind nicht verbunden. Dadurch ist kein Stromfluss möglich. Also leuchten diese beiden LED's nicht!

Es ist zwingend erforderlich, dass diese freien Anschlüsse Kontakt mit Masse haben. Baue zuerst die Schaltung auf, wie im Foto; also mit drei roten LED's. Ändere dann die Bestückung und ergänze die roten LED's durch eine gelbe und eine grüne. Wie ändert sich die Helligkeit?

Nun ändere noch den Vorwiderstand von der grünen und der gelben LED auf 220R.



Dadurch, dass die grüne LED einen größeren Innenwiderstand besitzt und somit einen größeren Spannungsabfall verursacht, muss der Vorwiderstand kleiner werden, damit genauso viel Strom fließt wie bei einer gelben oder roten LED. Denn an einer roten LED fallen nur ca. 2V Spannung ab. An einer grünen LED fallen ca. 3V an Spannung ab! Deshalb muss der Widerstandswert, wie in Beispiel 1 beschrieben (berechnet), abgeändert werden (220R).

Nun ein paar Anmerkungen zum Programm [blink2.ino](#). Bei den #defines wurden für jede LED eine eigene Wartezeit definiert.

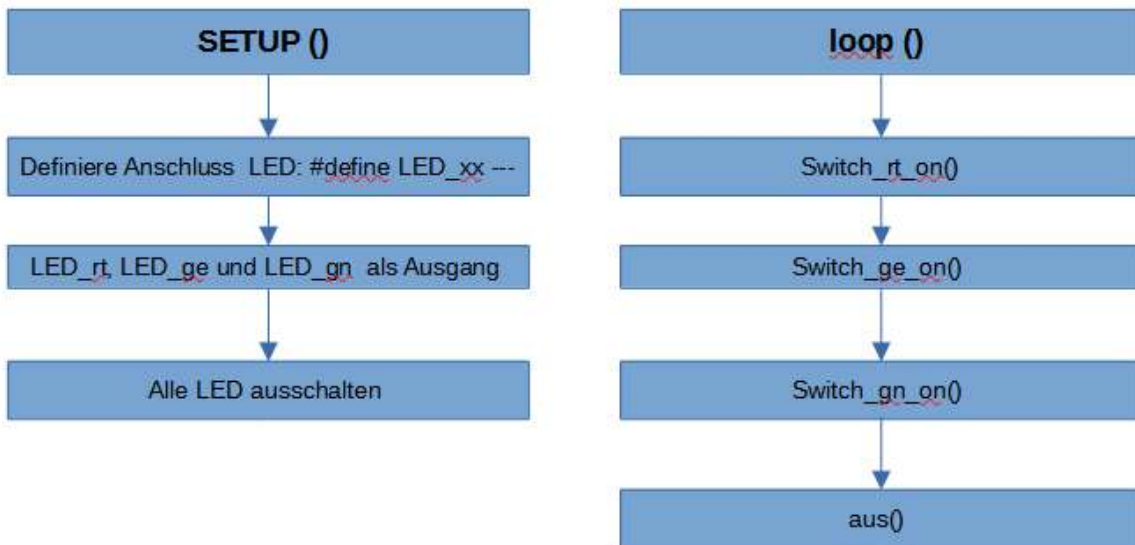
Neu ist in diesem Beispiel, die Verwendung von Unterprogrammen. Diese müssen zuerst deklariert werden: **void** switch_gn_on (**void**);

Dies ist ein Hinweis an die Arduino IDE, dass es ein Unterprogramm gibt, welches weiter unten dann geschrieben steht. Natürlich könnte das Programm auch gleich an dieser Stelle stehen.

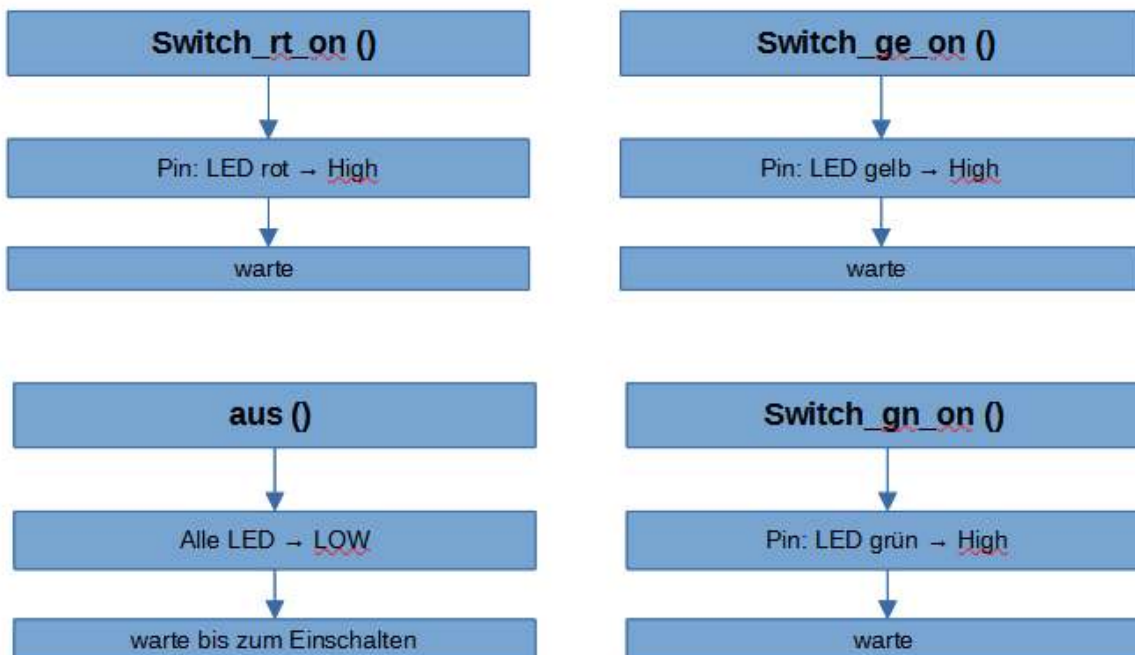
Das ist Geschmackssache des Programmierers. Die einen schreiben den Code gleich an die Stelle, an der der Programmname zum, ersten Mal erwähnt wird. Die anderen schreiben alle Unterprogramme nach der loop();

Das bleibt jedem Programmierer selbst überlassen, was er als übersichtlicher empfindet. Beides ist korrekt.

Hier nun das Flussdiagramm des Programmbeispiels blink2.ino :



Unterfunktionen:



Lasse die LED's nacheinander blinken und erweitere die Schaltung um weitere LED's deiner Wahl zu einem Lauflicht ähnlich dem von Knight Rider; Du kannst nun auch die Werte der Wartezeiten beliebig ändern.

Es ist ein großer Vorteil, die Zeiten über #defines festzulegen. Denn bei einem größeren Programm, bei dem die Zeiten mehrmals verwendet werden, braucht man diese nur einmal ändern und sie haben dann überall im Programm diesen Wert. Durch die Definitionen am Programmanfang, sind sie überdies leichter zu finden sind.

Was ist störend an diesem Programm?

Mich stört, dass die LED's nicht unabhängig voneinander blinken. Schöner wäre es, wenn man unabhängig für jede einzelne LED die Zeit definieren könnte, bei der sie an oder aus ist. Also braucht man eine Funktion, die die Zeit abfragen kann um festzustellen, wie lange die LED schon leuchtet. Dies wird in dem, nun folgenden Beispiel gelöst: [blink3.ino](#). Bei diesem Beispiel ist zunächst die Funktion `timer_alt_rt = millis()`; zu erwähnen. Wenn der Arduino an Spannung gelegt wird, fängt sofort ein Zähler im Prozessor an zu zählen. Er beginnt bei 0 und zählt so weit, bis er nach ca. 50 Tagen, wieder bei 0 beginnt. Mit der Funktion `millis()` kann man den aktuellen Zählerstand abfragen und in eine Variablen vom Typ `unsigned long` abspeichern. Denn nur mit diesem Variablen Typ lassen sich Werte von 0 ... 4.294.967.295 ($\cong 2^{32} - 1$) abspeichern. In Setup() wird der Anfangswert von millis() in `timer_alt` gespeichert. So dass dieser dann in loop() immer mit dem aktuellen Wert von millis() z.B. in `timer_rt = millis()` verglichen werden kann. Wenn der aktuelle Zählerstand größer ist, als die Summe von `warte_rt` und aktueller Zähler `timer_rt`. Wird der Zustand der LED geändert, entweder AUS → EIN oder EIN → AUS. Der Arduino kann einen Ausgangspin nicht zurücklesen. Mit einer Variablen z.B. `onoff_rt` merkt sich der Arduino den Zustand der LED. Deshalb muss der Programmierer auch darauf achten, dass nicht nur der Zustand der LED (Pin) sondern auch die Variable mit geändert wird. Wenn der Wert der Variablen ungleich Null ist, schaltet er den Pin auf aus und setzt die Variable auf 0 und wenn der Wert 0 ist, schaltet er den Pin auf HIGH und den Variablenwert belegt er mit 1. Bei Änderung des Zustands der LED muss der Wert von `timer_alt` neu einlesen werden, neuer Startwert für die Zeitmessung. Denn nach einem Schaltwechsel beginnt der Zählvorgang bis zum nächsten Schaltvorgang wieder von neuem! Im Beispiel [blink3.ino](#) ist dies mit einer LED dargestellt. Genauso kann man es mit jeder weiteren LED machen und jede mit einer eigenen Blinkfrequenz ausstatten.

Flussdiagramm blink3.ino :

Neu ist hier die Abfrage ob der Timer abgelaufen ist. Im Programm wird dies realisiert mit

if (Bedingung) // falls Bedingung erfüllt

{ Programmcode; }

else // Bedingung nicht erfüllt:

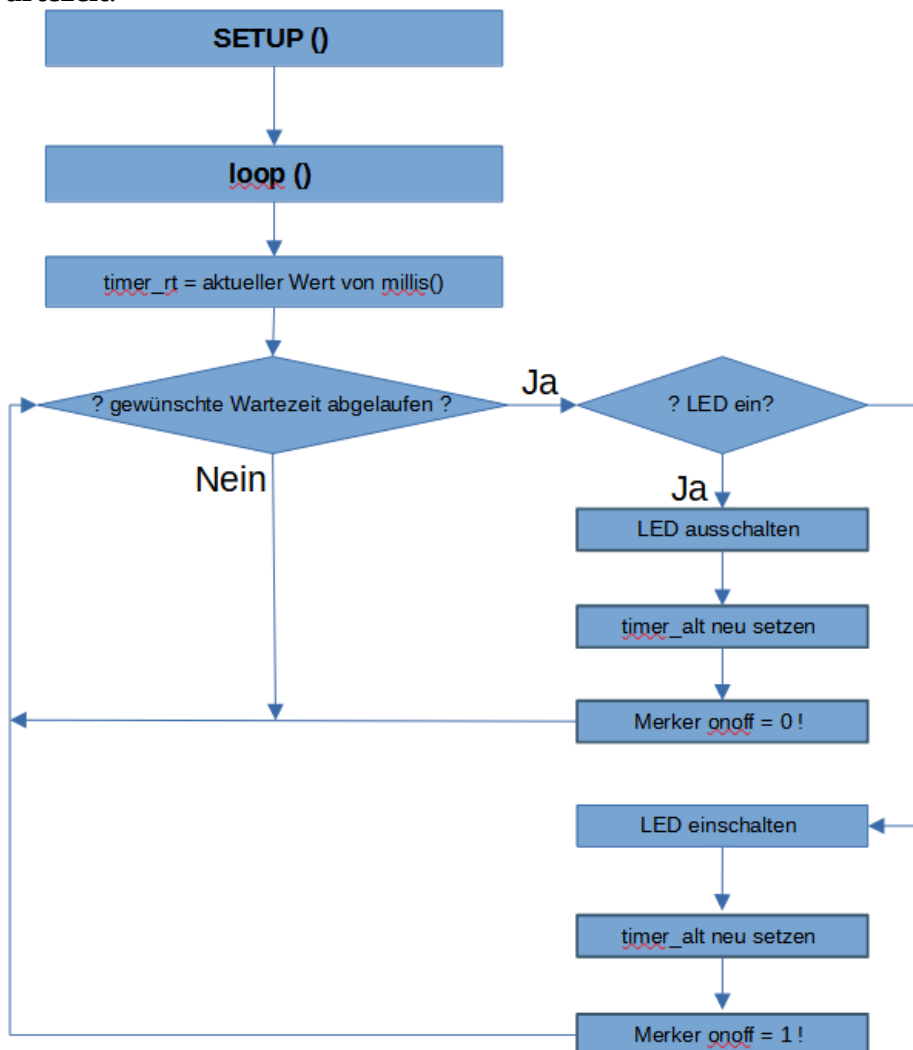
{ Programmcode; }

In der runden Klammer ist dabei eine Bedingung, deren Ergebnis entweder 0 oder 1 ergibt. Bei einer 1 werden die Befehle in der nachfolgend geschweiften Klammer abgearbeitet. Wenn die Abfrage nicht erfüllt ist, also 0 ist, wird die Befehlsreihe in der else Schleife abgearbeitet. Die Else Bedingung ist aber nur optional und wird dann benötigt, wenn genau definiert sein soll, was passiert, wenn die Bedingung nicht erfüllt ist. Bei einfachen Beispielen ist die oft nicht erforderlich, aber bei komplexeren Programmen, sollte immer bedacht werden, was passiert, wenn die Bedingung nicht erfüllt ist, damit der Prozessor keine unvorhergesehenen Dinge tut und Befehle ausführt, weil er plötzlich in einem Programmteil gelangt, der unzureichend definiert ist.

Oftmals genügt es, nur zu definieren, was passieren soll, wenn die Bedingung erfüllt ist, dann { Befehle in der Klammer abarbeiten; }

Betrachtet man die Zeitabfrage etwas genauer: **if(timer > timer_alt_rt + warte_rt)**

In der Variablen **timer** wird der Zählerwert laufend aktualisiert: **timer = millis()**; Die Wartezeit ist dann abgelaufen, wenn der aktuelle Zählerwert größer ist, als die Summe aus **alter Zählerstand + Wartezeit**.



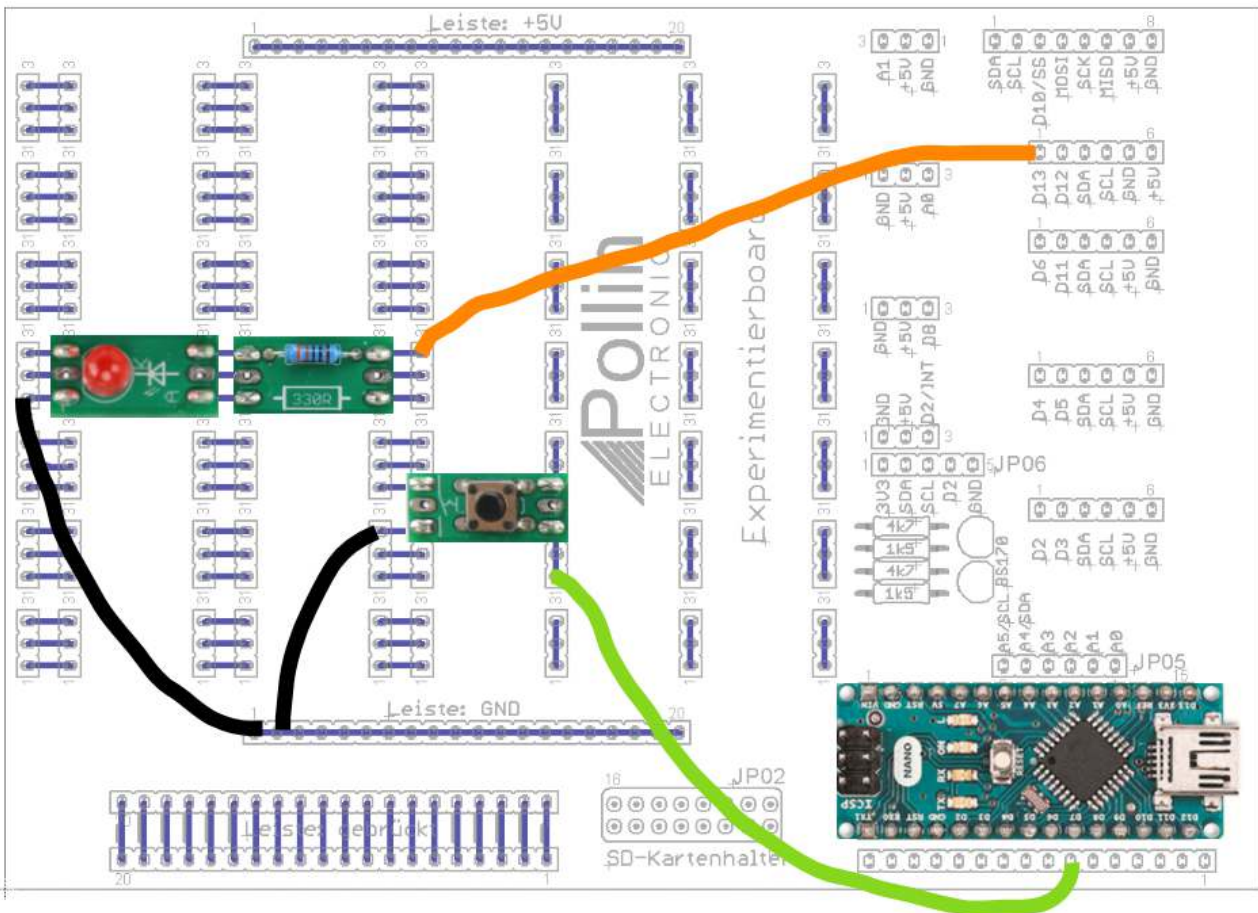
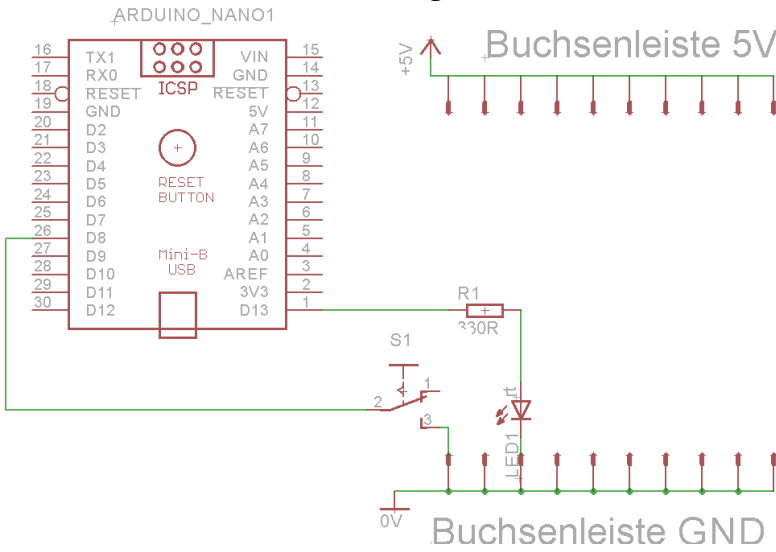
Dann soll sich der Zustand der LED (**HIGH, LOW** und die Variable **onoff_***) ändern und ab dann muss der Arduino wieder warten, bis die Wartezeit **timer_alt + warte** abgelaufen ist.

Das Beispiel **Blink4.ino** verwendet drei LED's mit unterschiedlichen Schaltzeiten.

11.1.3. LED Licht mit Taster ein und ausschalten: [Blink5.ino](#)

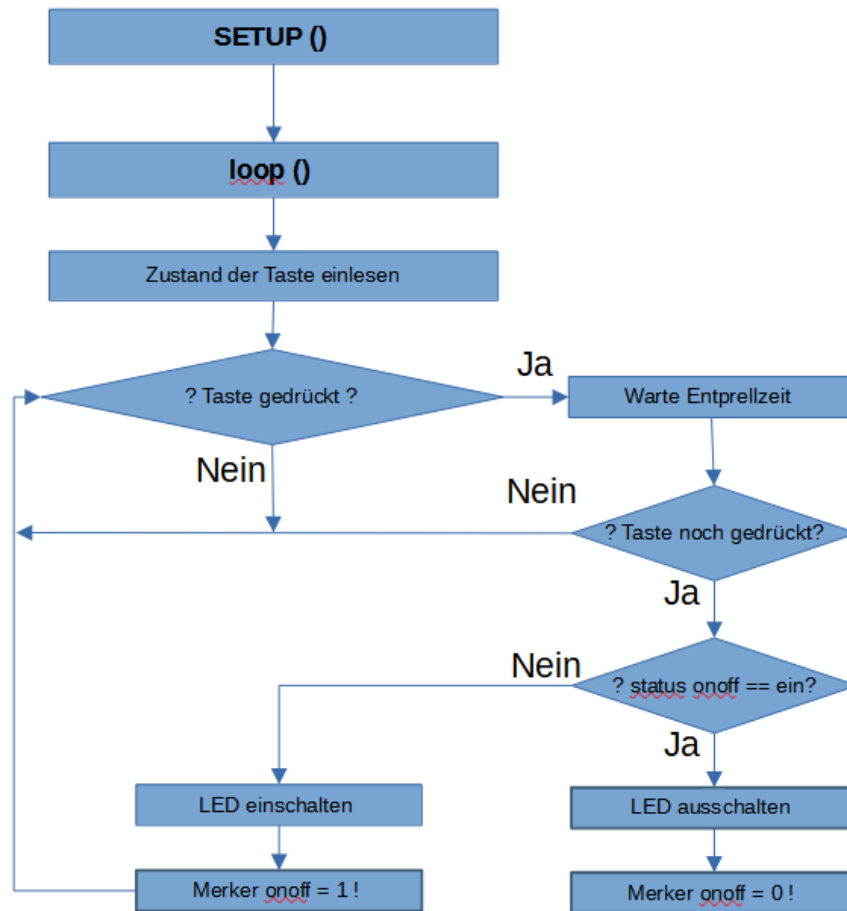
Nun soll mit einem Taster die LED ein- und ausgeschaltet werden; dazu wird wieder ein Port als Ausgang für die LED und ein zweiter Port als Eingang für einen Taster definiert. Normalerweise wäre ein Widerstand notwendig, der vom Taster nach 5V geschaltet wird. Wenn nun der Taster

betätigt wird, wird die Spannung am Widerstand gegen Masse geschaltet. Beim Loslassen, würde der Vorwiderstand den Pegel am Eingang wieder auf 5V ziehen. In diesem Beispiel ist ein Widerstand aber nicht notwendig, weil dieser Controller einen Widerstand am Eingang bereits integriert hat. Dieser kann durch einen Softwarebefehl aktiviert werden. Neben dem Arduino ist eine Buchsen-leiste platziert. Diese ist mit den Arduino Ports GND, D0 ... D13 und VCC belegt. Damit soll es erleichtert werden, den richtigen Port-Pin zu finden.



Gebraucht wird hier der Befehl `pinMode(PINNAME, INPUT_PULLUP)`, der den Eingang als pull-up definiert. Damit wird ein Widerstand im Prozessor aktiviert, der den Ausgang

automatisch auf 5V legt. Somit ist der Eingang „High“ und wird erst „LOW“, wenn der angeschlossene Taster gedrückt wird. Ein neuer Befehl, der hier verwendet wird ist: `digitalRead(pin)`; Damit kann man den logischen Zustand vom Eingang (Taster) einlesen.



Überlege, was am Flussdiagramm fehlt. Gehe das Flussdiagramm in Gedanken durch und du stellst fest, die LED leuchtet nur so lange der Taster gedrückt ist. Aber der ursprüngliche Ansatz ist gewesen, dass die LED mit einem Tastendruck eingeschaltet und mit einem weiteren Tastendruck wieder abgeschaltet werden soll. Deshalb benötigen man eine weitere Funktion: eine Schleife. Die Schleife dient dazu, zu warten, bis die Taste losgelassen wird. Das verhindert, dass unsere Loop permanent durchlaufen wird.

Anmerkung zu `do ... while(!Bedingung)` und zum Ausdruck `!pulled` im Beispiel [Blink5.ino](#). Die Befehle zwischen den geschweiften Klammern der `do ... while` – Schleife werden so lange ausgeführt, bis die Bedingung in der runden Klammer von `while (Bedingung)` nicht mehr wahr ist. Das bedeutet für das oben gezeigte Beispiel, dass die Schleife so lange durchlaufen wird wie die Variable `pulled == 0` ist. Denn mit dem Negierungszeichen `!` wird die 0 zu einer 1 und somit ist die Bedingung erfüllt.

Denn mit `!pulled` wird der Zustand von `pulled` negiert. Wenn `pulled == 0` ist, ist die Bedingung damit 1, also erfüllt und die Schleife wird weiter durchlaufen.

Wird die Taste losgelassen und `pulled == 1`, dann ist damit die Abfrage Bedingung in der Klammer nicht mehr wahr, weil das Negierungszeichen den Zustand von `pulled` umkehrt, somit kann das Programm die Schleife verlassen.

11.1.4. Reaktionstester: [blink6_zufall_reaktion.ino](#)

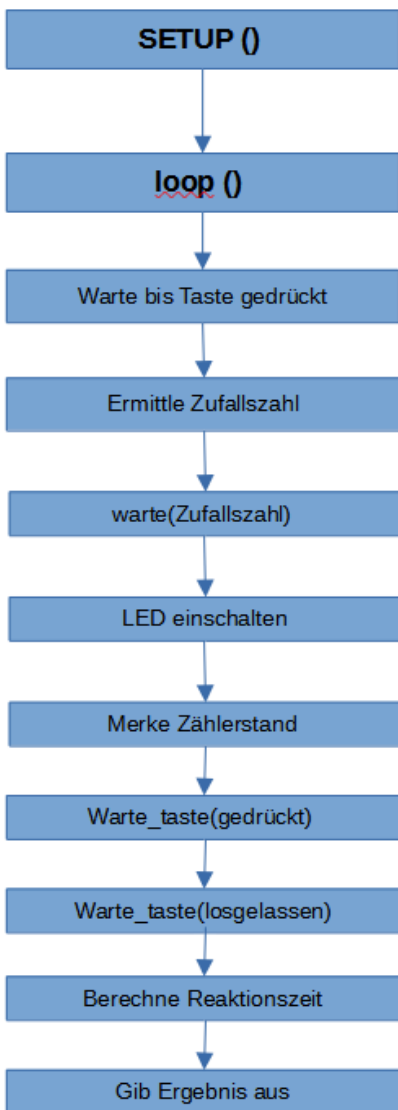
Es wird eine Zufallszahl erzeugt. Nach dieser Zeit wird eine LED verzögert eingeschaltet. Wenn die LED leuchtet, soll unmittelbar danach der Taster gedrückt werden. Diese Reaktionszeit vom Ansteuern der LED bis zum Drücken des Tasters, wird gemessen und über die serielle Schnittstelle ausgegeben.

Versuche nun, das Programm selber zu erweitern, dass Du eine Schwelle festlegst, bei der zusätzlich der Text ausgegeben wird „Gute Reaktion“ oder „Schlafmütze“.

Bei diesem Beispiel wird zuerst gewartet, bis die Taste gedrückt ist. Damit erfolgt der Programmstart. Es wird zuerst eine Zufallszahl für die Wartezeit ermittelt. Nach dieser Wartezeit wird eine LED eingeschaltet. Dann wird die Zeit bis zum Tastendruck gemessen. Dies ist quasi die Reaktionszeit. Sie wird über die Serielle Schnittstelle ausgegeben.

Mit der Funktion `Serial.print()`; kann ein Text, oder Variablenwerte über die Serielle Schnittstelle ausgegeben werden und über den seriellen Monitor dargestellt werden.

Dieser kann mit der Tastenkombination `<strg> +<shift> +<M>` geöffnet werden. Dabei muss aber die Baudrate im Fenster richtig eingestellt sein. Standardmäßig ist sie bei 9600. Deshalb wird auch normalerweise diese auch bei `Serial.begin(9600)`; festgelegt. Bei größeren Datenmengen ist es jedoch ratsam diese auf 115200 zu erhöhen.



`Serial.print('w');` ' ... ' gibt nur **ein** Zeichen aus

`Serial.print("Test");` " ... " gibt eine Zeichenfolge aus;

`Serial.println("Test");` gibt eine Zeichenfolge mit anschließendem carriage return und Line feed aus, beginnt also eine neu Zeile bei der nächsten Textausgabe. Dies ist noch ein Relikt aus der Zeit der Nadeldrucker, mit Zeilenrücklauf und Zeilenvorschub. Eine weitere Form Zahlen darzustellen ist `Serial.print (Wert, Format)` Format ist dabei die Art wie die Zahlen dargestellt werden: **DEC** als Dezimalzahl, **HEX** als Hexadezimalzahl und **OCT** als Oktalzahl.

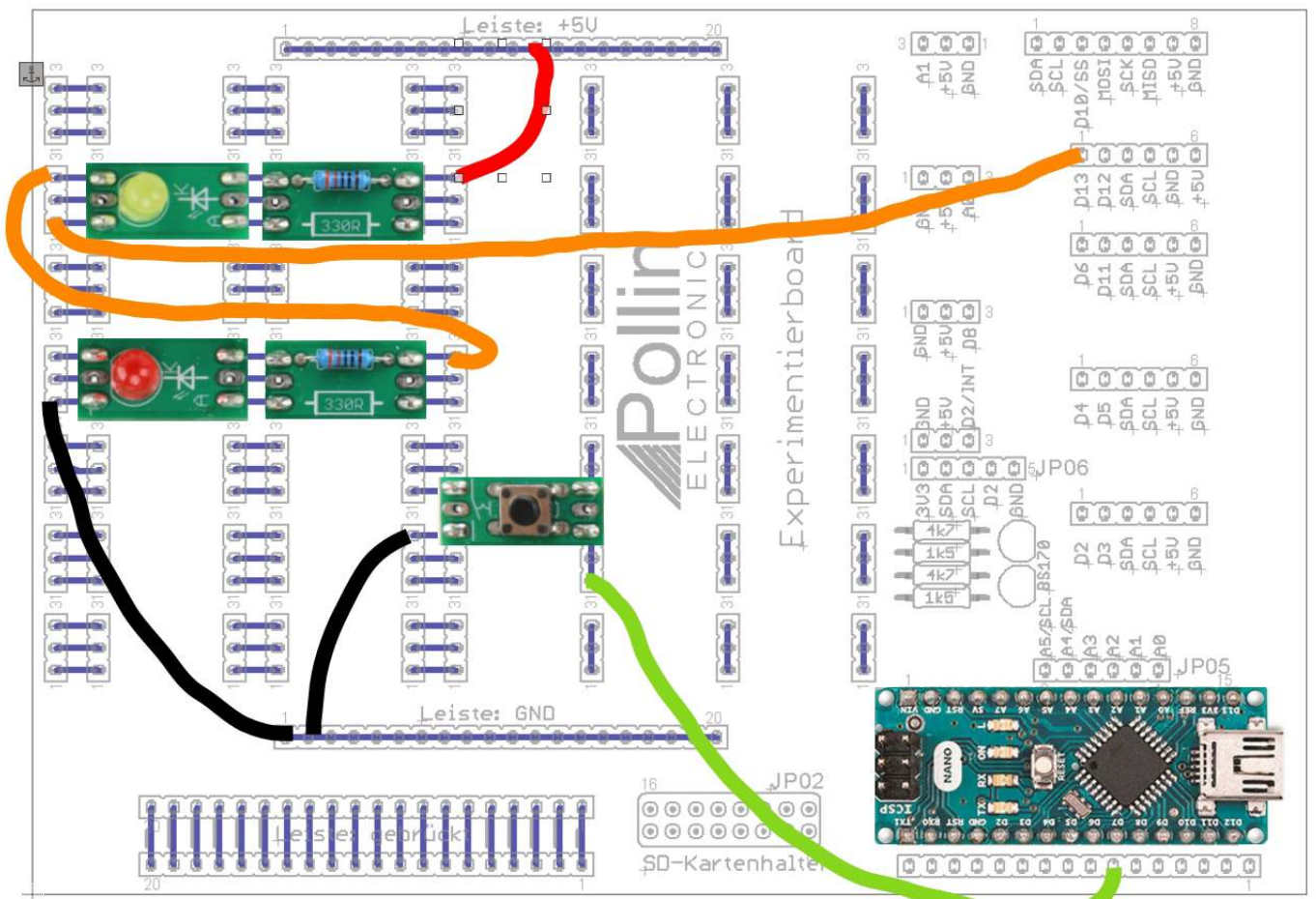
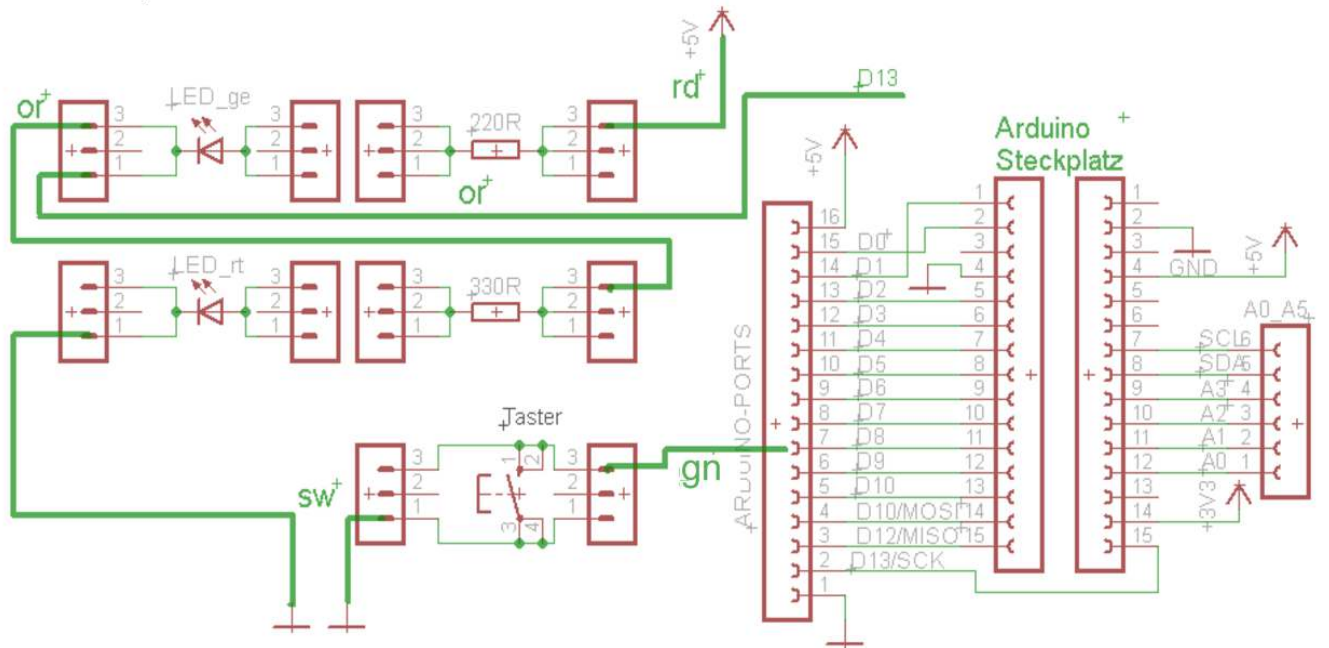
Um die Funktion `Serial.print` allerdings nutzen zu können, muss im `Setup()` zuerst die Funktion `Serial.begin(Baud)` aufgerufen werden. Damit wird die Schnittstelle des Arduino konfiguriert und die Übertragungsrate, also die Frequenz mit der die Bits übertragen werden festgelegt. Standardmäßig sind 9600 Baud üblich. Also 9600 Bit/s. Dabei besteht ein Datenstream nicht nur aus den 8Bit des Datenworts sondern auch dem Start- und Stopbit.

Anmerkung:

Eine weitere Anmerkung gilt der Funktion `void warte_taste(boolean status_taste)`. Wenn vor der Funktion das Wort `void` steht, dann bedeutet dies, dass die Funktion keinen Wert zurück gibt. Aber weil in der Klammer kein `void` steht, muss der Funktion bei deren Aufruf ein Wert vom Typ Boolean übergeben werden. Boolean bedeutet, die Variable kann nur die Werte 0 für false und 1 für true annehmen.

Links im Bild ist wieder der Programmablauf dargestellt.

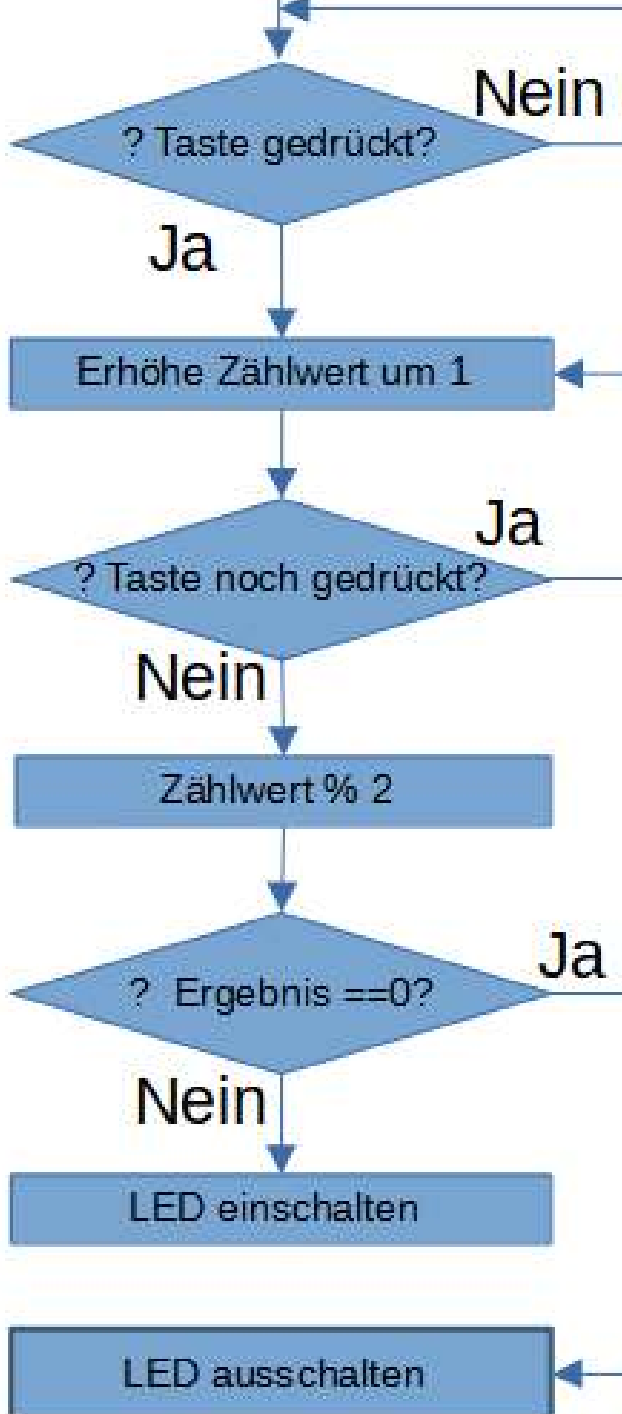
11.1.5. Kopf oder Zahl



Taste drücken: dabei wird ein counter hochgezählt.
 Wenn dann die Taste losgelassen wird, wird geprüft ob die Zahl gerade also durch zwei teilbar oder ungerade ist. Ist der Zählwert ungerade, dann wird die rote LED angeschaltet. Bei geradzahligem Ergebnis wird die rote LED ausgeschaltet.

SETUP ()

loop ()



Der exakte Code ist im Beispiel [Blink7.ino](#) abgelegt.

Als Anmerkung gibt es noch einen neuen Befehl zu erwähnen:

```
count = count % 2;
```

Dies ist eine Sonderform einer Division. Denn hierbei wird in die Variable der **Rest** der Division abgelegt. Das bedeutet bei **natürliche Zahl** $\text{count} \% 2 = \text{Rest}$. Ist der Rest dann entweder null bei einer geraden Zahl oder 1 bei einer ungeraden Zahl. So kann dann entweder die eine oder die andere LED angeschaltet werden. So bedeutet die LED entweder gerade oder ungerade oder ben Kopf oder Zahl. Erweitere das Programm [Blink7.ino](#) so, dass die gelbe LED dann leuchtet, wenn die rote LED erlischt und umgekehrt.

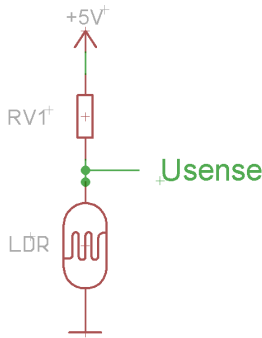
Hinweis:

```
#define LED_ge
```

Ergänze dabei im Setup()
pinMode()

```
In der if-Abfrage  
digitalWrite(LED_ge, HIGH)  
else  
digitalWrite(LED_ge, LOW)
```

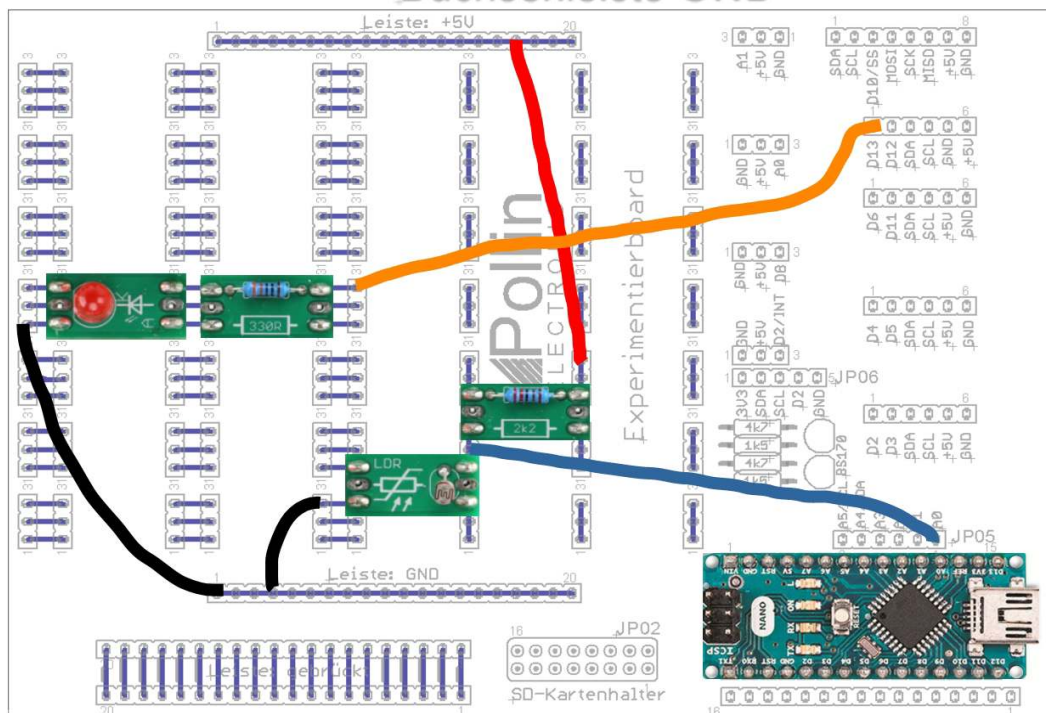
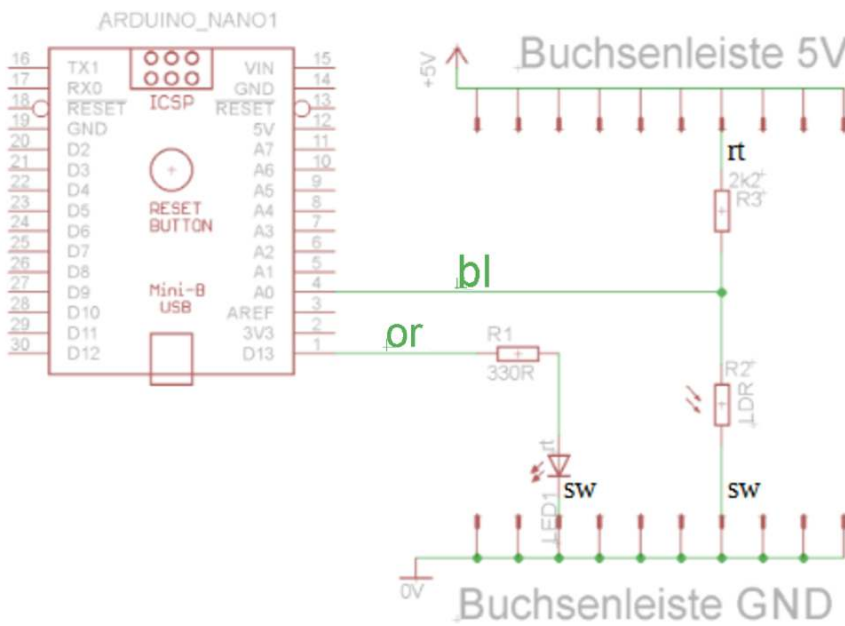
11.2. Der LDR Widerstand



Im Bild links ist ein Spannungsteiler dargestellt, der aus einem Vorwiderstand R_{v1} und einem LDR-Widerstand besteht. Der LDR ändert seinen Wert mit der Helligkeit der Umgebung. Daher sein Name Light Dependent Resistor. Je heller das Licht, desto kleiner der Widerstand. Je kleiner der Widerstand, desto kleiner der Spannungsabfall und desto kleiner die Spannung U_{sense} . Es ist schwer, den Wert zu kalibrieren und damit den exakten LUX-Wert der Helligkeit anzuzeigen. Aber man erkennt das Prinzip, wie die Messdaten zu nutzen sind, um z.B. einen Dämmerungsschalter zu bauen, oder eine Alarmanlage, wenn das Taschenlampenlicht von einem möglichen Einbrecher

darauf fällt.

11.2.1 Arduino als Helligkeitsanzeige



Betrachten wir uns nun das nächste Programmbeispiel und dabei zuerst, den Aufbau der Schaltung: Um den Widerstand des LDR (R2) bestimmen zu können wird ein Spannungsteiler benutzt. Der Widerstand R3 mit einen Wert von 2200 Ohm [2.2 kOhm] bildet mit dem LDR Widerstand R2 einen Spannungsteiler. Die Spannung von 5V verteilt sich auf beide Widerstände genauso, wie deren Verhältnis zueinander ist. Wenn beide Widerstände den gleichen Wert haben, dann halbiert sich auch die Spannung. Sinkt der Widerstand R2 mit zunehmender Helligkeit, dann wird auch die Spannung an diesem Widerstand kleiner und somit auch die Spannung die wir am Eingang zum Arduino messen können.

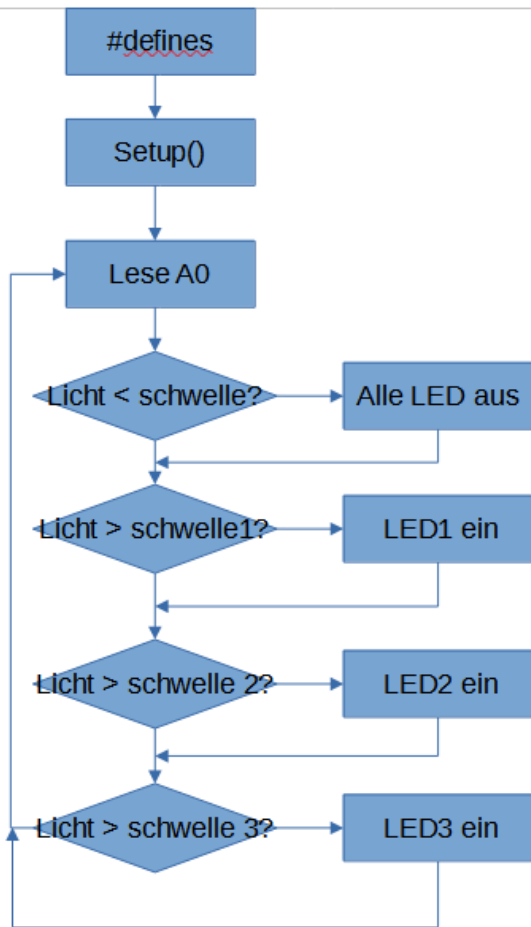
Um eine Spannung mit dem Arduino messen zu können, dient die Funktion: `analogRead(PORT)`; Als Port für eine Spannungsmessung sind nur die analogen Eingänge A0 .. A7 des Arduino geeignet. An einen dieser Eingänge wird der Spannungsteiler angeschlossen. Hier in unserem Fall ist es A0. Ein digitaler Eingang kann nur zwei Zustände 0 und 1, also LOW und HIGH annehmen. Analog bedeutet, dass der Arduino einem Spannungswert eine Zahl zuordnen kann. Der Spannungswert kann im Falle des Arduino Werte von 0...1023 annehmen. Der Wert 1023 entspricht dabei 5V. Entsprechend muss man dann die Werte die `analogRead()` liefert umrechnen, um den entsprechenden Spannungswert ermitteln zu können. Der Analog-Digital-Wandler hat nur eine Auflösung von 10Bit. Daher liefert er $2^{10} = 1024$ Messwerte und somit einen Wertebereich von 0 ... 1023, weil ja die Null der erste Wert ist. Bisher haben wir uns nur mit statischen Zuständen high und low befasst. So als wenn man mit dem Phasenprüfer eine Netzspannung prüft. Damit kann man nur sagen, ist eine Spannung an der Steckdose vorhanden, oder nicht. Manchmal jedoch möchte man schon wissen, welchen Wert die Spannung hat. An der Autobatterie kann durch Messen der Spannung in etwas ein Rückschluss auf den Zustand der Batterie gezogen werden. Bei vielen Sensoren möchte man auch oft einen genauen Meßwert kennen. Im nächsten Beispiel ist dies auch so. Da die Dämmerung nicht schlagartig eintritt, sondern es allmählich erst dunkel wird, möchte man die Helligkeit einstellen können, bei der das Licht eingeschaltet wird. Bisher haben wir noch nicht gelernt, eine Anzeige mit de Arduino anzusteuern. Deshalb nutzen wir wieder das Werkzeug „serieller Monitor“. Im Beispiel `LDR.ino` soll die Helligkeit als Spannungswert angezeigt werden.

```
Wert = analogRead(A0);
```

```
Spannung = Wert * (5.0 / 1023) [V];// nur bei 5.0 wird float ausgegeben, bei 5 nicht !
```

```
Serial.print(" Spannung: "); Serial.println(Wert,2);// 2 bedeutet, es werden zwei Nachkommastellen ausgegeben.
```

11.2.2.Helligkeitsanzeige mit mehreren Schaltschwellen:



[nachtlicht2.ino](#) hinterlegt.

Wenn die Dunkelheit einen gewissen Wert erreicht, soll die LED eingeschaltet werden. Der LDR ist hier so angeordnet, dass mit zunehmender Dunkelheit der Spannungswert an A0 steigt, weil der Widerstand des LDR zunimmt. Wenn die erste Schwelle überschritten, dann soll die erste LED leuchten. Dann die zweite und die dritte.

Ausschalten sollen die LED's erst, wenn eine gewisse Schaltschwelle unterschritten ist. Zwischen den Schaltschwellen sollte ein bestimmter Abstand, genannt Hysterese sein, damit es bei den Schaltschwellen nicht zu beliebigen Schaltvorgängen führt, weil der AD-Wandler auch Toleranzen hat. Dem obigen Schaltungsaufbau werden lediglich zwei weitere LED's hinzugefügt.

Aufgabe:

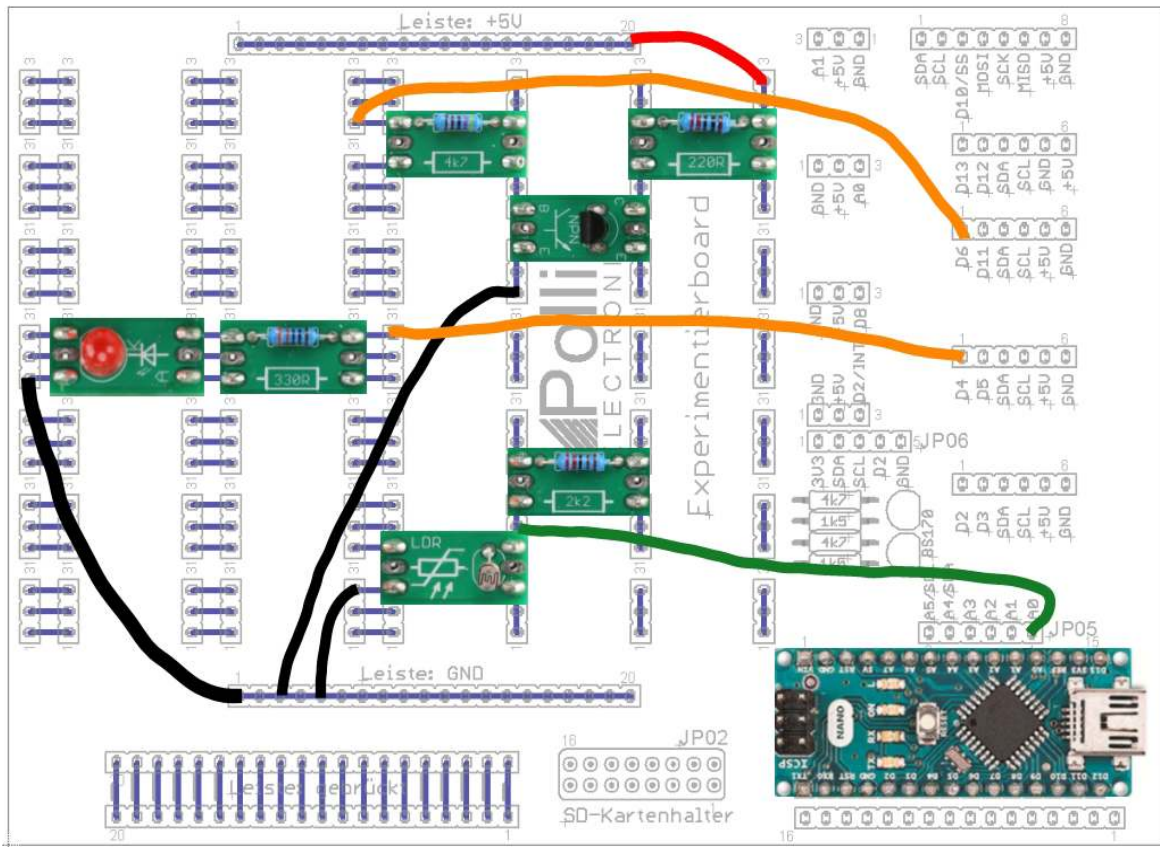
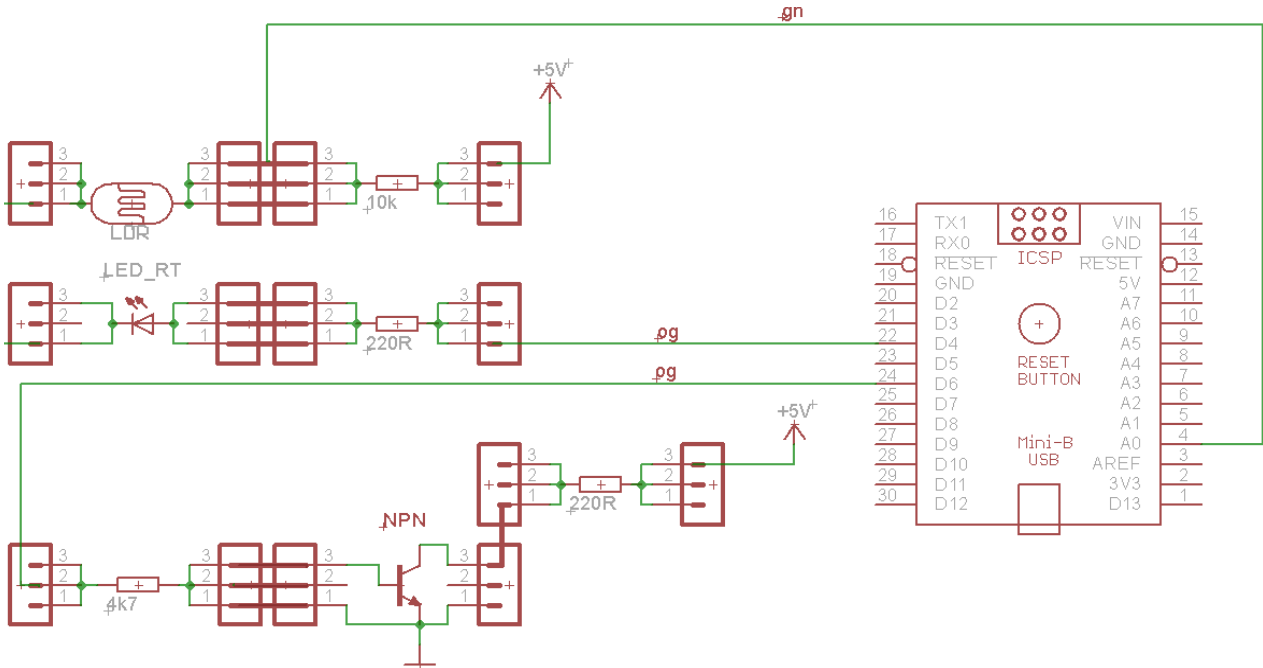
erweitere die Schaltung um zwei weitere LED's gegenüber dem Beispiel aus [nachtlicht1.ino](#).

Füge dann in der Software zwei weitere If Abfragen mit entsprechenden Schaltschwellen ein, so dass bei zunehmender Dunkelheit immer eine weitere LED zugeschaltet werden. Als Hilfestellung soll das nebenstehende Flussdiagramm dienen. Die Schaltschwellen kannst Du nach eigenem Ermessen wählen. Eine Lösung ist in der Datei

11.2.3. Geheimpfachwächter: *Schublade.ino*

Um zu überwachen, ob jemand in seine Schublade geschaut hat, kann man die gleiche Schaltung wie im Beispiel 4.1. verwenden, jedoch können die LED's entfernt werden. Allerdings ist eine weitere Option, die LED so oft blinken zu lassen, wie viele Male die Schublade schon geöffnet wurde.

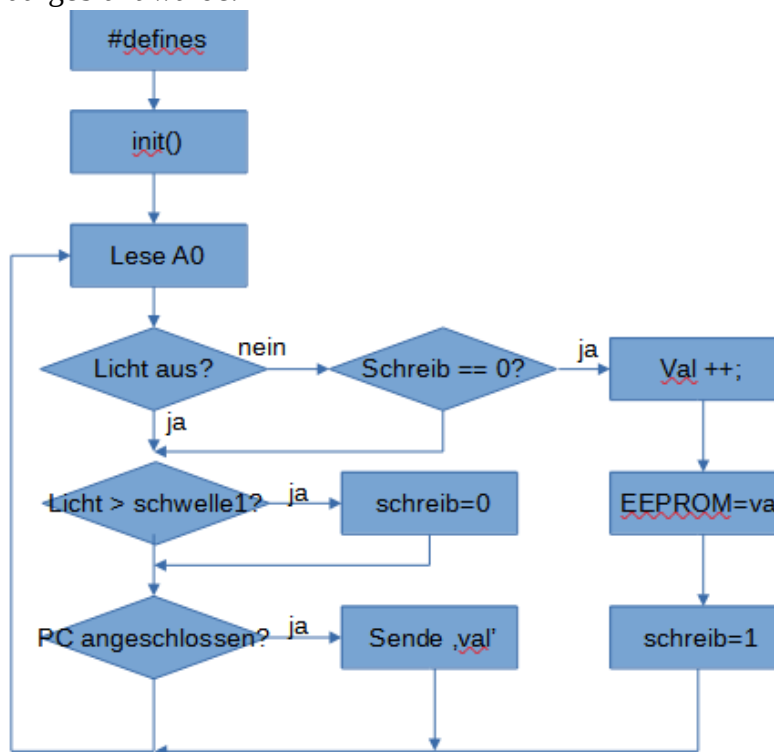
Allerdings braucht man noch zusätzliche Befehle und Funktionen. In der Schublade ist es überwiegend dunkel und so sollen nur die Ereignisse registriert werden, wenn ein Licht auf den Sensor trifft.



Dann kommt nun etwas ganz neues hinzu. Wir wollen einen Wert dauerhaft speichern. Selbst wenn der Neugierige den Arduino von der Batterie nimmt, soll das Ereignis gespeichert werden. Der interne Speicher, den wir verwenden ist ein EEPROM - ein elektrisch vom Prozessor lesbarer und löschbarer Speicher. Dieser Speicher behält aber seine Information, wenn der Arduino von der Spannung genommen wird. Um die Funktion in der Arduino IDE nutzen zu können muss man diese externe Bibliothek „einbinden“. Das bedeutet, man muss dem Übersetzungsprogramm (dem Compiler) mitteilen, dass diese Funktion benötigt wird. Es gibt Standardbefehle, die der Prozessor versteht, aber es gibt Funktionen, die nur in Bibliotheken bereitgestellt werden. Diese müssen dann exklusiv dem Übersetzungsprogramm mitgeteilt werden.

Der Befehl hierzu lautet `#include <...>`. Da stellt sich wahrscheinlich sofort die Frage, warum bindet man nicht gleich alle Bibliotheken ein. Denn erstens müsste man sich nicht kümmern welche Bibliothek man braucht und zweitens müsste man nicht wissen, wo die Bibliothek zu finden ist. Denn das ist manchmal auch notwendig, dass der Pfad mit im include Befehl angegeben wird, wo die benutzte Bibliothek abgelegt ist.

Aber wie gesagt, es ist nicht sinnvoll, alle Bibliotheken einzubinden, weil das Programm unnötig aufgebläht würde.



Der Befehl ein EEPROM zu beschreiben lautet: `EEPROM.write(addr, val);` die Variable `addr` beinhaltet den Wert, welche Zelle 0...512, im Speicher, beschrieben werden soll; die Variable `val` ist ein 8bit Wert: 0...255 . Es ist auch möglich größere Werte abzuspeichern. Dies geschieht dann mit dem Befehl `EEPROM.put(addr, val);` Dabei kann `val` jede Art von Variable sein. Mit dem Befehl `EEPROM.get(addr, val);` kann dann die Variable aus dem EEPROM zurückgelesen werden. Dies aber nur als Anmerkung am Rand. Wir werden uns in diesem Skript nur auf 8-Bit große Variablen beschränken. Anmerkung zur **if – Abfrage** im Programmbeispiel. Wenn es

notwendig ist, dass in einer Abfrage zwei Bedingungen erfüllt sein müssen, ist es nicht notwendig, diese ineinander zu verschachteln. Das würde funktionieren, aber das Programm schwerer lesbar machen. Also kann man die Bedingungen logisch verknüpfen. Eine logische Verknüpfung ist entweder `&&` wenn beide Bedingen erfüllt sein müssen, oder wenn nur eine der zwei oder mehreren Bedingungen ausreichend ist, dann gelingt dies mit `||` -Verknüpfung der Bedingungen. Die beiden senkrechten Striche werden mit ALT-Taste + 124 im Nummernblock erzeugt, wobei der Num-block natürlich auch aktiviert sein muss.

Wenn das Beispielprogramm läuft, möchte man ja auch irgendwie mit dem Arduino kommunizieren, um abzufragen, wie oft die Schublade während der Abwesenheit geöffnet wurde. Dann muss der Prozessor ja wissen, wann die Daten abgefragt werden, er soll diese ja nicht ständig senden.

Zudem muss es möglich sein, den Eintrag im EEPROM wieder zu löschen. Dies geschieht z.B. mit dem Buchstaben ‚l‘ dann wird das EEPROM gelöscht und mit ‚r‘ können die Daten aus dem EEPROM gelesen werden.

Energiesparmodus:

Im Beispiel [Schubblade2.ino](#) wird der Arduino für 8s in den Schlafmodus geschickt, um den Stromverbrauch zu minimieren. Das soll die Belastung der Spannungsversorgung (Batterie oder Powerbank) minimieren. Dafür benötigt man folgenden Befehl:

LowPower.idle(SLEEP_8S,

ADC_OFF,TIMER2_OFF,TIMER1_OFF,TIMER0_OFF,SPI_OFF,USART0_OFF,TWI_OFF,USART0_OFF,TWI_OFF);

Mit dem Befehl **LowPower.idle()** und dem Parameter **SLEEP_8S** wird der Arduino so konfiguriert, dass er alle 8s aufwacht; das Programm ausführt und wieder abschaltet. Es gibt natürlich viele weitere Möglichkeiten, Strom zu sparen. Aber dies würde den Rahmen hier sprengen.

Allerdings hat die Versorgung mit der Powerbank auch einen Pferdefuß. Bei zu geringer Belastung würde der Powerbank die Versorgung ganz abschalten. Dies gilt es zu verhindern, indem alle 8 sec der Stromverbrauch auf ca. 50mA ansteigt. Der Transistor T soll durch ein Einschalten eines Widerstandes ein Abschalten des Powerpack verhindern. Füge also einen Transistor der Schaltung hinzu. (siehe Bild links)

Um Daten mit dem Arduino über die Serielle Schnittstelle einlesen zu können, bedient man sich folgender Funktionen:

if (Serial.available() > 0)

Damit wird ein Wert zurückgeliefert, der angibt ob und wie viele Daten im Empfangsbereich angekommen sind. Wird als Wert 0 zurückgegeben, dann sind keine Daten vorhanden und die Befehle der nachfolgenden { } werden übergangen.

Variable = Serial.read();

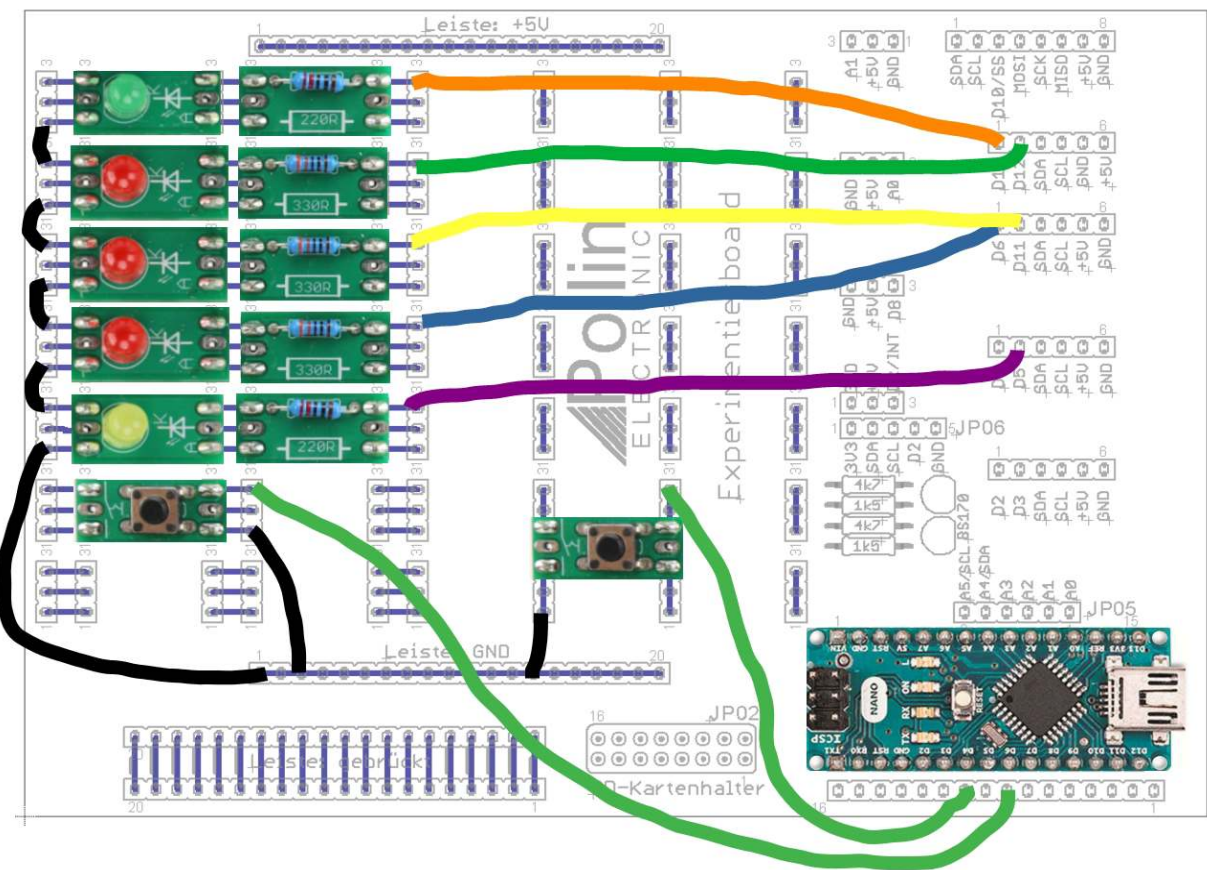
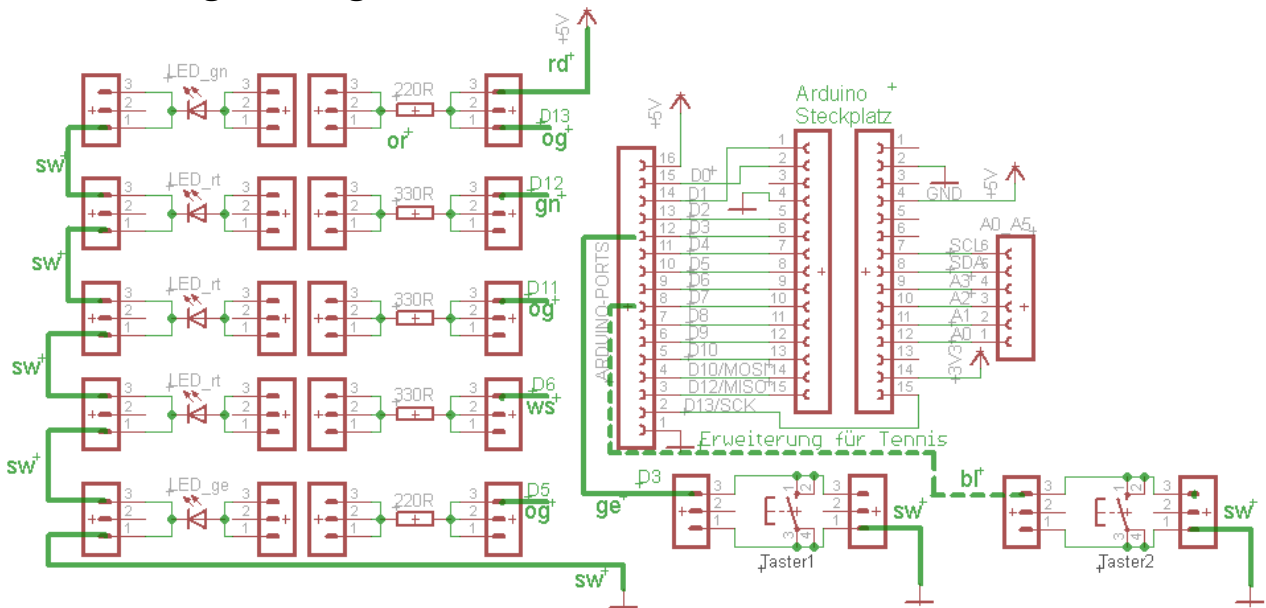
Damit können Daten vom Empfangspuffer in eine Variable eingelesen werden.

Hier der vollständige Aufbau:

11.3. Übungen mit Lauflicht

Als Vorbemerkung möchte ich einmal das Thema Variable ansprechen. Für den Anfang ist es zwar nicht ganz so wichtig. Aber ich möchte trotzdem nicht versäumen, es zu erwähnen. Ein Mikrocontroller, hat im Gegensatz zu einem PC nur sehr wenig Speicherplatz des RAM Speichers. Dies ist der Speicher, in dem alle Variablen, die der Programmierer anlegt, abgespeichert werden. Beim Arduino „uno“ und Arduino „Nano“ ist das 1kByte. Früher hatte ein PC einige MegaByte Speicher, also tausendmal so viel. Heute hat ein PC eine Million mal so viel Speicher, wie der Arduino. Also gilt es, diesen kostbaren Speicherplatz, zu sparen. Und dies geht am Besten, wenn nicht jede Variable gleich viel Speicherplatz für sich reserviert. Denn nichts anderes ist es, wenn eine Variable definiert wird. Es wird nur Speicherplatz reserviert. Deshalb ist es manchmal wichtig, einer Variablen auch gleich einen Wert zuzuweisen. Denn der Speicher kann bereits mit einem Wert von einem älteren Programm belegt sein. Der Prozessor kümmert sich nicht darum. Das ist die Aufgabe des Programmierers. Aber wieder zurück zur Größe des Speichers. Wenn wir den Zählerstand des timers abspeichern wollen, brauchen wir den meisten Speicher. Variable, die als **unsigned long** oder als Fließkommazahl **float** abgespeichert werden, reservieren den größten Speicher für sich. Sie benutzen vier Bytes des Speicherplatzes. Dafür kann aber eine Zahl von $0 \dots 2^{32}-1$ abgespeichert werden. Eine **unsigned int**-eger Zahl belegt 2 Byte des Speichers und kann Werte von $0 \dots 65535 (= 2^{16}-1)$ annehmen. Eine Zahl des Typs **unsigned char**, **uint8_t** oder **byte** belegen nur ein Byte und können dafür auch nur Werte $0 \dots 255$ annehmen. Das Wort **unsigned**, bedeutet Vorzeichenlos. Benötigt man eine Variable, deren Wert auch eine negative Zahl sein kann, lässt man das unsigned weg und der Zahlenbereich halbiert sich. Bei char bedeutet das, dass die dargestellte Zahl zwar 255 Werte hat aber eben nur von $-127 \dots 0 \dots +127$. Es ist von Vorteil ein **#define** für konstante Variable zu verwenden, weil diese im Programmspeicher abgelegt werden und somit keinen RAM-Speicher verschwenden. Was ist der Unterschied von Programmspeicher Flash, RAM-Speicher und dem EEPROM? Der Flashspeicher im Prozessor ist der Programmspeicher. Dieser Speicher wird beim Programmieren einmal beschrieben und kann zur Laufzeit des Programms nicht mehr verändert werden. Der RAM – Speicher kann jederzeit gelesen und verändert werden. Deshalb werden dort die Variablen abgelegt. Dieser ist aber sehr klein und abhängig vom Prozessortyp. Der RAM-Speicher verliert allerdings seinen Inhalt, wenn der Arduino von der Spannung getrennt wird. Das EEPROM ist ein Speicherbereich in dem Variable abgelegt werden können, die nach dem Trennen des Arduino von der Versorgungsspannung erhalten bleiben sollen. Dieser Speicher ist nur für eine bestimmte Anzahl von Schreib- und Lesezyklen ausgelegt und sollte für Variablen nicht verwendet werden. Er ist auch umständlicher zu beschreiben und zu lesen. Seine Größe ist auch abhängig vom Prozessortyp. In manchen Fällen kann es sogar sinnvoll sein, die Schreibzyklen auf das interne EEPROM als Variable abzulegen. Dann kann die Adresse nach z.B. 10000 Schreibzyklen erhöht werden. Somit kann die Verwendbarkeit des Arduino verlängert werden. Aber eben nur in Ausnahmefällen, denn 10.000 Schreibzyklen sind schon sehr viel.

11.3.1. Schlag die Fliege



Dies wird eine weitere Variante die Reaktionszeit zu testen. Dabei soll der Rhythmus der blinkenden LED beobachtet werden. Dann soll daraufhin abgeschätzt werden, wann die äußere LED leuchten wird. Ziel ist es, zeitgleich mit dem Aufleuchten der LED die entsprechende Taste zu drücken. Im ersten Programm „Schlag die Fliege“ ist nur ein Taster. Das „Tennis-Spiel“ ist für zwei Spieler. Spieler1 bedient die grüne LED und Spieler2 bediente mit dem Taster2 die gelbe LED. Das Spiel kann dann je nach Phantasie und können erweitert werden. Wer die ersten 10 erfolgten Schläge betätigt hat ist Sieger. Dabei kann der

Schwierigkeitsgrad auch gesteigert werden, in die Toleranz für die Reaktionszeit weiter verkleinert wird. Später kann dann versucht werden die Punkte über ein Display auszugeben, was jetzt einfach nur über die serielle Schnittstelle machbar ist. Eine weitere Möglichkeit den Schwierigkeitsgrad zu steigern ist es, die Taste nicht zeitgleich mit der letzten LED zu drücken, sondern um die Zeitdifferenz später diese zu drücken. Also wenn LED5 leuchtet + delay(zufall) abwarten und dann erst die Taste drücken.



Funktion, die den Startwert als „Zufallszahl“ generiert:

Ziel des Spiel soll sein, die Zeitdifferenz zu beobachten, zwischen der, jede LED zu leuchten beginnt. Denn dadurch soll abgeschätzt werden wann die rote LED leuchtet. Denn der Taster soll gedrückt werden nach dem Zeitintervall wenn die rote LED aufleuchtet. Dazu könnte man noch einen Summer bauen, der zu piepsen beginnt, wenn der Spieler es wirklich geschafft hat, das Intervall zu treffen und die Taste zu drücken. Erweitere die Schaltung und das Programm [schlag_fliege.ino](#) um diese Funktionalität. Es wird bei jedem neuen Durchlauf der LED, die Zeitdifferenz zwischen dem Aufleuchten der einzelnen LED's, durch Zufall neu festgelegt.

Aber zuerst noch ein paar Hinweise: Es gibt zur Funktion **random()** einige Anmerkungen. Die Funktion random() soll Zufallszahlen erzeugen. In den Klammern übergibt man dabei der Funktion Werte. Übergibt man der Funktion nur einen Wert x, so werden als Zufallszahl Werte von 0 ... x zurückgegeben.

Ein Funktionsaufruf (x,y); bewirkt, dass die zurückgegebenen Werte im Bereiche der Variablen x ... y liegen. Allerdings ergibt sich bei dieser Funktion ein Problem: wird der Arduino neu gestartet, dann erfolgt die Ausgabe der Zufallszahlen wieder in der selben Reihenfolge, genau wie beim vorigen Start. Es sind also nur bedingt zufällige Zahlen.

Um dies zu unterbinden gibt es noch eine

randomSeed(analogRead(A_{in}));

A_{in} soll dabei ein unbeschalteter Analogeingang A0 ... A7 sein. Dann wird zufällig der Wert eingelesen, der gerade gemessen wird. Durch das Rauschen am Signaleingang schwanken diese Werte.

Zu erwähnen ist noch, die Prozedur oder auch Unterfunktion genanntes Programm `warte_taste();` Neu ist dabei, dass dieser Funktion ein Wert übergeben werden kann. Dabei ist zu beachten, dass die Variable im Programmkopf z.B. als **boolean** definiert wird.

void `warte_taste (boolean status_taste);` Man kann dabei auch mehrere Variablen übergeben. Diese können auch verschiedenen Typs sein :

void `LED (boolean status_on, int timer1);` Es kann von der aufgerufenen Funktion auch ein Wert zurückgeben werden: `zustand = status (LED1);` dabei ist die Funktion dann folgendermaßen zu definieren:

vor der Funktion steht dann **nicht void** sondern der Variablentyp, der zurückgegeben wird:

Hier ein Beispiel für eine Funktion mit einem Rückgabewert:

```
char Funktionsname ( unsigned char variable, definition_Variable
deklaration_Variable, ... ...)
{
    ... variable1;
    unsigned char variable_2;
    ... Code Zeilen
    ...
    ...
    Return (variable_2); // damit wird ein Wert an die
                           Funktion
aufrufende
zurückgegeben!
}
```

Vor dem Funktionsnamen steht die Definition der Variablen, die dem Compiler mitteilt, wie groß der Speicherbedarf der Variablen ist, die von der Funktion zurückgegeben wird. (Beispiel **char**: 1Byte; **int**: 2Byte; **long**: 4Byte) In der Klammer nach dem Funktionsnamen werden die Variablen definiert, also die Speichergröße festgelegt und auch deklariert, also der Speicherzelle einen benutzerdefinierter Name vom Programmierer zugeteilt. Die Variablendefinition und Deklaration werden mit **,** voneinander getrennt.

Neu ist auch die Rückgabe der Werte mit `Return (variable1, variable2, ...);`

Der Vorteil von Funktionen mit Rückgabewert ist auch, dass Speicherplatz für RAM Variablen eingespart werden kann. Denn so ist es möglich globale Variable einzusparen. Variable in Funktionen sind nur so lange gültig, wie der Funktionsaufruf dauert. Anschließend wird der Speicher den die lokalen Variablen belegt haben wieder frei gegeben. Globale Variable, die im Programmkopf deklariert werden, haben immer ihre Gültigkeit und deren Speicherbedarf bleibt immer Reserviert. Dies ist der Unterschied von lokalen Variablen, die während der Laufzeit durch einen Unterprogrammaufruf deklariert werden und globalen Variablen.

11.3.2. Erweiterung mit 2 Tastern (Tennis.ino)

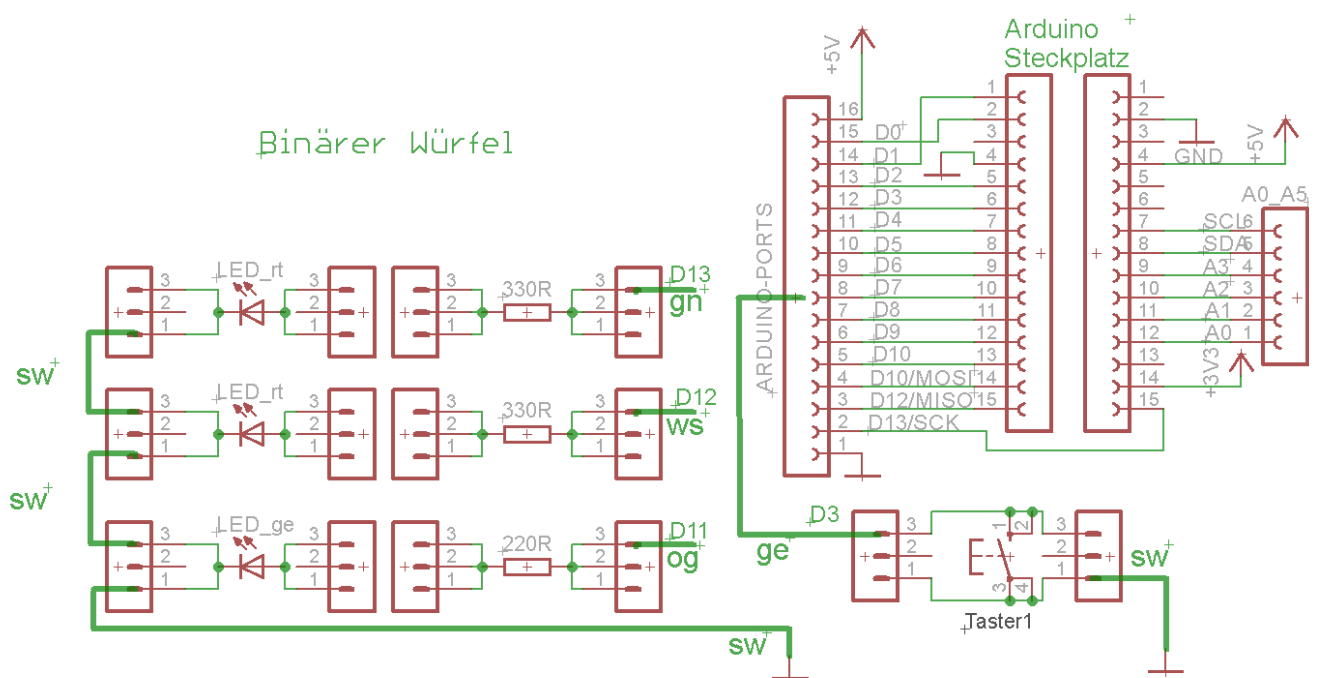
Bei Schlag die Fliege „bewegen“ sich die LED's von links nach rechts und dann soll im richtigen Moment die Taste gedrückt werden. Bei dem Beispiel [Tennis.ino](#) wird einfach die Funktion erweitert. Hat Spieler1 richtig getroffen, so beginnen die LED's wieder zurücklaufen zu Spieler2. Wenn auch dieser richtig drückt. Dann beginnt wieder von vorne. Die grobe Programmstruktur ist bereits vorgegeben. Jetzt ist nur noch die zweite Taste im Programm zu implementieren.

Vielleicht ist es möglich einige Variablen als `#defines` im Programmkopf einzubinden, damit das Programm leichter an steigende Anforderungen angepasst werden kann.

Beispiele: Reaktionszeit `v_r`; (variable) Toleranz `c_r`; (Konstante für erlaubte Reaktionszeit) `v_countError` (wie viele Fehler hat jeder Spieler schon gemacht) `c_limit` (Anzahl erlaubter Fehler) `v_high_score` (wann ist ein Spieler der Sieger, wie oft hat er die richtige Zeit erwischt) ; `v_statistik1`, `v_statistik2`: Zähler wie oft jeder Spieler schon gewonnen hat, wie ist der aktuelle highscore; entscheide Du, wird der highscore durch einen neuen überschrieben?

Die Möglichkeiten dieses Programm deinen Bedürfnissen anzupassen sind vielfältig. Also viel Spaß und viel Erfolg dabei.

11.3.3. binärer Würfel



Binäre Zahl bedeutet, dass zwei Zustände die Basis bilden. Im Gegensatz zum 10er Zahlensystem wird nicht erst bei Zehn, die Stelle erweitert. Die erste Stelle kann nur 0 oder 1 sein. Die erste Stelle wird dabei als Digit0, die nächste höherwertige ist Digit1 bezeichnet und dann folgt Digit 2 usw.

Binäre Zahl bedeutet, dass nur zwei Zustände die Basis bilden. Im Gegensatz zum 10er Zahlensystem wird nicht erst bei Zehn, die stelle erweitert. Die erste Stelle kann nur 0 oder 1 sein. Die erste Stelle wird dabei als Digit0, die nächste höherwertige ist Digit1 und dann folgt Digit 2.

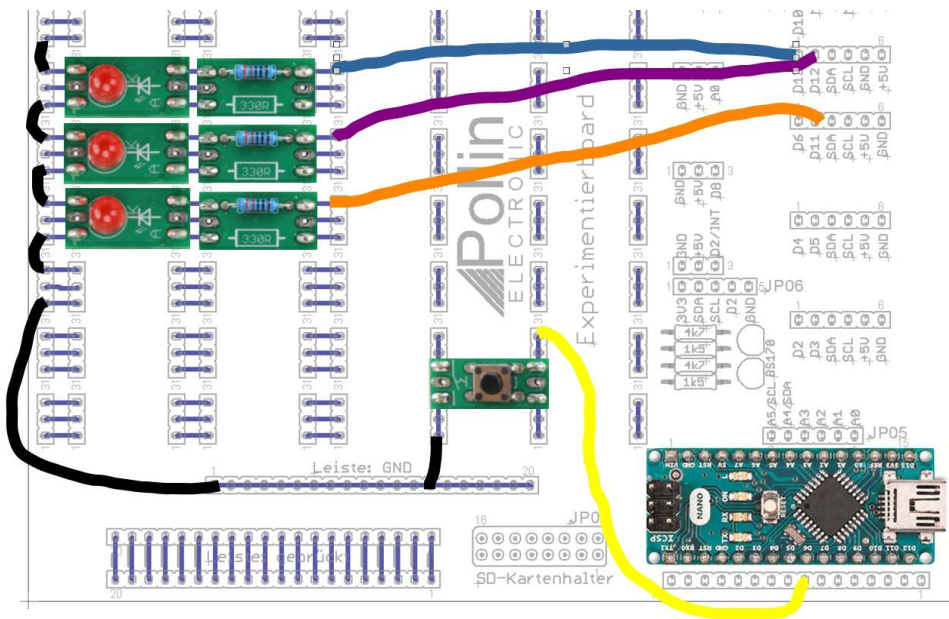
(MSB)

(LSB)

	DEC	HEX	Binär	Digit_3	Digit_2	Digit_1	Digit_0
Zahl:	0	0x0	0b0000	0	0	0	0
	1	0x1	0b0001	0	0	0	1
	2	0x2	0b0010	0	0	1	0
	3	0x3	0b0011	0	0	1	1
	4	0x4	0b0100	0	1	0	0
	5	0x5	0b0101	0	1	0	1
	6	0x6	0b0110	0	1	1	0
	7	0x7	0b0111	0	1	1	1

	8	0x8	0b1000	1	0	0	0
	9	0x9	0b1001	1	0	0	1
	10	0xA	0b1010	1	0	1	0
	11	0xB	0b1011	1	0	1	1
	12	0xC	0b1100	1	1	0	0
	13	0xD	0b1101	1	1	0	1
	14	0xE	0b1110	1	1	1	0
	15	0xF	0b1111	1	1	1	1

vier Bit ergeben ein Nibble. Dies ist die Hälfte eines Bytes, welches aus 8 Bit besteht. Mit einem Nibble kann wie oben abgebildet 16 Zahlen von 0 .. 15 dargestellt werden. Im Hexadezimalen Zahlenformat werden die Zahlen größer 10 als A... F (=dezimale 15) dargestellt. Ein Byte in Hexadezimaler Schreibweise wird dann die Zahl 255 als größter 8Bit Wert mit 0xFF und in binärer Schreibweise als 0b11111111 dargestellt. So weit also die Zahlenformate. Aber wir wollen ja nur als Zufallszahl für einen Würfel die Zahlenwerte zwischen 1 ... 6 ausgeben. Entsprechend der oben dargestellten Binärzahlen ist es auch möglich diese direkt über die Ausgänge des Prozessors auszugeben. Die Ports des Prozessors sind eben auch nach dem Binärsystem aufgebaut. Pin10 ist Digit2 des Port_B; Pin9 ist Digit1 von Port_B; und Pin8 ist Digit0 von Port_B; Im nächsten



Beispiel beim Laufflicht, wird auf diese Art LED's anzusteuern näher eingegangen und exemplarisch gezeigt. Doch zunächst betrachten wir das Beispiel bin_wuerfel.ino. Dabei steuern wir die LED's noch ein wenig umständlicher an, damit ein Unterschied und eine Verbesserung zum

Laufflicht deutlich zu erkennen ist.

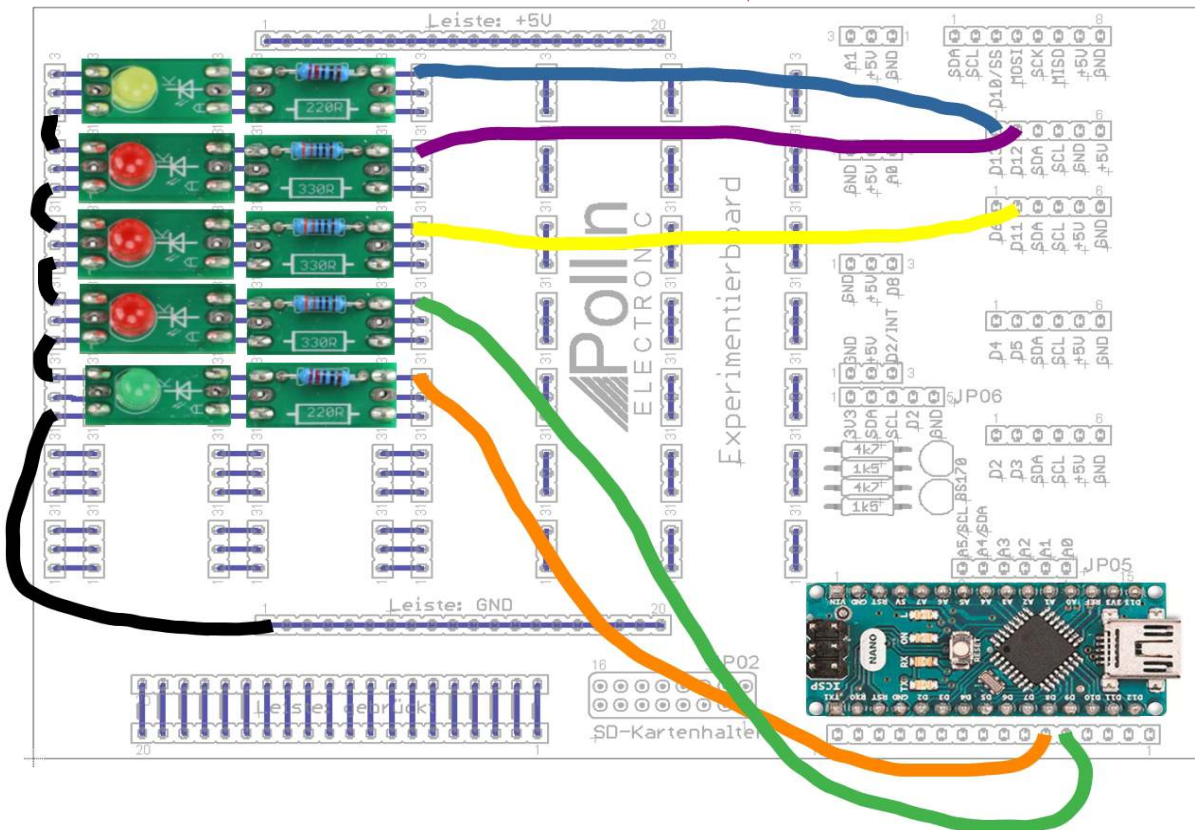
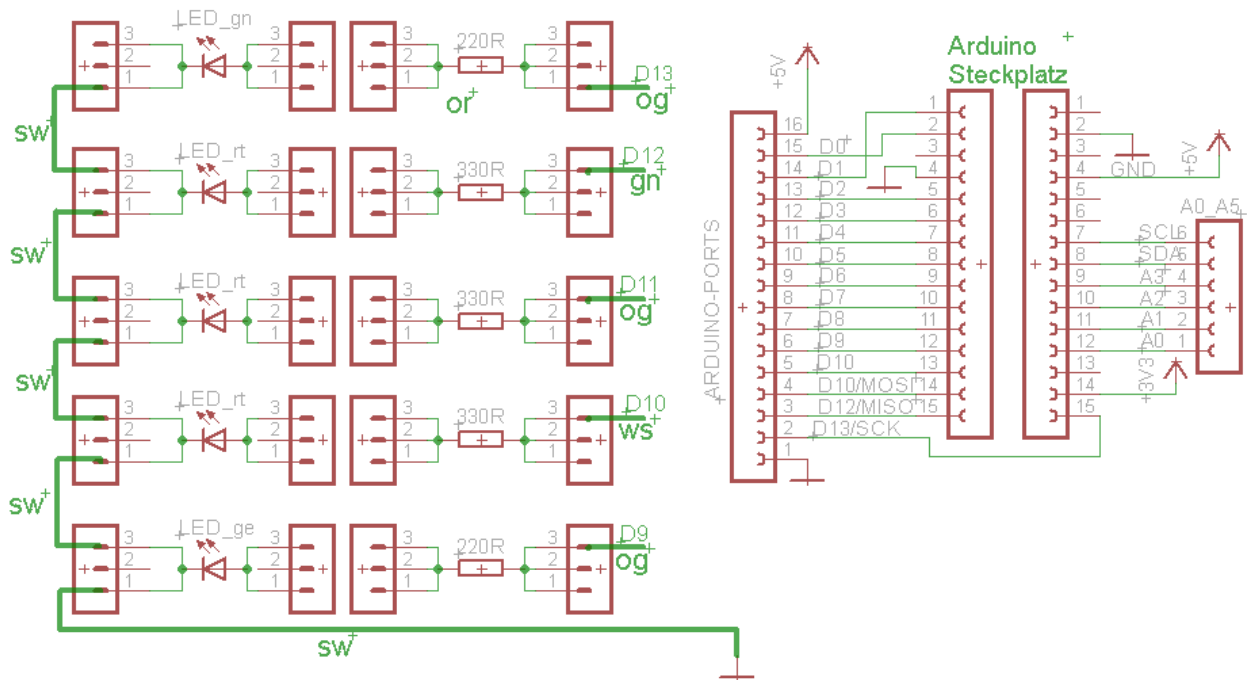
Was bedeutet LSB und MSB?

LSB bedeutet **L**owest **S**ignificant **B**it (niederwertigstes Bit) und MSB = **M**ost **S**ignificant **B**it (höchstwertiges Bit) das niederwertigste Bit ist quasi die niedrigste Stelle bei der zu zählen begonnen wird. Das LSB kann Werte von 0 und 1 annehmen. Das zweite Bit kann auch nur zwischen 0 und 1 schalten, aber von der Wertigkeit, ergeben sich die Werte 2 und 0. Pin D10 hat die Wertigkeit von 4. Das kommt daher, weil das Zahlensystem als binäres oder duales Zahlensystem bezeichnet wird. Eine Dualzahl ist dabei eine Zahl, die nur zwei Zustände annehmen kann. Beim dezimalen Zahlensystem kann eine Stelle zehn Werte annehmen. Mit jeder Stelle verzehnfacht sich die dargestellte Zahl. Beim Binären Zahlensystem verdoppelt sich die Zahl mit jeder zunehmenden Stelle. Es wurde definiert, eine Gruppe von 8 binären Ziffern (Bit) zu einem Byte zusammenzufassen.

Die höchstwertige (MSB) Ziffer (Bit) hat dabei einen Wert von 128. Werden alle Bitwerte in einem Byte zusammenaddiert, dann ergibt sich der Wert 255. Mit der Zahl 0 ergeben sich für acht Bit also 256 darstellbare Zahlen, also ein **char** (Character). Zwei Byte werden ein Double oder ein **word** genannt. Damit lassen sich integer Zahlen von -32768...+32767 darstellen oder ganzzahlige sogenannte **unsigned int** von 0...65535;

11.3.4. Lauflicht (Beispiel: [Lauflicht.ino](#))

Eine weitere Möglichkeit, ein Lauflicht zu programmieren, ist den Port als ganzes anzusprechen (siehe [Lauflicht2.ino](#)). Bisher haben wir jeden Port-Pin einzeln auf HIGH oder LOW gesetzt. Wie ist es aber möglich, alle acht Pins von einem Port gleichzeitig anzusprechen? Die einzelnen Pins: 8 (PB0), 9 (PB1), 10 (PB2), 11 (PB3), 12 (PB4), 13 (PB5) kann man auch auf einmal ansprechen, indem ein Byte in das Port-Register direkt geschrieben wird:



Die Ausgänge eines Port können mit dem Befehl **PORTB** angesprochen werden. Wobei zu beachten ist, dass der Buchstabe B hier für den Namen des Port steht. Es gibt beim Arduino noch einen PORTC A0 (**LSB**) ... A5 (**MSB**) und einen PORTD = PinD0 (**LSB**) ...Pin D7 (**MSB**)

```
PORTB = 0x01; // Pin D8 = HIGH; (LSB)
PORTB = 0x02; // Pin D9 = HIGH;
PORTB = 0x04; // Pin D10 = HIGH;
PORTB = 0x08; // Pin D11 = HIGH;
PORTB = 0x10; // Pin D12 = HIGH;
PORTB = 0x20; // Pin D13 = HIGH;
PORTB= 0x3F; // alle Pins = HIGH; (MSB)
```

Ein Port besteht normalerweise aus 8 Bit. Beim Port B sind allerdings nur 6 Pins (D8 ... D13) an die Steckerleiste geführt. Die beiden anderen sind nicht für den Anwender verfügbar!

Möchte man nun z.B. D9 und D11 gleichzeitig auf HIGH setzen, so tut man dies

durch $0x02 + 0x08 = 0x0A$;

```
PORTB = 0x0A → D9 und D11 sind HIGH;
```

Eine weitere Möglichkeit die LED's anzusteuern, ist die Manipulation der Variablen mit dem Shift Befehl. Der Shift oder auch Rotate Befehl genannt, agiert wie ein Schieberegister. Ein Schieberegister bewegt die Daten im Prozessor mit jedem Befehl um ein Bit, entweder nach links oder rechts.

Beispiel: Im Prozessor ist das binäre Wort 0b00110111 dies entspricht dem Dezimalwert: 55 bei einem Links-shift werden die Bits einfach eine binäre Stelle nach links verschoben, so wie im Dezimalen System einfach eine 0 angehängt wird. Der Zahlenwert ändert hier jedoch seinen Wert um das doppelte:

Im Prozessor ist Dann das binäre Wort 0b01101110 dies entspricht dem Dezimalwert: 110 der Schiebepfeil in der Programmiersprache C lautet » oder «

```
x = 1;
```

```
x = x << 1 // der Wert wird um ein Bit links geschoben → x ist jetzt 2
oder
```

```
x = 1 << 3 // der Wert wird um 3 Bit links geschoben → x hat nun den Wert 8
// x hat nun den binären Wert 0b00001000;
```

entsprechend gibt es auch den Befehl zum Rechtsshift, bei dem dann der Wert mit jedem Bit halbiert wird:

```
x = 8; // binärer Wert 0b00001000;
```

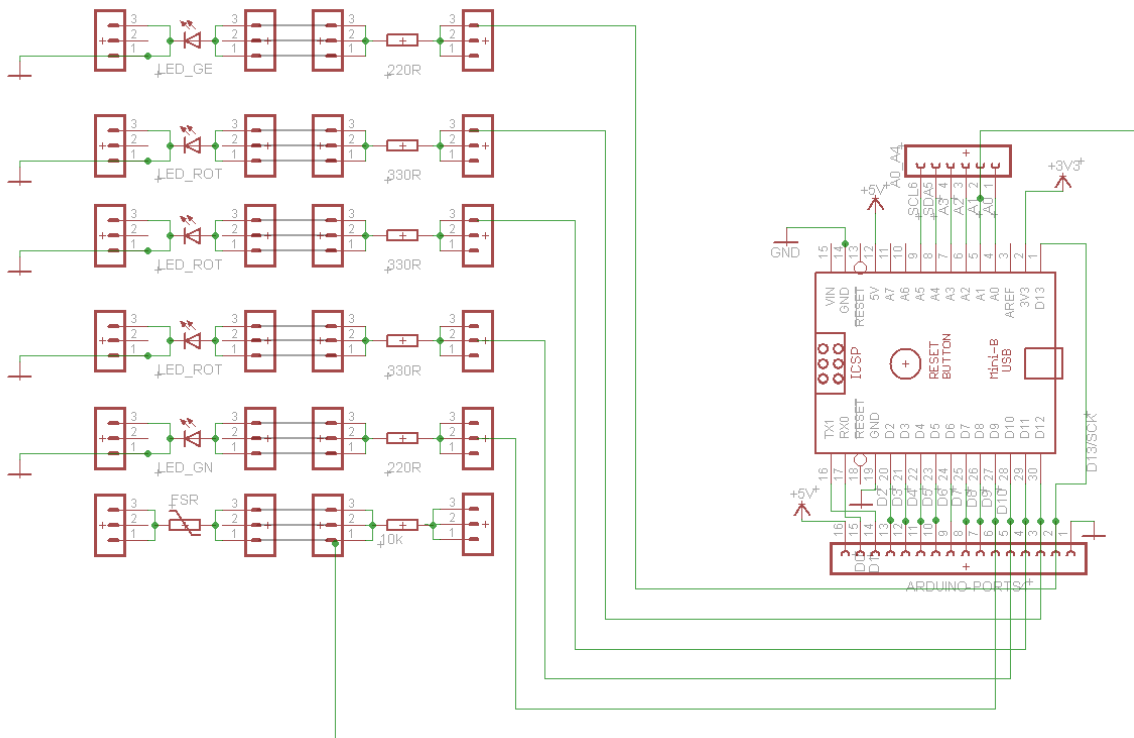
```
x = x >> 1 // der Wert wird um ein Bit nach rechts geschoben → x = 4
// x hat nun den binären Wert 0b00000100;
```

Eine Möglichkeit die PORTS anzusteuern ist so:

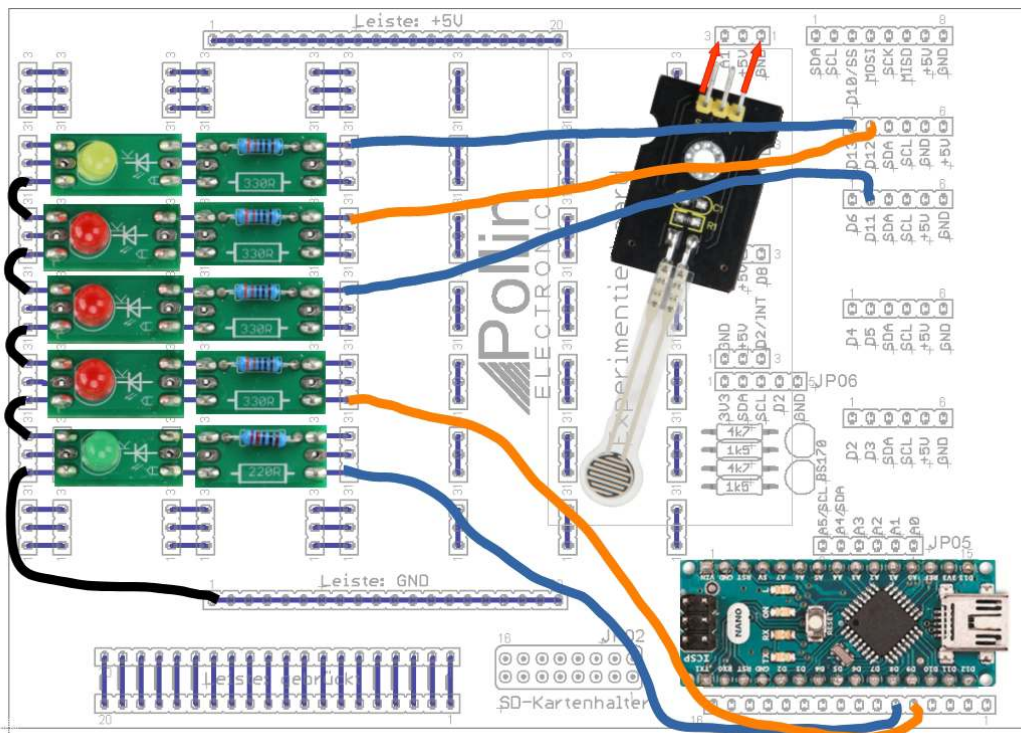
```
x=1;
for( y =0; y < 5)
{
    PORTB = x;
    x = x<< 1;
    delay(T_on);
}
```

Ändere dazu das Beispiel [Lauflicht3.ino](#)) entsprechend ab.

11.3.5. FSR-Sensor (Artikel 811 164) steuert LED band (FSR_beispiel.ino)



Ein FSR Sensor ist ein Widerstand, der seinen Wert dadurch ändert, dass Kraft auf ihn ausgeübt wird. Mit zunehmender Kraft reduziert sich dabei der Widerstand. Die Schaltung ist dabei ähnlich zu der vom Nachlicht. Deshalb verwende das Programm von [Nachtlicht.ino](#) und passe es den neuen Erfordernissen an.



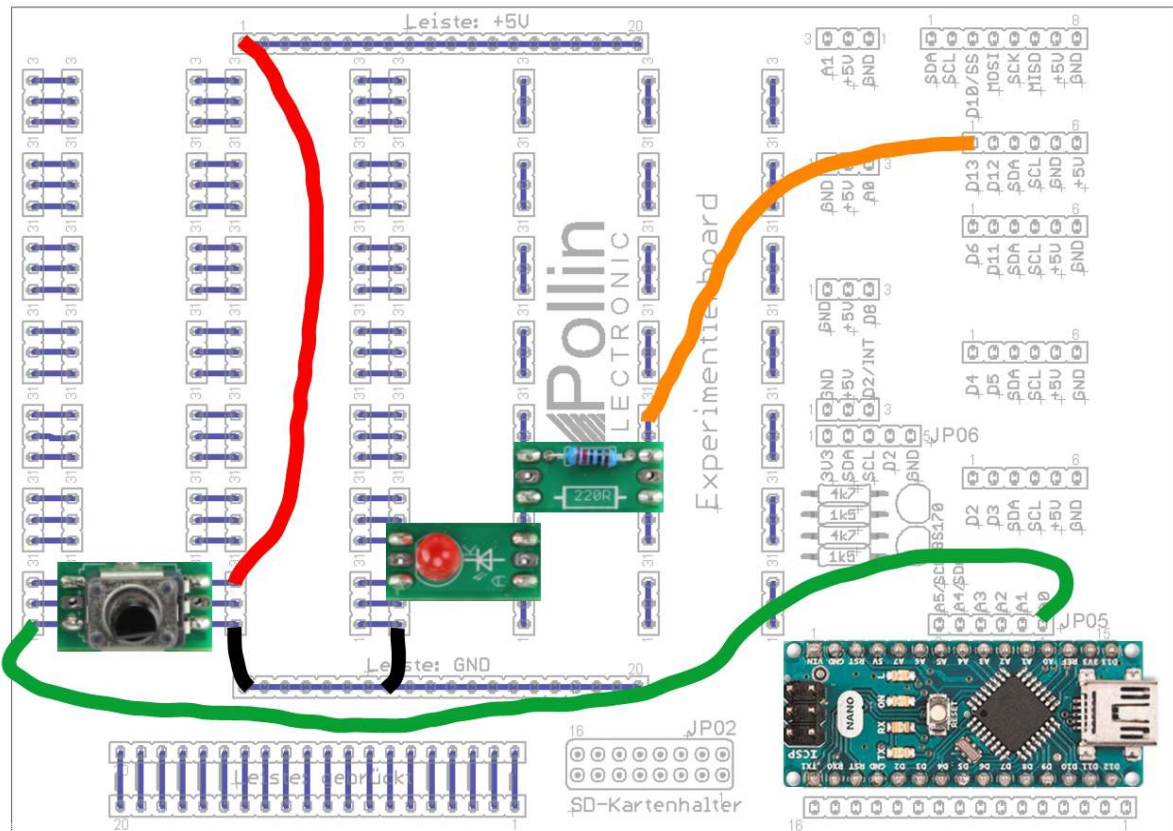
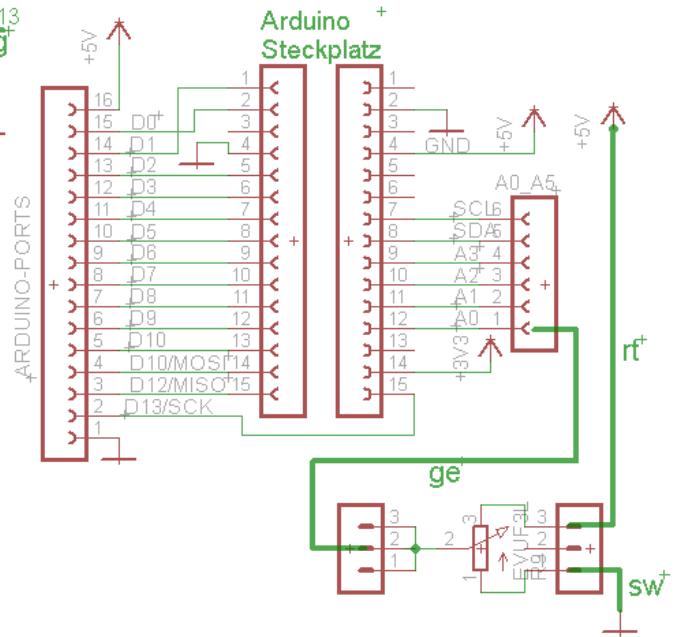
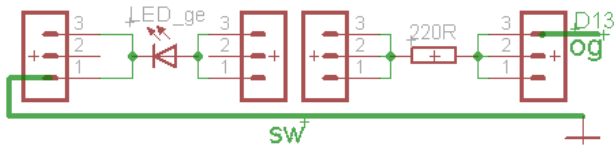
Der FSR-Sensor kann auf die Grundplatte aufgesteckt werden wie im Bild links angedeutet ist. Die Schaltung entspricht dann der, wie im Schaltplan oben dargestellt. Der eingeleseene Wert am Analogeingang des Arduino ist 1023, unbelastet und geht fast auf 0 bei voller

Belastung des Sensors. Deshalb kann zur Einstimmung in dieses Thema das Programm aus 4.2 verwendet werden. Der FSR ist hier jedoch an Pin A1 angeschlossen.

Die Aufgabenstellung in diesem Absatz ist, mit zunehmender Kraft, immer mehr LED's leuchten zu lassen. Probieren wir mal fünf Stück. Der Widerstand ändert sich. Bei einer bestimmten Schaltschwelle soll die erste LED einschalten. Zuerst ist festzulegen wie viele LED's, Dann können die einzelnen Schaltschwellen festgelegt werden, um dann beim Programmablauf diese zu überprüfen und gegebenenfalls korrigieren.

11.4. LED-Dimmen

Hier zunächst der Schaltplan:



Beachte die korrekte Platzierung des Poti:



11.4.1. einfaches Dimmen

Das Prinzip dahinter ist, die Gleichspannung, mit der eine LED normalerweise angesteuert wird, zu takten. Das heißt, die LED für einen kurzen Moment auszuschalten. Wenn die Zeit, in der die LED ausgeschaltet wird im Verhältnis zum Einschalten immer größer wird, dann leuchtet die LED immer schwächer. Allgemein wird dies als Puls Pausenverhältnis bezeichnet.

Werte=`analogRead`(Poti_pin) liefert einen Wert 0 ... 1023

Damit kann man z.B. ein `delay`(warte); von warte=0 ... 1023[ms] einstellen. Die LED blinkt entweder ganz schnell bis ganz langsam, je nach Stellung des Potentiometers.

Beispiel `potentiometer.ino`:

```
#define potiPin A0 //define analog pin0 an dem der mittlere Abgriff des Poti angeschlossen wird
#define ledPin 13 //define Ausgang, wo die LED angeschlossen ist
```

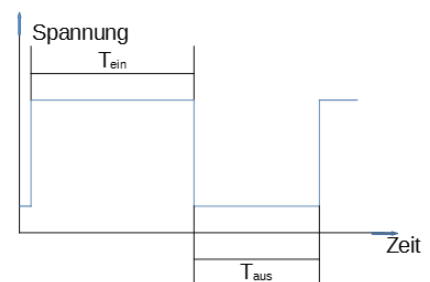
```
unsigned int val; // der Wert der Stellung des Potis wird in dieser Variable gespeichert.
```

```
void setup()
{
  pinMode(ledPin,OUTPUT); //LEDpin wird nun als Ausgang definiert
  Serial.begin(9600); //Set baud rate 9600
}
void loop()
{
  val = analogRead(potiPin); //
  digitalWrite(ledPin,HIGH); //LED ein
  delay(val); //warte bis zum abschalten
  digitalWrite(ledPin,LOW); //LED aus
  delay(val); //warte bis zum einschalten
  Serial.println(val); //zeige den aktuellen Poti wert
}
```

Das eigentliche Ziel aber war, die Helligkeit zu ändern. Was muss man ändern?

Bleibt der Wert von einem `delay`() konstant z.B. bei 1023 und nur einer wird durch das Poti variiert, dann lässt sich die Helligkeit ändern.

Ein Port kann entweder 0 „aus“ oder 1 „ein“ sein. Dann leuchtet eine angeschlossene LED oder sie ist aus. Wenn aber ein Rechtecksignal wie im rechten Bild erzeugt wird, indem man den Schaltvorgang an der LED sehr schnell macht, dann kann auch auf diese Art die Helligkeit der LED verändert werden. Je länger dabei der Pin eingeschaltet bleibt, desto heller leuchtet die LED.



Im obigen Beispiel ist val der Wert des Potis 0 ... 1023. Wenn nun

für Taus der Poti Wert verwendet wird, so erhält man bei einem Endwert des Poti 1023 und somit ein Tastverhältnis von 1:1. Beim anderen Endanschlag des Potentiometers 0, also leuchtet in diesem Fall die LED mit voller Helligkeit. Es ist also ein wenig experimentieren angesagt. Zumal die LED mehr blinkt, als permanent leuchtet. Denn die Frequenz, mit der die LED leuchtet berechnet sich aus dem Kehrwert der Summe von T_{ein} und T_{aus}. $Frequenz = 1 / (T_{ein} + T_{aus})$ Um dies zu vereinfachen, kann eine dem Arduino eigene Funktion der PWM benutzt werden.

Dabei wird bei einer festen Frequenz das Puls Pausenverhältnis verändert. Dazu wird der Befehl

analogWrite(pin, val) verwendet. Val hat dabei einen Wert von 0 ... 255 und dabei ist dann die LED ganz aus (0) oder eben voll angesteuert (255). Ansonsten kann val jeden beliebigen Wert annehmen. So ist bei einem Wert von val=100, der Port-Pin für 100 Zeiteinheiten eingeschaltet und für 255-100 = 155 Zeiteinheiten ausgeschaltet. Wobei die Zeiteinheiten nicht [ms] sind, denn die Grundfrequenz ist 490Hz und könnte bei Pin5 und Pin 6 auf 980Hz erhöht werden. Zu beachten gilt allerdings, dass diese Funktion nicht an jedem Pin nutzbar ist. Beim Arduino UNO, NANO und MINI nur an **Pin: 3, 5, 6, 9, 10 und 11 !**

11.4.2 dimmen mit PWM

Verwenden wir also am Beispiel [PWM_dimmung.ino](#) den Pin 11 statt Pin13, wie er in Beispiel 11.6. verwendet wurde.

Anzumerken ist weiter noch dass bisher die Funktion der for – Schleife noch nicht benutzt wurde.

```
for (val = 0 ; val <= 255 ; val += delta_val)
```

***= Multiplikation**

-= Subtraktion

:= Division

Die for Schleife wird dann verwendet, wenn genau bekannt ist, wie oft die Befehlsfolgen zu durchlaufen sind. Die Abbruchbedingung ist damit ein konkreter Zahlenwert.

Die allgemeine Definition einer for – Schleife schaut folgendermaßen aus:

for (Laufvariable **val** wird Startwert zugewiesen, hier=**0** ; **Abbruchbedingung** ; Laufvariable muss sich **ändern**). Dabei ist s nicht immer nur eine Addition, um den Wert zu ändern, auch Subtraktion Multiplikation oder Division wären möglich!

Die Schleife wird im oben gezeigten Fall $255 / \text{delta_val}$ mal durchlaufen. Wichtig sind die anschließenden geschweiften Klammern, denn nur diese Befehle, in dieser Klammer, werden bei jedem Schleifendurchlauf ausgeführt.

11.6.3 Dimmen durch steuern der PWM mit Poti ([PWM_dimmung_poti.ino](#))

ADC:	Bit -9	Bit -8	Bit -7	Bit -6	Bit -5	Bit -4	Bit -3	Bit -2	Bit -1	Bit 0
	1	1	0	0	1	1	0	1	1	1

Genauso, wie das Puls-Pausen-Verhältnis mit einem Poti gesteuert wurde, lässt sich auch die PWM steuern. Zu beachten gilt dabei allerdings, dass der Wert des

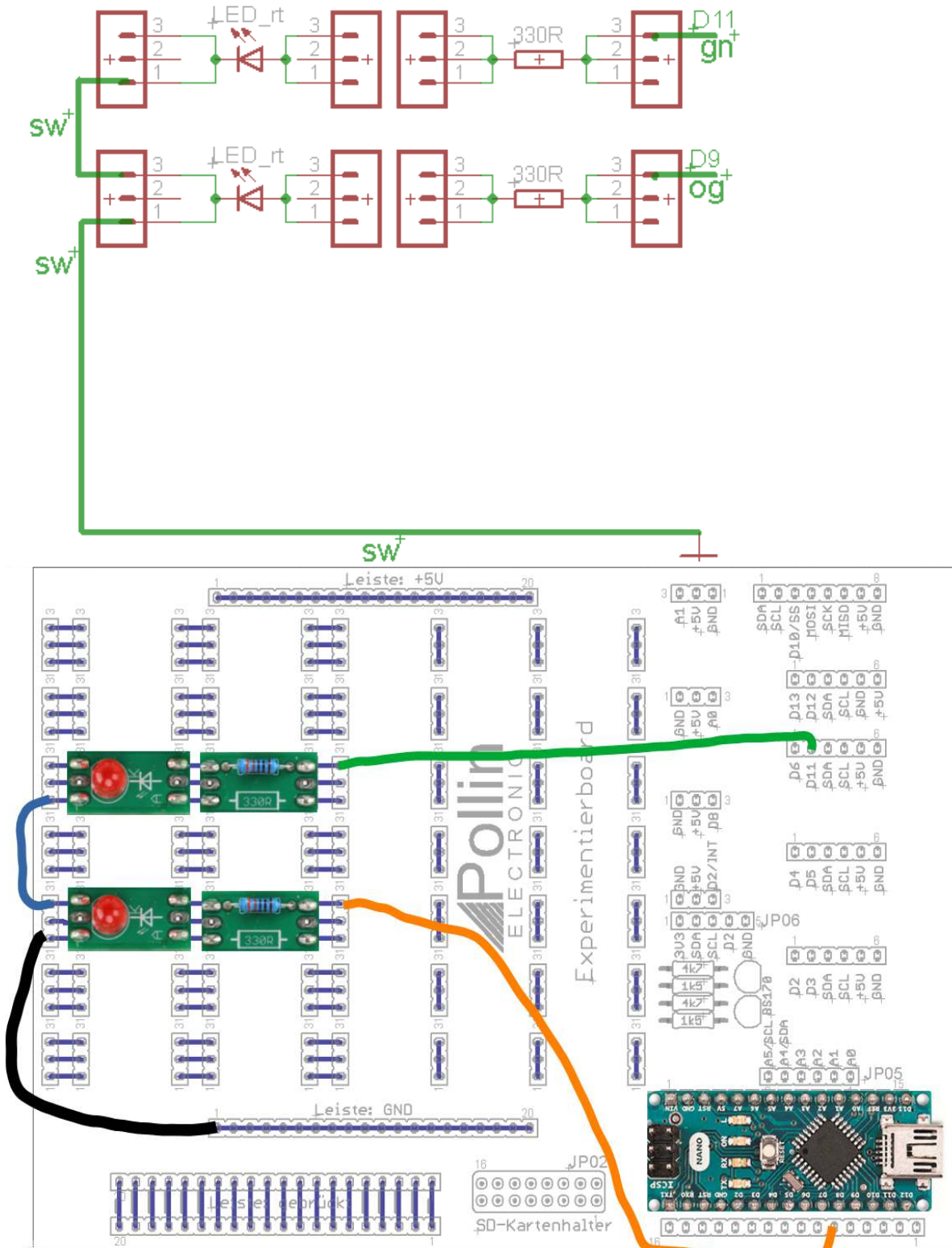
Poti 0... 1023 hat und die PWM nur Werte von 0... 254 erlaubt

Um nun den ADC-Wert in das PWM-Register schreiben zu können, muss der Wert durch vier geteilt werden. Dies geschieht durch eine Division oder durch zweimaligen Rechts-Shift! Aus Programmtechnischer Sicht ist der Schiebepfeil vorzuziehen, weil dies keine Rechenleistung des Prozessors verbraucht. In unserem Fall dürfte die Rechenzeit jedoch keine Rolle spielen.

PWM-Register:	Bit -7	Bit -6	Bit -5	Bit -4	Bit -3	Bit -2	Bit -1	Bit 0
	1	1	0	0	1	1	0	1

11.4.3. Wechselblinker am Andreaskreuz

An einem Bahnübergang blinkten früher zwei rote Lampen, eine wurde langsam immer heller, die andere im selben Maße dunkler. Im englischen nennt man dies Fading. Hier das Beispiel [Duo_fading.ino](#). Dies ist aber nur eine kleine Weiterentwicklung von [PWM_dimmung.ino](#);



11.4.5. Leuchtturm

Das Beispiel [PWM_Leuchtturm.ino](#) ist ein wenig komplexer, was die Programmierung angeht. Will man eine Liste von Werten anlegen, geht das natürlich mit wert1, wert2, wert3 usw. Um sich jedoch die Arbeit ein wenig zu vereinfachen, gibt es in jeder Programmiersprache die Möglichkeit solche Listen anzulegen und diese mit nur einem Variablennamen anzusprechen. Es ist eine Reihe von Variablenwerte. In der Programmierung werden überwiegend englische Begriffe benutzt und eine Reihe heißt unter Programmierern: **Array**.

Im einfachen Fall besteht so ein Array aus wenigen Werten. Ein Beispiel wäre die PWM-Ausgänge des Arduino über so ein array anzusprechen. Die Definition dafür würde folgendermaßen aussehen:

```
Variablentyp Variable [anzahl]; char pwm_out[6]={3, 5, 6, 9, 10, 11 };
```

In der geschweiften Klammer stehen dann die Werte der Pins, an denen ein PWM-Signal ausgegeben werden kann. Beim Arduino uno und beim Arduino Nano sind nur diese Pins für eine PWM verfügbar.

Der Variablentyp ist ein char, byte oder uint8_t weil der Wert, dem die Variable haben darf, im Bereich 0 ... 255 liegen muss. Größere Werte sind nicht erlaubt. **Beim Arduino** ist es egal, ob eine 8Bit Variable als char, byte oder uint8_t definiert ist. In jedem Fall ist es eine vorzeichenlose Zahl. Nun brauchen wir noch ein zweites Array, nämlich eines, in dem Werte gespeichert sind, die an den PWM-Ports ausgegeben werden sollen: `int8_t state [6];` // reserviere 6 Speicherstellen für 8-Bit Zahlen;

Will man nun einer Variablen in diesem Array einen Wert zuweisen, geschieht dies z.B. so: `state[0] = 200;`

Wir wissen, jeder Rechner beginnt mit 0 zu zählen, also weisen wir mit `state[0] = 200;` der ersten Variablen im Array den Wert 200 zu. Die Befehlszeile `state[5]=127;` weist der vierten Variablen im Array den Wert 127 zu. Wenn man einen Wert aus einem Array in eine Variable lesen möchte geschieht dies so: `variable = state[x];`

wobei x nur ein Wert von 0 ... 4 sein darf.

Wenn nun gefordert ist, dass die Ausgänge bestimmte Werte annehmen sollen, kann man diese in mehreren Reihen definieren. Das ist dann ein zweidimensionales Array, ein Array von einem Array:

```
int8_t state [3] [6];
```

Dabei wird eine zweite Zahl vor der bereits definierten Array Struktur in eckigen Klammern eingefügt. Nun bedeutet das im oberen Beispiel, dass es drei Arrays gibt, mit je sechs PWM-Zuständen, für die PWM-Ausgänge des Arduino.

Der Zugriff auf einen Wert im Array, geschieht nun, dass genau festgelegt sein muss, welcher Wert in welchem Array gemeint ist. Möchte der Programmierer auf den vierten Wert im zweiten Array zugreifen, dann geschieht das folgendermaßen: `state[1][3]=200;` 1 bedeutet die zweite Reihe und drei bedeutet, das vierte Element, weil wir ja bei 0 zu zählen beginnen. Möchte man nun den letzten Wert aus dem dritten array am Pin 6 des Arduino ausgeben, so kann dies folgendermaßen realisiert werden: `analogWrite(pwm_out[2], state[2][5]);`

Beim Leuchtturm Projekt wird wie schon erwähnt, eine Matrix verwendet, um die Zustände vorzudefinieren:

```
state [Zeile] [Stelle]
int state [10] [6] = {
    // Stelle0, Stelle1, Stelle2, Stelle3, Stelle4, Stelle5
    { aus,    aus,    aus,    aus,    aus,    aus}, // Zeile 0
    { halb,  aus,    aus,    aus,    aus,    aus}, // Zeile 1
    { halb,  halb,  aus,    aus,    aus,    aus}, // Zeile 2
    { aus,   aus,    aus,    aus,    aus,    aus}, // Zeile 3
    { halb,  aus,    aus,    aus,    aus,    aus}, // Zeile 4
    { halb,  halb,  aus,    aus,    aus,    aus}, // Zeile 5
    { aus,   aus,    aus,    aus,    aus,    aus}, // Zeile 6
    { halb,  aus,    aus,    aus,    aus,    aus}, // Zeile 7
    { halb,  halb,  aus,    aus,    aus,    aus}, // Zeile 8
    { halb,  aus,    aus,    aus,    aus,    aus}, // Zeile 9
};
```

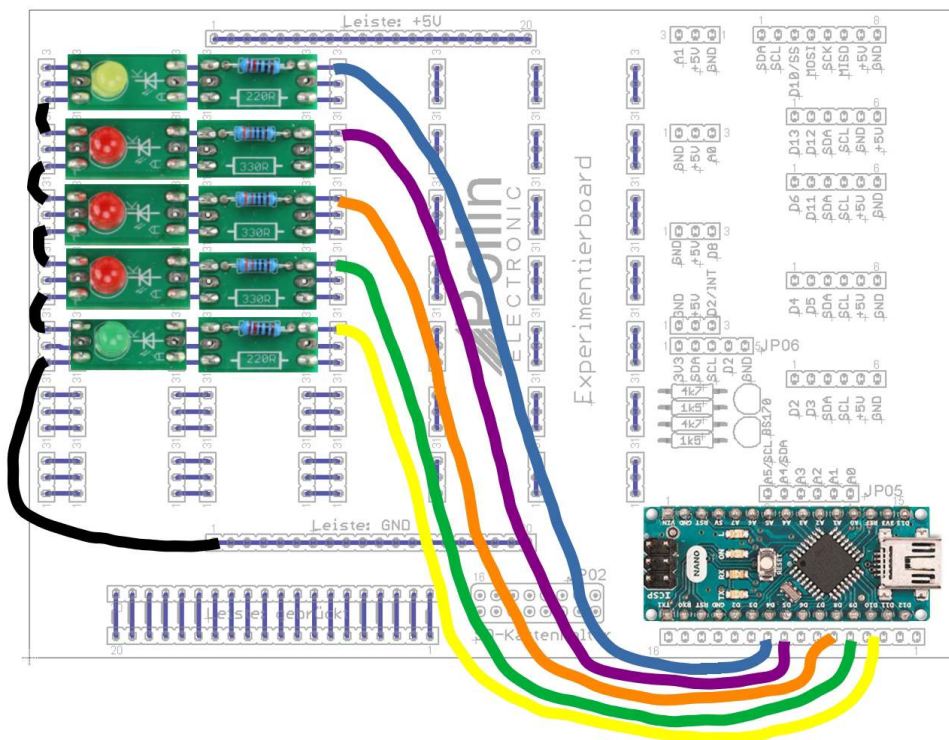
Die Werte für aus, halb und ein sind in den folgenden Zeilen festgelegt:

```
const int8_t aus=0;
const int8_t halb=60;
const int8_t ein=255;
```

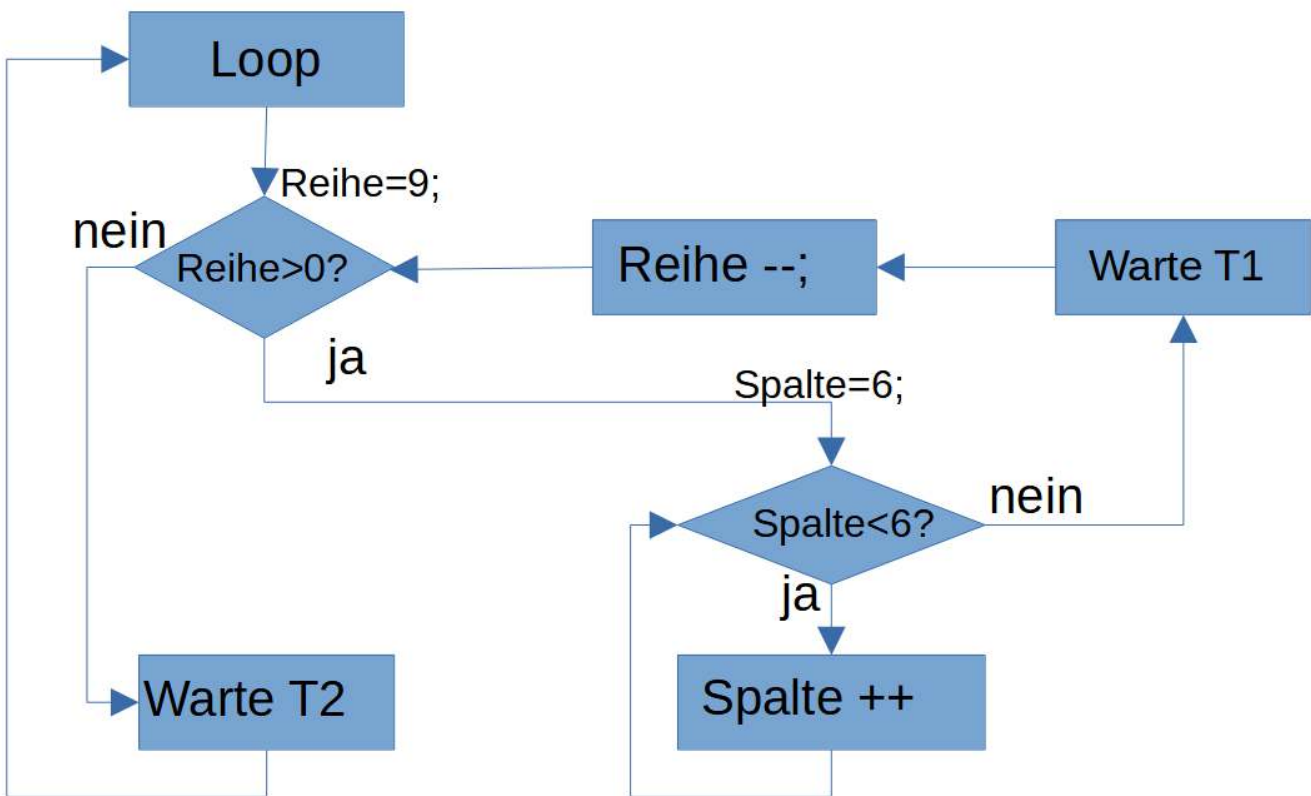
Diese Wertzuweisungen müssen im Programmcode natürlich vor dem Array stehen, damit der Compiler keine Fehlermeldung erzeugt.

Nun lassen sich die Werte aus dem zweidimensionalen Array aus **integer** Werten, bestehend aus **6** Spalten und **10** Zeilen bestimmen. Der erste Wert, der zweiten Zeile wird folgendermaßen ausgelesen wert = state[1][0], wert ist damit 60!

Unten sehen Sie das Foto, wie die LED's zu verschalten sind, damit das Programm PWM_Leuchtturm.ino auch fehlerfrei laufen kann:



die verschachtelte for – Schleife:



Die Funktionsweise einer for-Schleife wurde hier bereits behandelt und vorgestellt.

Im Programmbeispiel von [PWM_Leuchtturm.ino](#) wird eine for-Schleife in einer for-Schleife aufgerufen:

```
for(Reihe=9; Reihe >=0; Reihe--)
{
  for (Spalte=0; Spalte < 6; Spalte++)
  { analogWrite(PWM_out[Spalte], state[Reihe][Spalte]); }
  delay(T1); // Wartezeit zwischen den einzelnen Reihen
}
```

delay(T2); // wartezeit, bis die for-Schleife mit Reihe beginnt, also bis Programm von Neuem beginnt.

Dabei wird mit jedem Wert für Reihe, also von 9 ... 0 (insgesamt zehnmal), die innere for_schleife aufgerufen.

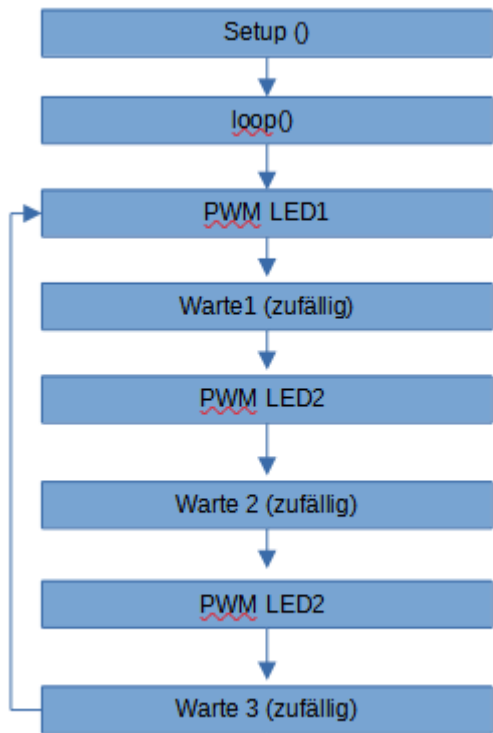
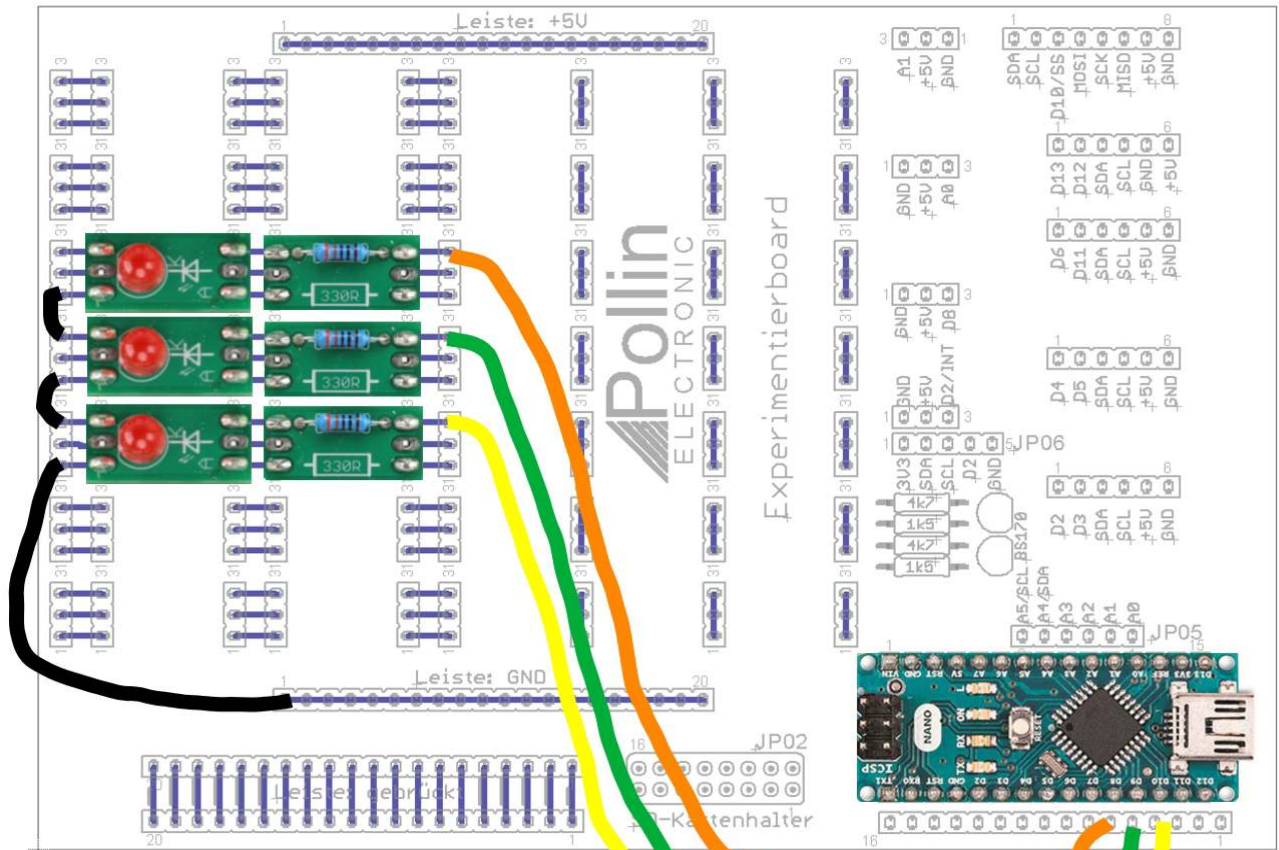
Also wenn Reihe=9 ist, dann wird die komplette for-Schleife bei Spalte von 0 ... 5 abgearbeitet;

Dann wird der Wert von Reihe erniedrigt, und nun wird wieder die komplette for-Schleife mit Spalte von 0 ... 5 abgearbeitet;

Dies geht so lange bis Reihe=0 und nach dem kompletten Durchlauf der for-Schleife mit Spalte, wird zuerst T2 Millisekunden gewartet, bis der Durchlauf mit Reihe=9 wieder von vorne beginnt.

11.4.6. Kerzenlicht.ino

Die PWM Werte werden auch hier wieder mit random() erzeugt. Es müsste sonst alles bekannt sein. Deshalb ist das Beispiel nur als Ergänzung und dazu gedacht die Werte für random() beliebig zu ändern, um zu sehen, was dabei dann passiert. Die Verkabelung ist wie beim Leuchtturm, ohne die gelbe und grüne LED.

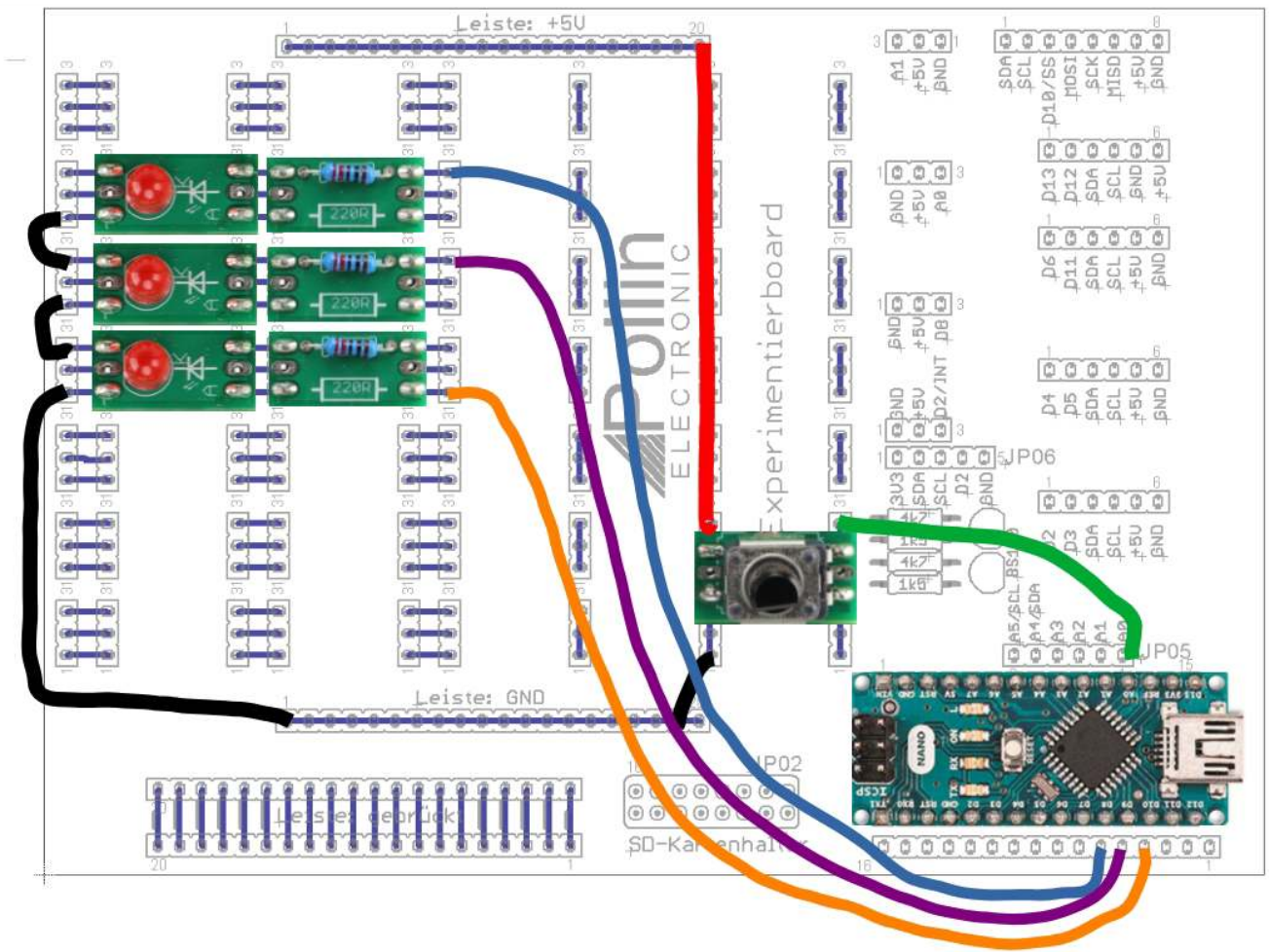
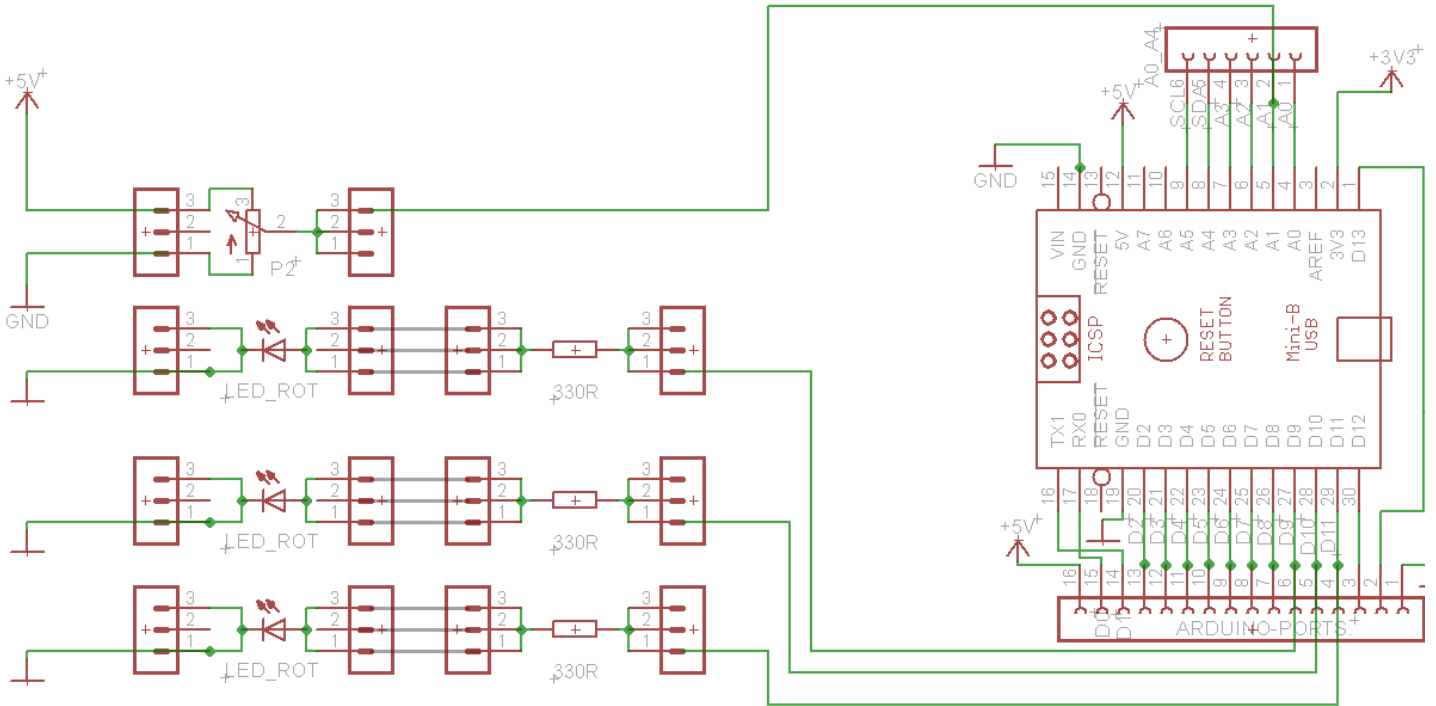


Am besten wirkt der Effekt, wenn die LED's nahe zusammen stehen und darüber eine aufgeraute Plexiglasscheibe gelegt wird. Zudem könnte auch eine gelbe LED noch einen zusätzlichen Effekt liefern.

Aus dem Flußdiagramm soll deutlich werden wie ein Kerzenlicht simuliert wird. Es wird eine PWM für die LED eingestellt und nach jedem Durchlauf des Programms wieder verändert. Dabei ist der Durchlauf bestimmt aus den drei Wartezeiten. Es ist auch möglich die Wartezeiten genau wie die PWM-Werte nur in engen Grenzen zu ändern, oder vielleicht auch eine LED dauerhaft leuchten zu lassen.

Der Phantasie und Experimentierfreude sollten keine Grenzen gesetzt sein.

11.4.7. Stroboskop

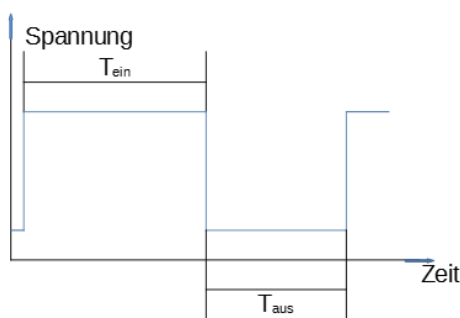


Bei einem Stroboskop möchte man, dass die LED möglichst hell leuchtet. Dazu wird der Einzustand konstant belassen. Geändert wird die Zeit für den „Aus“- Zustand. Dies ergibt dann

auch wieder eine Art PWM. Durch die Variation der „Aus“-Zeit ändert sich die Wiederholffrequenz für das Blinken. Wenn die Frequenz der LED und die des angestrahlten Objektes identisch sind, dann erscheint das Objekt als stehend. Dies ist die Schwierigkeit in diesem Projekt. Es sollte praktisch vorher schon bekannt sein, mit welcher Frequenz sich das Objekt dreht, oder wie bei einem Wasserhahn, wie oft dieser in der Sekunde tropft.

Der Unterschied zu einer PWM ist, dass bei einer PWM die Frequenz konstant ist und sich nur das Verhältnis von eingeschaltet zu ausgeschaltet ändert. Je länger eingeschaltet, desto heller leuchtet die LED. Die Funktion bei einem Stroboskop ist, dass die Wiederholffrequenz mit der eine LED angesteuert wird variiert werden kann. Wozu wird dies benutzt? Eine Anwendung ist das Fotografieren. Zum Beispiel kann man einen tropfenden Wasserhahn mit einem Stroboskop beleuchten. Wenn die Frequenz mit der der Wasserhahn tropft identisch ist mit der Blinkfrequenz des Stroboskop's, dann erscheint der Wassertropfen als stehendes Bild. Bei einem Motor kann auf diese Art die Drehfrequenz bestimmt werden. Denn wenn der Motor als stehend empfunden wird und die Blinkfrequenz des Stroboskop's bekannt ist, dann weiß man auch genau die Drehzahl.

Der Schaltungsaufbau kann direkt von 11.6. übernommen werden. Dazu können weitere LEDs dazugeschaltet werden, wie beim Kerzenlicht im vorigen Abschnitt 11.6.6. So kann u.U. die Helligkeit des abgestrahlten Lichts erhöht werden. Links im Bild ist die Verdrahtung für das Beispiel: [stroboskop.ino](#) dar-gestellt.



Im linken Bild ist der Impuls zu erkennen, den wir mit dem Arduino erzeugen müssen. Dazu ist es notwendig zwei Zeiten als Variable festzulegen. Das eine ist die Zeit, wie lange der Impuls High ist; nennen wir in pulse. Dieser Wert ist während der Laufzeit konstant, muss aber als define festgelegt werden, damit wir ihn anpassen können. Denn je länger dieser Impuls ist, desto heller leuchtet die LED. Allerdings ist durch dessen Länge auch indirekt festgelegt

wie schnell die Wiederholffrequenz ist. Deshalb verwenden wir mehrere LED's um eine größere Leuchtkraft zu erzielen.

`PORTB=0x1E;` damit lassen sich die LED's gleichzeitig einschalten.

Grundgedanke der Funktion [stroboskop.ino](#) ist, mit dem Poti die Wiederholzeit zu variieren.

Oder man kann, wie im Beispiel [stroboskop2.ino](#) die Puls-zeit und die Pausenzeit variieren. Eine Erweiterung der Funktion ist, mit einem zweiten Poti die Puls-zeit zu ändern. Neu angeschlossen wird das zweite Poti an A1 angeschlossen. Damit wäre es möglich die Helligkeit und die Wiederholffrequenz zusätzlich zu erhöhen und dabei die LED-Helligkeit auf ein Minimum zu reduzieren.

Der neue Befehl dabei ist **map**. Dieser ist unter Punkt 11.7.1. ausführlich erklärt. Die Funktion **map** liefert als Ergebnis einen Wert, der von einem größeren Wertebereich in einen kleineren umgerechnet werden kann. Ein Poti liefert einen Wertebereich von 0 ... 1023.

Um nun ganzzahlige Pulse von 1 .. 10ms zu erzeugen verwendet man nachfolgenden Befehl:

```
pulse = map(analogRead(PULSEPOTI), 5, 1000, pulse_max, pulse_min); // Erzeuge pulse  
1ms ... 10ms
```

aber betrachten wir nur den Befehl **map(variable , val1 , val2, val3, val4)**

variable: beschreibt dabei den Wert, den wir über das Poti
einlesen **Wertebereich Eingang:** **val1** : minimal Wert der von variable
berücksichtigt werden soll

val2 : maximaler Wert der von variable berücksichtigt
werden soll

val3 : minimaler gewünschter Wert der Ausgabe
gewünschter Wertebereich: **val4** : maximaler gewünschter Wert der Ausgabe

map rechnet also die aktuelle Variable so um, dass es eine lineare Abhängigkeit gibt zwischen dem Wert der mit dem Poti eingestellt wird und dem Wertebereich, den wir für die Ausgabe uns wünschen.

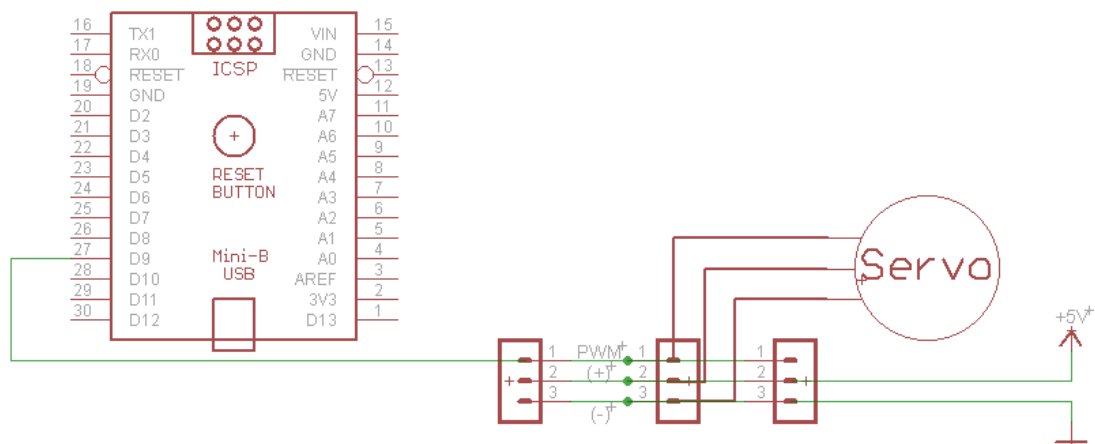
11.5. Servo (Artikel 820 570) ansteuern

Für die Abbildungen wurde ein Servo mit einem Metallgetriebe demontiert.

Ein Servomotor ist ein Gleichstrommotor mit Getriebe, der ein Spannungssignal direkt in eine Drehbewegung umwandelt. Dabei kann jedoch der Motor maximal eine Drehbewegung von 360° durchführen. Das besondere liegt in der Ansteuerung. Wie in der linken Abbildung zu sehen, ist der Winkel vom Puls Pausenverhältnis der anliegenden Spannung abhängig. Der Servomotor benötigt zur korrekten Ansteuerung ein sich alle 20 Millisekunden wiederholendes Signal. Es besteht wie vom Stroboskop bekannt aus einem HIGH-Impuls, der in diesem Fall zwischen 1 und 2 Millisekunden lang ist, und einem LOW-Impuls. Die Dauer des HIGH-Impulses bestimmt den Zielwinkel (normalerweise von 0 bis 180°). 1ms entspricht das bei einem Winkel von 0° und 2ms entspricht einem Winkel von 180° In der Arduino-Software kann hierfür der Befehl `delayMicroseconds(x);` verwenden:

```
ON = 1500;  
digitalWrite(myServo,HIGH);  
delayMicroseconds(ON);  
digitalWrite(myServo,LOW);  
delayMicroseconds(20000-ON);
```

Der Servo braucht eine gewisse Zeit, sich in die gewünschte Position zu bewegen. Daher muss das Signal eben so lange wiederholt werden, bis der Servomotor in dieser Position ist. Also ausprobieren, wie lange das ist und eben so lange dann eine Schleife durchlaufen lassen.



Servomotoren überprüfen ihren eigenen Stellwinkel mit einem eingebauten Potentiometer. Es könnte aber durchaus passieren, dass eine gewünschte Position nicht ganz erreicht wird. Der Servo korrigiert sich dann kontinuierlich selbst, was zu einer permanenten Bewegung des Servo führen kann. Um das zu verhindern, sollte der Impuls nach Erreichen des gewünschten Winkels auf LOW geschaltet werden. Aber einfacher ist es gleich die für den Arduino erhältliche Servo Bibliothek zu verwenden. Damit spart man sich Zeit und Ärger. Trotzdem kann man ja den Servo manuell programmieren um zu sehen, wie viel Vorteile eine Bibliothek hat.

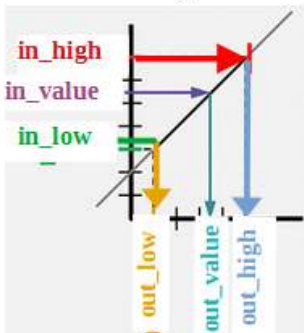
In Beispiel1 [servomotor1.ino](#) ist nur der Servomotor (SM) angeschlossen. Das Programm bewegt den Motor nur von 0 auf 180° und wieder zurück.

11.5.1. Servo mit Poti ansteuern

Mit dem Beispiel [servomotor2.ino](#) wird der Motor auf die Stellung bewegt, die ich mit dem Poti anwähle.

Bei diesem eigentlich sehr einfachen Beispielen verwenden wir wieder die Funktion `map()`: `pos = map(pos, 0, 1023, 0, 180)`; damit werden die Werte des Potis 0... 1023 in die für die Ansteuerung des Motors notwendigen Wertebereich 0...180 umgerechnet, ohne sich um die mathematischen Probleme zu kümmern.

Am besten kann man sich die Funktionsweise der Funktion `map` mit dem nebenstehenden Schema vorstellen.

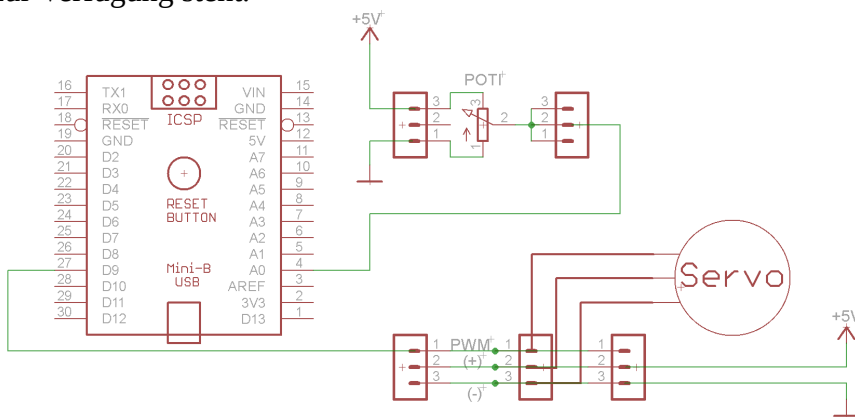


Die Funktion `map` wandelt den Wertebereich von `in_low` ... `in_high` über die eingezeichnete Gerade in den Wertebereich von `out_low` ... `out_high` um.

Um die Geradengleichung bzw. die Steigung der Geraden braucht sich der Programmierer nicht kümmern. Es ist nur wichtig sich die Belegung der Variablen beim Funktionsaufruf von `map()` zu merken: `out_value = map (in_value, in_low, in_high, out_low, out_high)`;

dann erhält man ganz automatisch den richtigen Wert der Variablen, für den korrekten weiteren Programmablauf.

Die Funktion `analogWrite(pin, value)` bietet dabei belegt `value` die Werte von 0 (ausgeschaltet) bis 255 (voll eingeschaltet; Allerdings gilt hierbei zu beachten, dass diese Funktion nicht an jedem Pin zur Verfügung steht.



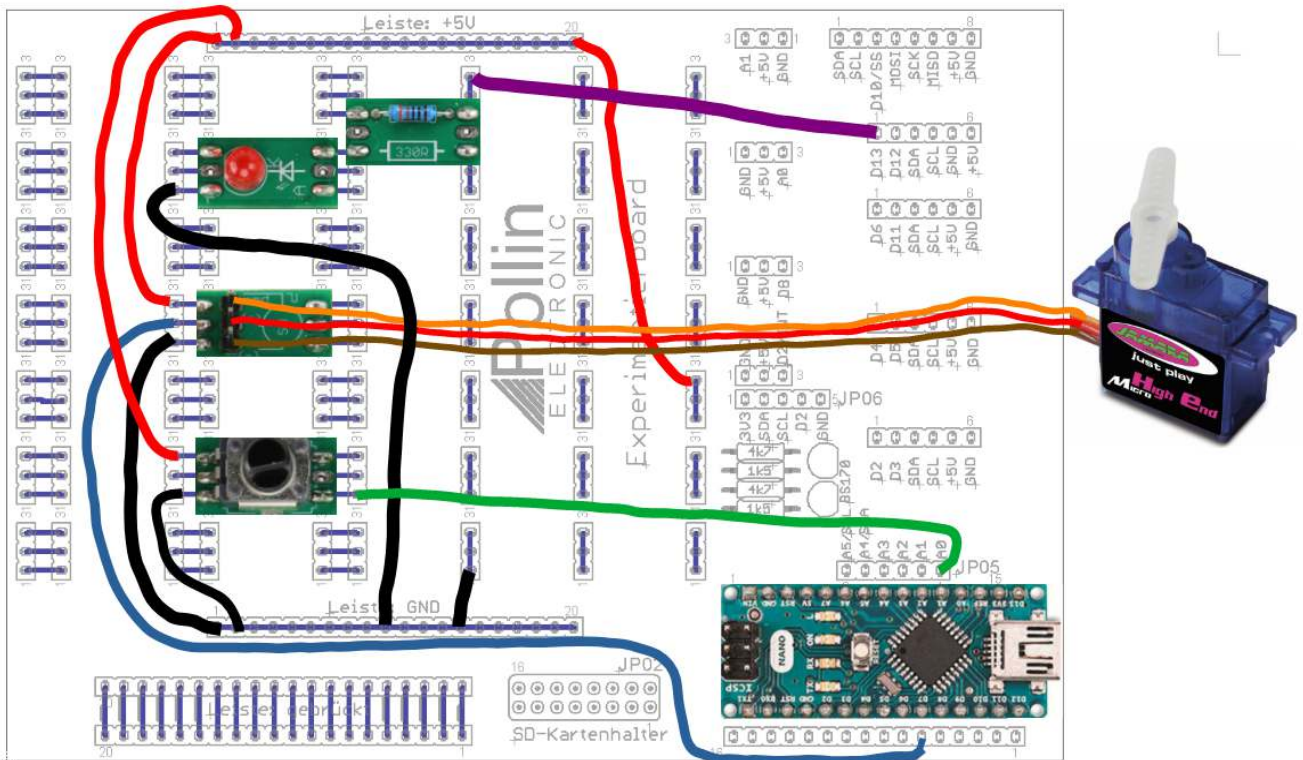
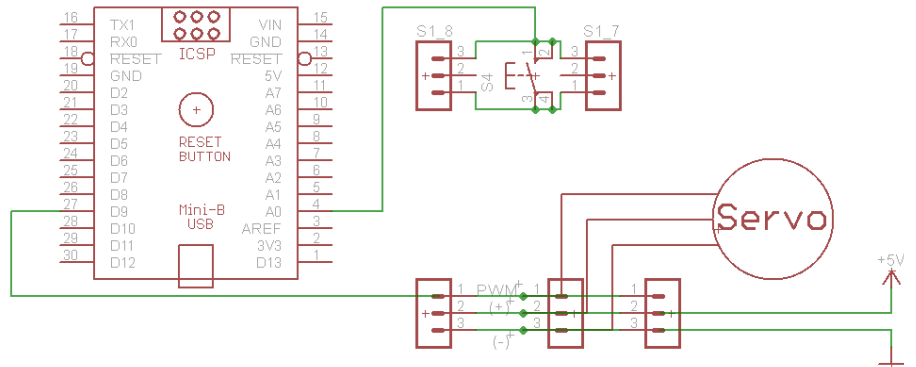
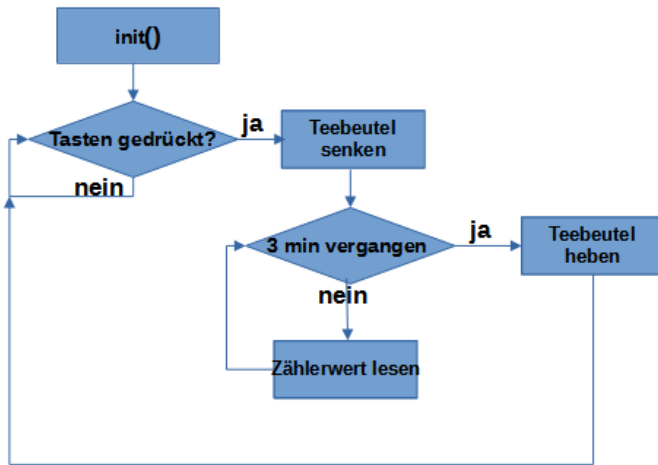
Im Beispiel2 [servomotor2.ino](#) wird der Servo (SM) mit einem Poti gesteuert. Der Wert des Poti soll in einen Drehwinkel umgesetzt werden.

Aufgabe:

Ein Programmbeispiel könnte so programmiert werden, dass während der Änderung des Potiwertes, eine LED leuchtet und somit eine Warnlampe sein für die Bewegung des Servo. Der Aufbau kann dem Beispiel 11.7.2. entnommen werden.

11.5.2. Teetimer

Die Aufgabe ist, drei Minuten nach dem Tastendruck, hebt sich der Arm des Servomotors. Damit soll der Teebeutel aus einem Glas entnommen werden. Als Basis wurde [servomotor1.ino](#) ist ein wenig abgewandelt in [teetimer.ino](#).



11.5.3. der NTC (NTC-10k Artikelnummer 220757)

NTC bedeutet negativer Temperaturkoeffizient. Damit steigt mit fallender Temperatur der Widerstandswert und umgekehrt. Früher benutzte man noch einen Parallelwiderstand, um die Kennlinie des Thermistors ein wenig mehr zu linearisieren. Heute übernimmt dies die Rechenleistung des Mikrocontrollers. Die Kurve des Widerstandsverlaufs gleicht einer e-Funktion. Deshalb benötigt man auch die Logarithmus Funktion des Arduino. Die mathematischen Zusammenhänge sind etwas kompliziert, so dass wir hier ein Beispiel von adafruit verwenden: <https://learn.adafruit.com/thermistor/using-a-thermistor>

Der Servo lässt sich zwischen 0 und 180° bewegen. Dadurch kann man durchaus ein nützliches Thermometer bauen. Der Temperaturbereich könnte liegen von -10 ... 30 °C. Das sind 40°C. Nehmen wir den Servo und bewegen ihn nur 160°, dann haben wir vier Winkelgrade um 1°C anzuzeigen. Das ist eine gute Auflösung und dürfte auch eine leicht zu zeichnende Skala ergeben.

Als Annäherung kann man auch eine Tabelle mit Werten verwenden und diese linear interpolieren. Damit erpart man sich die komplizierte Formel mit der sogenannten „Steinhart-Hart-Gleichung“. Diese wurde 1968 veröffentlicht und für die Temperaturmessung in den Ozeanen verwendet. Damit wurde eine gute Näherung erzielt, den Widerstandswert in eine Temperatur umzurechnen. Die Formel benötigt allerdings mehrere Parameter, die entweder aus dem Datenblatt des Sensors oder aus einer Messung zu berechnen sind.

Eine kleine Erleichterung ist die Verwendung der Beta-Formel. Diese ist eine Näherung für den Temperaturbereich bis 50°C. Bei höheren Temperaturen oder bei Temperaturen im Minus-Bereich steigt dann der Messfehler an. Leider ist im Datenblatt kein anderer Wert für B angegeben. Manchmal gibt es einen zweiten Wert. Dann müsste das Programm so ergänzt werden, dass bei einer niedrigeren Temperatur eine andere Formel angewandt wird. Hier geht es jedoch nicht um besondere Genauigkeit, sondern nur, um das Messprinzip vorzustellen.

Als Näherungsformel zur Bestimmung des Widerstandswertes gilt:

$$R_T = R_N * e^{B*(1/T - 1/T_N)}$$

R_T : aktueller Widerstandswert bei bestimmter Temperatur

R_N : Widerstand des NTC bei Raumtemperatur

B: Betawert des NTC aus dem Datenblatt: hier $B_{25/50}=3380$ (Artikelnummer NTC: 220 757)

T: aktuell gemessene Temperatur

T_N : Raumtemperatur 25°C in Kelvin, also 298,15 K

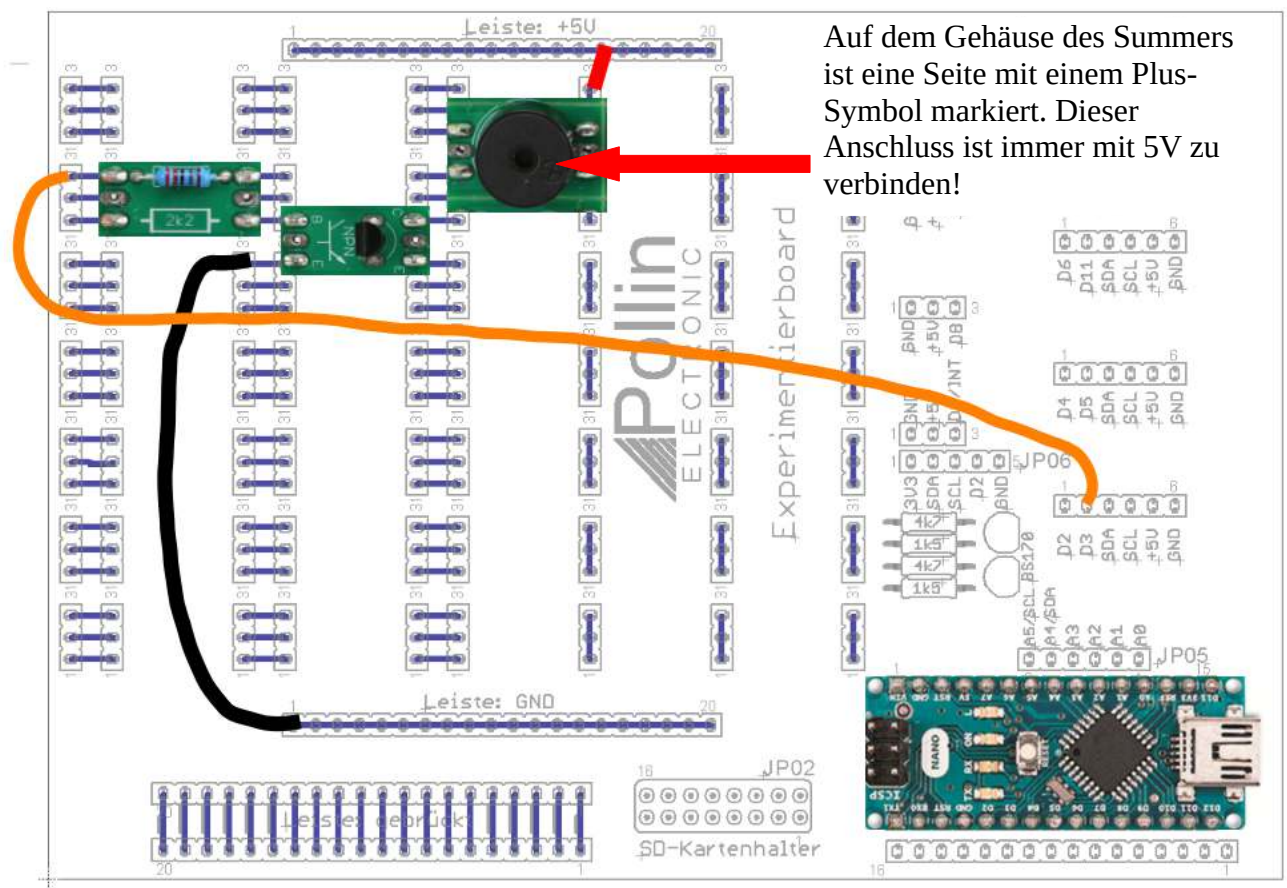
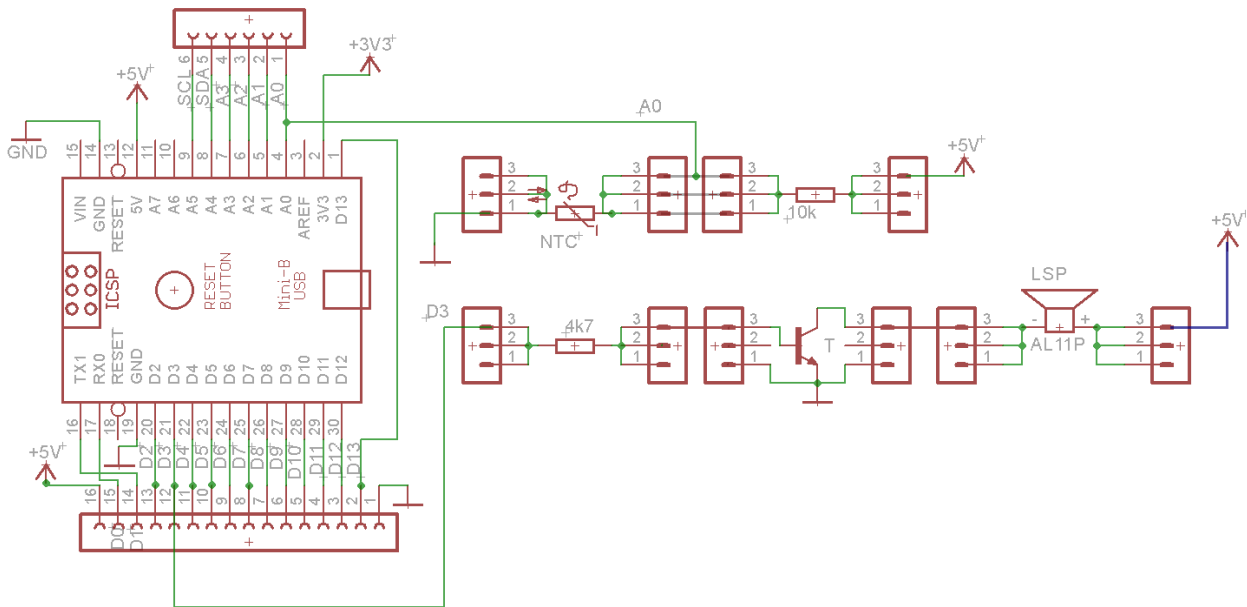
roß.

Das Beispiel [NTC_bsp1.ino](#) ist ein Beispiel die hier gezeigte Theorie in ein Beispielprogramm umzusetzen.

Dabei wird am Pin A0 die Spannung gemessen, die am NTC-Widerstand abfälltgr Vorwiderstand von 10k und der NTC bilden einen Spannungsteiler. Bei Raumtemperatur sind beide Widerstände in

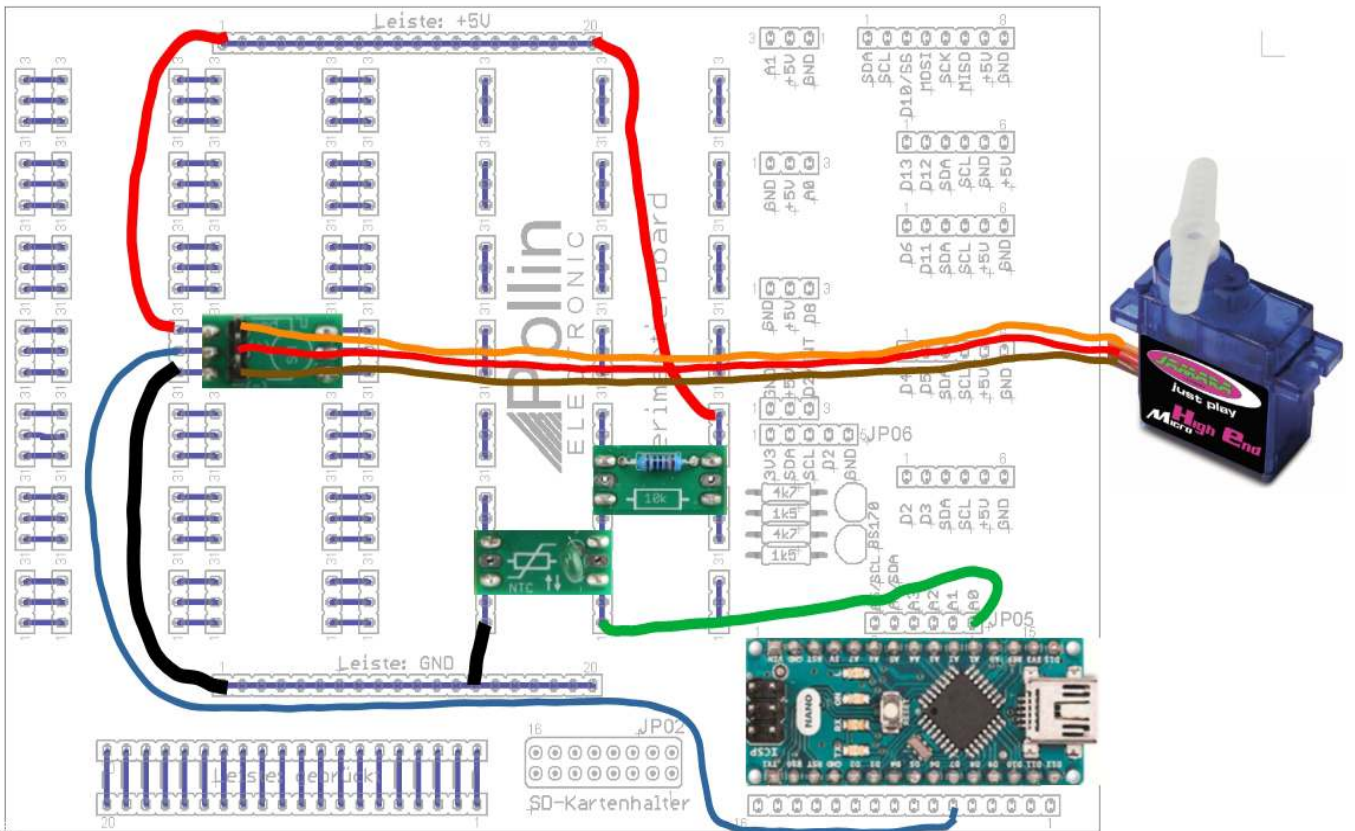
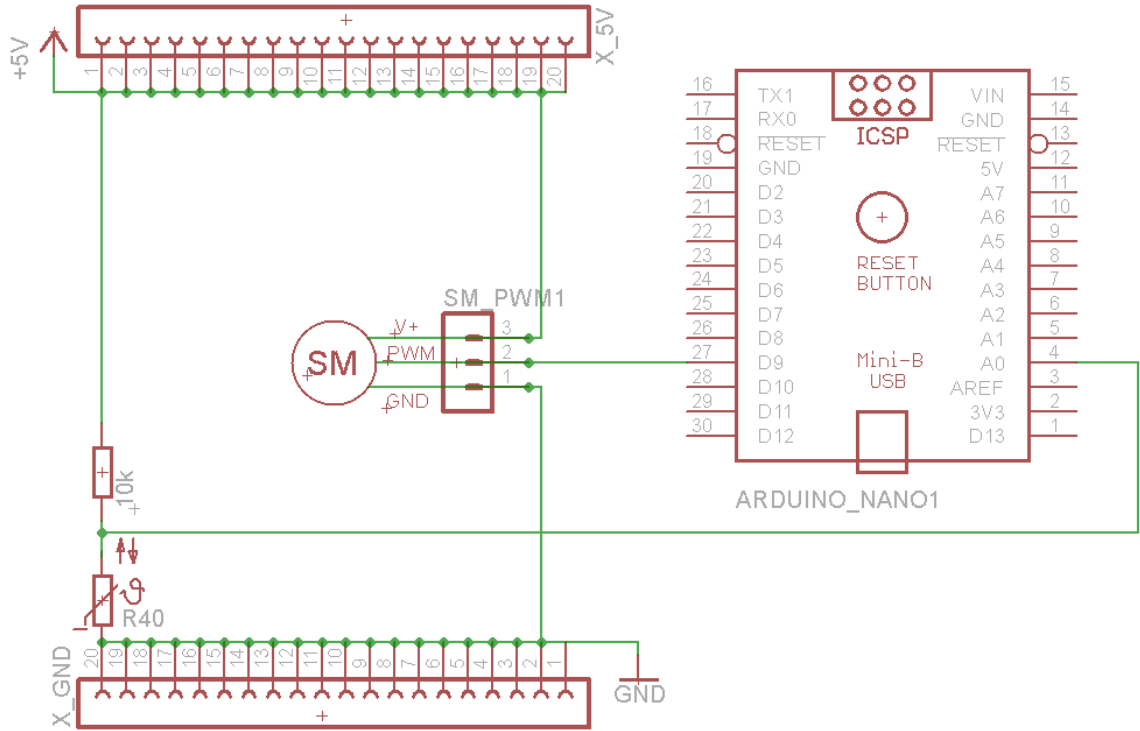
etwa gleich groß. Bei zunehmender Temperatur wird der Widerstand kleiner. Somit wird an A0 eine kleiner Spannung gemessen. Mit sinkender Temperatur, wird die Spannung größer.

Ergänze das Beispiel, um einen Signalton, wenn die Spannung an A0 einen von Dir bestimmten Wert überschreitet oder unterschreitet. Die Verdrahtung ist nachfolgend dargestellt. Dabei ist Pin D9, der Ausgabepin. Du kannst dabei eine PWM erzeugen mit `analogWrite()`, oder mit `tone()` und `notone()` arbeiten, oder aber auch mit `digitalWrite(D9,HIGH)`; `delay()` und `digitalWrite(D9,LOW)` in einer Schleife.



Auf dem Gehäuse des Summers ist eine Seite mit einem Plus-Symbol markiert. Dieser Anschluss ist immer mit 5V zu verbinden!

11.5.4. Temperaturmessung mit Servo anzeigen



Das Beispiel [Servomotor2.ino](#) kann dazu als Grundlage benutzt werden. Nach dem Lesen des Analogwertes ist nur die Temperatur aus dem Messwert zu berechnen. Die Parameter der Funktion `map()` müssen dann noch angepaßt werden.

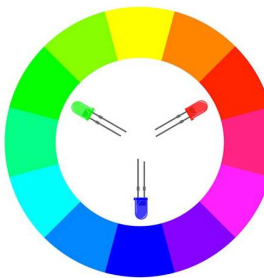
11.6. RGB LED

11.6.1. Grundlegendes



Eine analoge RGB-LED ist aufgebaut aus drei LED's und wie der Name schon sagt, sind dies: eine **Rote**, eine **Blaue** und eine **Grüne** LED in einem gemeinsamen Gehäuse.

Gut zu erkennen ist auch, wie durch Farbmischung der einzelnen LED's unterschiedliche Farbwerte erzeugt werden können. Im Ersten Programmbeispiel wird auch darauf zurückgegriffen. Durch Einschalten von immer zwei LED's kann violett, gelb und türkis erzeugt werden. Wenn alle drei LED's aktiviert werden, erhält man weiß.



Im linken Bild ist ein Farbkreis dargestellt, wie sich die Farben mischen lassen. Mit einer PWM-Ansteuerung der einzelnen LED's lassen sich die die Farben in Stufen einstellen. Theoretisch kann man die Spannung an einer LED in 256 Spannungswerte einteilen. Da jedoch die PWM nicht besonders stabil ist, wird es bei jeder LED zu Farbschwankungen kommen. Es ist auch für einen ungeübten schwer einzelne Farbnuancen zu unterscheiden. Aber das Prinzip ist das gleiche wie bei einem LED-Fernseher. Mit dieser Methode lassen sich theoretisch über 16 millionen Farben darstellen. Sobald aber die dritte LED mit angeschaltet wird, wird die Farbe schwächer und es ergibt sich ein zunehmender Weißanteil. Sind alle

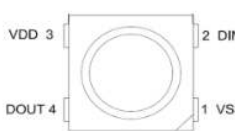
drei LED's voll angeschaltet, ergibt sich ein helles weißes Licht. Ähnlich funktioniert eine LED Lampe. Da werden auch „künstlich“ Farbanteile dazugemischt, um ein warm erscheinendes Licht zu erzeugen. Allerdings geschieht dies bereits in der LED, durch deren kristallinen Aufbau und nicht durch eine spezielle Ansteuerschaltung von außen.

Für die Ansteuerung eine RGB-LED ist zu beachten, dass jede Farbe ihren eigenen Vorwiderstand braucht, weil ja die Durchlassspannung, wie wir bereits gelernt haben, von der Farbe abhängig ist. Mit einem Vorwiderstand kann gewährleistet werden, dass auch einem High-Signal an der LED, diese nicht überlastet wird. Natürlich ist es auch möglich eine LED ohne Vorwiderstand zu betreiben. Dann jedoch ist darauf zu achten, dass der PWM-Wert nicht zu groß wird. Das bedeutet, dass das Puls-Pausen-Verhältnis nicht zu groß wird, um die LED zu schädigen.

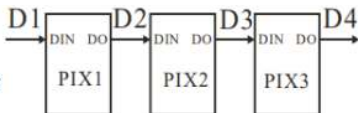
Es gibt auch digitale RGB-LED's, die eine Ansteuerung bereits eingebaut haben und vom Prozessor nur ein



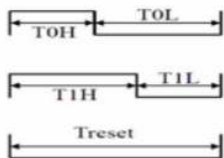
Digitalsignal benötigen, damit sie die einzelnen LED's mit der gewünschten Helligkeit ansteuern und so die gewünschte Farbe erzeugen. Es sind dies z.B. die **WS2812** von world-semi im 5050 Gehäuse; **APA106** im sogenannten Standardgehäuse mit 5mm Durchmesser und vier Anschlussbeinchen oder auch mit 8mm durchmesser; **PD9823** im 5mm und 8mm Gehäuse und auch als 5050 Variante mit der Bezeichnung PL9823-5050; und die **SK6812** ist auch noch zu erwähnen.



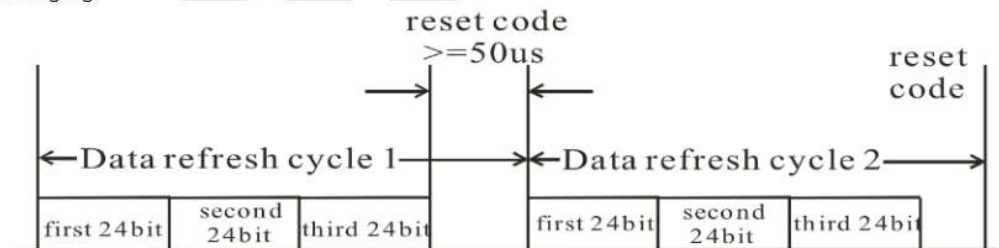
Links im Bild ist eine „intelligente“ RGB-LED dargestellt, mit der Anschlussbelegung.



Es kann nicht festgelegt werden, wie viele LED's in Serie geschaltet werden können. Dies wird von der Stromaufnahme begrenzt.



oben im Bild ist dargestellt, wie ein High, ein LOW- und ein Reset Signal übertragen werden-



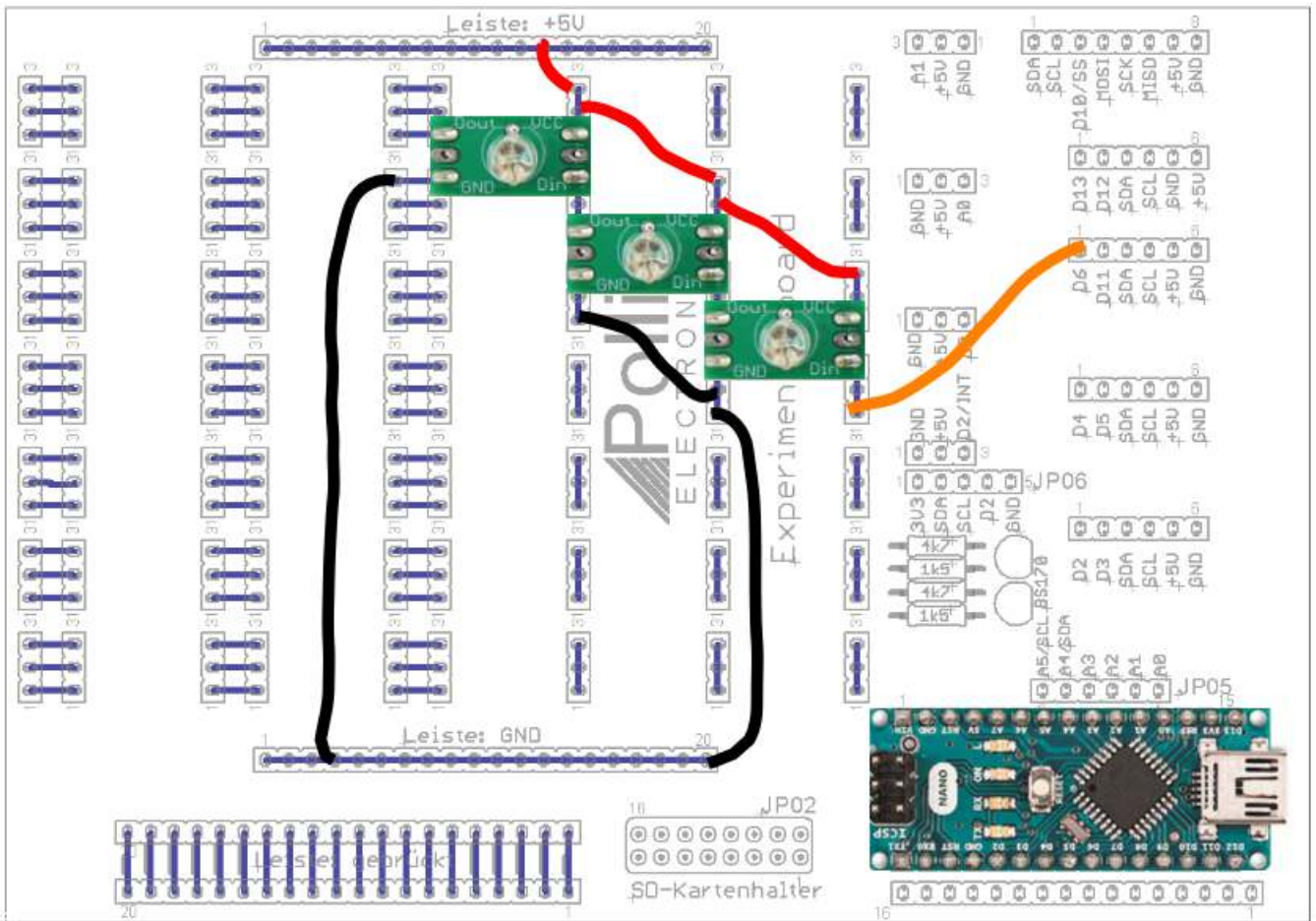
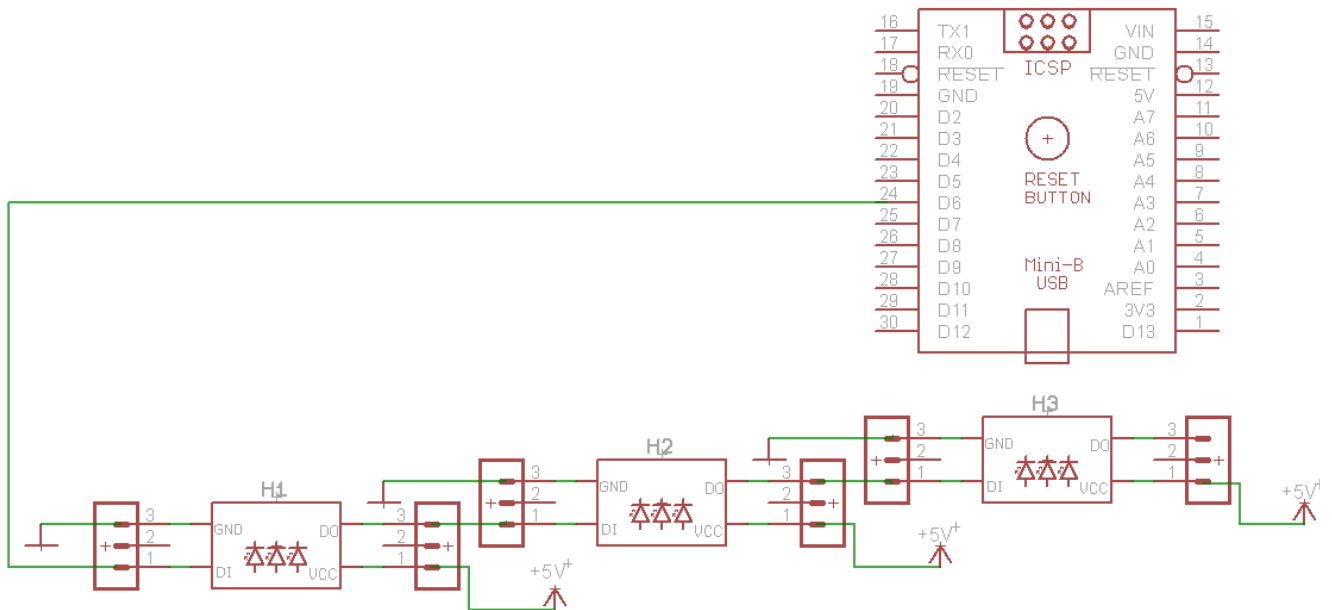
Im Bild oben ist schematisch gezeigt, wie eine Datenübertragung zu drei nacheinander geschalteten LED's erfolgt.

Im Bild unten ist gezeigt wie ein 24Bit Datenframe aufgebaut ist. Dabei werden zuerst das MSB von Grün, dann die Daten für die rote LED und letztendlich die Information für die blaue LED.

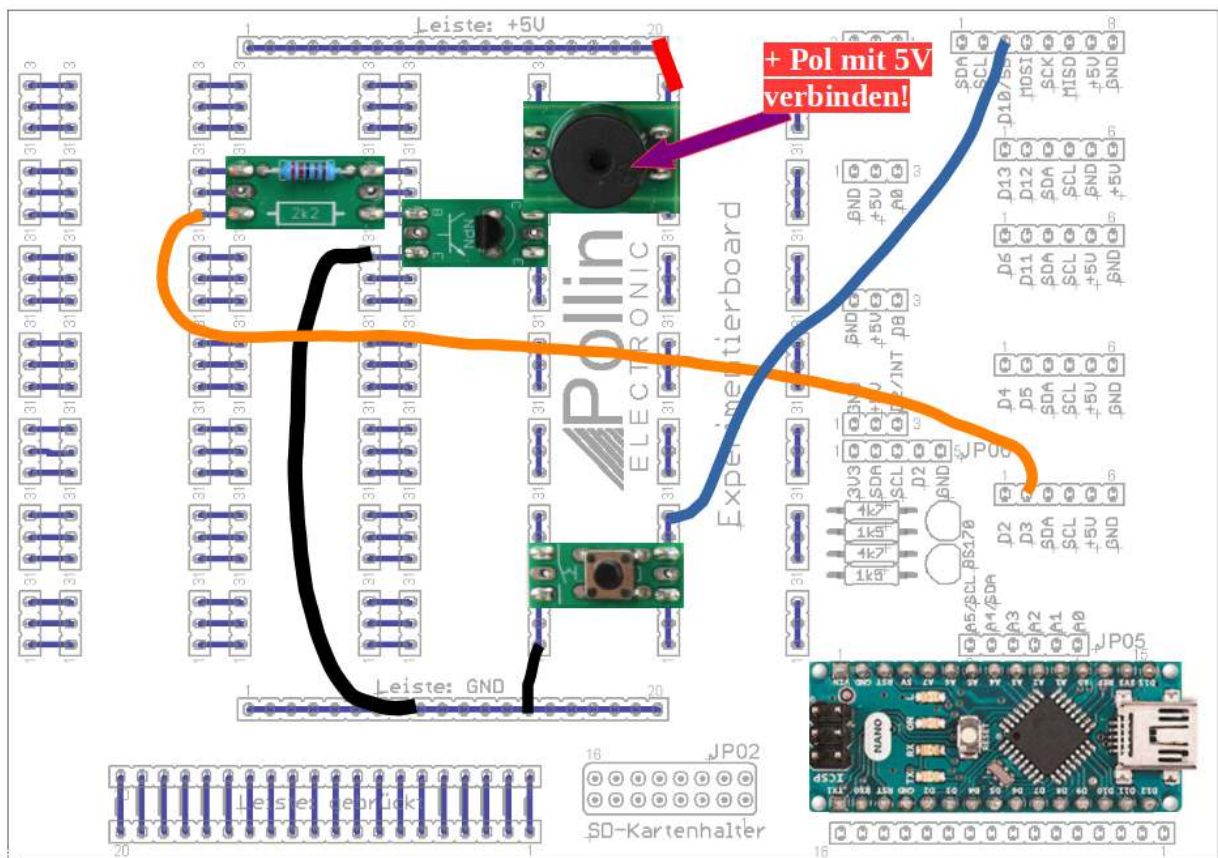
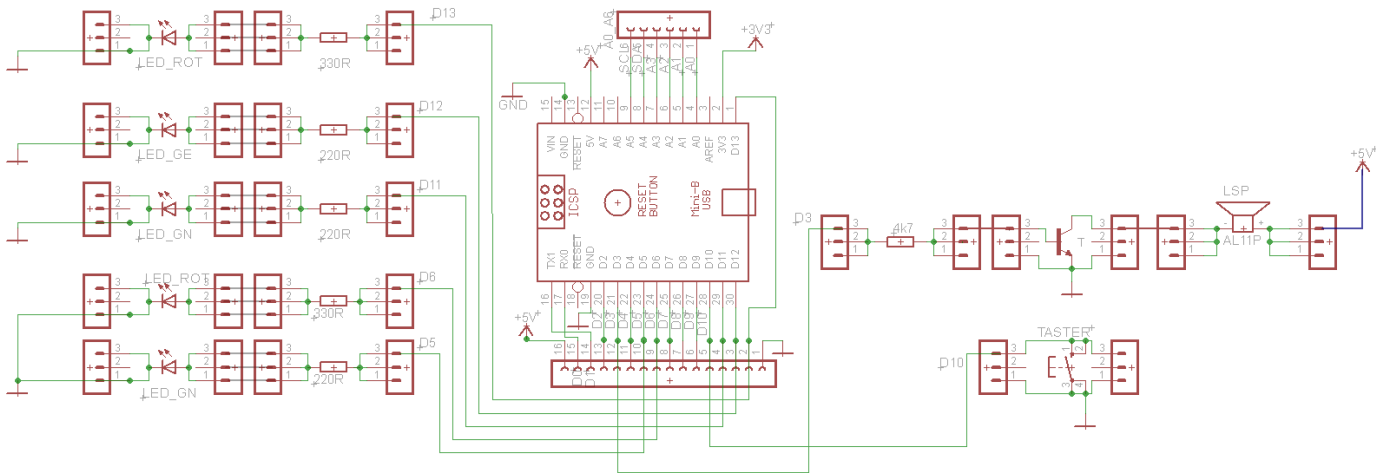
G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

11.6.3. Programmbeispiel für digitale RGB-LED

Im Beispiel `rgb_digital.ino` ist gezeigt, wie so eine LED angesteuert werden kann:



11.7. Tonausgabe:



Um Töne mit dem Arduino zu erzeugen, gibt es auch eine einfache Funktion `tone()` :
`tone (pin, frequenz);`
`tone (pin, frequenz, dauer)` bei `pin` ist entweder `pin 3` oder `pin 11` auszuwählen; als `frequenz` ist ein integer Wert einzugeben, mit einem Mindestwert von 31Hz; bei `dauer` kann man einen Wert als long in [ms] eingeben. Die Verwendung dieser Funktion erklärt sich am Beispiel [MelodienGong.ino](#) am besten.
 Wenn keine `dauer` eingegeben wurde, kann der Ton mit `noTone()`; wieder abgeschaltet werden.
 Beim Beispiel [MelodienGong2.ino](#) wird die Melodie nach dem Tastendruck gespielt. Bei Verwendung von `tone()` kann die Funktion `millis()` **nicht** verwendet werden.

11.7.1. Kurzeittimer (mit Taster bestückt)

Nach einer Zeit `delta_t` soll der Arduino 3 mal mit 4 kHz jeweils 200ms lang piepsen:

Kurzeittimer.ino. Der Aufbau ist identisch, mit dem in Abschnitt 11.9.

11.7.2. Martinshorn **Martinshorn.ino**

Ein Martinshorn besteht aus zwei Tönen, die in einer bestimmten zeitlichen Abfolge gespielt werden. Es ist das gleiche Beispiel wie bei `Klingelton.ino`. Lediglich werden nach dem Funktionsaufruf von `tone()` die Pausen mittels eines `delay` erzeugt. Der Wert für die Spieldauer ist in der Konstanten `k_pause` vordefiniert und kann jederzeit verändert werden. Im Beispiel **Martinshorn1.ino** erfolgt die Ausgabe der Töne erst nach dem Betätigen des Tasters.

11.7.3. vorbeifahrendes Martinshorn (Dopplereffekt) **Martinshorn_doppler.ino**

Damit soll ein vorbeifahrendes Martinshorn simuliert werden. Die Frequenz der Töne fällt aufgrund des sogenannten Dopplereffektes schlagartig ab, sobald sich ein Objekt von Beobachter entfernt. Dies ist bei jedem sich bewegenden Objekt zu beobachten. Bei einem vorbeifahrenden Einsatzfahrzeug mit Martinshorn ist dies jedoch besonders deutlich. Die Änderung der Frequenz ist abhängig von der Geschwindigkeit des Objektes. In dem Beispiel mit dem Martinshorn, wurde eine Geschwindigkeit von 100 km/h angenommen. In diesem Beispiel wurde nicht die Funktion `tone()` verwendet, da damit die gewünschte Frequenz nicht in dieser Auflösung und Genauigkeit erzeugt werden kann. Eine Schwingung besteht bekanntermaßen aus einer positiven und einer negativen Halbwelle. Die Periodendauer ist der Kehrwert der gewünschten Frequenz. Schalten wir also die Halbe Periodendauer den Ausgang am Arduino auf HIGH und dann die halbe Periodendauer auf LOW, dann ergibt sich die gewünschte Frequenz am Lautsprecher. Wenn man nun das positive Signal verkürzt und das kürzere Signal um diesen Wert verlängert, dann bleibt die Frequenz gleich, aber es vermindert sich die Lautstärke des erzeugten Signaltons. Sichtbar machen kann man diesen Effekt, in dem an den Ausgang des Arduino statt des Lautsprechers eine LED (z.B.: rot) mit Vorwiderstand (z.B.: 330R) angeschlossen wird. Dann ändert die LED ihre Helligkeit im Rhythmus des ausgegebenen Signals.

11.7.4. US Polizeisirene: **US_police.ino**

Im Internet wurde der Tonverlauf einer US-Polizeisirene folgendermaßen beschrieben:

Schleife 3x durchlaufen: 500 Hz bis 1600 Hz jede 3ms wert um 1 Hz erhöhen
 dann von 1600 Hz bis 500 Hz alle 3ms um 1Hz dekrementieren

Schleife 15 mal durchlaufen: 500 ... 1800 Hz und wieder 1800 Hz ... 500 Hz.

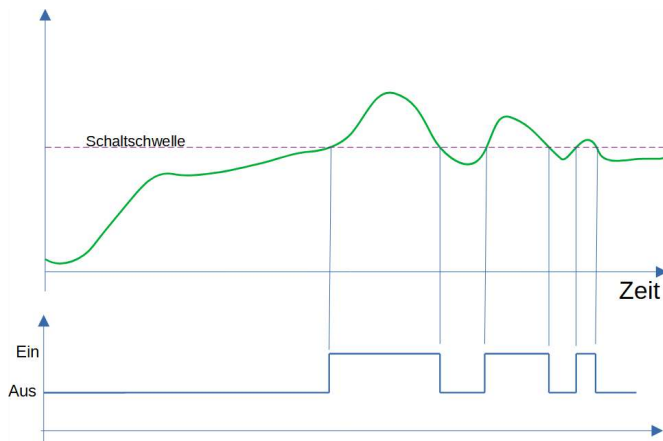
Dann beginnt alles wieder von vorne.

11.7.5. Frostwarner-Simulation **Frost_sim.ino**

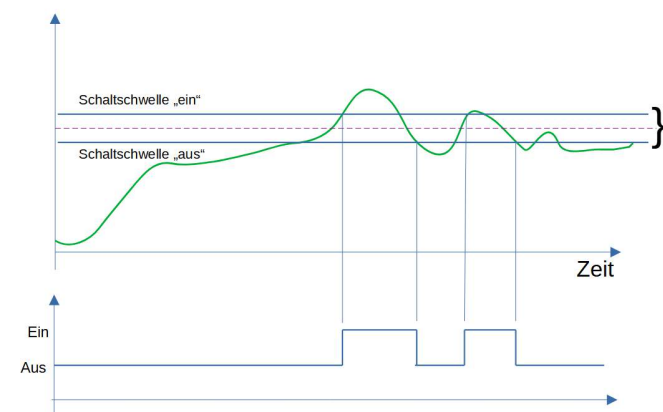
Unter 3°C soll ein Piepstön ausgegeben werden. Die Temperatur kann über einen speziellen Widerstand gemessen werden. Diesen simulieren wir mit einem Potentiometer. Wenn das Poti in

Mittelstellung gedreht wird, soll dann ein Piepston hörbar sein. Wird das Poti weiter gedreht, bleibt er. Wird das Poti allerdings zurückgedreht soll er erst nach einer gewissen Hysterese wieder erlöschen.

Im Beispiel [Frost_sim2.ino](#) wurde das Programm ergänzt, dass es nur einmal piepst, wenn der Schwellwert unterschritten wurde. Der nächste Piepston wird erst wieder ausgegeben, wenn die Schwelle wieder unterschritten wird, nachdem sie aber zuvor im normalen Betriebszustand gewesen ist. Unten ist der Schaltungsaufbau dargestellt. Statt des Spannungsteilers bestehend aus NTC und 10k kann auch ein Poti eingebaut und die Mittelanzapfung an den Analoginput A0 angelegt werden.



In Bild links ist zu sehen, wie sich ein Messwert in einer bestimmten Zeit verändern kann. Wäre das ein Temperaturverlauf, dann würde die Heizung oft aus und einschalten. Denn beim Überschreiten einer Schaltschwelle würde der Brenner „Ein“-geschaltet und beim Unterschreiten in den Zustand „Aus“ geschaltet. Dies möchte man verhindern, in dem man Schaltschwellen, wie im rechten Bild einführt. Je weiter diese Schaltschwellen auseinander liegen, desto größer, sagt der Fachmann, ist die Hysterese. Damit ist eine längere Brennerlaufzeit gewährleistet, was die Effizienz einer Heizungsregelung erhöht.

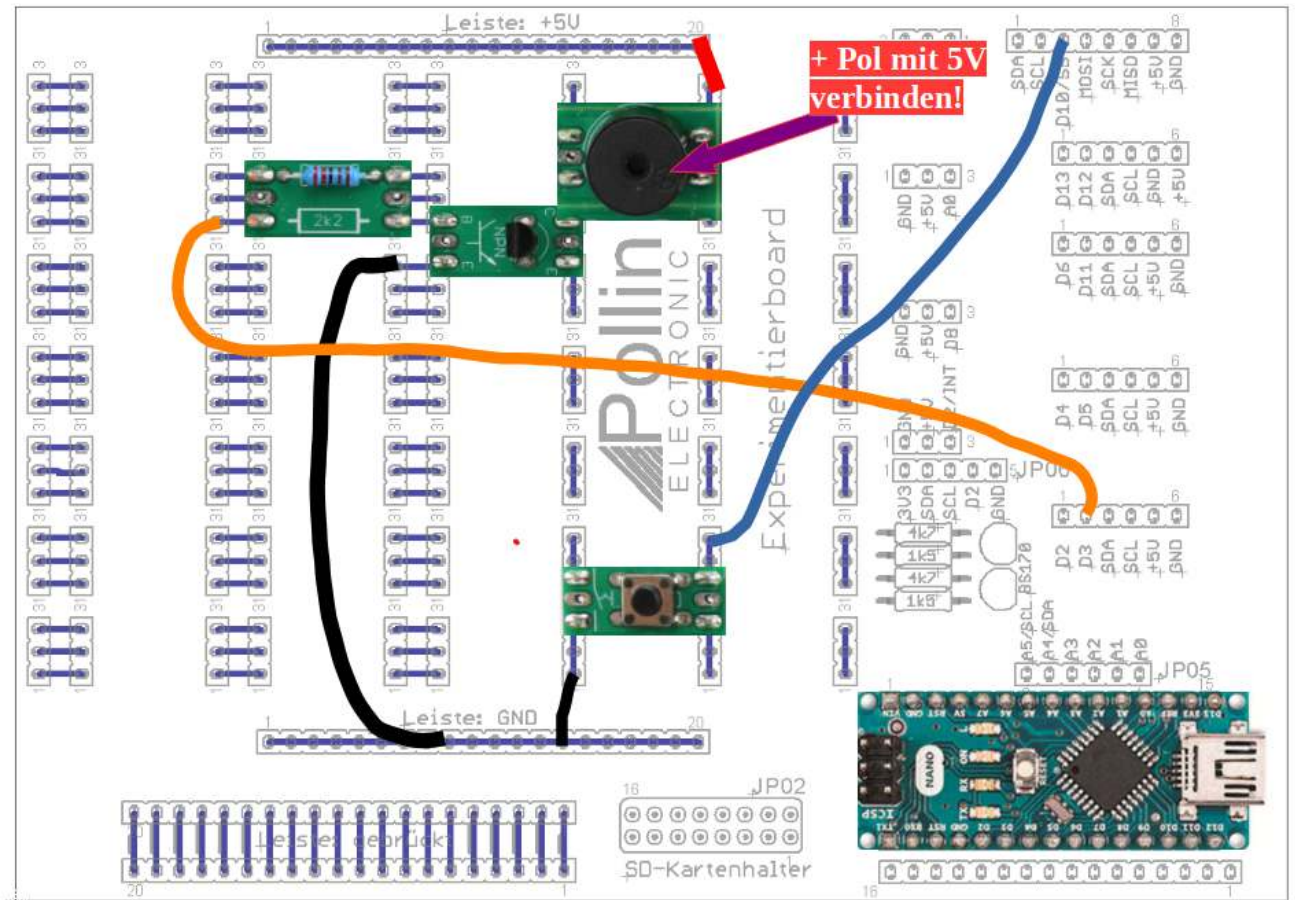
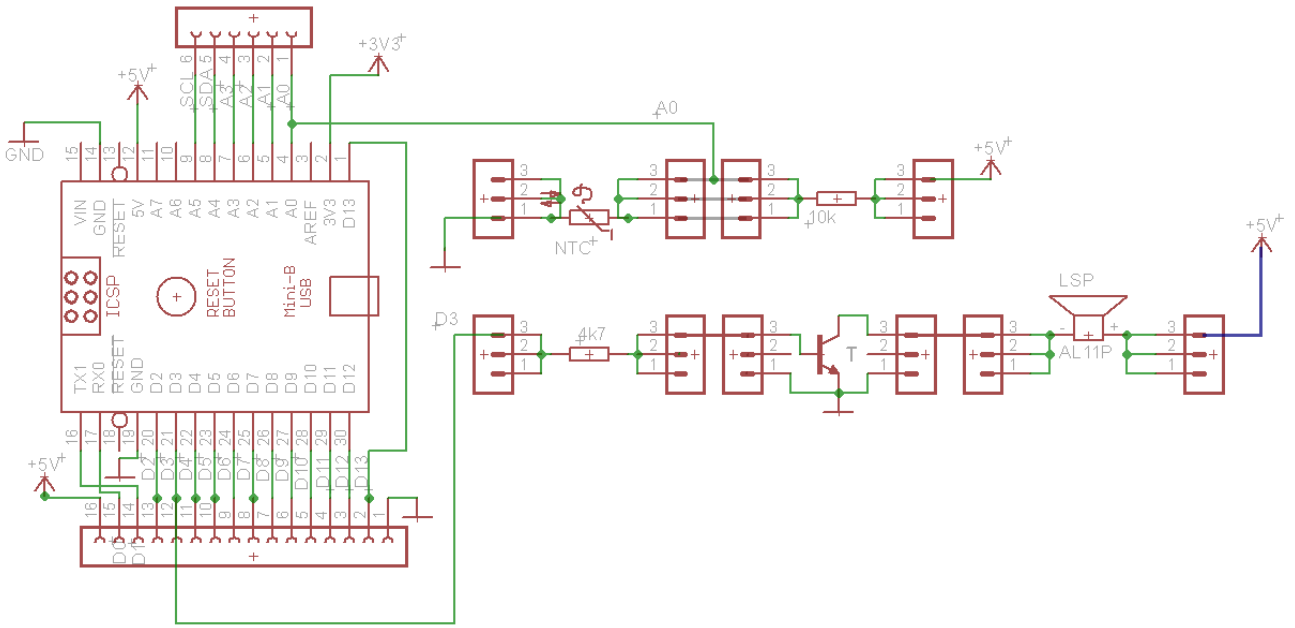


Im linken Bild ist das gleiche Messsignal wie im linken Bild dargestellt. Es wurde lediglich eine zusätzliche Schaltschwelle eingeführt. Erst bei Unterschreiten dieser Schaltschwelle wird in den Zustand „Aus“ geschaltet. Damit soll auch verhindert werden, dass das System zu „schwingen“ beginnt und im Endeffekt soll verhindert werden, dass das System so oft an- und abschaltet wird, dass keine Reaktion im Ergebnis erfolgt und die Heizung kalt, der Kühlschrank warm, oder eine Lampe aus bleibt, oder gar ein Gerät durch zu häufiges Schalten einfach kaputt geht.

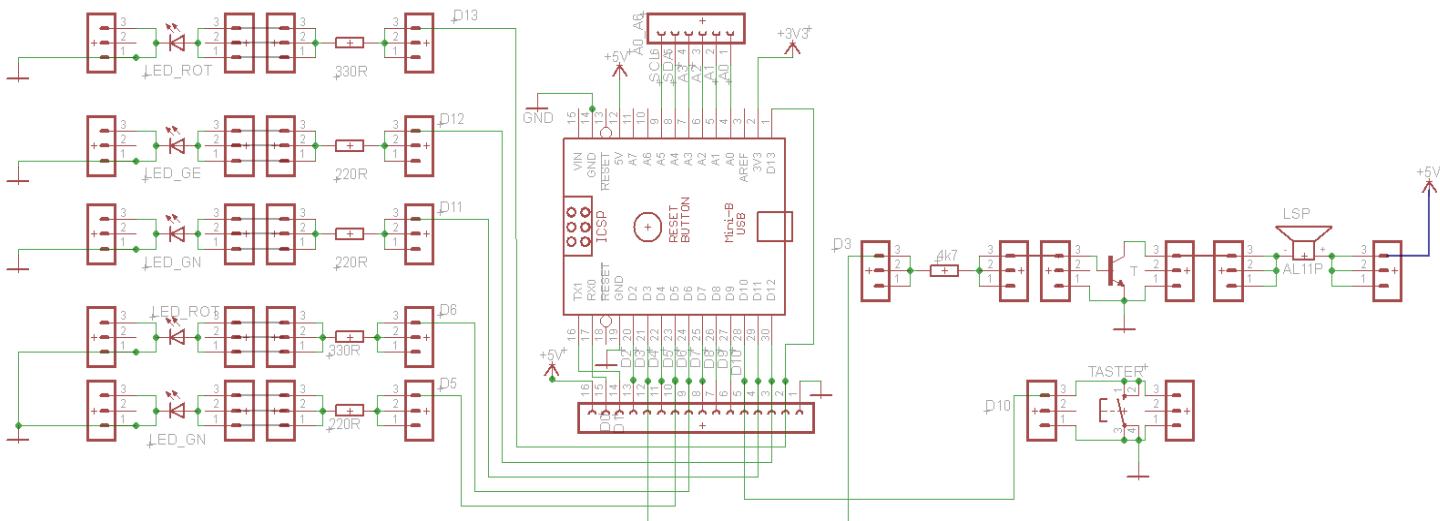
Nun zurück zu unserem Frostwarner-Beispielprogramm.

Bei einem 10k Ohm Vorwiderstand auf +5V würde an einem 10k NTC laut Datenblatt bei +3°C eine Spannung von 3,54V anliegen. Bei +4°C sind es 3,49V. Mit zunehmender Temperatur wird der Widerstand kleiner und somit auch die an ihm abfallende Spannung. Definiere, ab welcher Schwelle eine Zustandsänderung erfolgen soll und ändere selber die Werte für die eingestellte Hysterese.

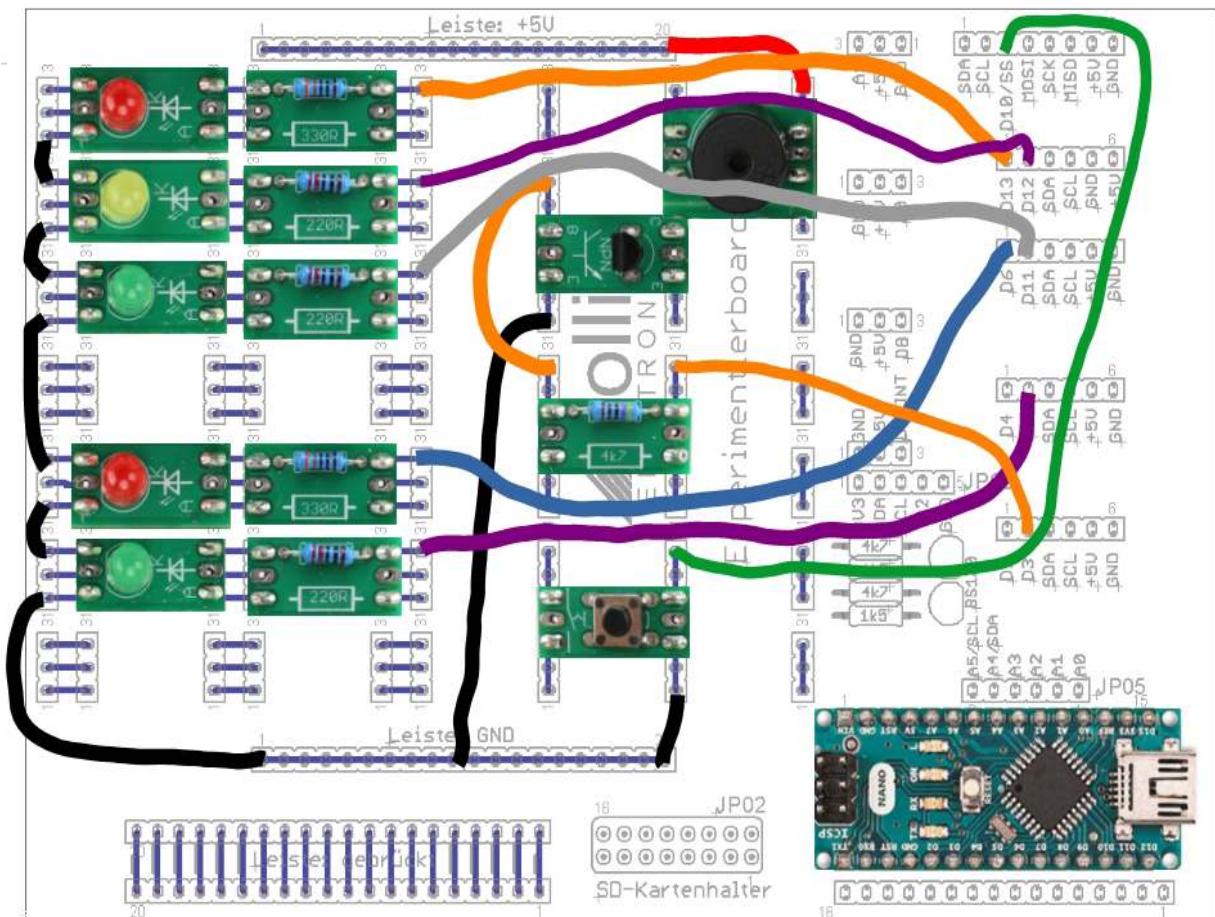
Dann bekommst Du selber ein Gefühl dafür, wie sinnvoll eine Hysterese ist.



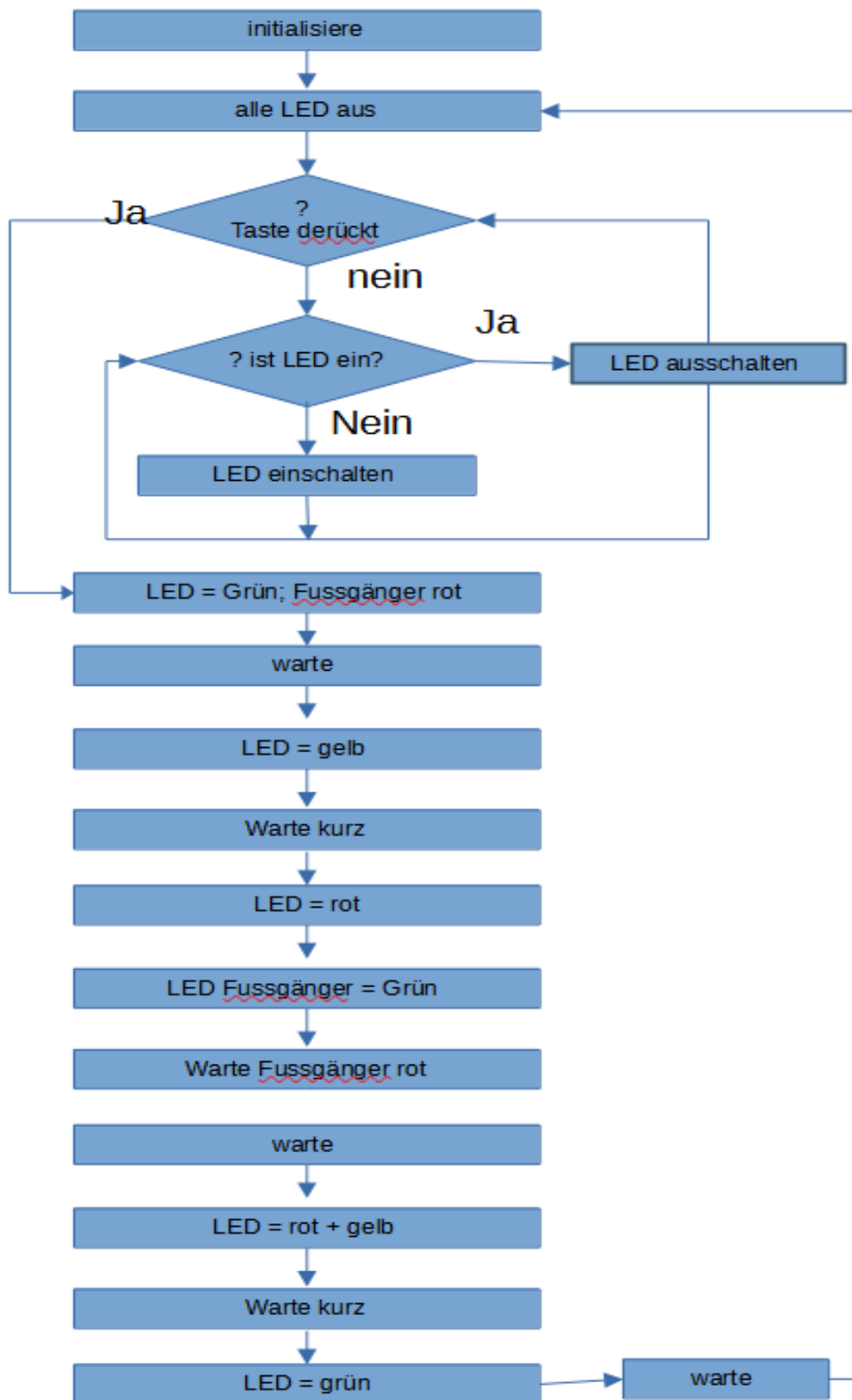
11.7.6. Ampelschaltung für Fußgänger mit Summer



Ein Taster soll mindestens 2s lang gedrückt werden. Daraufhin wird die Ampelanlage aktiviert. Autos haben grün, die Fußgänger noch rot. Nach twarte1, wird gelb, dann rot. Nach twarte2, wird grün für die Fußgänger und es soll ein hörbar sein. Falls außer Betrieb, blinkt nur die gelbe LED für die Autofahrer. Das Beispiel befindet sich in der Datei [Ampelsteuerung.ino](#).



Auf dem Gehäuse des Summers ist eine Seite mit einem Plus-Symbol markiert. Dieser Anschluss ist immer mit 5V zu verbinden!



Da der Tastendruck bei einem delay-Befehl ignoriert wird und die Taste oftmals länger als 2s gedrückt werden muss, wurde im Beispiel [Ampelschaltung2.ino](#) wieder auf die Abfrage von millis() zurück gegriffen. Was noch ein Schönheitsfehler war, ist die grüne LED der Verkehrsampel. Diese sollte natürlich ausgeschaltet sein, wenn die gelbe LED blinkt.

Komplexere Beispiele

11.8. IR Empfang

11.8.1. Das Übertragungsprotokoll

Decodieren diverser Fernbedienungen bei 38 khz Trägerfrequenz. Die Beispiele sind direkt aus dem Internet übernommen, da sie sehr kompliziert, aber doch interessant sind. Eine sehr wichtige Funktion hierbei ist:

Beschreibung misst die Dauer eines Pulses an einem Pin. Es kann entweder der Wert LOW oder HIGH übergeben werden. Wenn der übergebene Wert HIGH ist, wird gewartet bis der Pin auf HIGH gezogen wird, startet einen Timer und wenn der Pin wieder auf LOW fällt, stoppt der Timer und die Pulslänge in Mikrosekunden (μs) wird zurückgegeben. Wenn ein Timeout spezifiziert wird, gibt die Funktion nach dieser Dauer den Wert 0 zurück. Die Funktion wurde nur berechnet und kann bei längeren Pulsen zu Fehlmessungen führen. Die Funktion kann Pulslängen von $10\mu\text{s}$ bis 3 Minuten messen. Syntax `pulseIn(pin, wert); pulseIn(pin, wert, timeout);` Parameter `pin`: die Pinnummer des Pins an dem der Puls gemessen werden Sollwert: der Zustand bei dem der Timer gestartet werden soll, HIGH oder LOW `timeout`: die Zeit in Mikrosekunden die die Funktion auf einen Puls wartet, bis sie 0 zurückgibt und beendet wird Rückgabewert ist die Pulslänge in Mikrosekunden oder 0 wenn kein Puls innerhalb des timeouts gestartet wurde.

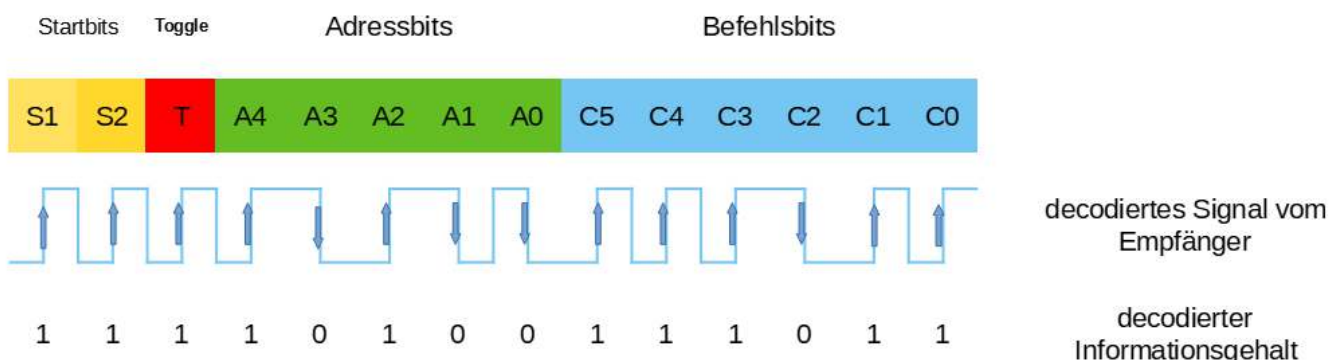
Beispiel:

```
int pin = 3;
unsigned long dauer;

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

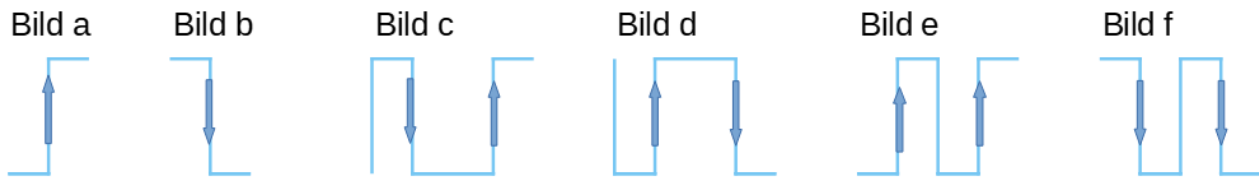
void loop()
{ dauer = pulseIn(pin, HIGH); }
```

Aufbau eines RC5 Infrarotbefehls



Das obige Bild stellt das decodierte Signal dar, das vom Receiver chip ausgegeben wird. Das toggle Bit ändert seinen Zustand von 0 auf 1 bzw. von 1 auf 0, wenn eine Taste erneut gedrückt wird. Die beiden Startbits sind immer 1; die address Bits kategorisieren das Gerät, das angesprochen werden soll; das command Bit den Befehl, also Lautstärke, Kanalwahl, ein- ausschalten etc.

Diese Art Signale so zu codieren wurde Anfang der 1980er Jahre von der Firma PHILIPS eingeführt und hat sich dann zu einer Art Standard entwickelt. Mittlerweile wurde das Prinzip auch von anderen Firmen in abgewandelter Form übernommen.



Ob ein Zustand als HIGH oder LOW erkannt wird entscheidet bei diesem Übertragungsprotokoll nicht der Spannungspegel, so wie z.B. bei der seriellen Kommunikation. Bei diesem Protokoll ist die Richtung der Spannungsflanke des Signals entscheidend; steigt die Flanke, bedeutet dies eine logische eins (Bild a), also HIGH, fällt die Flanke, bedeutet dies eine logische 0, also ein LOW-Signal (Bild b).

Wenn man die möglichen Signalfolgen nebeneinander stellt, erkennt man schnell den Unterschied. Einmal bedeuten zwei lange Pulse 1 0 (Bild d) und ein anderes Mal 0 1 (Bild c); Ähnlich verhält es sich bei kurzen aufeinander folgenden Pulsen. Diese bedeuten einmal 1 1 (Bild e) und das nächste Mal 0 0 (Bild f). Deutlich zu erkennen ist dabei, welchen Ausgangspegel die Signale dabei haben. Kommen sie von HIGH oder von LOW.

Ganz entscheidend, die Signale zu decodieren ist das Timing. Ein kurzer Puls hat eine Länge von 889 us; ein langer Impuls hat eine Länge von 1778 us. Diese Timings gelten allerdings nur bei 36kHz Trägerfrequenz. Mittlerweile gibt es verschiedene Trägerfrequenzen üblicherweise zwischen 35 und 40kHz.

11.8.2. Decodieren des IR-Signals

Das Decodieren eines Infrarotsignales ist sehr kompliziert. Deshalb ist es am besten, sich voll und ganz auf die library, die es für den Arduino gibt zu verlassen.

Wie bereits bekannt, wird diese mit `include <Iremote.h>` eingebunden.

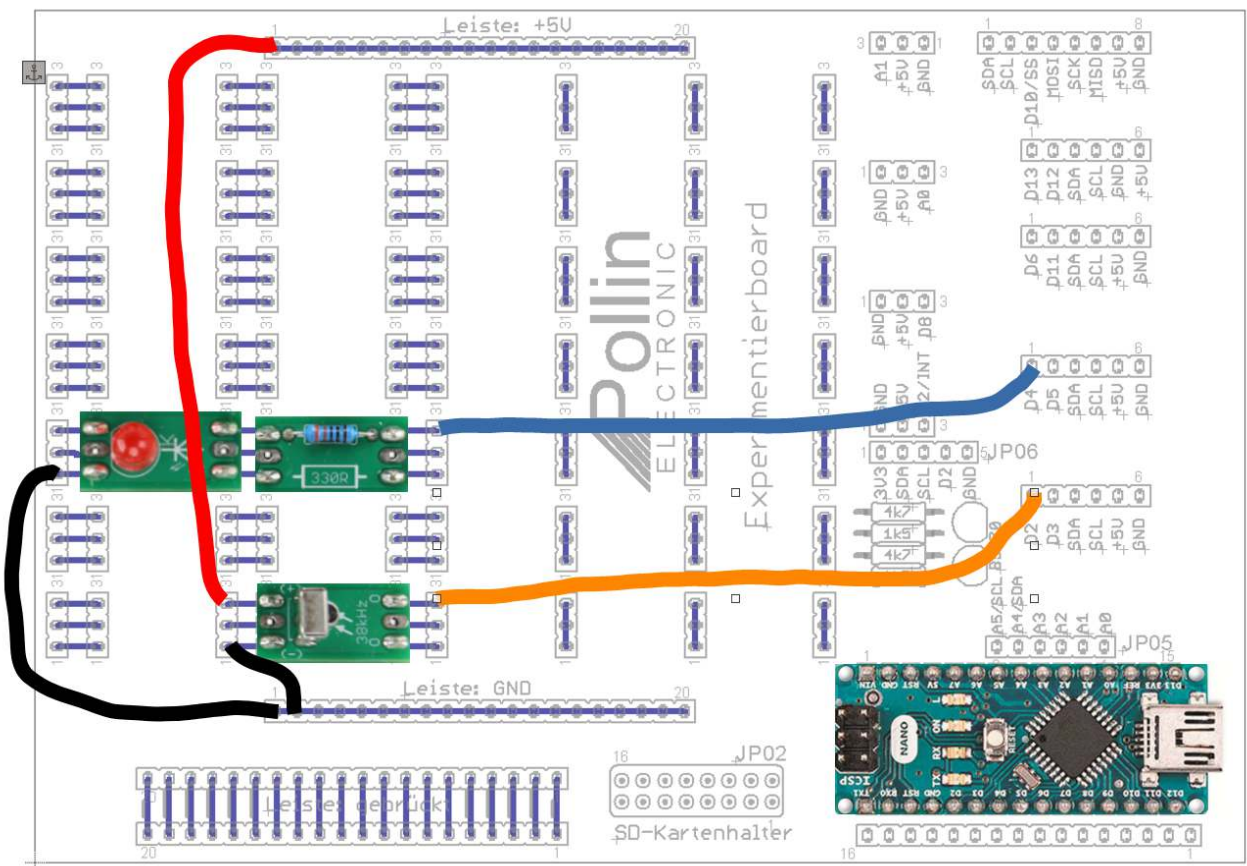
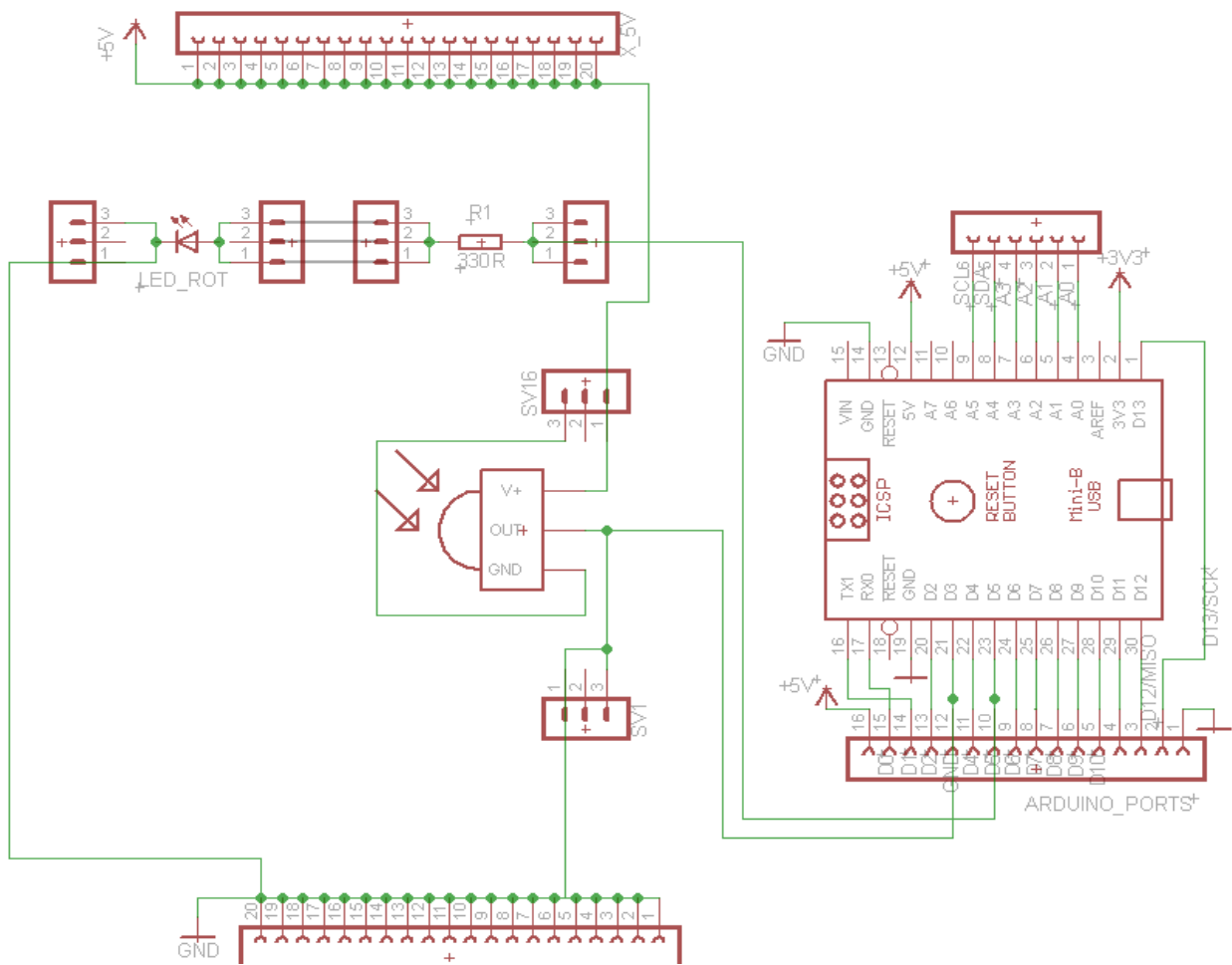
Mit `#define RECV_PIN 2` wird festgelegt wo der 38kHz IR – Empfänger an den Arduino angeschlossen wird.

Mit der Befehlszeile `Irrecv irrecv (RECV_PIN);` wird eine Funktion definiert, mit der man die Funktionen der Bibliothek nutzen kann. Dabei wird mit `(RECV_PIN)` der Pin definiert und der Bibliotheksfunktion mitgeteilt. Der Name `irrecv` ist frei wählbar und ist notwendig, weil damit alle Funktionsaufrufe der Bibliothek erfolgen. Dies wird im Beispiel [ir_beispiel0.ino](#) deutlich. Damit lassen sich dann mit den Nummerntasten verschiedene LED's ein und abschalten.

Das Beispiel [IR_beispiel1.ino](#) zeigt die Dekodierung eines Infrarotsignals entsprechend den Befehlen und Adressen, die übertragen werden. Damit kann man unterschiedliche Fernbedienungen von Fernsehern, Receivern u.ä. vergleichen.

Ursprünglich wurde das Infrarotprotokoll von der Firma PHILIPS erfunden und mit RC5 bezeichnet. Mittlerweile gibt es von verschiedenen Herstellern Geräte mit deren eigenen Protokollen.

Im Beispiel [IR_handset2.ino](#) sind verschiedene Protokolle hinterlegt und können bei vorhandener Fernbedienung auch getestet und decodiert werden.

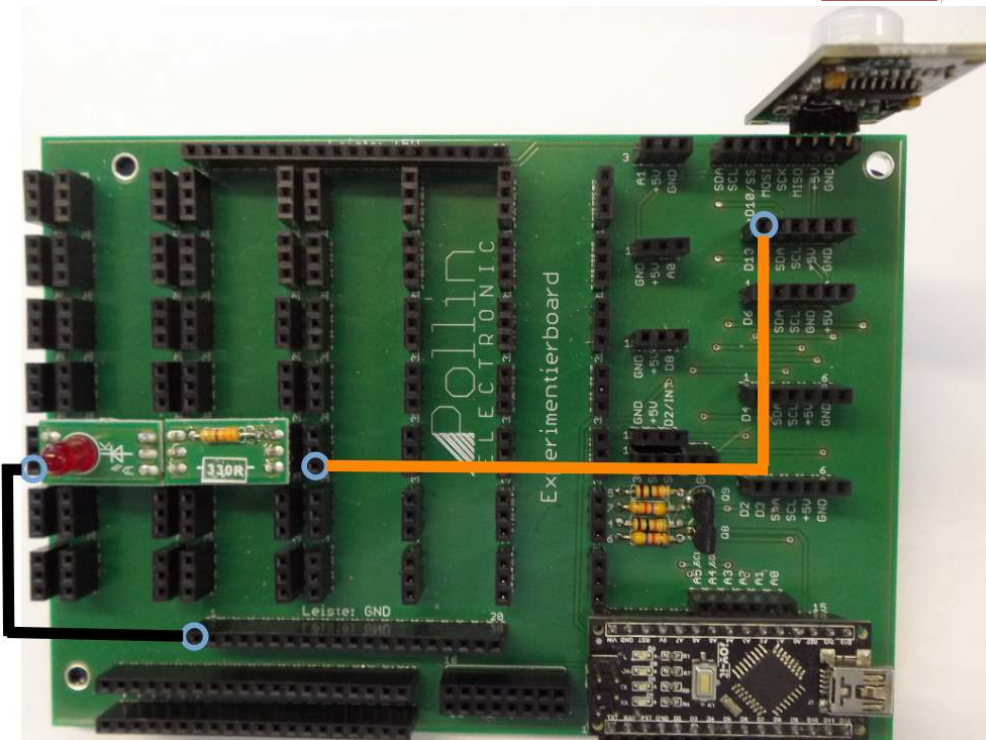
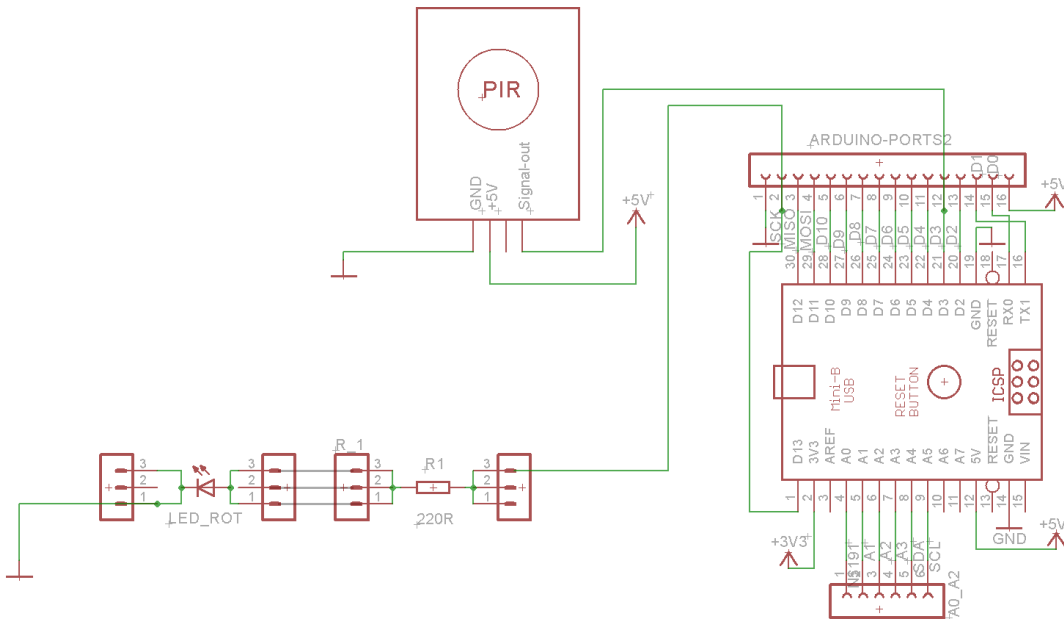


Für die fortgeschrittenen Programmierer:

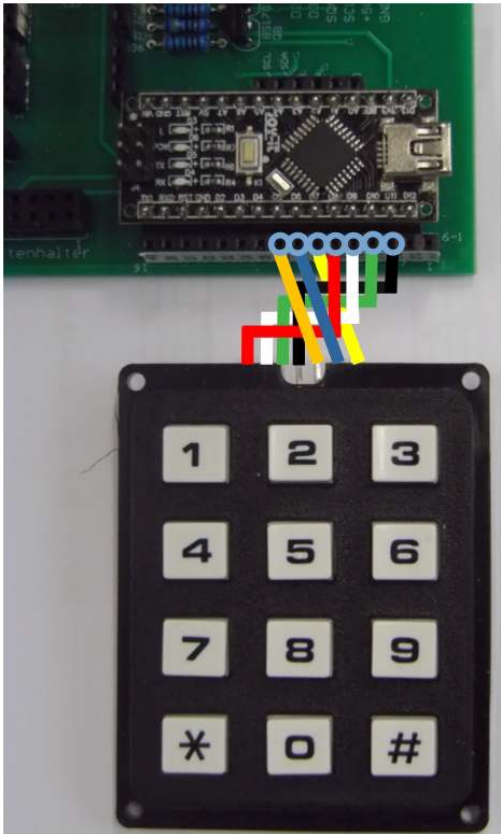
12. Diverse Module an den Arduino anschließen

12.1. PIR-Sensor (Artikelnummer 811274)

Im unteren Bild, ist die Verdrahtung zu sehen, wie der PIR Sensor mit dem Arduino zu verschalten ist. Der Sensor braucht nur eine Spannungsversorgung und liefert für ca. 3s ein 5V Signal, wenn er eine Person erkennt. Mit dem Poti kann die Reichweite (Empfindlichkeit) des Sensors eingestellt werden. Das Beispiel [PIR01.ino](#) ist ein Ereigniszähler. Das Beispiel [PIR02.ino](#) ist [PIR01.ino](#) nur um einen Funktion ergänzt, die die Abfrage des Eingangssignal übersichtlicher macht.



12.2. Tastenfeld (Artikelnummer 421221)

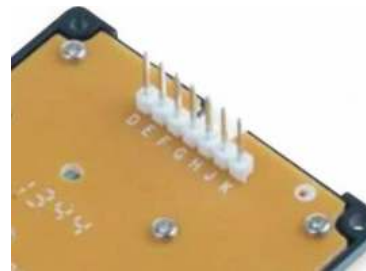


Die Verdrahtung erfolgt nach folgender Tabelle:

- Tastefeld: D → D8 (gelbe Leitung)
- Tastefeld: E → D7 (blaue Leitung)
- Tastefeld: F → D6 (orange Leitung)
- Tastefeld: G → D12 (schwarze Leitung)
- Tastefeld: H → D11 (grüne Leitung)
- Tastefeld: J → D10 (weiße Leitung)
- Tastefeld: K → D9 (rote Leitung)

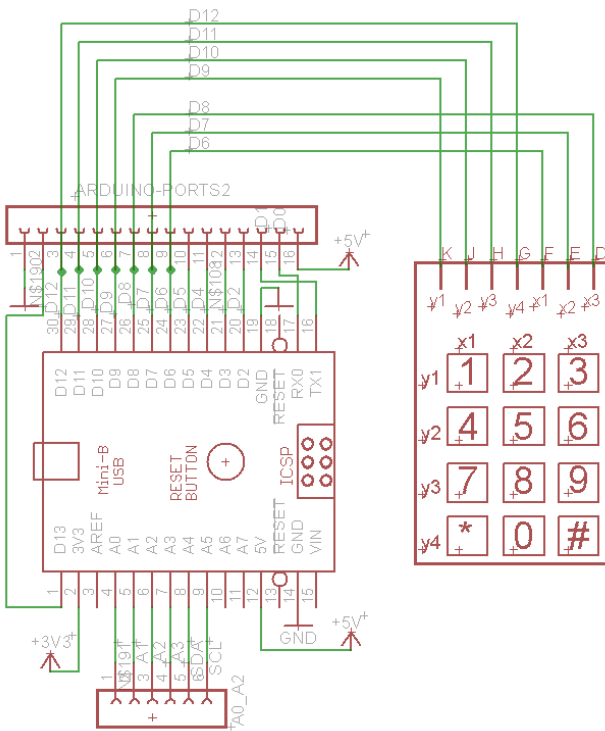
Ansicht: Tastenfeld Rückseite

Dabei können die Pins nach vorne gebogen werden, damit die Platine besser aufliegt



	X1	X2	X3
Y1	1	2	3
Y2	4	5	6
Y3	7	8	9
Y4	*	0	#

Matrix XY



Anmerkungen zum Programmcode **Tastefeld.ino** :

Das Tastenfeld ist als Matrix mit Zeilen (Y1 .. Y4) und Spalten (X1 ... X3) aufgebaut. Die Zeilen werden dabei abwechselnd auf das Potential von 5V (High-Zustand) gesetzt und die Spalten dabei eingelesen, um sie auf eine Zustandsänderung zu prüfen. Damit kann festgestellt werden, welche Taste momentan gedrückt ist.

In der Beispieldatei wird der Tastendruck ausgewertet. Es kann also ein definierter Code abgefragt werden. Dazu wird der Code, der eingegeben werden muss, in dem Array `code_def[]` abgespeichert. Die Tasten werden immer eingelesen, bis # betätigt wird. Dies ist eine Art Quittierung der Eingabe. Wenn der davor eingegebene Code richtig ist, dann wird über die serielle Schnittstelle „OK“ ausgegeben. Es kann auch ein Piepser angeschlossen werden, oder eine LED zum Leuchten gebracht werden, um die korrekt erfolgte Eingabe anzuzeigen.

oben im Foto rechts zu sehen. Die ausgelesenen Kartendaten von [mifare_01.ino](#) werden in [mifare_02.ino](#) gebraucht, um sie im Vergleichsarray abzuspeichern, um so festzustellen, wer Zugang bekommt und wer nicht. Beispiel [mifare_03.ino](#) ist ein wenig verspielter, was die Auswertung betrifft.

12.4. SD Karte (Artikel 810359)

Mit der Erfindung der SD-Karte wurde es möglich die Flash-Speicher-Technologie auch auf portablen Geräten und Mikrocontrollern zu nutzen. Viele Mikrocontroller besitzen standardmäßig ein SPI-Interface. Damit ist es möglich Daten schnell und sicher zur SD-Karte zu übertragen und abzufragen. Der USB-Stick wurde ca. fünf Jahre vor der SD-Karte erfunden, konnte aber mit herkömmlichen Mikrocontrollern nicht gelesen und beschrieben werden, weil dieser eine USB-Schnittstelle hat. Das Interface zu einer SD-Karte ist mit dem SPI-Bus leichter zu implementieren. Da es sich dabei um eine rein Serielle Datenübertragung handelt. Die SD Karte muss nur als FAT32 formatiert werden und sollte nicht größer als 32GB sein.

Dann die SD-Karte, richtig herum, in den Slot stecken, bis sie verriegelt und dann den Kartenhalter auf die Arduino Starter Kit Hauptplatine stecken (siehe Abbildung unten).

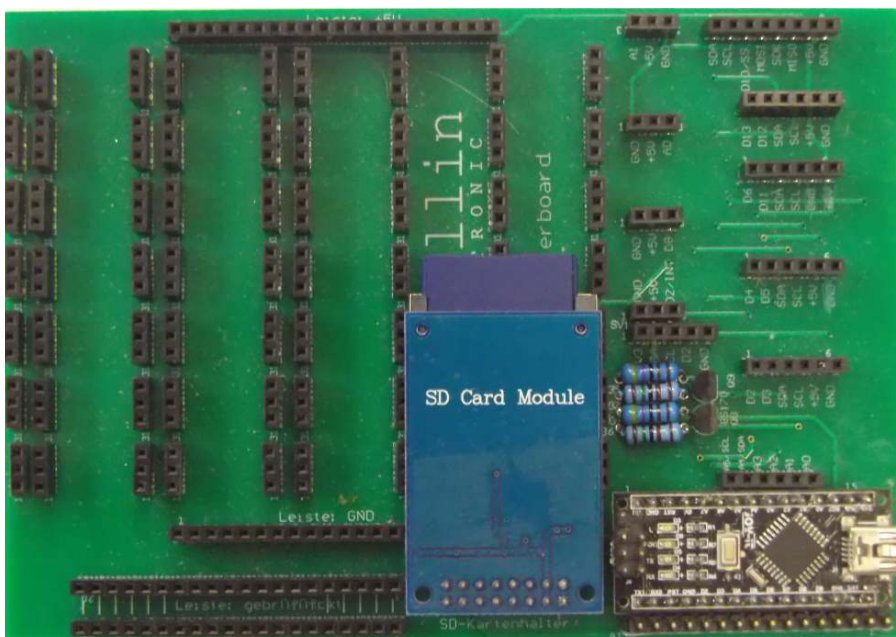
Der Steckplatz ist nur für die Artikelnummer 810359 geeignet. Andere SD-Karten müssen mit separaten Kabeln an die jeweiligen Portpins des Arduino angeschlossen werden. D10 ist der Eingang für das ChipSelect-Signal.

Beispiele: [cardinfo.ino](#) und [ReadWrite.ino](#)

Es gibt unter dem Menüpunkt Datei → Beispiele → SD noch weitere Beispielprogramme als die beiden oben genannten. Beim Beispiel [cardinfo.ino](#) werden nur die Bibliotheken der SD-Karte: SD.h und der SPI-Schnittstelle: SPI.h verwendet. Damit ist es möglich u.a. den Kartentyp abzufragen, die Größe des verfügbaren Speichers und so. Im Programmcode ist aber eine kleine Anpassung zu machen. In Zeile 36 ist der Wert für das ChipSelect Signal abzuändern: `const int chipSelect = 10;`

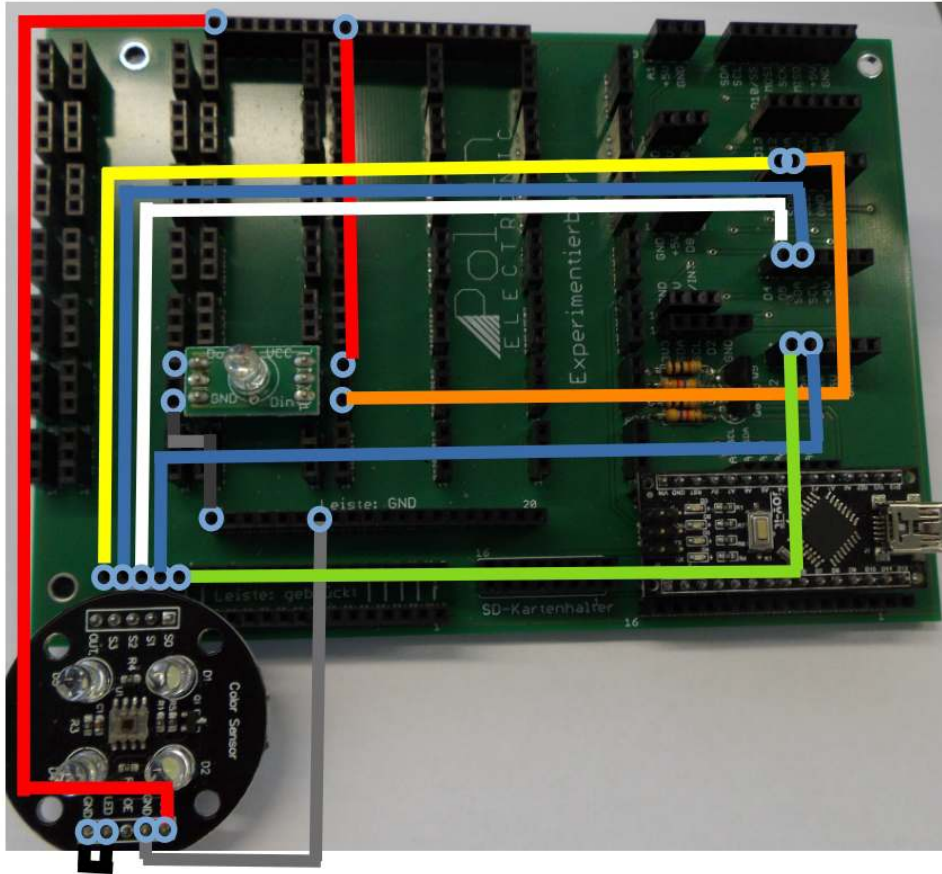
Um nun auf Daten einer Datei auf der SD:Karte zuzugreifen, braucht man das Objekt **FILE** myFile, wie es im Beispiel [ReadWrite.ino](#) gezeigt wird. Mit der erstellten Instanz **myFile** vom Objekt **FILE**, ist es problemlos und einfach möglich, Dateien zu erstellen, zu beschreiben, zu lesen und wieder zu schließen. Aber auch in diesem Beispiel ist eine kleine

Korrektur notwendig. In Zeile 36 wird `if (!SD.begin(4))` auf `if (!SD.begin(10))` geändert! Nun sollte auch dieses Beispiel fehlerfrei ablaufen.

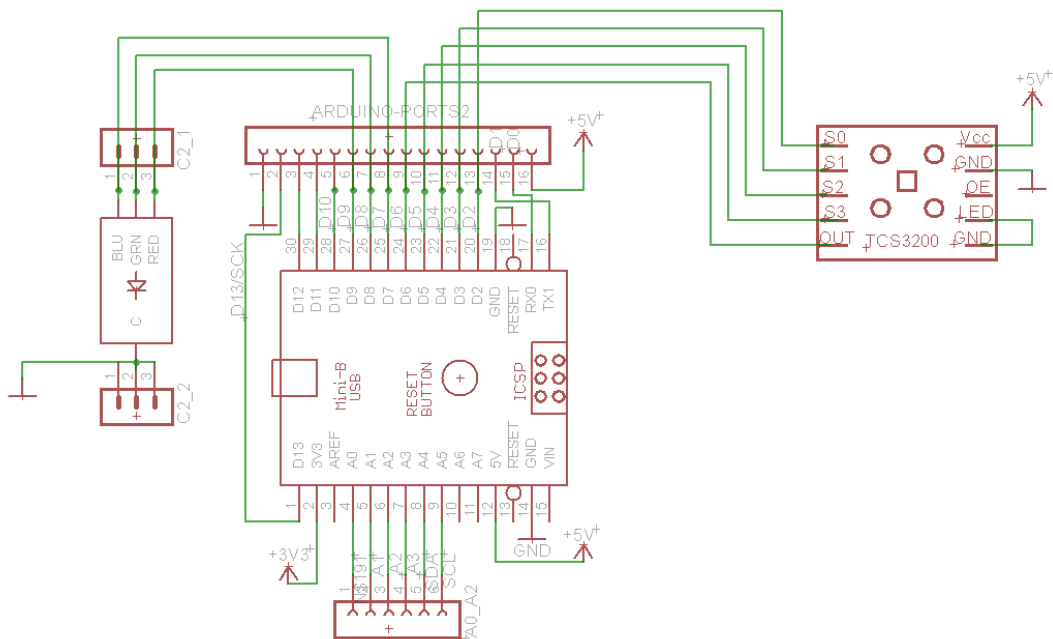


12.5. Farben erkennen mit dem Farbsensor TCS3200 (Artikelnummer 810681)

Das Prinzip des Sensors ist, dass verschiedene Farben unterschiedliche Ausgangsfrequenzen erzeugen.



Beispiel [TCS3200_01.ino](#) soll dies verdeutlichen.



Im Prinzip funktioniert der Farbsensor so, dass im Sensor Farbfilter für rot, grün und blau aktiviert werden können, die das empfangene Farbspektrum untersuchen. Dabei gibt der Sensor für den aktuell eingestellten Farbfilter, eine der empfangenen Helligkeit proportionale Frequenz aus. Dieses Rechtecksignal kann dann leicht vom Arduino detektiert und vermessen werden. Jede Farbe setzt sich aus der Summe der Anteile der drei Grundfarben zusammen. Da der hier verwendete Sensor sehr günstig ist, liefert er nur grobe Werte. Aber es gibt auch hochauflösende Farbsensoren, die sehr empfindlich sind und durchaus eine Farbe besser analysieren können als ein Laie. Ist das Signal OE mit GND verbunden, so wird ein Rechtecksignal am OUT-Pin generiert. Mit den Signal S2 und S3 werden die Filter ausgewählt:

S2	S3	Filtertyp
LOW	LOW	Rotfilter
LOW	HIGH	Blaufilter
HIGH	HIGH	Grünfilter
HIGH	LOW	kein Filter

kein Filter bedeutet, es kann die Helligkeit der Umgebung gemessen werden; Je nachdem welcher Filter aktiv ist, wird am Ausgang ein Rechtecksignal erzeugt. Dessen Frequenz ist ein Maß für die Helligkeit des jeweiligen Lichtes. So kann anhand der Frequenzen die Farbe bestimmt werden. Mit S0 und S1 läßt sich der Bereich der Ausgangsfrequenz (2kHz – 500kHz) festlegen:

S0	S1	Verhältnis
LOW	LOW	ausgeschaltet
LOW	HIGH	2%
HIGH	LOW	20%
HIGH	HIGH	100%

Bei vielen Beispielen im Internet wird 20% gewählt.

Anmerkung:

Beim Hersteller des Sensors kann zur Kalibrierung eine Farbblatt herunter geladen werden:

<https://joy-it.net/files/files/Produkte/SEN-Color/RGB-Test-Sheet.pdf>

Als Beispiel eine Farbe festzulegen kann z.B. folgendermaßen vorgegangen werden: Das Programm starten und dann den Sensor an eine Farbe halten z.B. rot. Dann die Nachrichten beobachten:

$R = 27$ $G = 85$ $B = 81$ Farbe: xxx

...
....

Aus den obigen Messwerten dann die Schwellen definieren:

```
if ((fqu_rot > 24) && (fqu_rot < 30) && (fqu_grn > 80) && (fqu_grn < 90) && (fqu_blaue > 75) && (fqu_blaue < 85))
  { color = 4;} //definiert Farbe: rot
```

Die Grenzwerte anfangs lieber ein wenig großzügiger festlegen. Die Farben festzulegen ist ein wenig Experimentierarbeit. Auch nicht jeder Blauton wird als Blau erkannt. Jede Farbe bräuchte eigentlich eine eigene Schwellenfestlegung. Also ist viel Schreiarbeit nötig.

Manchmal liegen die Messwerte außerhalb des definierten Bereiches.

Die Funktion: **constrain**(x, minimum, maximum) könnte da Abhilfe schaffen.

Falls die Zahl x größer maximum ist, dann wird als Wert für x maximum zurückgegeben;

Falls die Zahl x kleiner minimum, dann wird als Wert für x minimum zurückgegeben;

Wenn der Wert für x: minimum <= x <= maximum, dann bleibt x unverändert.

Oder mit einer if-Abfrage verwirft man die Werte und liest den Datensatz nochmal neu ein.

12.6. Beschleunigungssensor MPU5060 (Artikelnummer 811107)

Dieser Beschleunigungssensor ist ein sogenannter 6 Achsen-Sensor. Das bedeutet, er besteht im Prinzip aus zwei Sensoren, einem Gyroskop (Kreiseilinstrument) und einem Beschleunigungssensor.

Der Beschleunigungssensor kann nicht von einer Beschleunigung durch eine Bewegung und der Erdbeschleunigung unterscheiden. Der Kreiselsensor (Gyroskop) schon. Das Gyroskop, gibt Werte aus, wenn der Sensor in Bewegung ist. Der Beschleunigungssensor gibt auch Werte aus, wenn er in Ruhe ist, er misst die Erdbeschleunigung. Ein 9 Achsensensor, würde zudem x, y, z – Werte liefern für das Erdmagnetfeld.

Wir wollen uns jedoch hier auf den Beschleunigungssensor beschränken. Dieser hat 16-Bit Auflösung, Messbereiche +/- 2, 4, 8 oder 16 g; wobei mit g, die Erdbeschleunigung gemeint ist. Es gibt zu diesem Sensor ein Datenblatt und eine sogenannte Registermap, in der jede Funktionalität für einen Programmierer beschrieben ist. Insgesamt sind das fast 100 Seiten Dokumentation, was ein Indiz für die Komplexität dieses Sensors ist. Ein 9 Achsensensor ist noch ein vielfaches komplexer, weil die Berechnung des Magnetfeldes sehr viel Mathematik erfordert.

Deshalb ist es schwierig im Internet eine Bibliothek zu finden, die all diese Funktionen abdecken kann.

Die Messdaten sind zu dem noch temperaturabhängig!

Im ersten Programmbeispiel [MPU6050_gyro_simple.ino](#) , gilt auf eine Programmzeile besonders hinzuweisen:

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

Welcher Wert wird in dieser Zeile der Variablen accY zugewiesen?

Dazu wollen wir uns zunächst eine Skizze betrachten:

Speicherbedarf einer 16 Bit Integer-zahl															
Wertebereiche 0 ... 65534 (vorzeichenlos) oder mit Vorzeichen -32768 ... 32767															
high Byte								low Byte							
high Nibble				low Nibble				high Nibble				low Nibble			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Das nebenstehende Bild soll den Speicheraufbau einer vorzeichenlosen integer Zahl darstellen. Ein Integerzahl besteht aus 2 Byte und ein Byte aus wiederum zwei nibble, also vier Bit. Low Nibble 0...3 und High-Nibble 4..7.

Die Bits 0 ... 7 werden als LowByte bezeichnet. Die Bits 8 ... 15 werden als High-Byte bezeichnet. Wenn man nun einer integer Variablen einen Wert zuordnet, der kleiner als 256 ist, dann wird dieser in den Bytes 0 ... 7 abgelegt. Die Funktion `wire.read()` gibt einen Wert von 8 Bit zurück. Das erste Byte, das nun vom Sensor zurückgegeben wird ist das High-Byte. Dieses wird also nach dem Einlesen im unteren Bereich (low Byte) gespeichert. Nun ist der Befehl `Wire.read() << 8` notwendig.

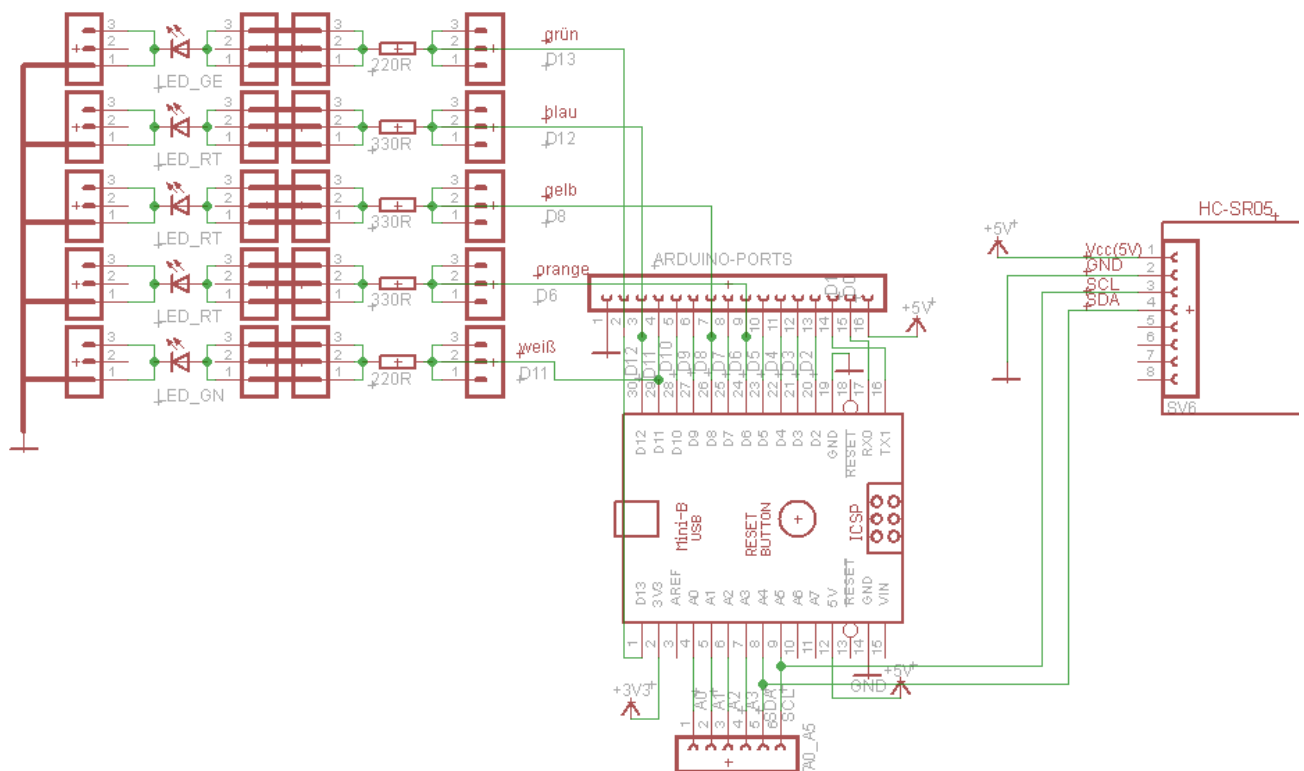
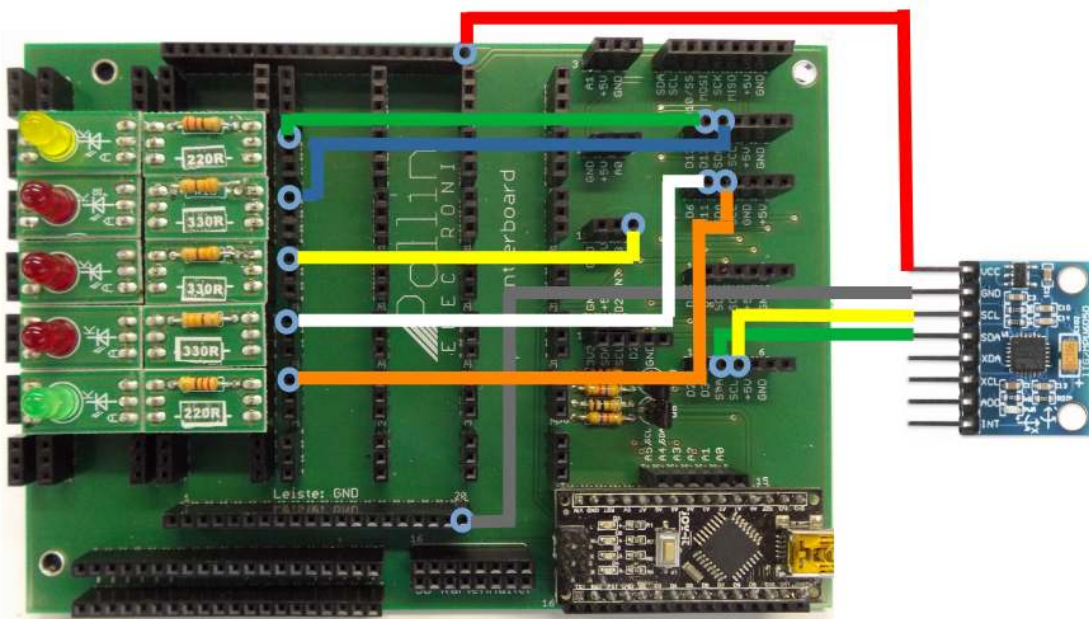
Denn `<< 8` bedeutet, dass in der Variable `accY` der eingelesene Wert um 8 Stellen nach links verschoben wird. Er steht dann im oberen, dem sogenannten High-Byte. Wenn man nun den zweiten Byte-Wert mit der oder-Verknüpfung `|` in die Variable `accY` einliest, dann wird er im unteren Byte abgelegt. Die Oder-Verknüpfung ist quasi eine Addition. Das ist deshalb so umständlich, weil der Arduino nur mit 8Bit arbeitet.

Digitale Wasserwaage MPU6050_wasserwaage.ino :

Wenn 0, dann leuchtet mittlere LED, wenn negativ LED links, wenn positiv LED rechts;
Das Programm soll hier nur der Verdeutlichung des Messprinzips dienen. Die exakten mathematischen Herleitungen wurden weggelassen.

Es wurde lediglich im setup() Mittelwerte gebildet und der Beschleunigungswert geteilt, damit in etwa eine Grad-Anzeige herauskommt und auch und die LEDs sich bei einer kleinen Bewegung nicht sofort großartig verändern.

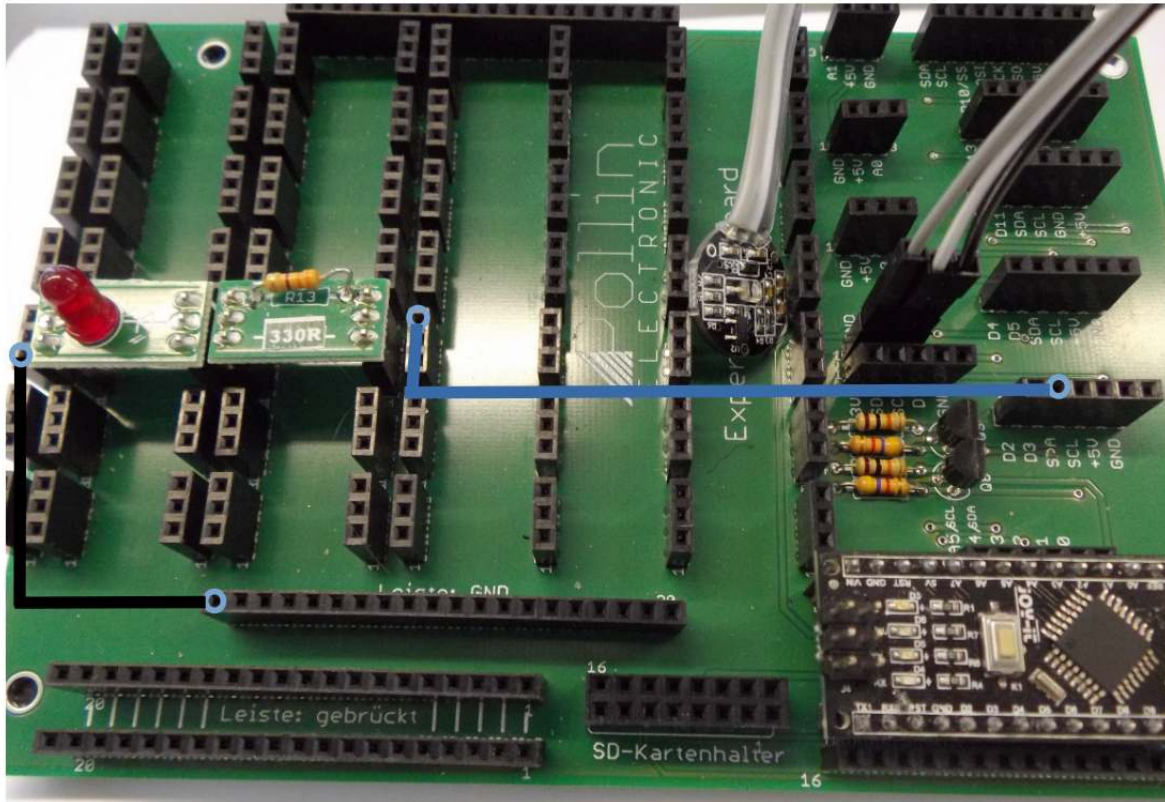
In der Funktion **void** init_MPU6050(**void**) wird der Sensor initialisiert. Die Initialisierung könnte viel mehr Register umfassen, aber für den Anfang reicht es aus, diese Werte nach eigenen Wünschen zu ändern und zu probieren, welchen Einfluss die Änderung auf das Messergebnis bewirkt.

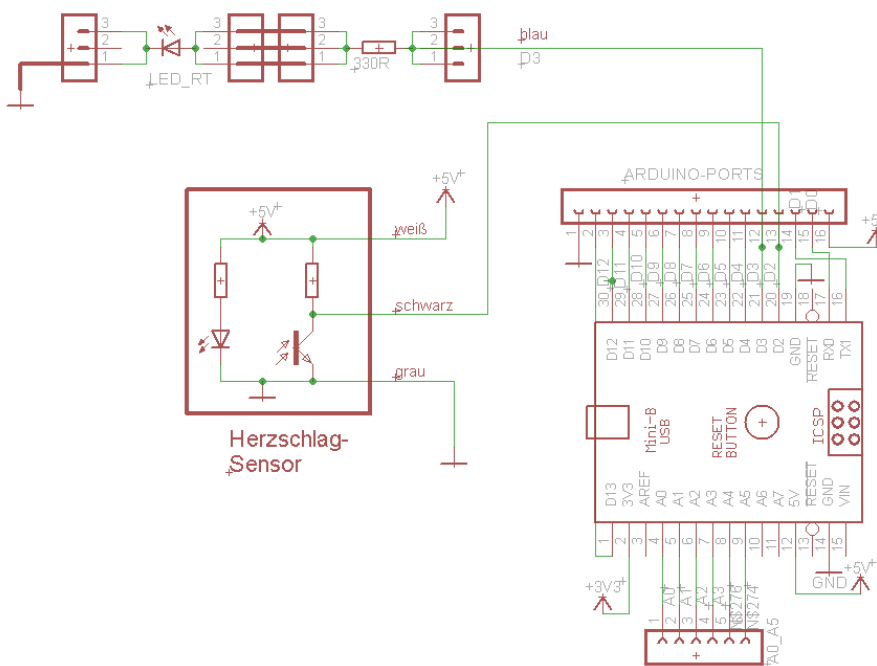


12.7. Herzschlagsensor (Artikelnummer 810917)



Der Herzschlagsensor wird mit drei Leitungen zum arduino verdrahtet. GND (schwarz) 5V (rot) und A0 (blau); Das Programm [Herzschlag.ino](#) liest die Daten an A0 ein. Die Werte kann man sich mit dem Werkzeug → serieller Plotter darstellen lassen.





Der Herzschlag-Sensor funktioniert nach dem Prinzip einer Lichtschranke. Nur wird an der Empfangs-LED der Spannungsabfall gemessen. Denn der Spannungsabfall an der Diode steigt mit dem Strom. Der Strom durch die Diode wird beeinflusst von der Stärke des empfangenen Lichtstrahls. Das Signal wird zudem noch verstärkt und liefert so ein Signal, das vom durchströmten Blut abhängig ist. Bei der Einstellung des Grenzwertes für die LED ist ein bisschen Experimentierfreude gefragt.

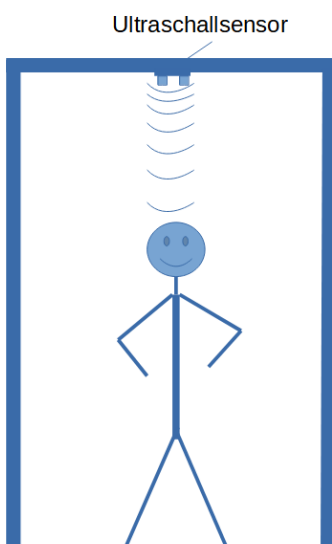
12.8. Ultraschallsensor (Artikelnummer 810481)



Hierbei nutzt man die Schallgeschwindigkeit, um Entfernungen zu messen. Es wird ein 10µs kurzer Impuls erzeugt und die Zeit gemessen, die vergeht, bis das Echo empfangen wird. Aus der Hälfte der Laufzeit kann die Entfernung berechnet werden, weil man annimmt, die Schallgeschwindigkeit ist immer gleich. In gewissem Rahmen ist sie das auch, aber eben auch von der Raumtemperatur abhängig. Bei konstanter Raumtemperatur

bleibt der Fehler dann konstant und könnte so auch herausgerechnet werden. Das Beispiel [HC-SR04.ino](#) setzt diese Theorie um.

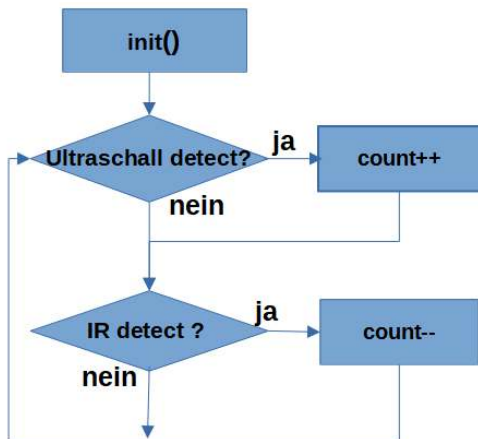
12.8.1. Durchgangshöhenmesser



Für das Beispiel [HC-SR04_altitude.ino](#) wurde eine Funktion `trigger()` eingeführt um die Befehle zur Generierung des Triggerimpulses in eine Funktion zu bringen. Um die Größe einer Person bestimmen zu können, muss in der Konstanten Variablen die Höhe in [mm] eingegeben werden, auf der der Sensor montiert ist. Der Sensor muss natürlich so montiert werden, dass er nach unten zeigt. Die Größe der Person wird bestimmt in dem laufend die „Distanz“ gemessen wird. Eine Verbesserung des Programms ist, wenn Daten gesammelt werden und der kleinste Wert angezeigt wird. Dann kann man wirklich davon ausgehen, dass die Körpergröße einer Person erfasst wurde. Wenn man nun davon

ausgeht, dass die Personen die durch die Tür kommen, eine Mindestkörpergröße haben, dann kann auch der Messwert diesbezüglich eingeschränkt werden.

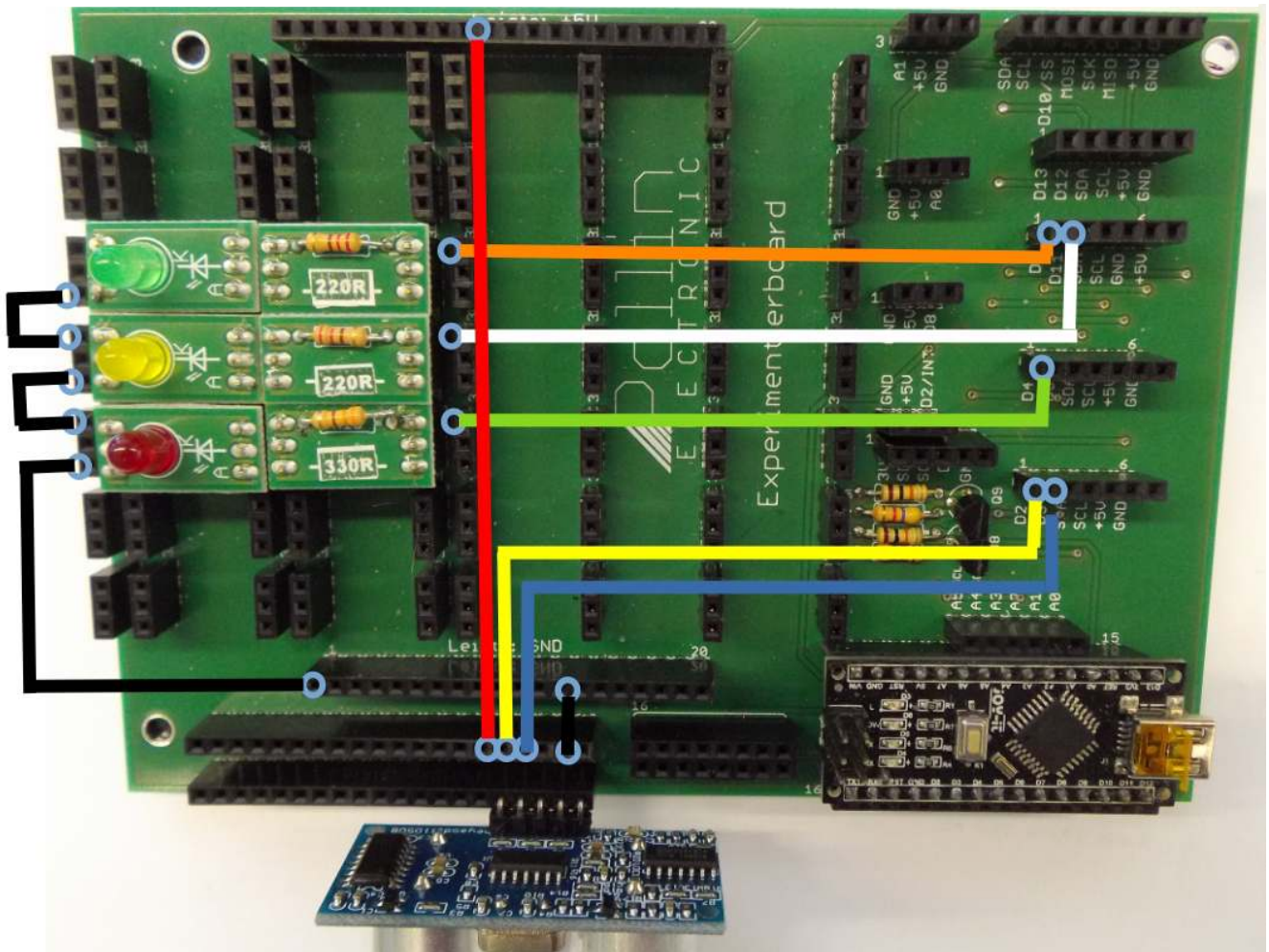
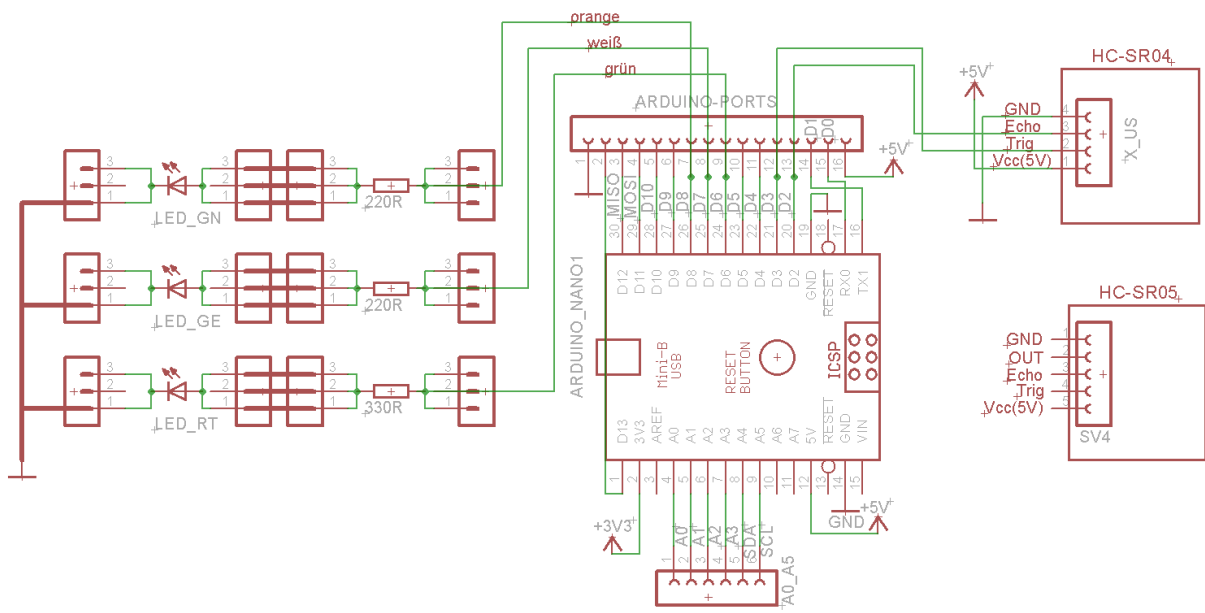
12.8.2. Personenzähler



Im Beispiel [HC-SR04_counter.ino](#) ist das Programm [HC-SR04_altitude.ino](#) nur ein wenig ergänzt worden. Wenn eine bestimmte Körpergröße gemessen wurde, wartet das Programm, bis die gemessene Entfernung wieder eine bestimmte Entfernung überschreitet, dann kann davon ausgegangen werden, dass eine Person den Sensor durchschritten hat. Allerdings ist so nicht feststellbar, ob eine Person den Raum betreten oder verlassen hat. Dazu bräuchte man einen zweiten Sensor. Sind beide auf einer bestimmten Höhe nebeneinander montiert, dann könnte man feststellen aus welcher Richtung die Person sich in den Raum bewegt.

Dazu könnte als zweiter Sensor der Abstandsensor aus Kapitel 12.9. verwendet werden. Dazu müssten die beiden Sensoren in gleicher Höhe am Türrahmen befestigt werden; einer im Raum (z.B. IR) und der andere außen (z.B. Ultraschall); je nachdem welcher Sensor zuerst auslöst, kann festgestellt werden, ob die Person den Raum betritt oder verlässt.

12.8.3. Einparkwarner



Mit einem Ultraschallwandler, drei LED's (grün, gelb, rot) und einem Piepser soll ein Einparkwarner aufgebaut werden. Diesen kann man an einer Garagenwand anbringen und als Einparkhilfe, beim rückwärts einfahren nutzen. Die Anzeige soll bis zu einer Entfernung von 50cm

grün sein; ab 25 cm gelb leuchten, eine Ton mit 50 zu 50 Puls Pausenverhältnis ausgeben; und bei 10 cm rot leuchten und einen Dauerton ausgeben.

Im Beispiel **HC-SR04_parking.ino** ist das Herzstück die Funktion `measure()`. Zuerst wird geprüft, ob die Daten eine gewisse Distanz unterschreiten. Diese Distanz wird mit `#define ignore xxx` festgelegt. Dann werden mit einer `if`-Bedingung die Entfernungen festgelegt, ab wann welche LED angesteuert wird und ab welcher Distanz der Piepser ein Signal abgibt.

Ein wichtiger Punkt, der sich erst bei der Programmierung ergab ist die Art der `if` Abfragen:

```

if (Bedingung)
  { ... }
else if (Bedingung)
  { ... }
else if (Bedingung)
  { ... }
else
  { ... }

```

würde `else` vor dem `if` fehlen, wären z.B. bei einer Distanz `== 5 cm` alle Bedingungen erfüllt und es würden alle Befehle ausgeführt, weil dann alle abgefragten Bedingungen erfüllt sind. Deshalb das `else` vor dem `if`, dann wird praktisch nur eine der aufgeführten Bedingungen erfüllt.

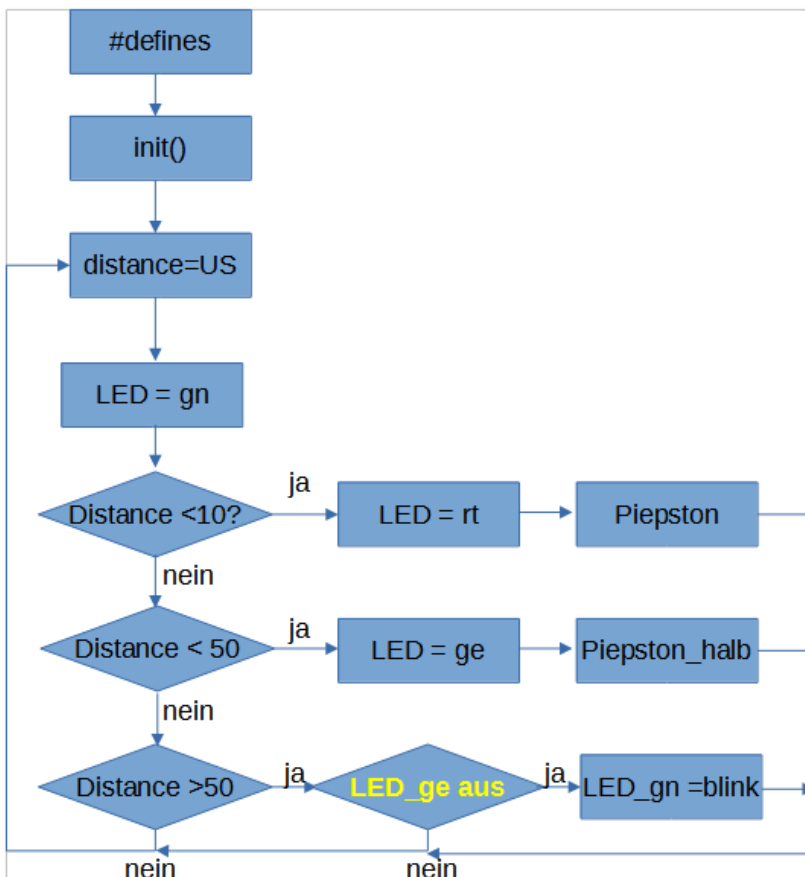
Ferner gilt es zu erwähnen, dass zwei neue Funktionen benutzt wurden:

`tone(AusgabePin, Frequenz [Hz], Dauer [ms])`

und `noTone(AusgabePin)`.

Bei der Funktion `tone()` ist die Variable `Dauer` nur eine Option. Wird diese Variable nicht an die Funktion übergeben, ertönt der Ton so lange, bis er mit `noTone` abgeschaltet wird.

Warum ist es wichtig, dass eine Variable für die gelbe LED abgefragt wird, also ob **LED_ge aus** ist ?



Das soll eine Art Software-Hysterese sein, dass im Grenzbereich nicht gelb und rot abwechselnd leuchten;

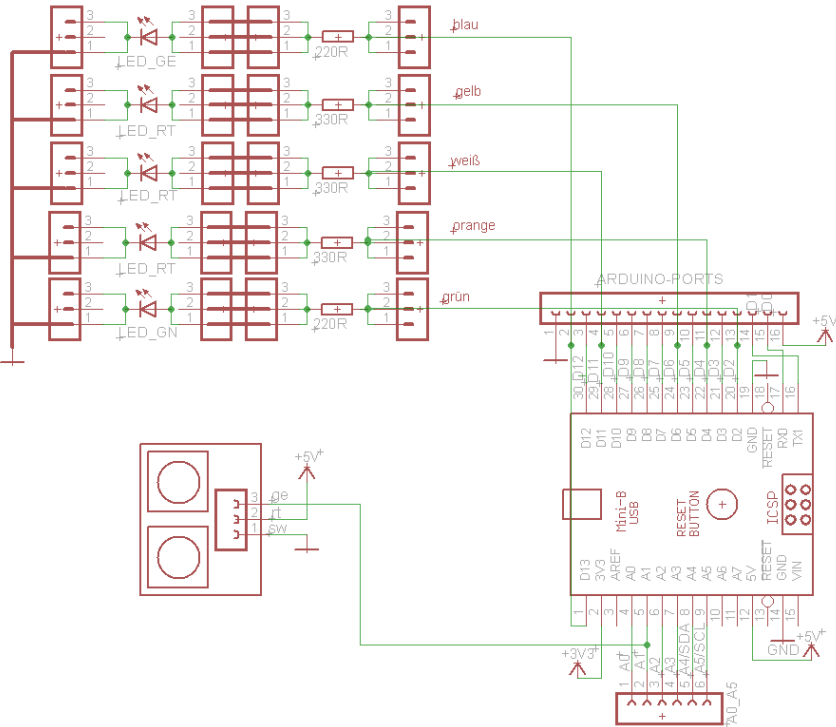
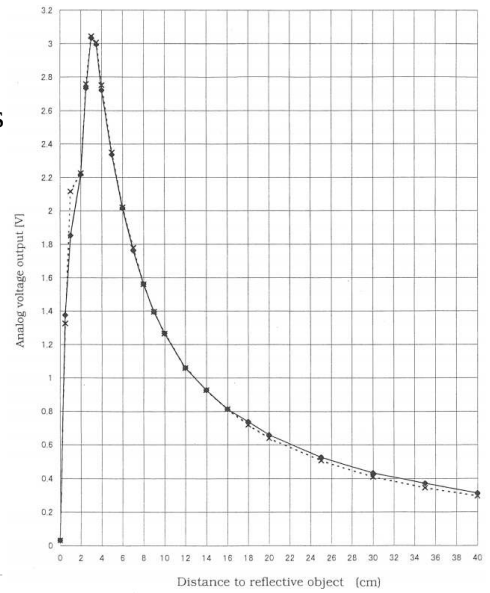
Annahme: beim Rückwärtsfahren kann ja nicht sein, dass sich der Abstand plötzlich wieder vergrößert.

12.9. Infrarotsensor (Artikel 810480)

Abstandsmessung



Um den Abstand korrekt anzuzeigen, muss man wissen, wie die Kennlinie des Sensors aussieht. Im Gegensatz zum Ultraschall Sensor ist dieser Sensor nicht linear. Der Messbereich dieses



Sensors liegt im Bereich unter 70cm. Allerdings hat der Sensor eine hohe Genauigkeit und ist Temperaturunabhängig. Diese Art von Sensor wird überwiegend in der Automatisierungstechnik verwendet, da er sehr zuverlässig ist und die Daten analog abgefragt werden können und nicht von einer Laufzeit abhängig sind.

Der Sensor ist einfach anzuwenden, weil das Ausgangs-Signal des Sensors eine analoge Spannung ist. Leider ist es nicht linear, kann aber mit folgender

Näherungsformel beschrieben werden:

$$IRDistance = 28.153 * \text{pow}(\text{voltage}, -1.175)$$

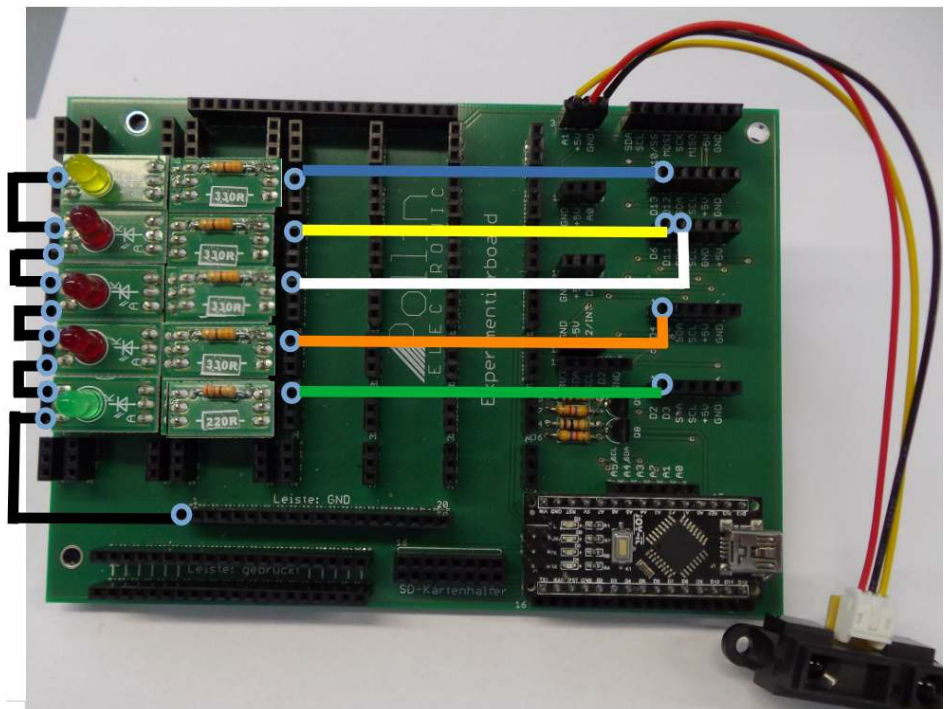
Das Programm

[IR_mess.ino](#) verwendet eine Annäherung an die Kurve. Diese gilt für den Sensor von Joy-it. Nicht für andere z.B. von sharp.

Die Funktion Exponentialfunktion `pow()`:

$$z = \text{pow}(x, y);$$

bedeutet $z = x^y$;



12.10. Infrarotlichtschranke SEN-KY033LT (Artikel 811269)

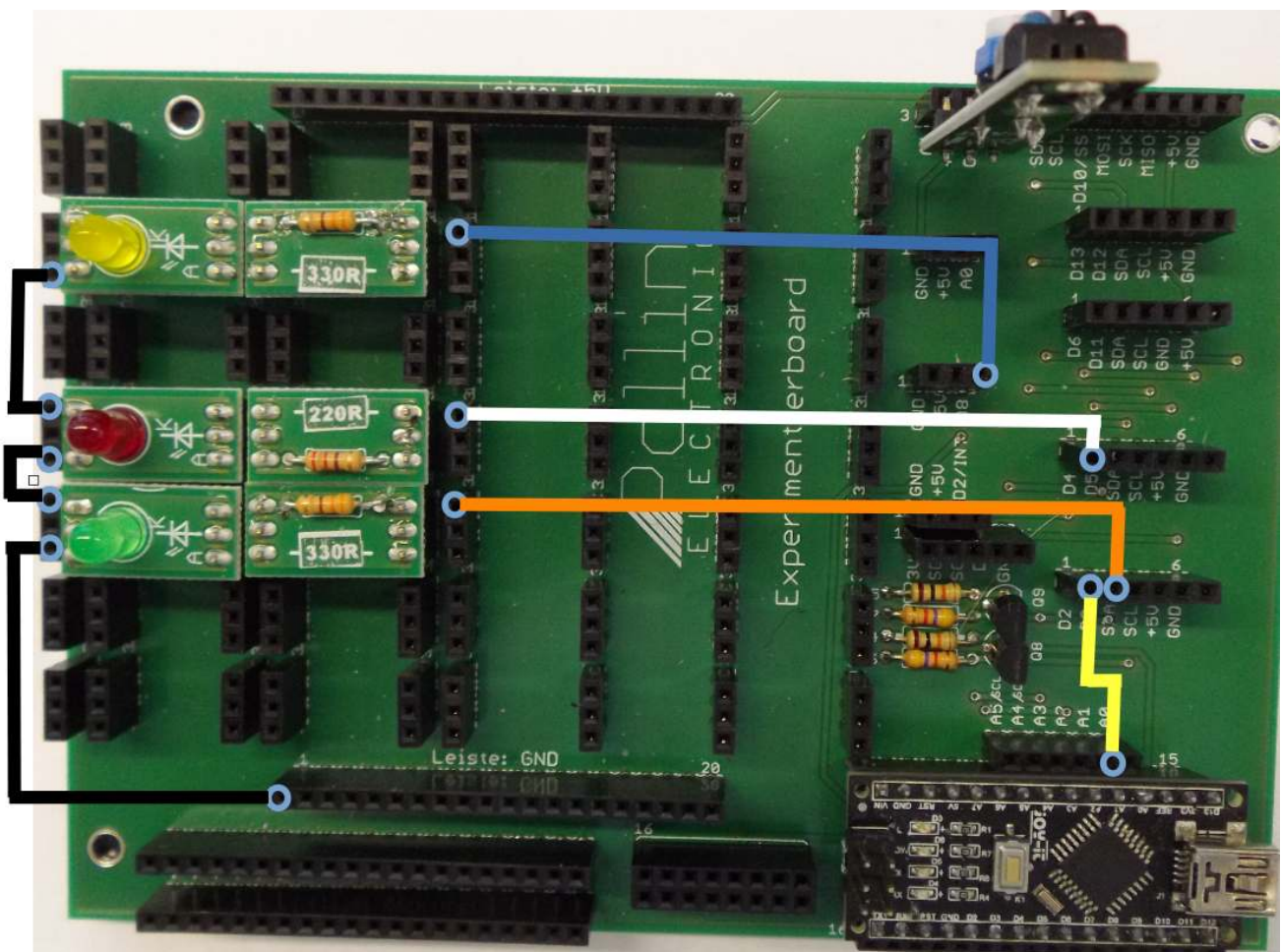
Dieser Sensor sendet und empfängt Infrarotsignale. Dabei kann er erkennen, ob er eine reflektierende Fläche (der Sensor empfängt sein eigenes Signal) oder eine absorbierende Fläche (der Sensor empfängt kein Signal) vor sich hat. Die Empfindlichkeit des Sensors ist mit Hilfe eines Potentiometers einstellbar. Roboter zum Beispiel können mit Hilfe von zwei dieser Sensoren automatisiert einer Line folgen.

Wir versuchen mit diesem Sensor einen Strichcode zu erkennen.

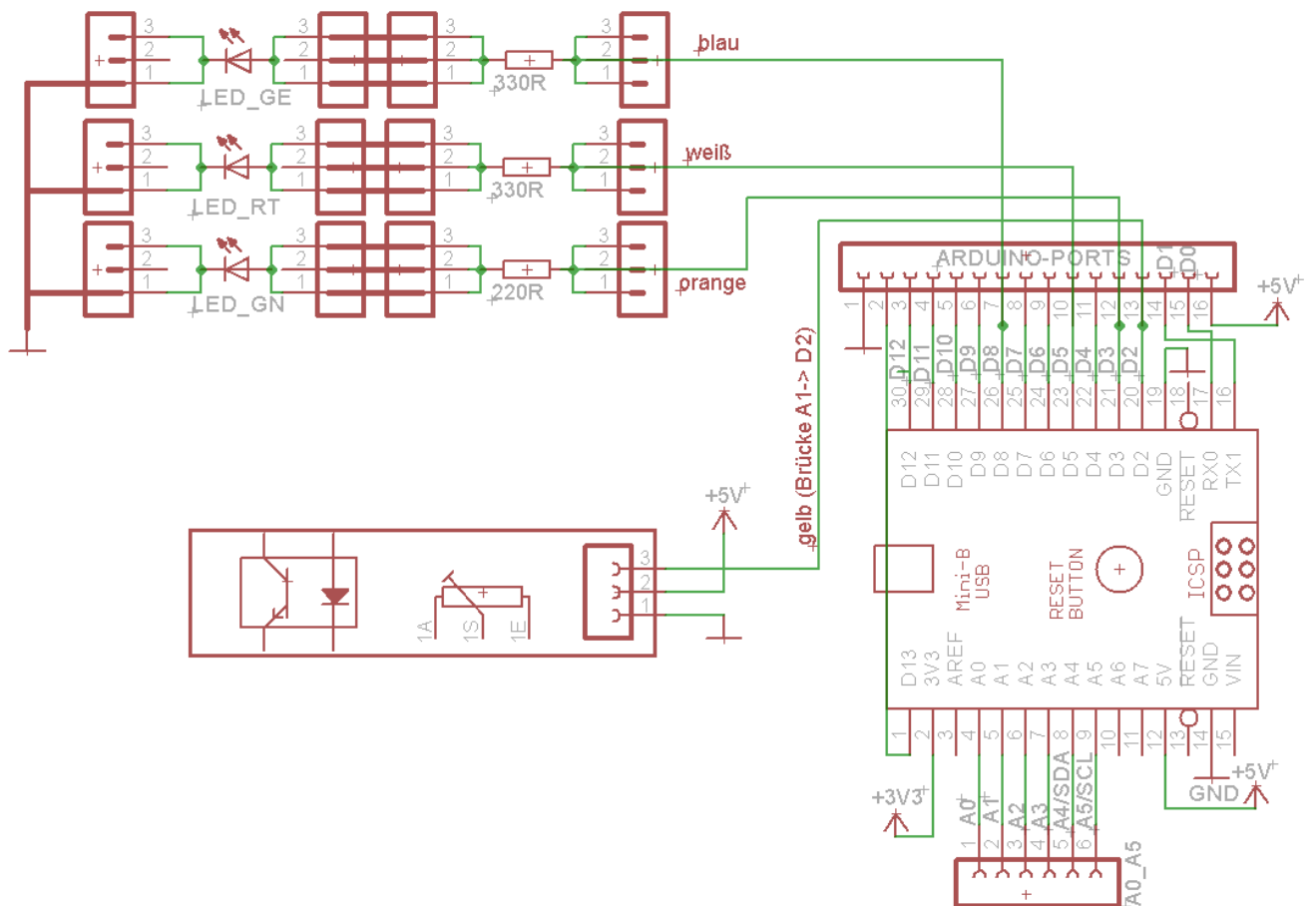
Die Anschlüsse:



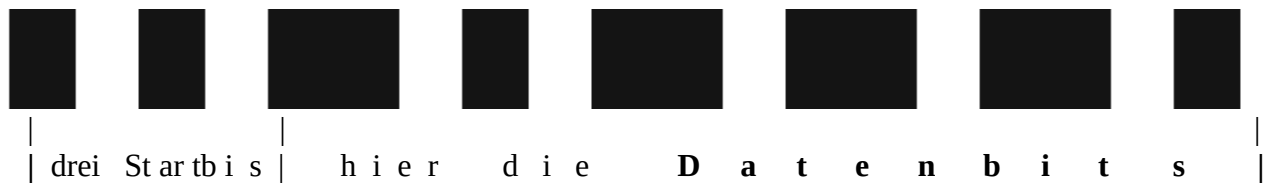
S: Signal an arduino Pin2
V+: an arduino +5V
G: an arduino GND



Beim Schaltungsaufbau ist zu beachten, dass der Sensor in eine Buchsenleiste am Rand gesteckt ist. Damit soll es erleichtert werden, den Strichcode vorbeizuziehen. Allerdings ist das Signal an Pin_A1 des Arduino geschaltet. Dieser Pin ist ein Eingang, deshalb stört er nicht, weil das Signal an D2 weitergeschleift werden muss. Es benötigt somit die Brücke von A1 nach D2. Die LED's sind als Signal gedacht, wenn ein Strichcode gelesen wurde (LED gelb), ein Strichcode erkannt wurde (LED grün) oder der Strichcode nicht zum Zugang berechtigt (LED rot).



Zuerst ist der Sensor abzustimmen. In diesem Fall ist er so eingestellt, dass das Poti so weit nach links gedreht wurde, bis die LED leuchtet, wenn sich der Finger ca. 1cm vor dem Sensorpaar befindet. Wird das Poti nach rechts gedreht, erhöht das die Empfindlichkeit des Sensors, z.B. wie er im obigen Bild eingestellt ist. Als Beispiel zur Nutzung dieses Sensors gibt es [Strichcode.ino](#). Am Sensor wird dabei ein Blatt Papier mit Strichcodes vorbeigezogen. Die Strichfolge beginnt immer mit zwei gleich großen schwarzen Flächen. Daraus versucht der arduino die Zeit zu berechnen, mit der er den Sensor abtasten muss, um den korrekten Code zu erhalten. Als Erweiterung könnte nun eine Funktion sein, die bei einem bestimmten Code eine LED einschaltet. Lösungsvorschlag ist [Strichcode2.ino](#). Zu beachten ist, dass zu einer Ansteuerung eines Relais der passende Transistor verwendet wird. Hier noch ein Beispiel für einen Pseudo- Barcode:

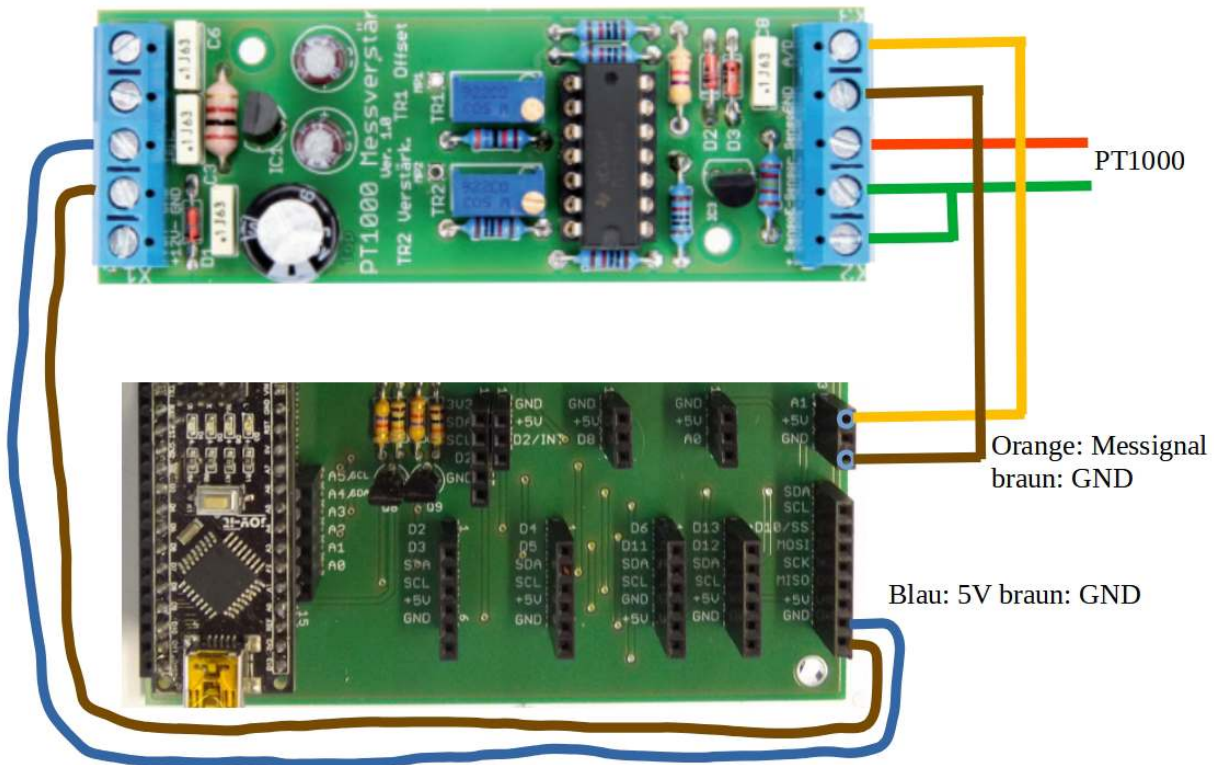
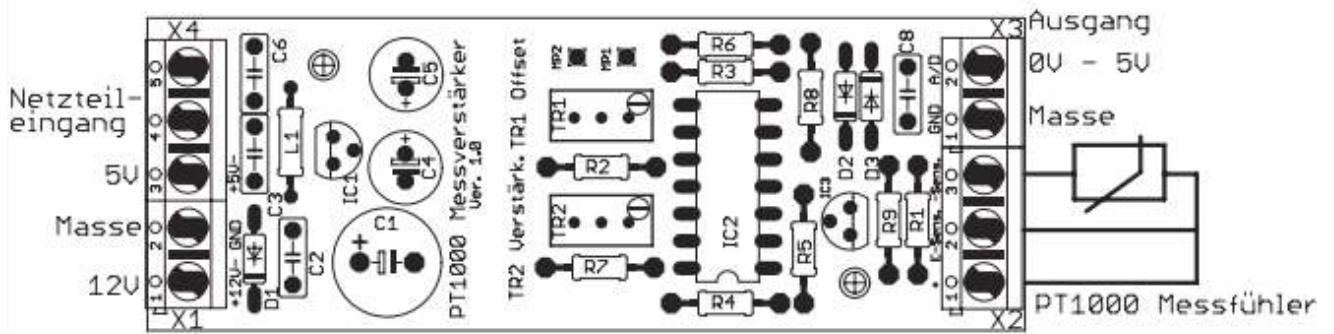


Ende

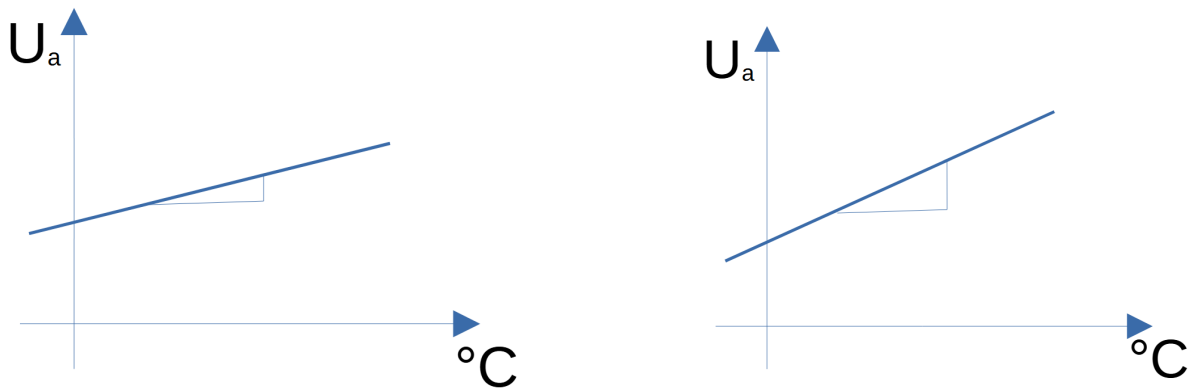
Allerdings ist bei dieser großen Anzahl an Bits, die Schwierigkeit, den Streifen in gleichmäßiger Geschwindigkeit am Sensor vorbeizuziehen. Vielleicht wäre es am Anfang besser, weniger Bits zu verwenden und auch die Flächen größer (länger) zu machen. Es ist wie immer, ein wenig Experimentierfreude gefragt.

12.11. Temperaturmessung mit PT1000

Hierzu verwenden wir einen PT1000 Sensor mit der Artikelnummer 180016 und dem Messwandler Bausatz mit der Artikelnummer 810144. Der Bausatz hat die Aufgabe, das Signal des Sensors zu verstärken und somit für den Arduino messbar zu machen. Der PT1000 ist ein sehr genauer Sensor, aber die Signaländerung bei geringer Temperaturschwankung ist sehr klein. Zum Filtern des Ausgangssignals kann ein 47µF Elko zwischen Masse X3:1 und X3:2 (Messsignal) nicht schaden.



Der Messwandler ist ein Bausatz. Daher ist nach dem zusammen Löten zwingend notwendig, die Platine abzugleichen. In der Beschreibung des Bausatzes wird als Ableichwiderstand 1k für 0°C und 1940R für 250°C empfohlen. Wir wollen hier jedoch keine so hohen Temperaturen messen. Daher kann ein niedrigerer Temperaturwert und dementsprechend ein niedrigerer Widerstand verwendet werden. Der Hintergedanke ist, dass der Arduino die Messspannung nur unzureichend genau messen kann. Er hat einen Messbereich von 0 ... 5V (V_{ref}). Dieser ist in 1024 Abstufungen unterteilt, was eine Auflösung von 5mV entspricht. In dem hier verwendeten Arduino Nano wurde am Pin: V_{ref} eine Spannung von 4,3V gemessen. Also kann der Arduino $4,3V/1024$ ist in etwa 4mV je Spannungsstufe auflösen.



Um eine größere Spannungsänderung zu erreichen, kann der Verstärkungsfaktor am Messwandler erhöht werden, was aber den Messbereich verkleinert. Die Steigung der Geraden $\Delta U_a / \Delta ^\circ\text{C}$ ist ein Maß für den Verstärkungsfaktor. Das ist in den oben dargestellten Grafen zu erkennen. Links im Bild ist der Verstärkungsfaktor relativ klein. Ein großer Temperaturunterschied bewirkt eine relativ kleine Spannungsänderung.

Rechts im Bild ist der Verstärkungsfaktor etwas größer gewählt.

In beiden Darstellungen ist die Temperaturänderung in etwa gleich. Aber in der rechten Abbildung ist die Spannungsänderung fast doppelt so groß wie im linken Bild. Das liegt an der Steigung der Geraden, also am Verstärkungsfaktor.

Eine Temperaturänderung ist somit leichter für den Arduino zu erkennen. Aber bei zu großer Temperaturänderung wird der Messbereich des Arduino natürlich schneller überschritten. Dies muss vermieden werden, sonst könnte der Arduino beschädigt oder gar zerstört werden, vor allem wenn die Messspannung die 5V Versorgungsspannung des Arduino übersteigt. Deshalb wird in diesem Fall die Spannungsversorgung der Messwandlerplatine von den 5V des Arduino übernommen. So sollte dann auch der Arduino geschützt sein.

Wir definieren als einen gewünschten Messbereich der Temperatur: 0 ... 50°C.

Um den Messwandler für unseren Anwendungsfall anzupassen, benutzen wir Festwiderstände, um die Temperaturen zu simulieren. Bei der oberen Temperatur nehmen wir einen mit 1k, also 1000 Ohm und schalten einen 220R in Serie. (entspricht ca. 57°C) und einen Widerstand bestehend aus 1000R und 39R in Serie geschaltet. Dies soll ca 8°C entsprechen. Da die Widerstände Toleranzen unterliegen, muss man entweder den Widerstandswert mit einem Multimeter nachmessen, oder die Toleranzen akzeptieren. Diese liegen bei jedem Widerstand bei 5%. Somit könnte sich Schlimmstenfalls ein Messfehler von 10% ergeben.

Beginnen wir nun mit der Kalibrierung der Messwandlerplatine. Wer kein Multimeter zur Hand hat, kann das Programm [PT1000.ino](#) benutzen. Sobald das Beispielprogramm geflasht und gestartet ist, kann der serielle Monitor mit <STRG> + <SHIFT> + <M> geöffnet werden und der aktuelle Messwert vom Pin AN1 des Arduino im gerade geöffnetem Fenster abgelesen werden.

Wir klemmen an den Messeingang X2:2 und X2:3 einen Widerstand von 1k und 39R in Serie und verändern mit Trimmer TR1 die Spannung am Messpunkt MP1. Diese messen wir entweder mit einem Multimeter oder verbinden ihn mit dem Arduino:AN1.

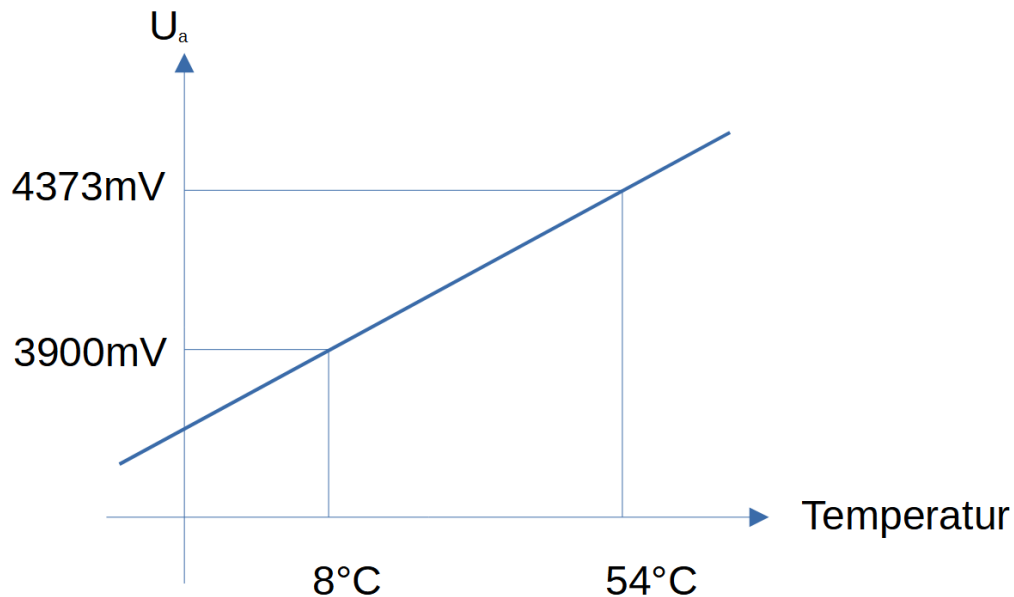
Jetzt reduzieren wir die Spannung durch Drehung nach links so lange, bis keine Änderung mehr feststellbar ist.

Nun wird der Widerstand von 39R durch einen mit 220R ausgetauscht und somit ergibt sich ein Wert von 1,22 Kohm. Der analoge Eingang des Arduino wird mit dem Messausgang verbunden (gelbe Leitung). Jetzt ändern wir am Trimmer TR2 die Messspannung: Wir messen die Spannung am Messausgang X3:1 (GND) und X3:2 und drehen dabei so lange TR2 nach links, bis auch die Messspannung nicht mehr erhöht. Dies wird bei ca. 4.3V der Fall sein, abhängig was wir als Multiplikator eingestellt haben: $\text{multiplikator} * 1024 = \text{Spannungswert des Arduino}$.

Jetzt beginnt das Kapitel mit der Mathematik. Doch zunächst messen wir mit dem Multimeter die

Werte der Widerstände und nach erfolgtem Abgleich notieren wir die Spannungen, die der Arduino anzeigt : R1 (1k+220R): **1209** Ohm → T1: **54** °C; Spannung U_a : **4373** mV
 und bei R2 (1k + 39R): **1033** Ohm → T2: **8** °C; Spannung U_a : **3900** mV

Die Werte der Tabelle tragen wir in ein Diagramm ein:



Bestimmen wir nun die Parameter für die Gerade:

Zuerst bestimmen wir die Werte der Widerstände mit dem Multimeter: 1209 Ohm ergibt aus der Tabelle_1 in der Bausatzbeschreibung einen Temperaturwert von ca. 54°C.

Der andere Widerstand, bestehend aus der Reihenschaltung ergab 1033 Ohm, was einer Temperatur von ca. 8°C entspricht. Die Spannungswerte werden dann ins Diagramm eingetragen. Siehe Abbildung oben.

Um die Geradengleichung $y = a * x + b$ zu bestimmen, widmen wir uns zuerst der Steigung der Geraden: $a = \Delta x / \Delta y$; die Steigung ergibt sich aus der Division von der Spannungsdifferenz und der Temperaturdifferenz.

$a = (4373\text{mV} - 3900\text{ mV}) / (50^\circ\text{C} - 8^\circ\text{C})$ damit ergibt sich eine Änderung der Spannung: von ca. 11,2mV je Temperaturänderung von einem °C.

Um den Nulldurchgang bestimmen zu können setzen wir eine Koordinate in die Gleichung ein. Somit ergibt sich für $y=4373\text{mV}$ und $x=50^\circ\text{C}$

$$b = y - a * x = 4373\text{ mV} - 11,2\text{mV}/^\circ\text{C} * 50^\circ\text{C} = 3813$$

Diese Werte tragen wir in die Formel im Programm ein:

Nullspannungsdurchgang vom Messwert abziehen $t_{\text{mess}} = (t_{\text{mess}} - 3813)$;

Nun kann der Spannungswert in einen entsprechenden Temperaturwert umgerechnet werden:

$$\text{Temp} = \text{Messwert} / 13\text{mV}/^\circ\text{C};$$

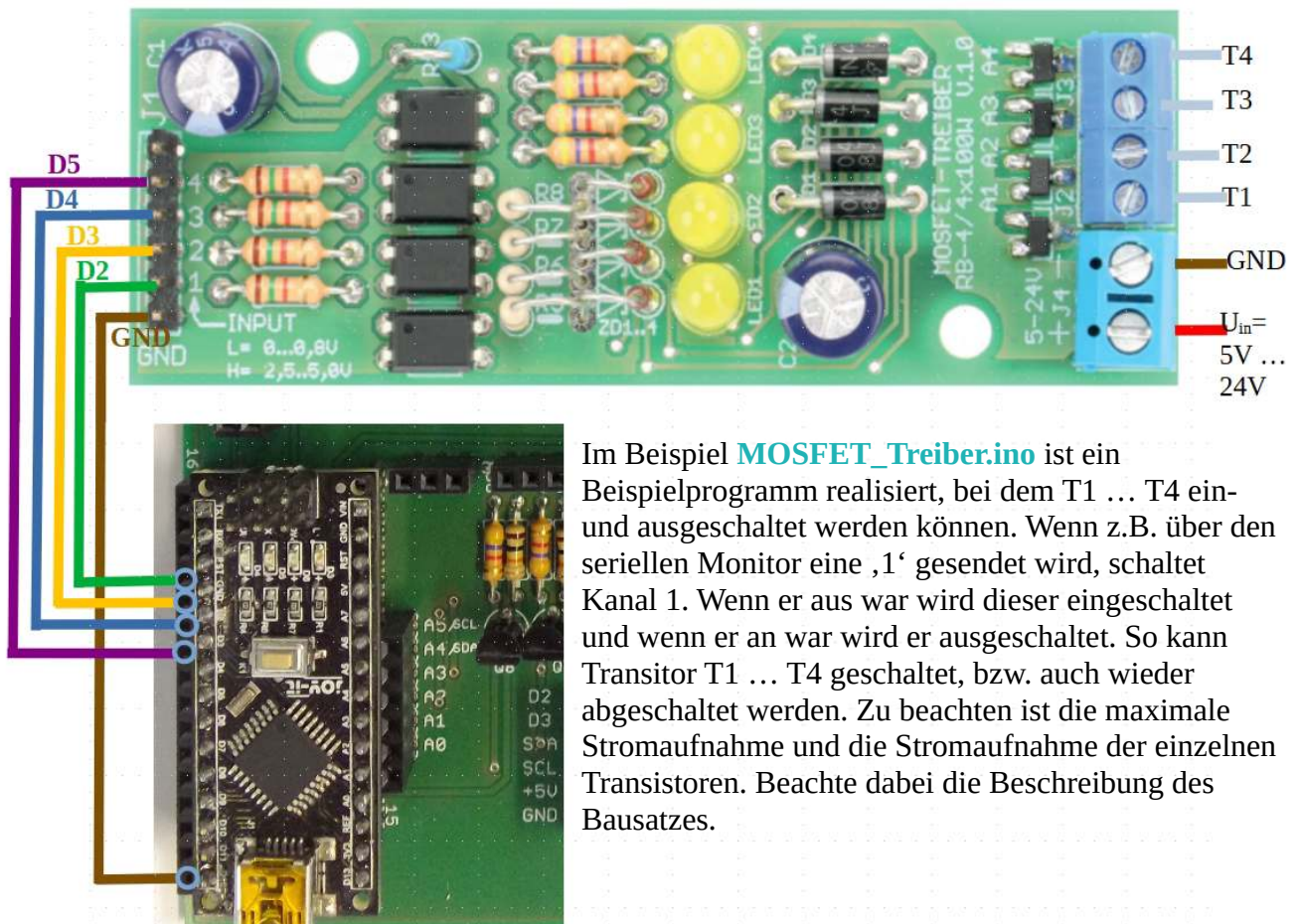
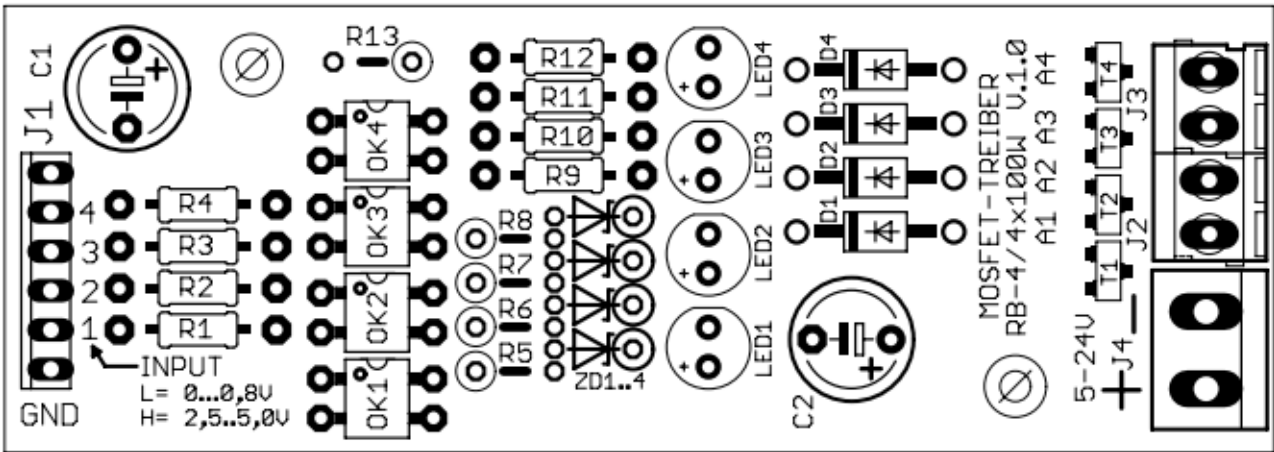
Nun klemmen wir wieder die beiden Festwiderstände an den Messeingang X2:2 und X2:3. Nun kann kontrolliert werden, ob die Widerstände auch den Temperaturen aus der Tabelle 1 in der Bausatzbeschreibung entsprechen. Denn jetzt müsste jeweils die Temperatur angezeigt werden, die dem Widerstandswert entspricht. Ansonsten ist ein neuer Abgleich oder eine Neuberechnung der Werte erforderlich.

Falls die Werte entsprechend den Toleranzen passen, kann der PT1000 angesteckt werden und so sollte dann auch die Raumtemperatur angezeigt werden.

Im Programm **PT1000_02.ino** wurde nur die Darstellung der Messwerte verändert.

12.12. MOSFET-Treiber (Artikelnummer 810329)

Da ein Mikrocontroller nur sehr geringe Ströme liefert, können größere Leistungen nur mit Relais oder MOSFET's geschaltet werden. Eine Möglichkeit größere Lasten zu schalten bietet der Bausatz MOSFET-Treiber. Damit können bis zu acht Lasten (Bausatz 810403) unabhängig voneinander geschaltet werden. Hier benutzen wir den Bausatz 810329 mit nur vier Kanälen.

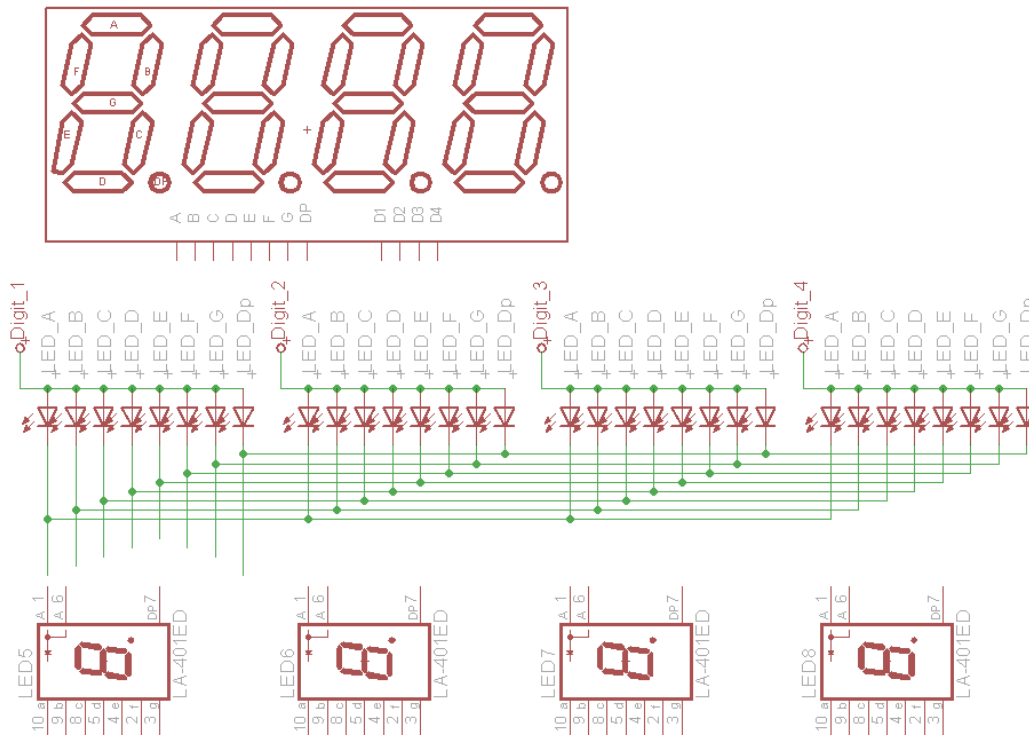


Im Beispiel [MOSFET_Treiber.ino](#) ist ein Beispielpogramm realisiert, bei dem T1 ... T4 ein- und ausgeschaltet werden können. Wenn z.B. über den seriellen Monitor eine ‚1‘ gesendet wird, schaltet Kanal 1. Wenn er aus war wird dieser eingeschaltet und wenn er an war wird er ausgeschaltet. So kann Transistor T1 ... T4 geschaltet, bzw. auch wieder abgeschaltet werden. Zu beachten ist die maximale Stromaufnahme und die Stromaufnahme der einzelnen Transistoren. Beachte dabei die Beschreibung des Bausatzes.

Dieses Beispiel kann auch verwendet werden, um die Relaiskarte mit der Artikelnummer 810273 anzusteuern.

12.13. Die Sieben Segment Anzeige

Aufbau und Funktionsweise



Eine 7-Segmentanzeige besteht, wie der Name schon sagt, aus sieben LED-Balken. Es gibt sie in zwei Ausführungen: common-anode oder common-cathode. Das bedeutet, entweder haben die einzelnen LED's eine gemeinsame Masse, dann muss zur Ansteuerung an den Anschlüssen ein HIGH Signal anliegen. Bei einer gemeinsamen Anode, werden zur Ansteuerung der Balken, die entsprechenden Pins auf Masse geschaltet. Im Schema ist auch zu sehen, welcher Pin, welchen Balken (a ... g) ansteuert. Der Anschluss p ist der Punkt nach jedem Segment. Es gibt auch Anzeigen, mit einem **:** zwischen zwei Segmenten. Damit lassen sich Zeiten gut darstellen. Im obigen Bild ist der Schaltplan der 7-Segment Anzeige 5641BS dargestellt, welche wir hier verwenden. Die Pins 12, 9, 8 und 6 sind die einzelnen Anoden der Segmente. Es ist gut zu erkennen, wie die Kathoden aller vier Segmente zusammen gefasst sind. Diese Schaltungstechnische Maßnahme hilft dabei, Pins des Prozessors einzusparen. Der Vorteil an so einer Art Display ist, das sehr geringe Preis. Aber dabei mehrstellig Zahlen darzustellen, fordert ein wenig mehr Programmieraufwand, als nur einen Ausgang auf HIGH oder auf LOW zu legen. Es ist zu beachten, dass die Segmente nicht zeitgleich angesteuert werden können. Der Programmierer ist dafür verantwortlich, dass zum richtigen Zeitpunkt, wenn eine gemeinsame Anode auf „High“ geschaltet wird, das richtige Signal an die Kathoden angelegt wird.

Da dieses Problem viele betrifft, wurde dafür ein IC entwickelt, das dem Programmierer diese Aufgabe abnimmt. Es ist das TM1637 von Titan Microelectronics. Dies ist für den Arduino weit verbreitet. Damit werden nicht nur alle Segmente angesteuert, sondern es ist auch möglich die Helligkeit der Anzeige zu steuern. Denn bei Dunkelheit z.B. könnte es blenden, also wird das Display bei Dunkelheit gedimmt. Die dafür nötigen Funktionen werden im nächsten Abschnitt an einem Beispiel erläutert.

12.13.1. erstes Beispiel

Zu der installierten Bibliothek gibt es ein Beispielprogramm [TM1637Test.ino](#)

Hier einige Anmerkungen zu diesem Beispiel:

Zeile 18: `TM1637Display display(CLK, DIO);` //erzeuge eine Instanz display der Klasse TM1637 Display.

CLK und DIO, wobei mit CLK und DIO die Pins definiert werden, an denen CLK-Anschlusspin und DIO-Anschlusspin angesteckt werden. BitDelay ist standardmäßig auf 100ms festgelegt. Diese kann so belassen werden. Also bleibt in der Regel der dritte Parameter frei. Wenn kein Parameter übergeben wird, wird automatisch der in der Bibliothek vordefinierte verwendet.

Mit der Befehlsfolge:

```
display.setBrightness(0x07,false);
```

```
display.setSegments(data);
```

wird das Display abgeschaltet. Dabei ist der erste Parameter nach `display.setBrightness` ein Wert von 0 bis 7, wobei 0 bedeutet dunkel und 7 bedeutet volle Helligkeit. Der zweite Parameter schaltet das Display an = true oder aus = false.

Dem Befehl `display.setSegments(x, Y, z)` können drei Variablen übergeben werden

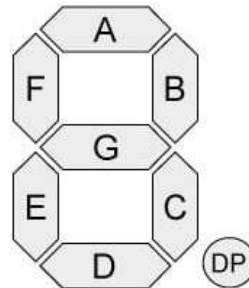
x ist die Zahl die dargestellt werden soll; zu beachten ist dabei, dass die Zahl aus 4 Zeichen bestehen kann. Jedes Zeichen ist dabei separat in einem Array zu speichern `x[0], x[1],x[2],x[3]`; Dabei wird jede Ziffer als Zeichen behandelt. Der Arduino erkennt dies nicht als Zahl. Es muss also jede Stelle definiert werden. Und jede Stelle besteht aus 7 Segmenten, wie der Name der Anzeige vermuten lässt.

Das Array wird nachfolgend definiert:

```
uint8_t data[] = { 0xff, 0x7f, 0xff, 0xff };
```

uint8_t besagt, dass jeder Wert im Array ein 8Bit integer Wert ist. Dabei ist es besonders übersichtlich die Hexzahlen in binärer Form darzustellen:

```
#define SEG_A 0b00000001
#define SEG_B 0b00000010
#define SEG_C 0b00000100
#define SEG_D 0b00001000
#define SEG_E 0b00010000
#define SEG_F 0b00100000
#define SEG_G 0b01000000
#define SEG_DP 0b10000000
```



Dabei ist in binärer Darstellung der Zahlen, gut zu erkennen, welches Stelle der Zahl welchem Segment des Displays zugeordnet ist. DP ist in unserem hier verwendeten Display der Doppelpunkt zwischen dem zweiten und dritten Digit. Deshalb wird er mit dem höchstwertigen Bit von `data[1]` angesprochen. Der wert hat `data[1]` den Wert `0xFF` sind alle Segmente inklusive des Doppelpunktes eingeschaltet. Hat `data[1]` den Wert `0x7F`, sind alle Segmente exklusive des Doppelpunktes eingeschaltet. So verwenden wir gleich zur Erklärung die Definition des Wortes „done“ aus dem Beispielprogramm [TM1637Test.ino](#) :

```
const uint8_t SEG_DONE[] =
{
    SEG_B | SEG_C | SEG_D | SEG_E | SEG_G, // data[0] = d
    SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F, // data[1] = 0
    SEG_C | SEG_E | SEG_G, // data[2] = n
    SEG_A | SEG_D | SEG_E | SEG_F | SEG_G // data[3] = E
};
```

const int8_t bedeutet, dass dieses definierte Array aus 8Bit integer Werten besteht, die im Programm nicht mehr verändert werden können. Dafür steht der Begriff **const**;

SEG_DONE ist der Name der Variablen, die das Array bezeichnet. Die eckigen Klammern [] signalisieren, dass es sich bei der Variablen um ein Array handelt. Array ist eine Reihe von Variablen, die unter einem Namen angesprochen werden können. Dabei ist jede Variable durch ein | getrennt. Beim letzten Wert entfällt dieses natürlich. Dafür muss das Ende der Variablenliste mit }; abgeschlossen werden. Der Anfang der Liste wird mit einer { festgelegt.

Die Aneinanderreihung der Konstanten SEG_C | SEG_E | SEG_G ergibt einen Wert. Dieser wird folgendermaßen ermittelt:

Der Senkrecht Strich wird erzeugt mit <ALT> + 124 , dabei sind 1, 2 und 4 die Tasten im aktivierten NUM-Block, die bei gedrückter ALT-Taste eingegeben werden.

Der senkrechte Strich | bedeutet für die Übersetzung des Programms, dass die Variablenwerte „verodert“ werden.

SEG_DONE[2] = SEG_C | SEG_E | SEG_G bedeutet, der dritten Variablen wird ein Wert zugewiesen. Die dritte Variable deshalb, weil jeder Computer, Mikroprozessor und Mikrocontroller mit 0 zu zählen beginnt. Eine Zuweisung von Werten einer Variablen oder einer Konstanten in binären Zahlen erfolgt durch folgende Anweisung:

SEG_DONE[2] = 0b00000100 = Seg_C | 0b00010000 = Seg_E | 0b01000000 = seg_G

verodern bedeutet eigentlich nichts anderes als diese Werte zu addieren. Allerdings ist es so, dass 0 und 1 ergibt 1, 1 und 1 ergibt auch nur eins! Es entsteht bei dieser Art von Addition kein Übertrag. Es wird nacheinander Stelle für Stelle „addiert“.

So ergibt sich quasi als „Summe“ folgender Wert für die Variable SEG_DONE[2] = 0b01010100; Dies entspricht im Hex-Format dem Wert 0x54;

Y ist die Variable für die Länge der dargestellten Zahl, also die Anzahl der Stellen von 1 ... 4;

z ist die Variable für die Position, aber der die Ziffern dargestellt werden 0..3 wobei 0 links bedeutet und 3 ist ganz rechts.

Der Befehl **display.clear()**; setzt alle Werte der Variablen data[] auf 0. Das Display ist in diesem Fall noch eingeschaltet. Es wird aber kein Segment aktiviert. In der Klammer () steht kein Wert, weil diese Funktion keine Variable als Übergabeparameter erwartet.

Da es recht umständlich wäre, eine Zahl in ihre Stellen zu zerlegen und der Variable data[] zuzuordnen, gibt es mehrere Funktionen, die diese Arbeit für den Programmierer erledigen.

Der Funktionsaufruf **display.showNumberDec**(zahl, führende_Null, Länge, Position); stellt die Variable Zahl im Display dar. Dabei ist Zahl eine Variable einen Wert von 0 ... 9999 annehmen kann. Länge ist die Anzahl der Stellen, wenn die Variable <1000 ist. Position ist die Position, ab der die Zahl ausgegeben wird, dabei steht die 0 für links beginnend. Der Wert der Variablen führende_Null ist entweder 0 für **false** und 1 für **true**, also wahr. Wenn der Wert 1 ist, werden die Stellen vor der eigentlichen Zahl mit Nullen aufgefüllt. Falls führende_Null =0 ist, wird nur die auszugebende Zahl am Display dargestellt.

Der Funktionsaufruf **display.showNumberDecEx**(zahl, punkte, führende_Null, Länge, Position); Hier ist lediglich die Variable punkte dazugekommen. Dabei könnte die Variable Punkte folgende Werte annehmen:

0b10000000 ergäbe als Stelle für den Dezimalpunkt folgende Anzeige: 0.000

0b01000000 ergäbe als Stelle für den Dezimalpunkt folgende Anzeige: 00.00

aber in unserem Fall ergibt sich folgende Anzeige: 00:00, weil unser Display keine weiteren Punkte besitzt.

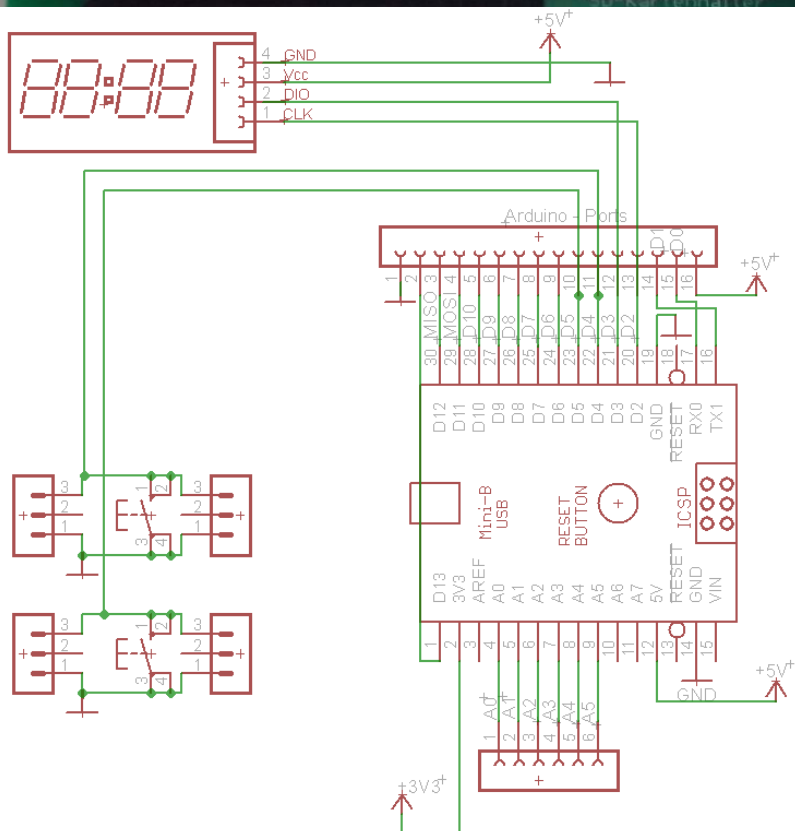
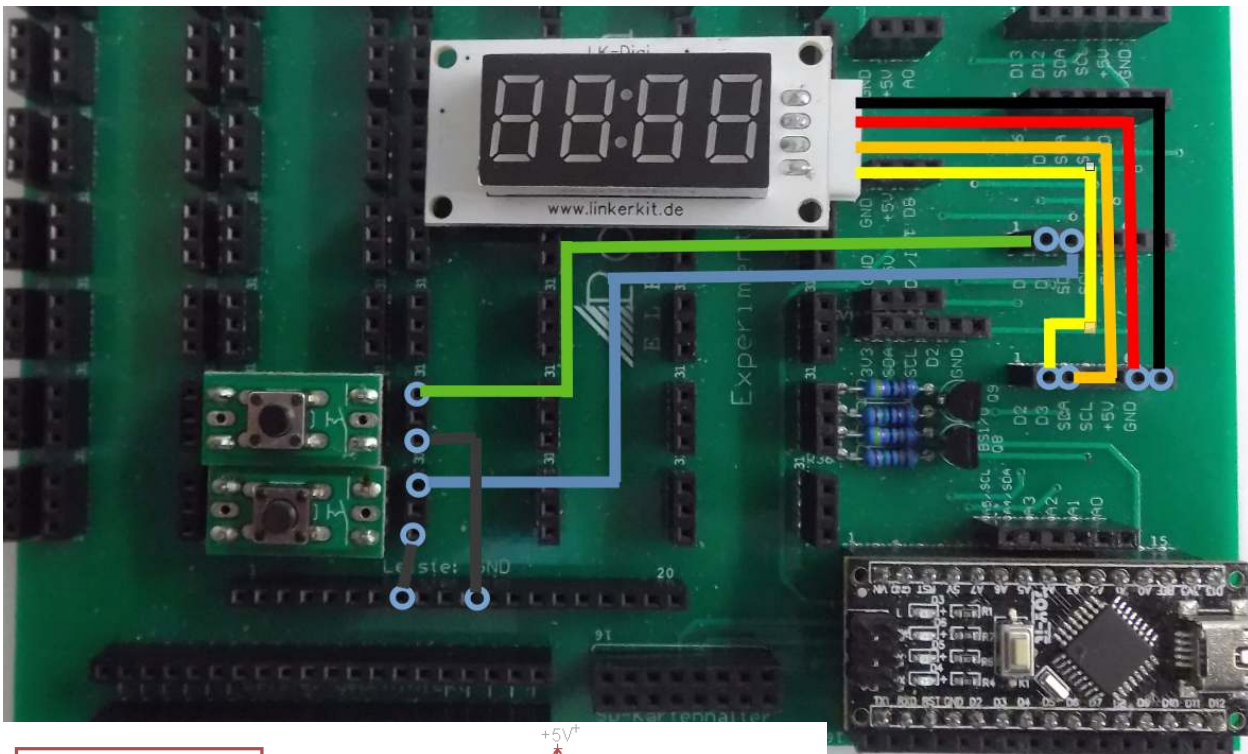
0b00100000 ergäbe als Stelle für den Dezimalpunkt folgende Anzeige 000.0

0b11100000 ergäbe als Stelle für den Dezimalpunkt folgende Anzeige 0.0.0.0 Die dargestellten Zahlen dürfen dabei auch **negative Werte** annehmen.

Mit der nachfolgenden Variablendefinition kann man - - - - am Display darstellen:

```
const uint8_t SEG_xxxx[] = {
    SEG_G,
    SEG_G,
    SEG_G,
    SEG_G
};
```

12.13.2. Uhrzeit mit LED-Modul (Artikelnummer 810453)



Im Beispiel [TM1637_Uhr01.ino](#) ist die Funktion einer Uhr programmiert. In diesem Beispiel blinkt der Doppelpunkt im Sekundentakt. Dabei wird Sekunde hochgezählt und die Uhrzeit aktualisiert. Dabei wurde als Zeitbasis die Funktion delay() verwendet. Der Nachteil der delay()-Funktion ist, dass während dieser Wartezeit, der Arduino auf keine Aktion reagiert. Der Startwert der Uhrzeit kann nur im Programmcode eingestellt werden. Ebenso kann nur dort eine Umstellung von 12 auf 24h Anzeige erfolgen.

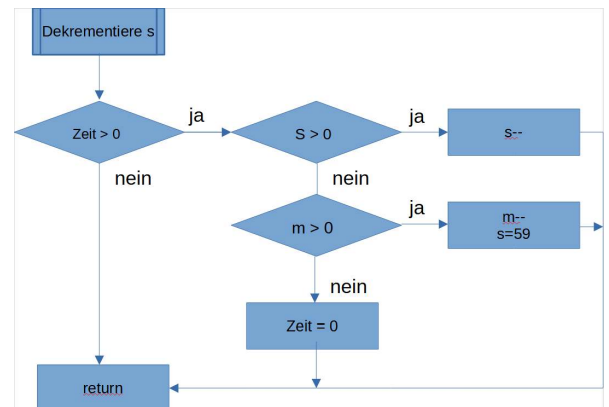
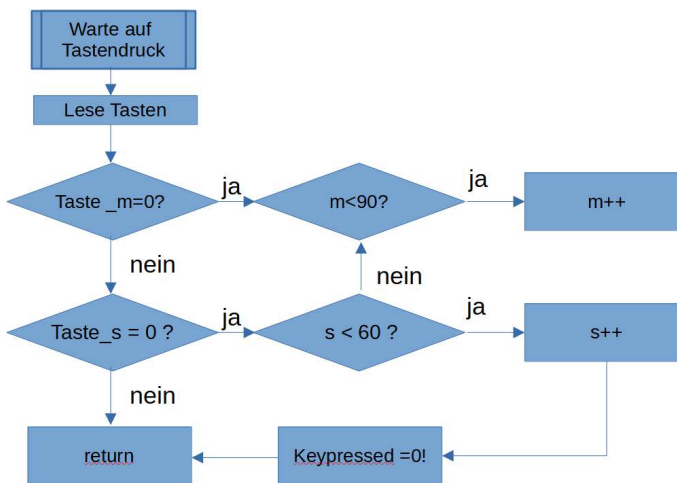
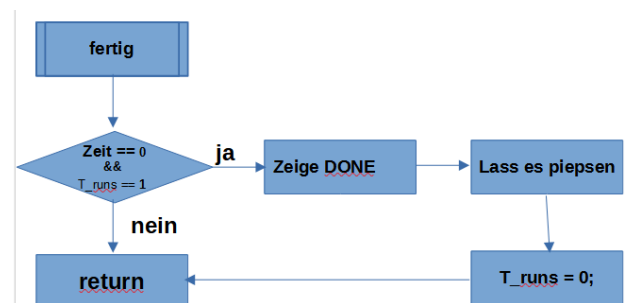
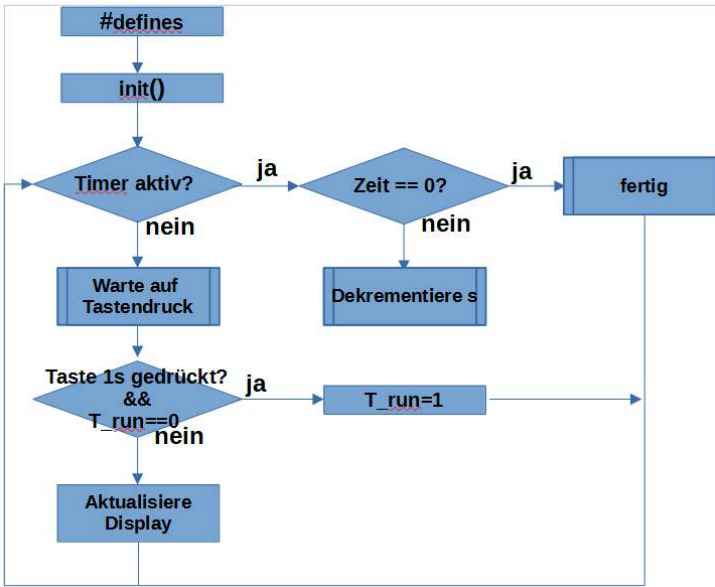
Im Beispiel **TM1637_Uhr02.ino** wurde die Funktion der Uhr erweitert. Nun wird die Uhr nicht mehr nach einem festen `delay()` weiter gezählt. Es wird ein Timer abgefragt. Wenn die Zeit vergangen ist, wird der Sekundenwert erhöht. Dies hat den Vorteil, dass im Beispiel **TM1637_Uhr03.ino** die Funktion der Tastenabfrage eingefügt werden konnte. Ansonsten müsste die Taste eine Sekunde gedrückt werden, bis der Controller darauf reagiert. Jetzt kann der Controller sofort auf den Tastendruck reagieren. Im dritten Beispiel wurde zusätzlich noch eine `while`-Schleife eingefügt, die wartet, bis die Taste wieder losgelassen wurde. Ansonsten würde die Zeit laufend hochgezählt so lange gedrückt ist, so wie in **TM1637_Uhr04.ino**.

Wie kann man erreichen, dass auch die Anzeige von 12h auf 24h erfolgt und umgekehrt?

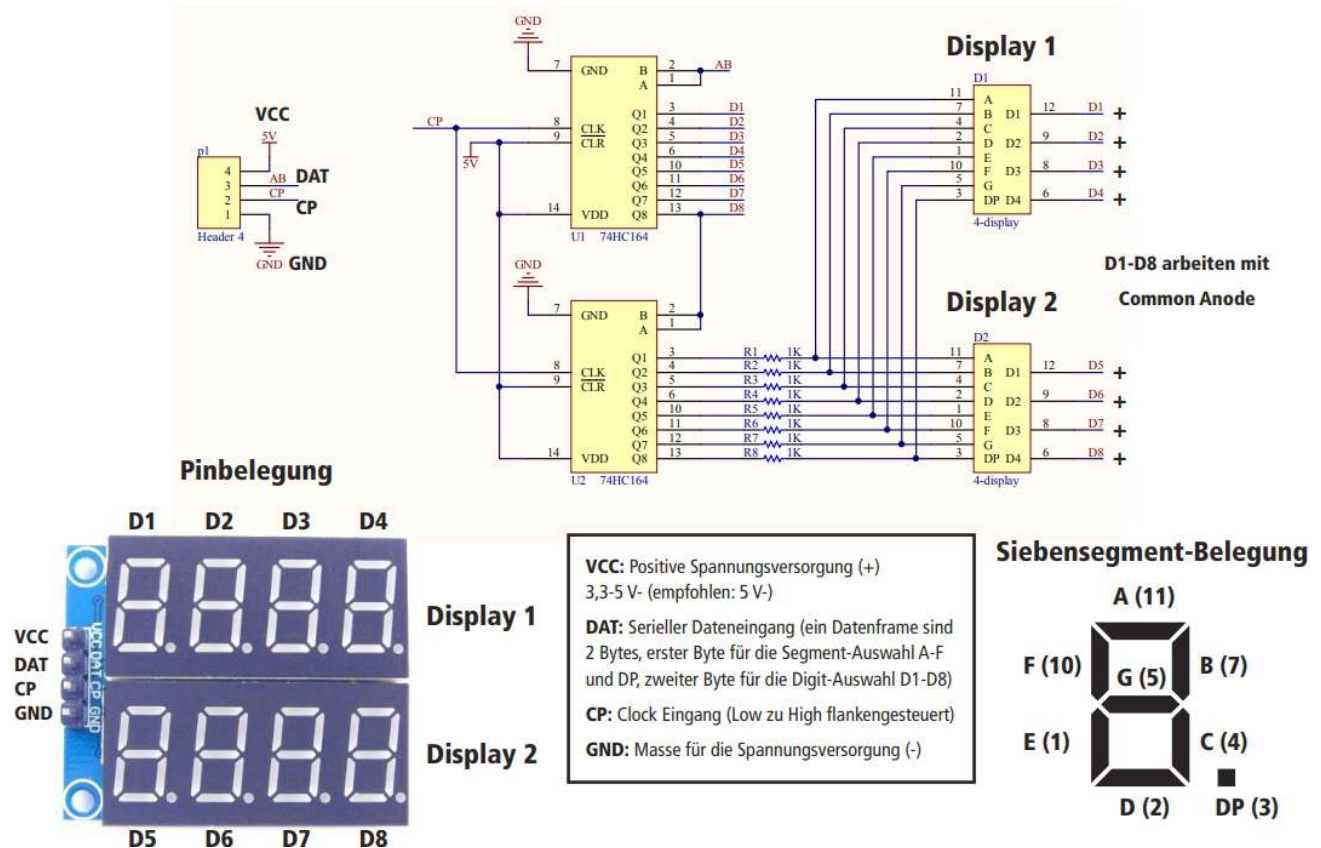
Baue die Schaltung so um, dass ein LDR zur Einstellung der Helligkeit verwendet werden kann.

12.13.3 Countdown timer

Bei diesem speziellen Uhrenprogramm **TM1637_Uhr05.ino** soll mit Tasten eine Wartezeit eingestellt werden. Mit der Taste 1 sollen die Anzahl Minuten und mit Taste 2 die Anzahl an Sekunden eingestellt werden. Nach dem Loslassen, läuft der Timer rückwärts. Erreicht er Null, soll er stoppen und einen Piepston erzeugen. Im Display soll bis zum nächsten Tastendruck „done“ stehen. Ergänze die Schaltung und das Programm um die Piepfsfunktion selbständig.

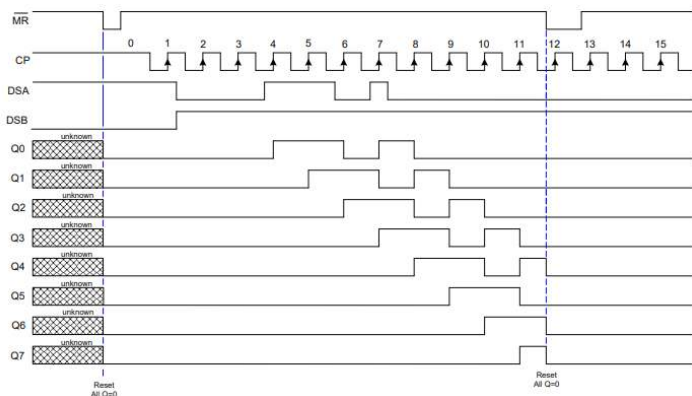


12.13.3. Uhrzeit mit Digitalanzeige-Modul (Artikelnummer 810568)



Diese 7-Segement Anzeigen werden über zwei Logik-Bausteine 74HC164 angesteuert. Der 74HC164 ist ein Schieberegister. Die Daten werden nach dem Prinzip „serial in and parallel out“ in diesem IC verarbeitet. Die Dateneingabe erfolgt über den Pin: DAT. Bei jedem Flankenwechsel 0 → 1 am CLK-Eingang werden die Daten, welche an DAT anliegen eingelesen und am Ausgang ausgegeben. Gleichzeitig, wird der zuvor eingelesene Zustand zum nächsten Ausgang weitergeleitet.

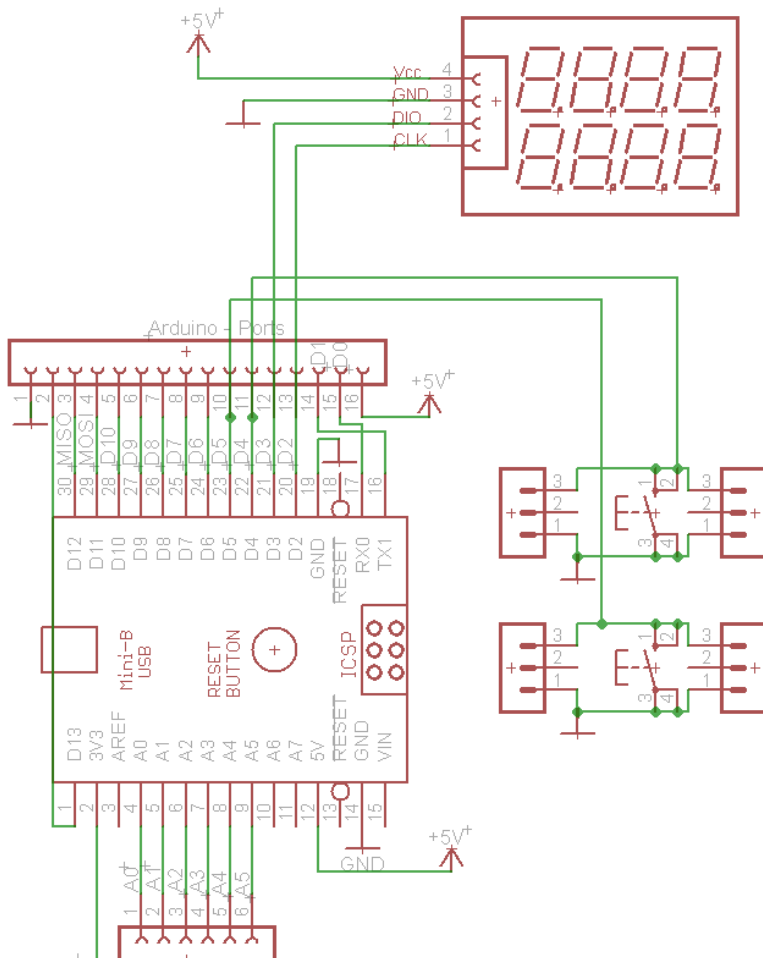
Links im Bild ist dabei zu sehen, wie nacheinander die einzelnen Ausgänge mit jedem



Flankenwechsel von 0 → 1 ihren Zustand am Ausgang ändern und weiter geben. In unserem Fall sind dabei DSA und DSB verbunden, was die Funktion ein wenig einfacher macht, weil so nur ein Eingang geschaltet wird. MR ist in unserer Anwendung immer auf eins und muss so auch nicht berücksichtigt werden. Also benötigen wir nur CLK und DAT. Da in unserem Fall zwei 74HC164 in Serie geschaltet sind, werden statt 8 eben 16

Flankenwechsel benötigt, um alle Ausgänge in einen definierten Zustand zu schalten. Wie dann die Zustände am Eingang DSA an die einzelnen Ausgänge durchgeschaltet werden ist dann im obigen Diagramm gut zu erkennen. Dabei schaltet ein Schieberegister die einzelnen Segmente eines Digits (A ... G, DP) und das andere Schieberegister schaltet das jeweilige Digit, also die Stelle D1 ... D8. Um nun das Bild einer stabilen Anzeige zu gewährleisten, müssen die einzelnen Anzeigen sehr schnell nacheinander durchgeschaltet werden. Dadurch, dass die Anzeigen nur einen kurzen Moment immer aktiv sind, nimmt die Helligkeit der Anzeige natürlich mit zunehmender Anzahl der

angezeigten Stellen ab. Doch nun genug der Theorie. Wenden wir uns einem Beispiel zu, um das ganze ein wenig klarer werden zu lassen.



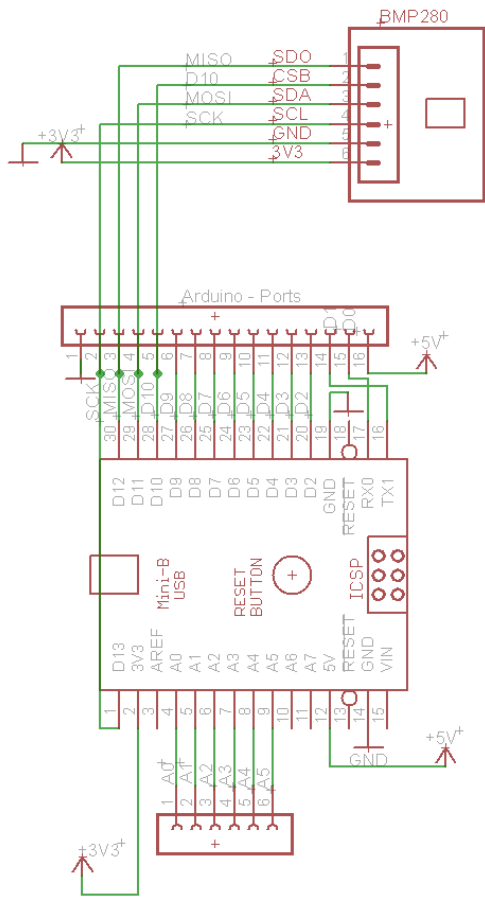
Beim Beispiel [Pollin_Display_Bsp0.ino](#) werden nacheinander die Segmente durchgezählt. Ändere bei `#define` timer 300 auf 3 und siehe, was dann passiert. Beim Beispiel [Pollin_Display_Bsp1.ino](#) werden vier Ziffern dargestellt. Mit `data[0] ... data[10]` können die Ziffern 0 ... 9 und der Dezimalpunkt aktiviert werden. Mit `digit[0] ... digit[7]` wird festgelegt, an welcher Stelle im Display dieser Wert angezeigt werden soll. Bedenke, der Arduino beginnt immer bei 0 mit dem Zählen! Beim Beispiel [Pollin_Display_Bsp2.ino](#) wird die Ansteuerung der einzelnen Digits in eine for-Schleife zusammengefasst. Dies soll zeigen, wie ein Programmierer sich manchmal die Schreibarbeit erleichtern kann. Beim Beispiel [Pollin_Display_Bsp3.ino](#) wird die Uhrzeit am Display ausgegeben.

Dabei werden die Stunden und die Minuten, wegen der Übersichtlichkeit, in zwei Zeilen ausgegeben.

Beim Beispiel [Pollin_Display_Bsp4.ino](#) wird die Uhrzeit am Display ausgegeben. Dabei werden die Stunden und die Minuten, wegen der Übersichtlichkeit, in zwei Zeilen ausgegeben. Zusätzlich werden `."` und `"-` werden benutzt, um die Sekunden zu zeigen. Besonders deutlich wird dabei im Gegensatz zum Beispiel [Pollin_Display_Bsp3.ino](#), wie die Displayhelligkeit abnimmt, mit jeder zusätzlich angezeigten Stelle des Displays.

Im Beispiel [Pollin_Display_Bsp5.ino](#) ist ein Countdown-Timer realisiert.

12.14. Luftdruckmesser (Artikel 810914)



Der BMP280 Sensor wurde im Testbeispiel verwendet. Das Beispiel

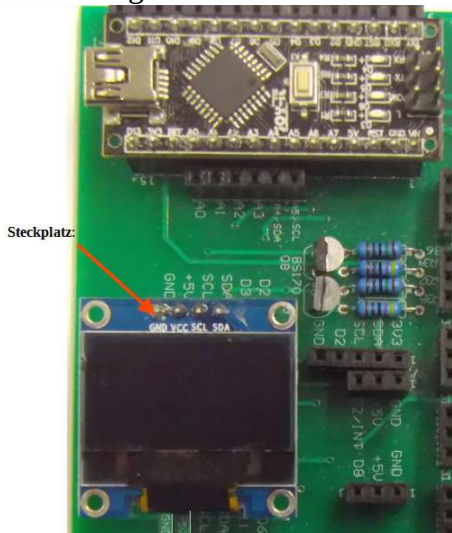
[BMP280_Hoehenmesser.ino](#) dient als kleines einführendes Beispiel. Dies ist abgewandelt vom Beispiel aus der installierten Bibliothek. Statt der I2C-Schnittstelle verwenden wir jedoch hier die SPI-Schnittstelle. Besonders zu beachten, ist dass beim Aufbau der Schaltung der Sensor mit 3,3V versorgt wird.

Die Höhenberechnung funktioniert über den Luftdruck. Dabei wird vorausgesetzt, dass dieser immer konstant bei 1023.25 mbar liegt. Allerdings ist dieser vom Wetter abhängig, ob Tiefdruck oder Hochdruck. Diese Schwankung birgt eine gewisse Unsicherheit.

12.15. OLED Displaymodul (Artikel 811108)

Luftdrucksensor am OLED Display als Höhenmesser; besonders zu beachten ist, dass drei Bibliotheken dafür installiert werden müssen: `Adafruit_SSD1306-master.zip`, `Adafruit-GFX-Library-master.zip` Diese beiden libraries werden auch in den Programmcode mit der `#include`-Anweisung eingebunden. Aber es ist noch eine zusätzliche Bibliothek erforderlich, die von der `Adafruit-GFX-library` aufgerufen wird, nämlich `Adafruit_BusIO-master.zip`.

Die Verdrahtung ist dafür sehr einfach, weil es einen Steckplatz gibt, in dem das Display nur in richtiger Polarität eingesteckt werden braucht. Der Sensor wird so angeschlossen, wie in 12.14. bereits dargestellt.



Vom Hersteller dieses Displays gibt es zwei Beispielprogramme.

Diese sind [Joy-it_Arduino_text.ino](#) zur Textdarstellung und [Joy-it_Grafikdisplay.ino](#) als Beispiel für Grafik.

Auf der Rückseite der Display-Platine ist zu sehen, welche Adresse für den I2C-Bus einzustellen ist, hier also 0x3C.

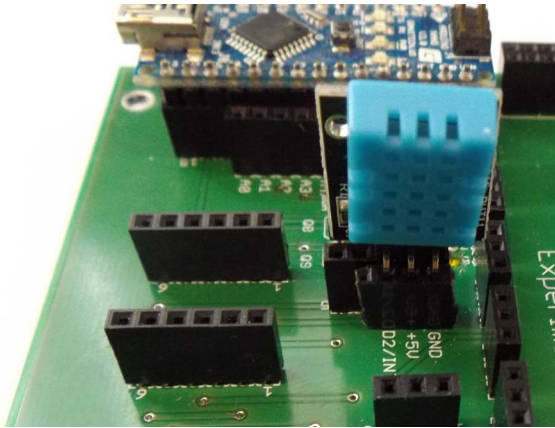
Das Programm [BMP280_Display.ino](#) ist aus dem Beispiel 12.14. und dem Textbeispiel von Joy-it entstanden.

Es können weitere Sensoren aus dem Pollin-Programm in gleicher Weise verwendet werden:

	Bezeichnung	Messbare Substanzen	Ausgänge	Artikel-Nr.:
Gassensor	SEN-MQ5	Flüssiggase (LPG), Propan, Methan, Butan, CO, H ₂ , Alkoholdampf	Analog / digital	811161
Luftqualitäts-Sensor	SEN-MQ135	Benzol (C ₆ H ₆), Ammoniak (NH ₃), Sulfid, Rauch, andere Luftverunreinigungen	Analog / digital	811159
Gassensor	SEN-MQ4	Erdgas und Methan	Analog / digital	811160
CO-Sensor	SEN-MQ7	Kohlenmonoxid	Analog / digital	811162
Flüssiggas-Sensor	SEN-MQ6	Flüssiggase (Liquefied Petrol Gas), wie Propan, Methan, Butan und brennbare Gase	Analog / digital	811169
Alkoholsensor,	SEN-MQ3	Ethanol, Alkohol über die Wasserstoffkonzentration	Analog / digital	811168
Gassensor	SEN-MQ2	Flüssiggas (LPG), i-Butan (C ₄ H ₁₀), Propan (C ₃ H ₈), Methan (CH ₄), Wasserstoff (H ₂), Alkohol, Rauch	Analog / digital	811167

Bei den Ausgängen gibt es ein analoges Signal, welches proportional zur Konzentration des Gases ist. Mit dem Potentiometer auf jedem Sensor Modul lassen sich Schaltschwellen einstellen, ab der bei Überschreiten der eingestellten Konzentration des jeweiligen Gases, der digitale Ausgang auf logisch „LOW“ (0V) schaltet. Dieser ist dann recht einfach mit dem Mikrocontroller zu erfassen. Vielleicht reicht es sogar für manche Anwendungsfälle, nur einen Transistor nach zuschalten und dann den Zustand mit einer LED anzuzeigen.

12.17. DHT11 Feuchte und Temperatursensor (Artikel 810914)



Mit dem Sensor DHT11 können die Temperatur und die Luftfeuchte gemessen werden. Die Luftfeuchtigkeit wird in % angegeben, weil sie ein Verhältnis-wert ist. Es wird das Verhältnis angegeben, wie viel Feuchtigkeit die Luft aufgenommen hat im Verhältnis, wie viel sie tatsächlich aufnehmen kann, bis zur Sättigung. Allerdings ist dieser Sensor sowohl bei der Genauigkeit, als auch von der Lesegeschwindigkeit eingeschränkt. Die Lesegeschwindigkeit beträgt

eine Messung innerhalb zwei Sekunden und die Messgenauigkeit im Temperaturbereich liegt nur bei 2°C Genauigkeit und im Feuchte Bereich bei $\pm 5\%$.

Falls noch nicht installiert, sollte die library für diesen Sensor von Github heruntergeladen werden unter:

https://github.com/adafruit/Adafruit_Sensor

Der Sensor sollte nicht öfter als alle zwei Sekunden abgefragt werden.

Benötigte Bibliotheken einbinden:

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

```
#include <Adafruit_Sensor.h>
```

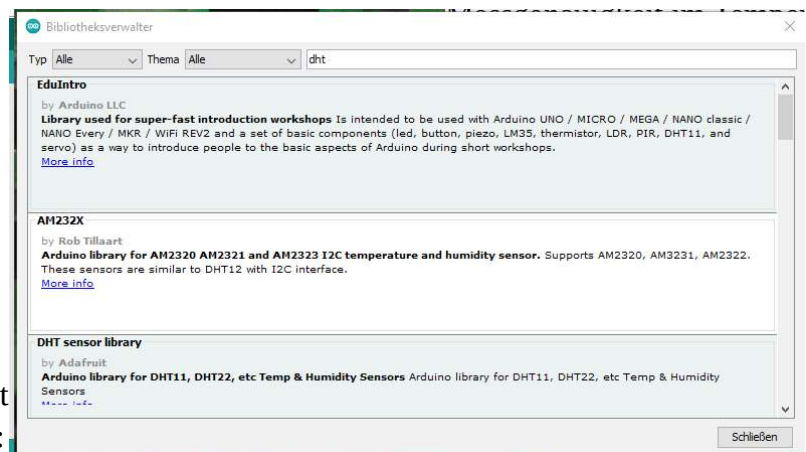
Dazu muss aber für den DHT11

Sensor noch eine Bibliothek

installiert werden. Dies erfolgt mit

Sketch → Bibliothek einbinden →

Bibliotheken verwalten. Jetzt erscheint das Fenster des Bibliotheksverwalters:



oben rechts im Suchfenster den Begriff: dht eingeben

Anschließend unten auf den Ausschnitt **DHT Sensor library** klicken und die Bibliothek installieren,

indem auf den **Installieren** Button gedrückt wird.

Wenn die Bibliothek korrekt installiert ist soll unter Datei → Beispiele → DHT Sensor library → DHTtester.ino aufgerufen werden können.

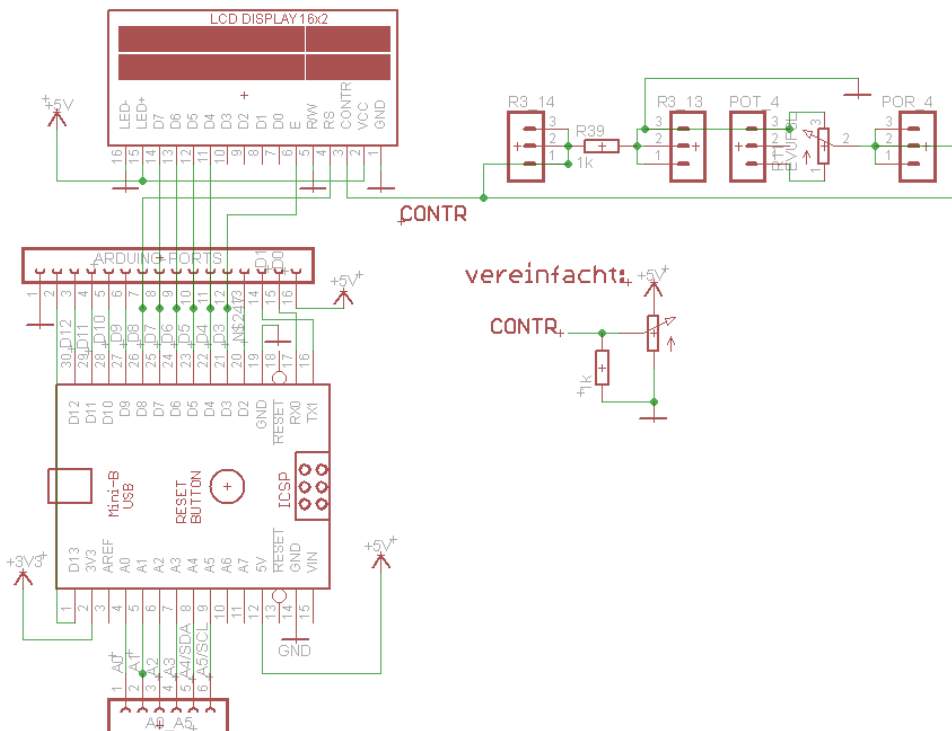
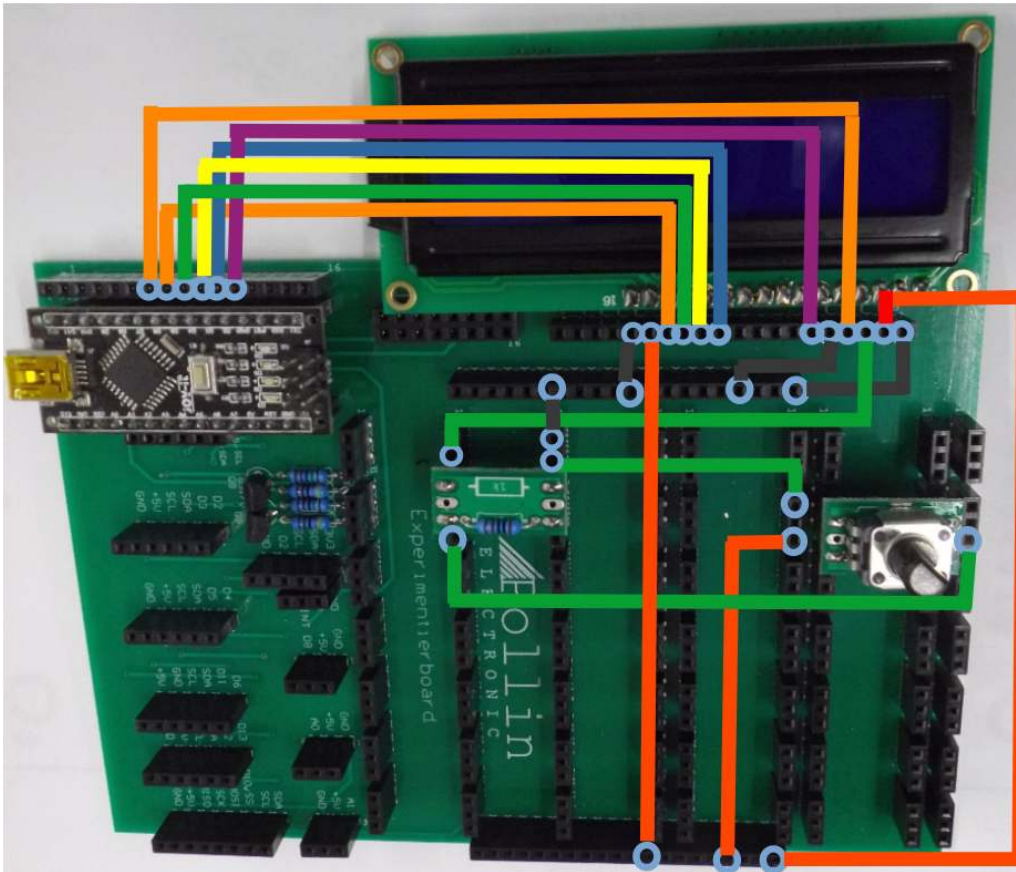
Nun kann die Bibliothek DHT.h in unser Programmbeispiel **DHT11bsp.ino** am Dateianfang eingebunden werden. Das besondere in diesem Programm ist die Verwendung eines sogenannten cast Operators. Damit wird eine Variable in einen anderen Typ umgewandelt. In diesem Fall wird eine floating point Variable in eine vorzeichenlose Ganzzahl umgewandelt.

Die Genauigkeit des Sensors DHT11 liegt bei $\pm 5\%$ bei der relativen Luftfeuchte und bei $\pm 2^\circ\text{C}$ bei der Temperatur. Um genauere Messergebnisse zu bekommen, sollte der **DHT22** mit der **Artikelnummer 811093** verwendet werden. Dessen Genauigkeit liegt bei $\pm 2\%$ bei der relativen Luftfeuchte und bei $\pm 0.5^\circ\text{C}$ bei der Temperatur. Beide Sensoren sind gegeneinander austauschbar, also Pinkompatibel.

12.18. LC-Display (Artikelnummer 121738)

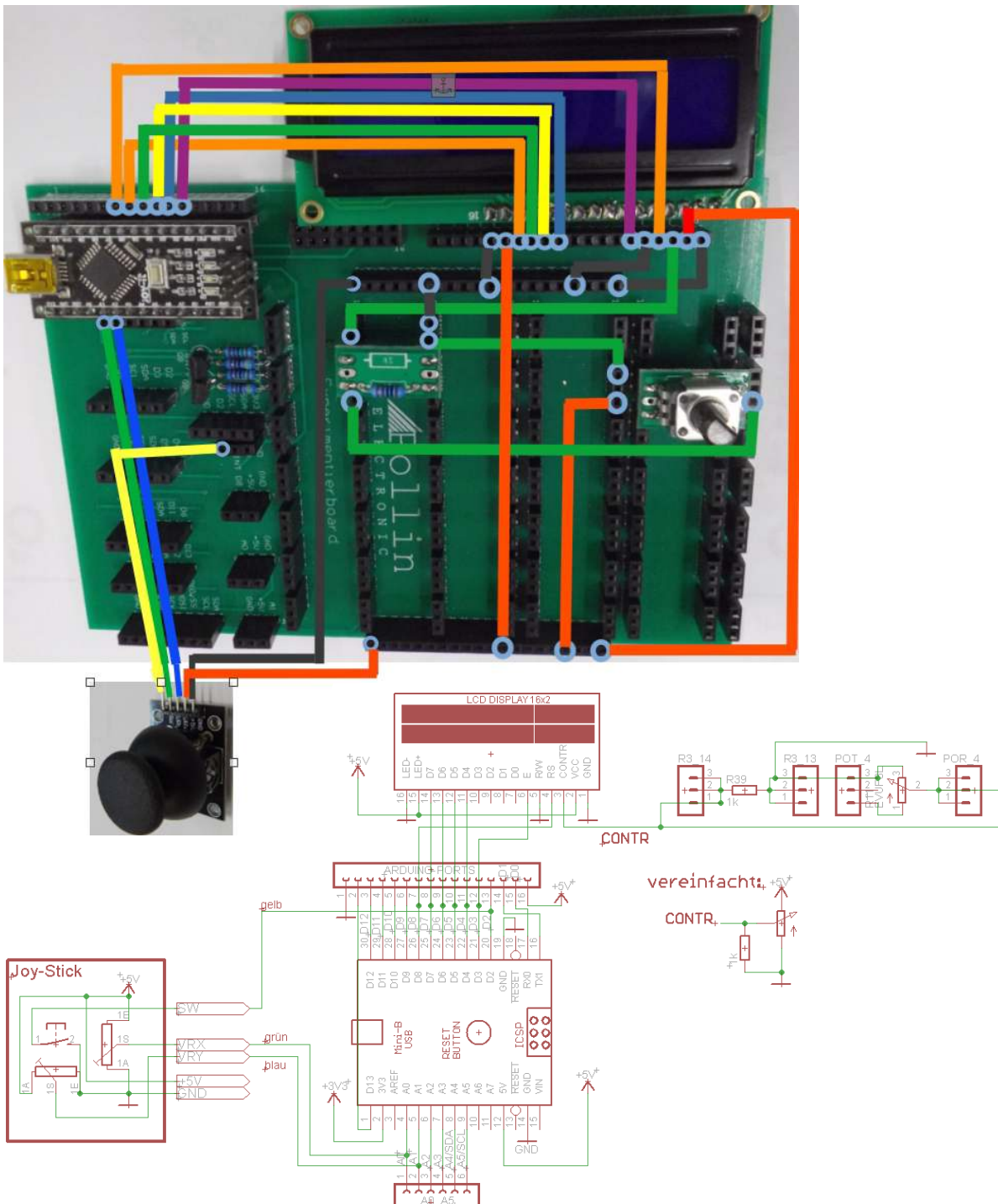
Das JOY-IT LCD-Modul 16x2 besitzt zwei Zeilen, in denen je 16 Zeichen angezeigt werden können. Es können dabei auch selber definierte Zeichen dargestellt werden, aber dieses Display ist nicht grafikfähig.

12.18.1. Uhrzeit anzeigen

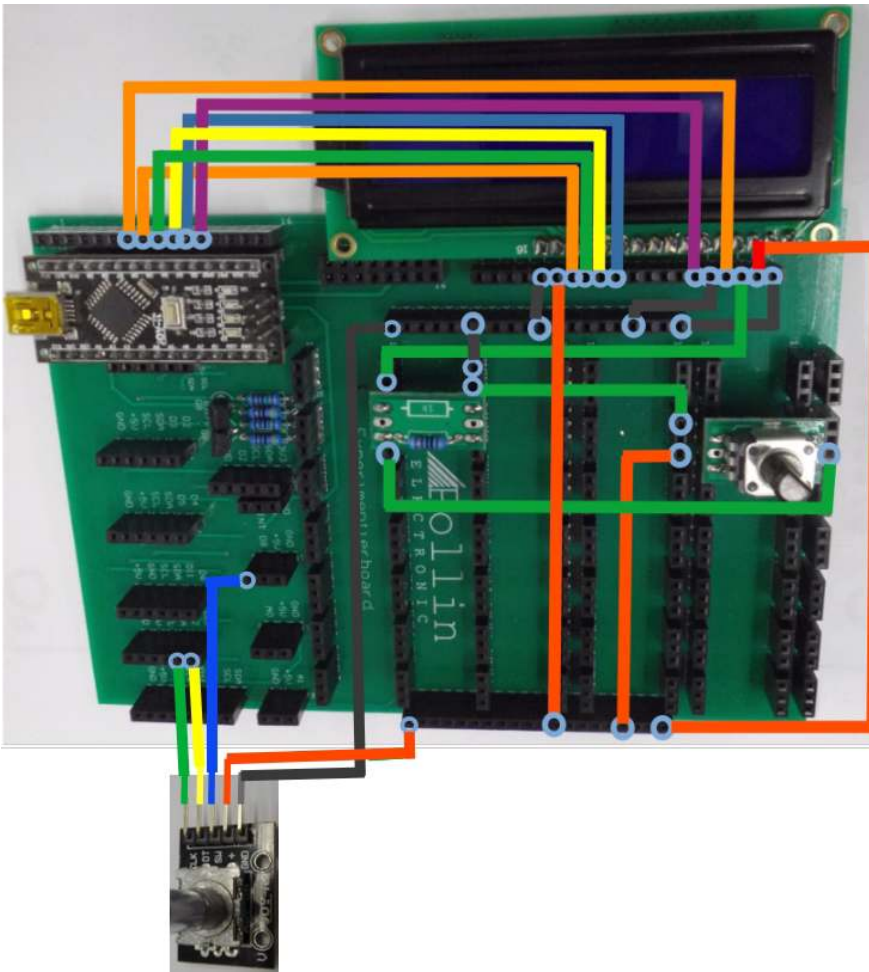


12.18.2. Uhrzeit Stellen mit Joystick (Artikelnummer 810922)

Ein Joystick besitzt zwei Potentiometer. Das eine liefert einen Wert beim Verändern der x-Position. Der zweite ändert seinen Wert beim Verändern der y-Position; und noch ein Taster, der beim Drücken des Joystick den Ausgang "SW" nach GND schaltet. In diesem Beispiel wird der Taster-Pin "SW" an den Pin D8 des Arduino verbunden. Der Joystick braucht zusätzlich eine Verbindung nach Masse (GND) und eine weitere Verbindung nach Versorgung (hier: +5V). Die mittleren Angriffe der beiden Potentiometer werden mit den analogen Eingängen A0 und A1 verbunden. Das Beispiel [Joystick01.ino](#) zeigt die grundsätzliche Funktionalität des Joystick und wie man diese im Programm realisieren kann. Eine ganz praktische Anwendung zeigt das Programm [LCD_uhr_Joystick.ino](#). Darin wird gezeigt, wie man die Uhrzeit mit einem Joystick ändern kann.

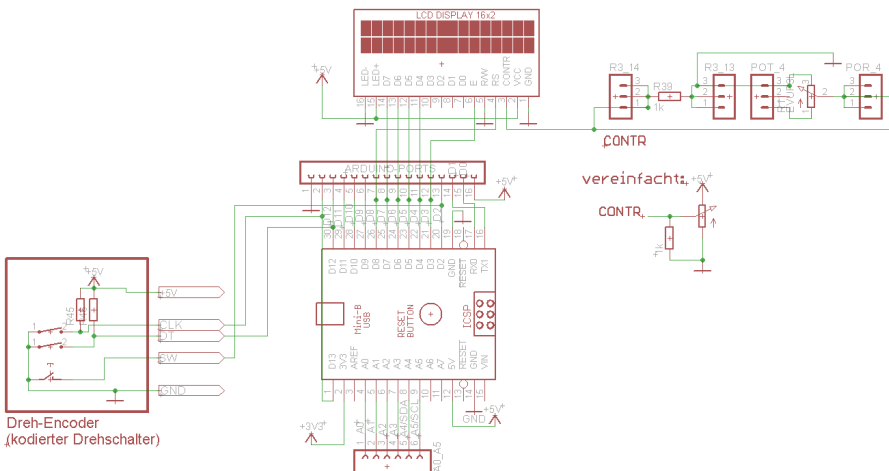


12.18.3. Beispiel mit codiertem Drehschalter (Digitalencoder Artikelnummer 810923)

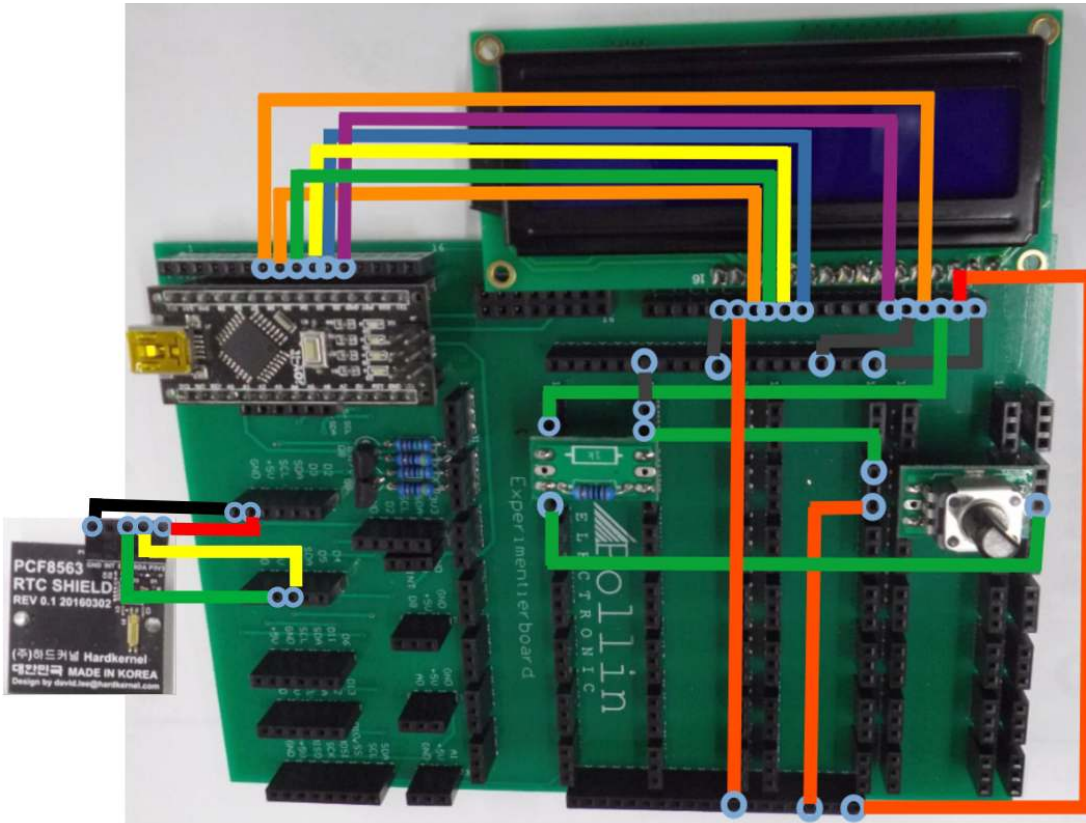


Ein codierter Drehgeber oder Encoder ist ein Drehschalter, der zwei Rechtecksignale schaltet. Beim Drehen rastet dabei der Encoder an bestimmten Positionen ein. Bei jeder dieser Positionen (Schritte) ändern beide Ausgänge ihre Spannungspegel. Sie werden abwechselnd High und Low, also 5V und GND. Je nachdem welcher Ausgang zuerst seinen Zustand ändert, kann daraus die Drehrichtung bestimmt werden. Die Drehgeschwindigkeit bestimmt die Flankenfolge. Ein zu schnelles Drehen kann der Arduino unter Umständen nicht mehr erkennen, bzw. nicht mehr

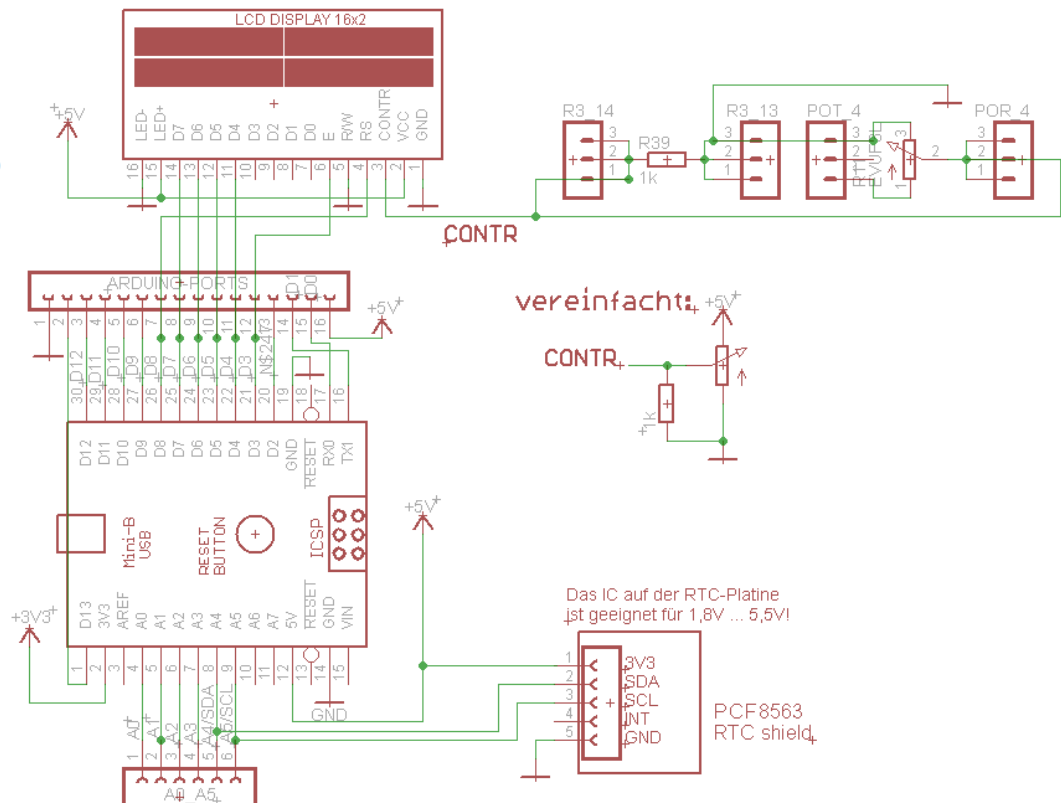
vom Programm ausgewertet werden. Das Beispiel [bsp_drehschalter.ino](#) zeigt die grundsätzliche Funktionalität des Drehschalters und wie man diesen auswerten kann.



12.18.4. Uhrzeit mit RTC Baustein (Artikelnummer 810515)

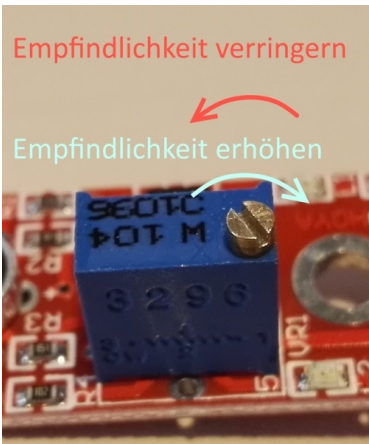


Im Beispiel [rtc_bsp01.ino](#) ist gezeigt, wie man ein RTC-Modul zum abspeichern der Uhrzeit und vom Datum verwenden kann. Wichtig ist aber, dass eine intakte Batterie im Modul enthalten ist. Die Zeile `rtc.adjust(DateTime(2021, 12, 29, 14, 8, 20));` in der Funktion `setup();` ist nur einmalig zum Setzen der Uhrzeit und des Datums einzukommentieren; Das Modul ist batteriegepuffert und somit sollte die Uhrzeit gespeichert sein. Im Beispiel [LCD_uhr02.ino](#) wird das Uhrprogramm um den Uhrenbaustein PCF8563 erweitert.



12.19. Hallsensor (Artikelnummer 810913)

Ein analoger Hallsensor dient zum Messen der magnetischen Feldstärke.



Mit dem Poti auf der Platine kann die Empfindlichkeit, bzw. die Verstärkung des Operationsverstärkers erhöht werden. Drehrichtung: siehe Abbildung links;

Bei der Messung ist zu beachten ist, dass die Höhe des Spannungswertes umgekehrt proportional zum Signal ist. Das bedeutet eine großes Magnetisches Feld liefert einen kleinen Spannungswert und umgekehrt.

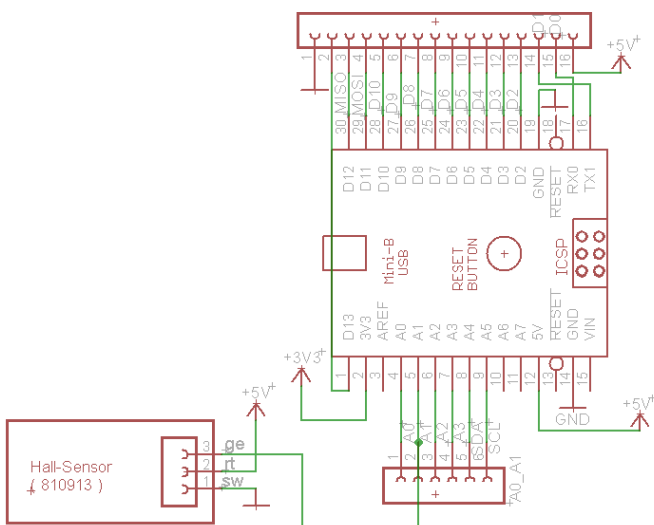
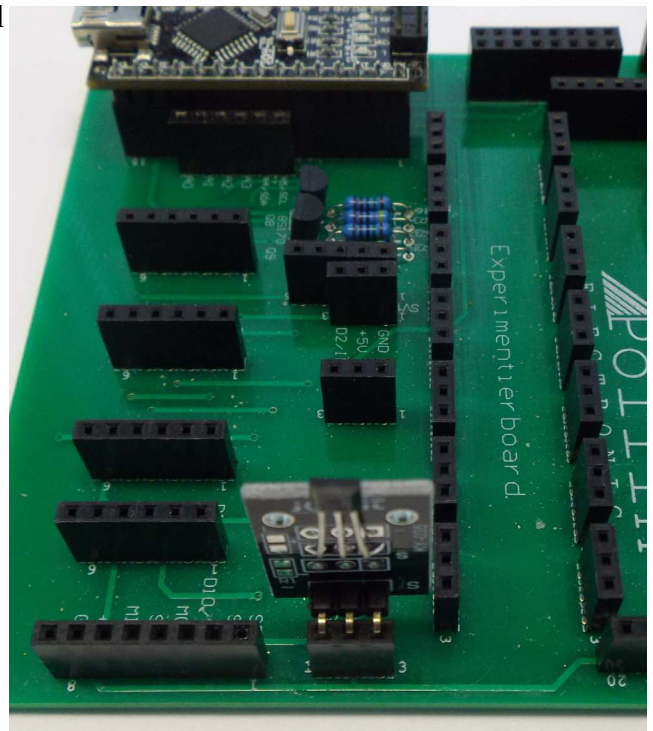
Ein HIGH-Pegel am digitalen Eingang signalisiert, dass der Grenzwert für die magnetische Feldstärke überschritten wurde. Allerdings lässt sich dieser Schwellwert nicht verändern.

Auf der Sensorplatine befinden sich noch zwei LED's. Die LED1 nahe dem Verstärker-IC zeigt an, ob der Sensor mit Spannung versorgt wird.

Die LED2 zeigt an, wenn ein Magnetfeld detektiert worden ist.

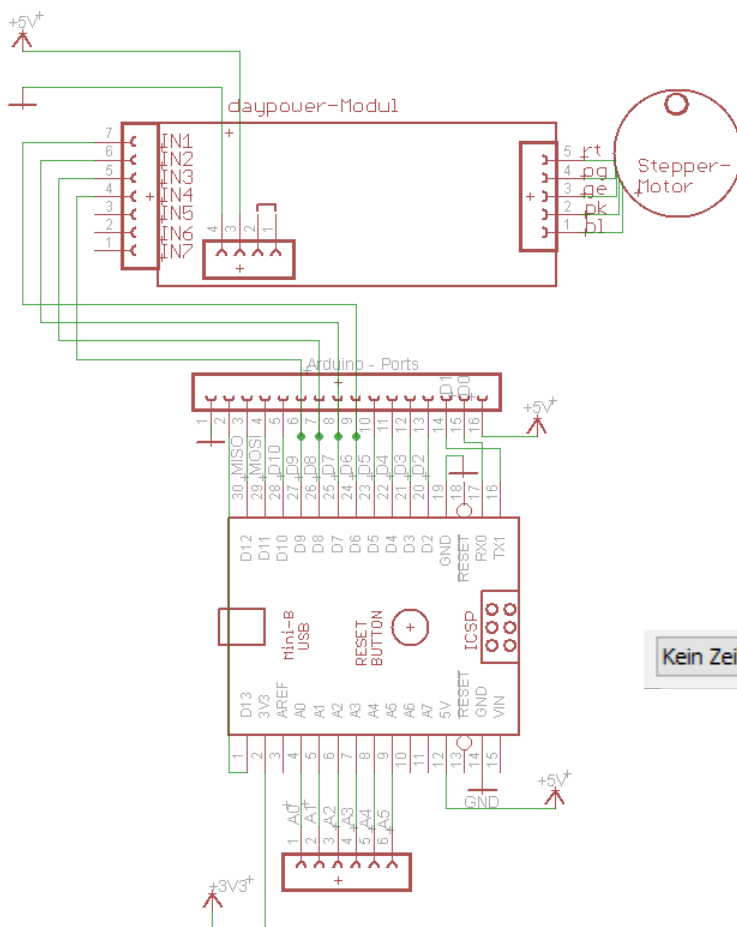
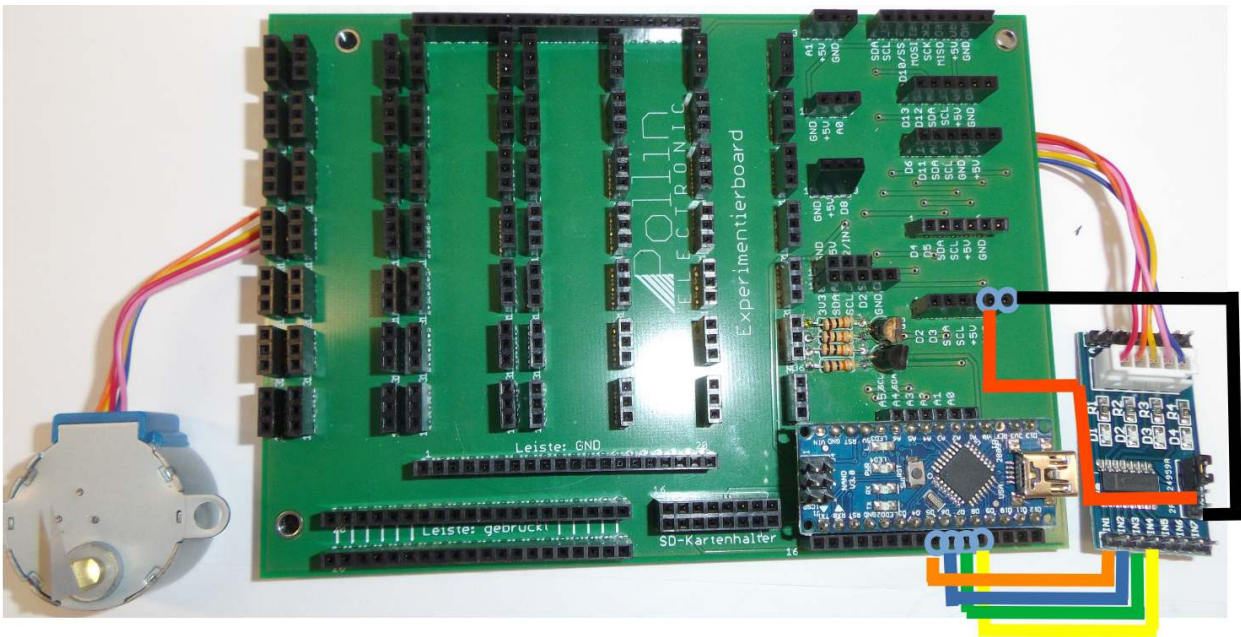
Das [Beispiel_hallSensor.ino](#) dient nur zum Experimentieren mit dem Sensor. Am besten benutzt man dazu einen Permanentmagneten um das Programm und das Sensormodul zu testen. Es gibt im Pollin-Programm einige Sensoren für die das selbe Programmbeispiel angewandt werden kann wie oben beim Hallsensor. Sie besitzen nur drei Anschlüsse für Masse, 5V und das analoge Ausgangssignal. Die Verstärkung des Ausgangssignal erfolgt über ein Poti.

Es sind dies ein weiterer Hallsensor (810567), ein Flammendetektor bzw. Infrarot-Sensor für Wellenlängen im Bereich: 760-1100 nm (810580) und einen Helligkeitssensor (LDR: 810577). Diese Sensoren sind von der Firma DAYPOWER und leider sind diese nicht Pinkompatibel mit den Modulen von JOY-IT. Die Module von DAYPOWER haben den Analogausgang in der Mitte. Das Programmbeispiel könnte dann auch für diese Sensoren verwendet werden. Achtung beim Verdrahten des Sensors mit dem Experimentierboard ist auf die Belegung der Anschlüsse des Sensorboards zu achten!



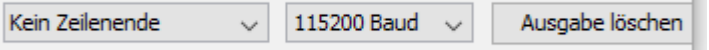
12.20. Schrittmotor (Artikelnummer 310543)

Das Beispiel [stepper_servo.ino](#) kann von www.pollin.de heruntergeladen werden. Dazu den Artikel 310543 aufrufen und bei verfügbare Downloads die Software herunterladen.



Beim Aufbau ist zu beachten, dass die Spannungsversorgung von 5V an der Schrittmotor Treiberplatine anliegt und die Eingänge mit den Arduino Ports gemäß der Definition in der Software verbunden sind. (IN=D6, IN2=D7, IN3=D8, IN4=D9)

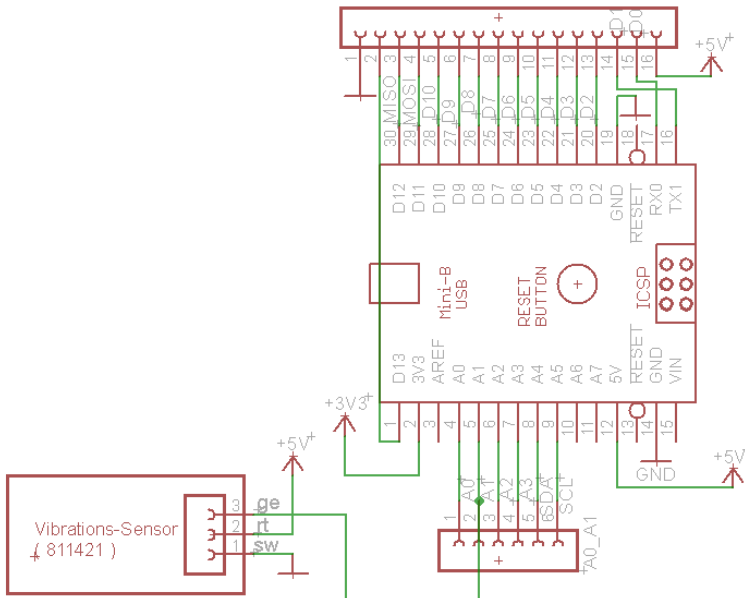
Nun kann das Programm [stepper_servo.ino](#) in den Arduino geflasht werden. Über das Werkzeug → serieller Monitor kann es mit Tastatureingaben gesteuert werden. Allerdings ist dabei wichtig, dass unten im Monitorfenster folgende Einstellungen gemacht sind:



mit der Einstellung „Kein Zeilenende“ wird sicher gestellt, dass nur die Tastenbefehle und keine weiteren handshake – Befehle wie <CR> carriage return oder <LF> line feed übertragen werden. Denn diese Sonderzeichen

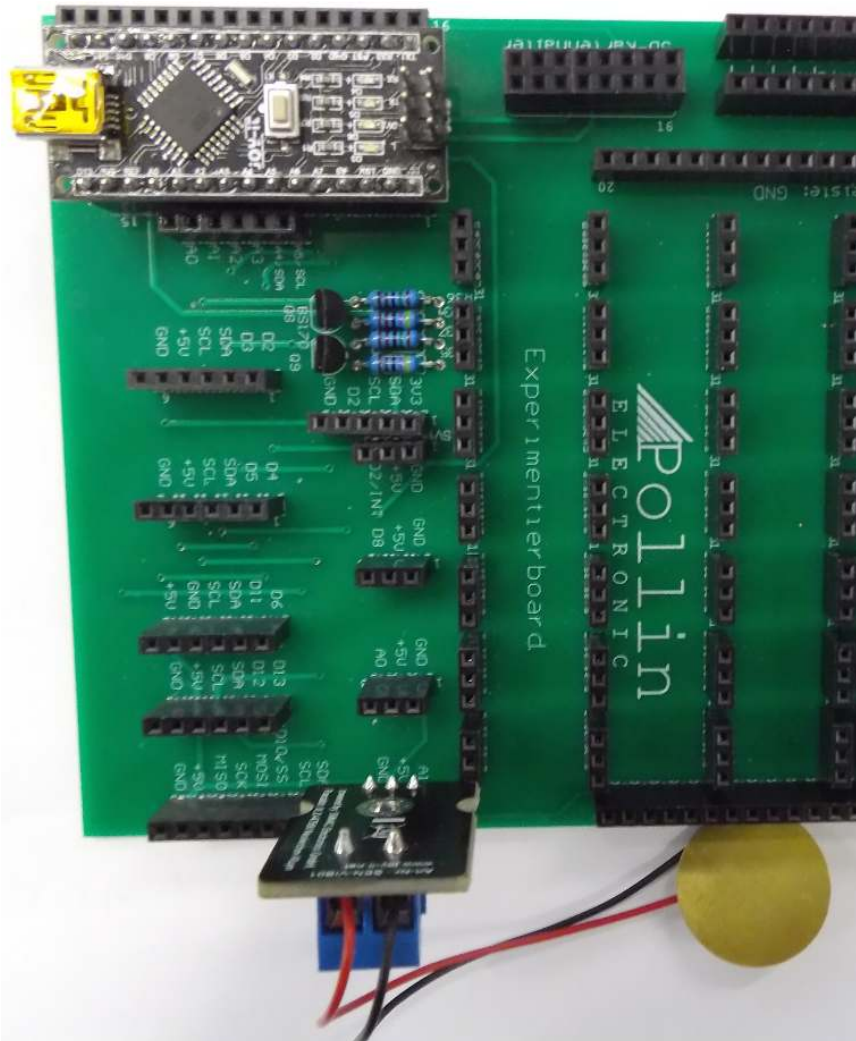
würde das Arduino Programm fehlerhaft auswerten und damit falsch reagieren.

12.21. analoger Vibrationssensor (Artikelnummer 811421)



Der Vibrationssensor ist wie der Name schon sagt ein Klopfsensor, der eine analoge Spannung liefert. Die Amplitude des Signals ist dabei abhängig von der Intensität der Erregung. Je stärker das Klopfen, desto größer die Amplitude. Das Beispiel [PiezoSensor.info](https://www.piezosensor.info) beschreibt dessen Anwendung. Unter dem Menüpunkt Werkzeuge → serielle Plotter, erfolgt die zeitliche Darstellung des Signals als Grafik und nicht als Text. So bekommt der Anwender ein Gefühl dafür wie stark das Klopfen sein muss, um eine bestimmte Signalstärke zu erhalten. Die

Schwelle, ab der eine LED leuchten soll wurde unter `#defines` festgelegt. Die Schaltung ist sehr einfach gehalten: G ist Masse, also mit der Leiste GND verbunden und der Anschluss S am Vibrationsmodul ist direkt mit dem Port A1 des Arduino verbunden. Eine Spannungsversorgung ist für dieses Modul nicht nötig, weil durch die mechanische Beanspruchung ein Spannungssignal erzeugt wird und das Modul rein passiv ist, das bedeutet, dass das Signal auch nicht verstärkt wird.

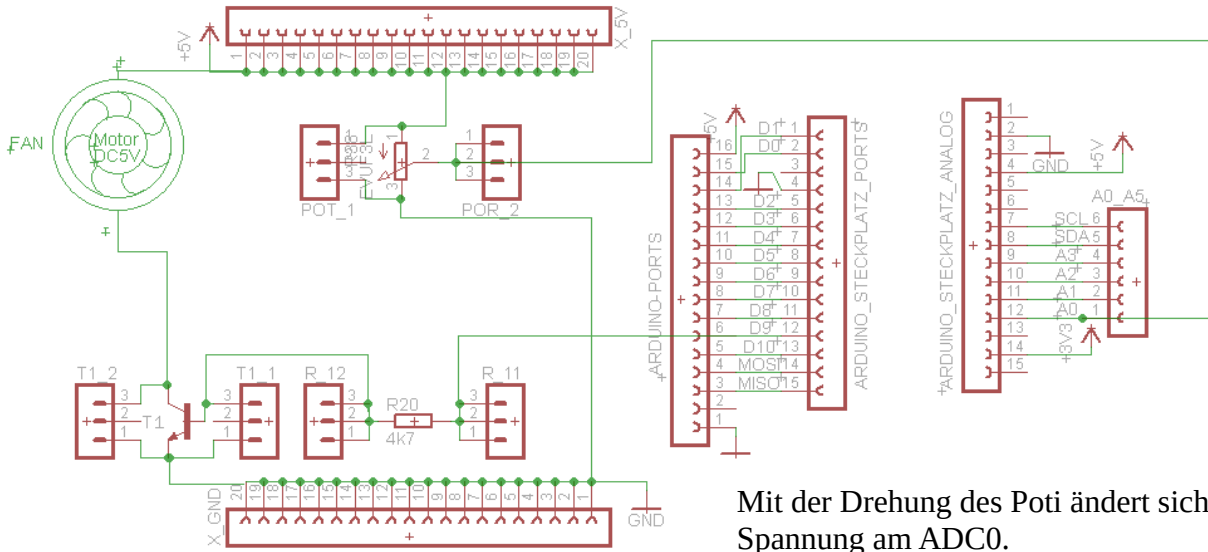
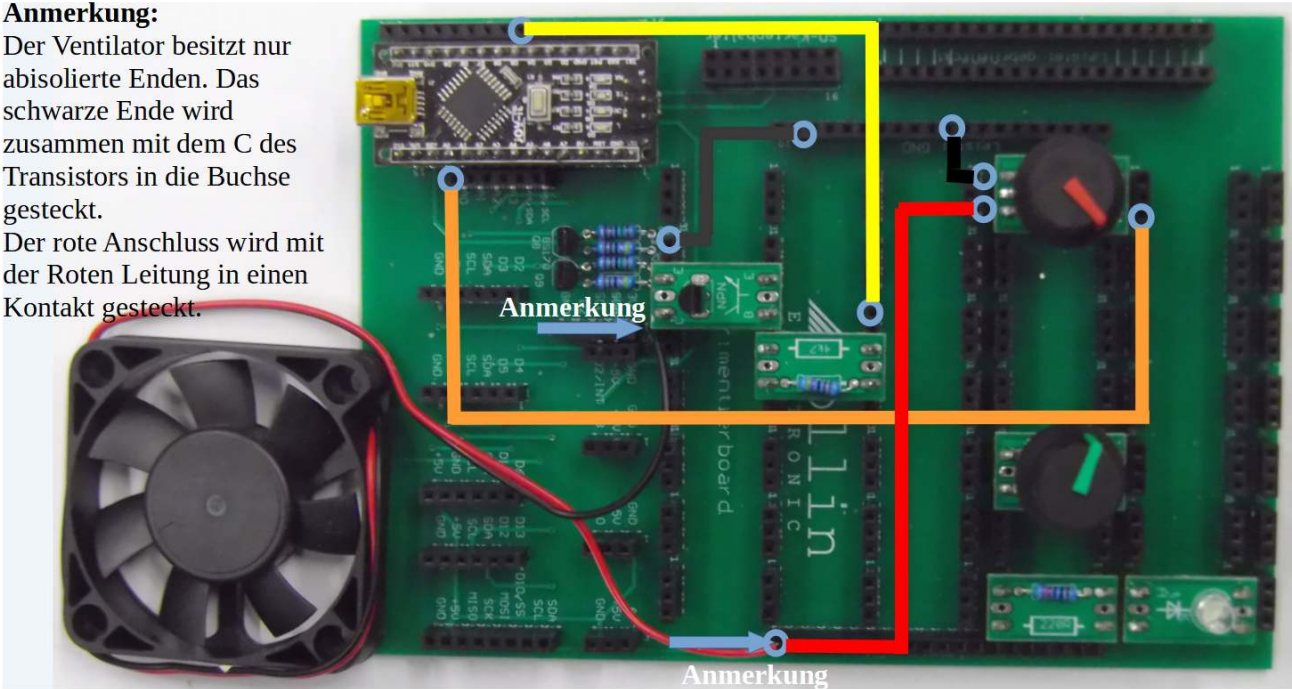


12.22. Lüfteransteuerung

12.22.1. Drehzahlregelung (PWM_Motor.ino)

Anmerkung:

Der Ventilator besitzt nur abisolierte Enden. Das schwarze Ende wird zusammen mit dem C des Transistors in die Buchse gesteckt. Der rote Anschluss wird mit der Roten Leitung in einen Kontakt gesteckt.



```

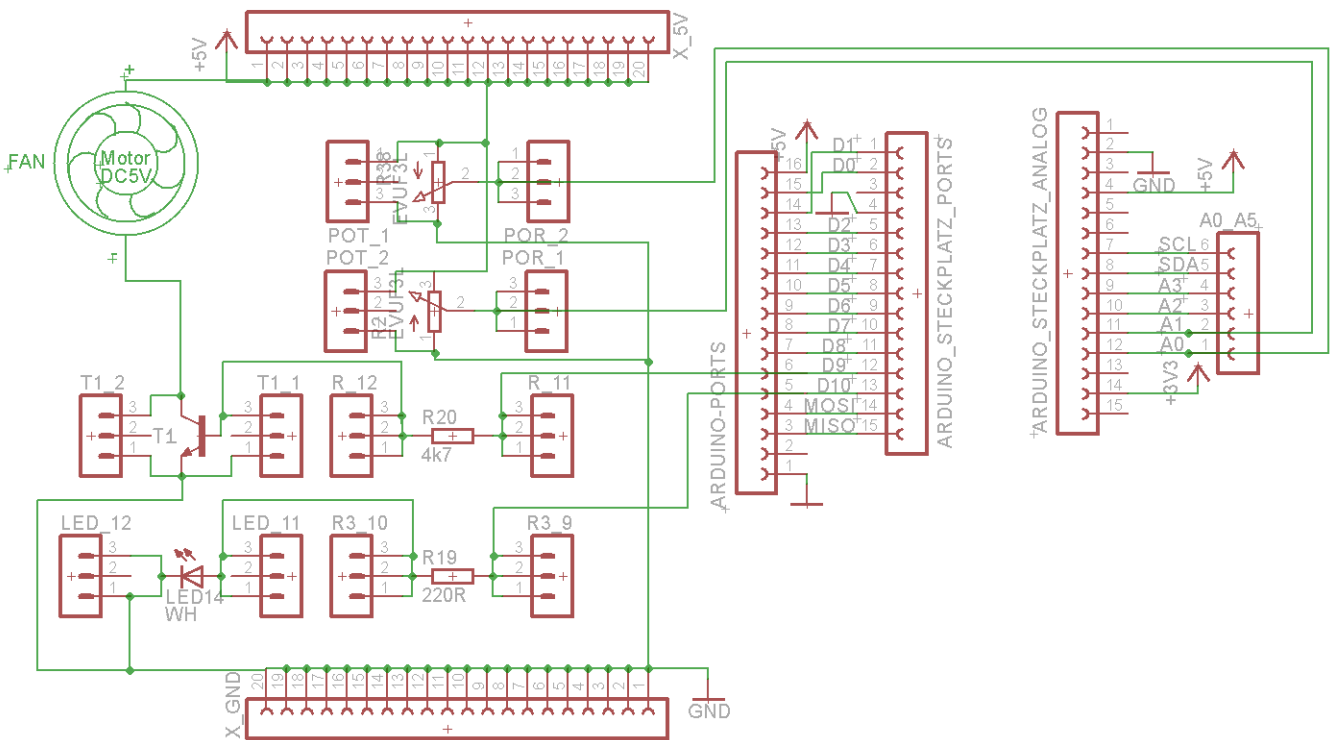
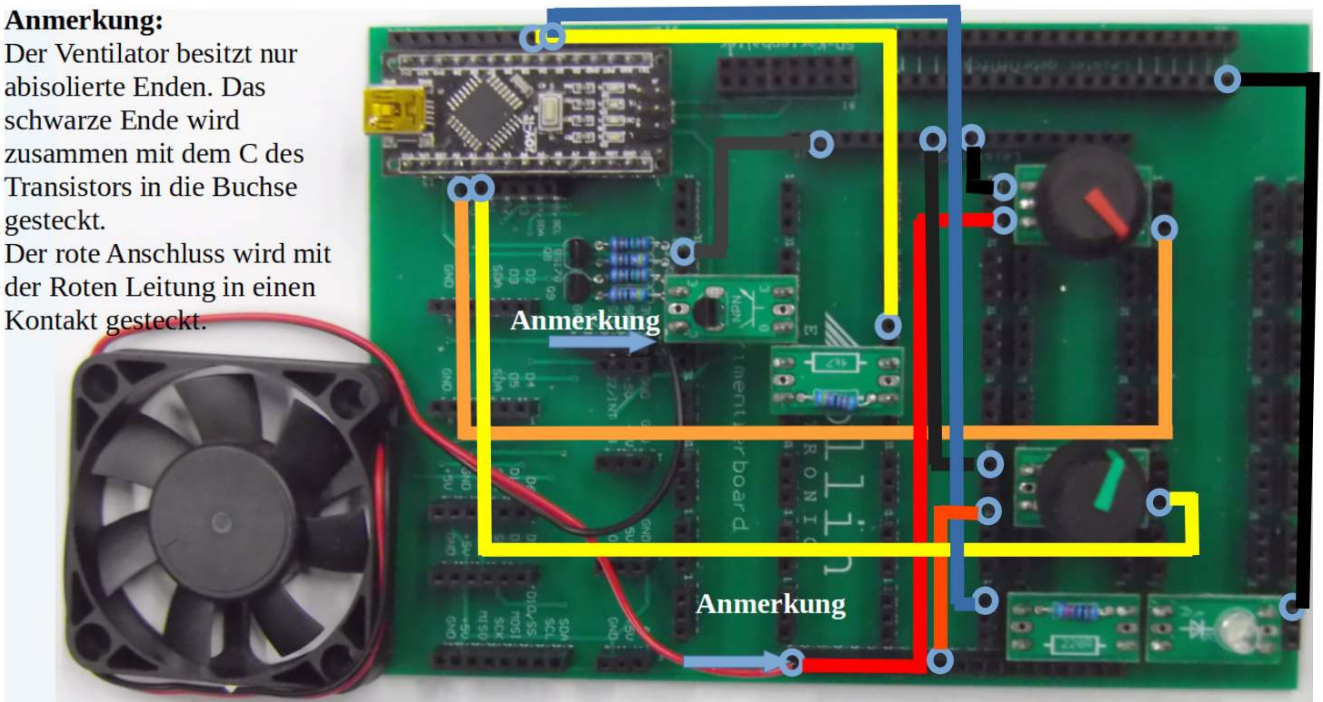
PWM_Motor
1 #define LED 13 // Definiere den LED Pin
2 #define schwelle 1000 // lege die Schaltschwelle fest
3 #define OnTimer 500 // wie lange soll die LED leuchten
4 #define motor 3
5
6 unsigned int poti; // integer Variable für den AD-Wandler
7
8 void setup() {
9 // put your setup code here, to run once:
10 pinMode(LED, OUTPUT); // Ausgänge initialisieren
11 pinMode(motor, OUTPUT);
12 Serial.begin(115200); // Baudrate festlegen
13 }
14
15 void loop() {
16 // put your main code here, to run repeatedly:
17 poti=analogRead(A0);
18 poti = poti /4; // PWM Wert liegt zwischen 0 ... 255
19 analogWrite(3, poti); // an den Pins 3, 5, 6, 9, 10, 11 ist die PWM verfügbar
20 Serial.println(poti,DEC); // gib den aktuellen PWM-Wert aus
21
22 }
    
```

Mit der Drehung des Poti ändert sich die Spannung am ADC0. Diesen Wert liegt im Bereich 0 ... 1024. Deshalb wird dieser durch 4 geteilt. Nun kann er an den PMW-Port weitergegeben werden. Der Wert für den PWM-Ausgang muss im Bereich 0 ... 254 liegen. Dadurch wird das Puls-Pause Verhältnis des PWM-Signals gesteuert. Somit ändert sich der Effektivwert der Gleichspannung variabel von 0 ... 5VDC. So variiert die Drehzahl des Lüfters.

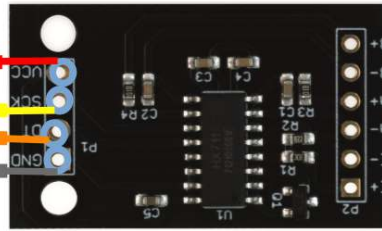
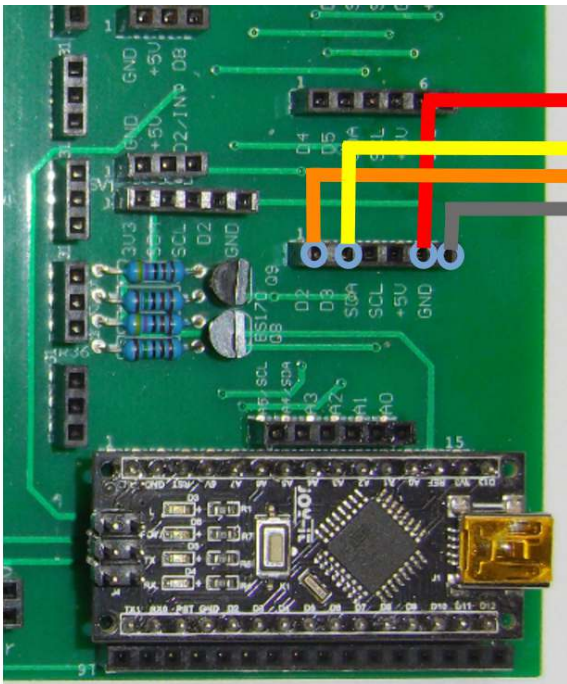
12.22.2. Drehzahlbestimmung mit Stroboskop (PWM_Motor_LED.ino)

Anmerkung:

Der Ventilator besitzt nur abisolierte Enden. Das schwarze Ende wird zusammen mit dem C des Transistors in die Buchse gesteckt. Der rote Anschluss wird mit der Roten Leitung in einen Kontakt gesteckt.



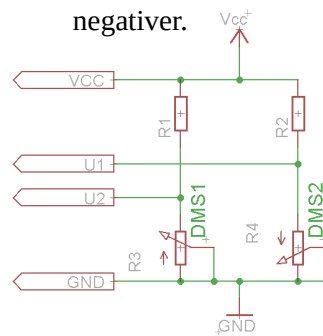
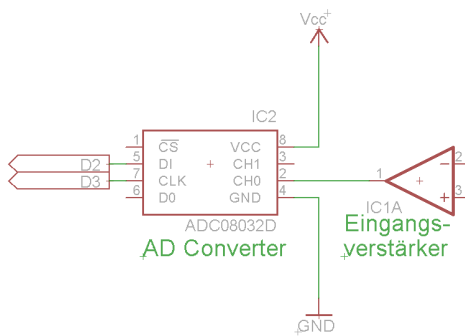
12.23. Wägezelle mit HX711 (Artikelnummer 811419)



Frei
Frei
Grün
weiss
schwarz
rot

Hier werden die Dehnungsmessstreifen der Wägezelle angelötet

Herzstück einer jeden Wägezelle sind die Dehnungsmessstreifen (DMS1, DMS2). Diese sind einer oben und der andere unten auf dem Biegebalken angebracht. Die Messung erfolgt wie bei einer Wheatstone Brücke in dem sich jedoch hier zwei Widerstandswerte ändern und somit die Genauigkeit und die Empfindlichkeit erhöhen. Die Spannung ist proportional zu den Widerständen. Wenn R4 größer wird, z.B. bei mechanischer Beanspruchung durch Druck auf den Biegebalken, so wird R3 im gleichen Maße kleiner. Somit wird die Spannung U_{12} größer. Wird der Biegebalken auf Zug beansprucht, dann stieg die Spannung an R3 mit zunehmender Kraft. Die Spannung wird dann immer negativer.



In den Beispielen [bsp01_HX711.ino](#) und [bsp02_HX711.ino](#) sind die Ansteuerung über die Arduino IDE dargestellt.

Um die Werte in Gewichtskraft umzurechnen muss der Mittelwert bei unbelasteter Zelle ermittelt werden. Dieser Wert ist dann von den Messwerten zu subtrahieren. So kann der Messwert als Gramm Angabe verwendet werden. Bei Druck bekommt man dann negative Werte.

12.24. RGB-LED-Matrix (Artikelnummer 810664)

Anmerkung: bei der Auswahl der Farben oder zu großer Helligkeitswerte, kann es vorkommen, dass die Farben nicht korrekt wiedergegeben werden, weil der Versorgungsstrom vom Arduino Nano Board nicht ausreicht. In diesem Fall könnte man das shield separat mit 5V versorgen. Dann aber das Basisboard nur mit GND und Pin 8 mit dem shield verbinden. **Nicht BEIDE 5V Spannungen zusammenschalten außer beide sind vom gleichen USB-Hub oder Computer.**

12.24.1. Farbenspiel:

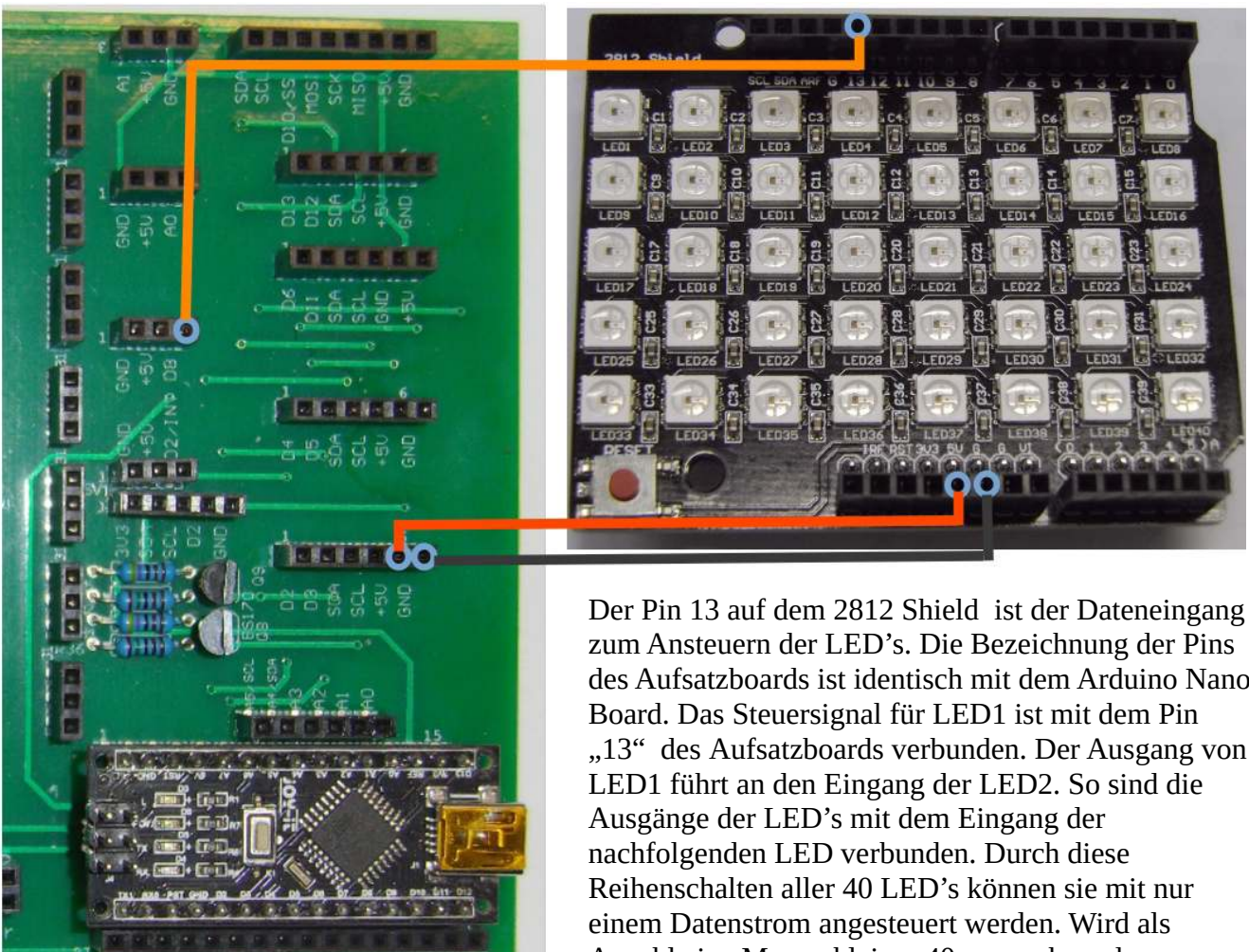
Im Beispiel [strandtest_wheel.ino](#) das unter dem Menüpunkt DATEI → BEISPIELE → adafruit_NEOPIXEL zu finden ist, müssen nur zwei Anpassungen vorgenommen werden: in der Zeile 6:

```
#define PIN 8 ändern. Dies ist der Pin, an dem die Daten vom Arduino ausgegeben werden.
```

In der Zeile 16:

```
Adafruit_NeoPixel strip = Adafruit_NeoPixel(40, PIN, NEO_GRB + NEO_KHZ800);
```

Die Zahl **40** gibt an wie viele der LED's in Serie geschaltet sind



Der Pin 13 auf dem 2812 Shield ist der Dateneingang zum Ansteuern der LED's. Die Bezeichnung der Pins des Aufsatzboards ist identisch mit dem Arduino Nano Board. Das Steuersignal für LED1 ist mit dem Pin „13“ des Aufsatzboards verbunden. Der Ausgang von LED1 führt an den Eingang der LED2. So sind die Ausgänge der LED's mit dem Eingang der nachfolgenden LED verbunden. Durch diese Reihenschalten aller 40 LED's können sie mit nur einem Datenstrom angesteuert werden. Wird als Anzahl eine Menge kleiner 40 angegeben, dann werden dementsprechend weniger LED's angesteuert.

12.24.2. Würfelspiel (gleicher Aufbau wie 12.22.1.)

Im Beispiel [neopixel_bsp1.ino](#) ist ein Würfel programmiert. Das Programm ist aufgeteilt in zwei Funktionen. Die Funktion `würfeln` ermittelt dabei die Augenzahl und aktiviert die entsprechenden LED's im Array.

Die Funktion `ledstripe()` ermittelt zuerst aus dem Array `led[]` und der gewünschten Farbe die Daten für das Array `pixels.Color()` und gibt die Daten am Port 8 aus und schreibt sie in den Pin“13“ des LED-shields.

12.24.3. Zeilen und Spalten ansteuern (gleicher Aufbau wie 12.22.1.)

Aufgabenstellung: Als erstes sollen die LED1 ... LED8 angesteuert werden. Dann soll die ganze Spalte unter jeder LED1 ... LED8 angesteuert werden. Im Beispiel [neopixel_bsp2.ino](#) ist gezeigt, wie Spalten nacheinander, oder beliebig mit der Funktion `spalte(col)` angesteuert werden. Col ist dabei die Variable 1 ... 8 mit der dann die entsprechende Spalte aktiviert wird. Zuerst `erase_ar()` aufrufen, um das LED-Array auf null zu setzen. Anschließend werden mit jedem Aufruf von `spalte(col)` die entsprechenden LED's aktiviert. Mit `ledstripe()` werden alle aktivierten LED's eingeschaltet.

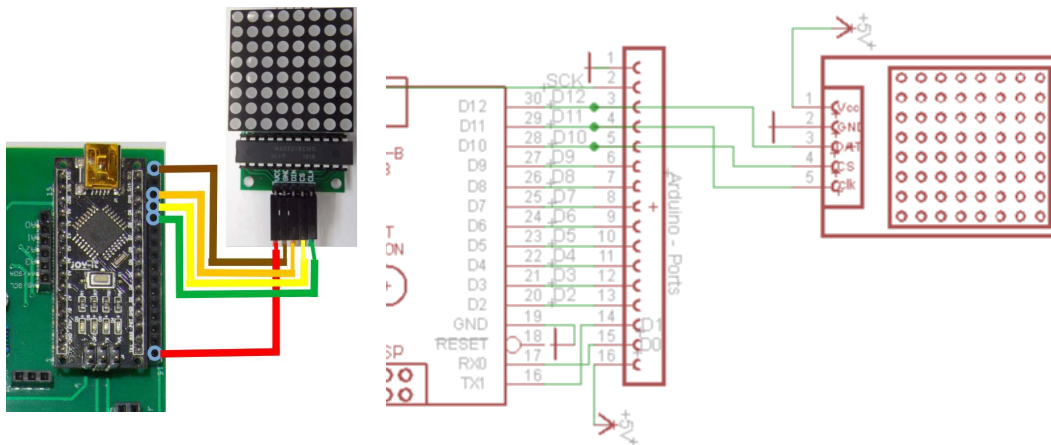
Mit dem Beispiel [neopixel_bsp3.ino](#) wird die Funktion noch einmal vertieft und um die for-Schleife erweitert. Gleichzeitig ist in diesem Programm eine kleine Falle eingebaut:

Die Schleife `for(i=7; i >= 1; i--)` zählt von 7 herunter bis 1. Allerdings, wenn man `i >= 0` setzt, so folgt nach 0 wieder 255. Das rührt daher, weil diese Laufvariable als `unsigned char` definiert ist. Die Werte werden also nicht negativ. Nach 0 beginnt der Wert bei 255 weil die Bedingung `> 0` immer erfüllt ist. Also darauf achten, ob bei der Abfrage, der Wert für die Abbruchbedingung auch wirklich erreicht werden und damit die Abbruchbedingung erfüllt werden kann.

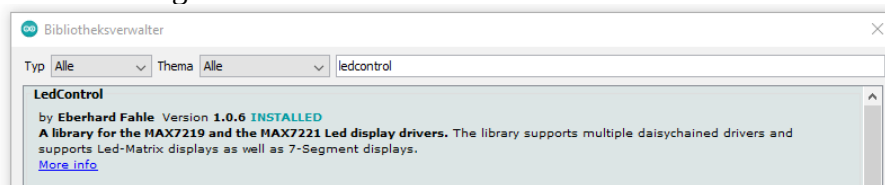
Das Beispiel [neopixel_bsp4.ino](#) zeigt, wie die Zeilen anzusteuern sind.

Was machen die Beispiele [neopixel_bsp5.ino](#) und [neopixel_bsp6.ino](#) ?

12.25. MAX712 LED Matrixanzeige (Artikelnummer 810692)



Bei der Matrix-Anzeige gibt es schon sehr viele Beispiele, die als Beispielsketch nach dem Einbinden der jeweiligen Bibliothek, aufgerufen und dann in den Arduino geflasht werden können. Bei allen Beispielen ist zu beachten, dass die Signale mit den richtigen Ports belegt sind. Es werden auch die Anzahl der Anzeigen abgefragt. Man kann nämlich mehrere (bis zu 7) Anzeigen nacheinander schalten. Allerdings nur, wenn eine zusätzliche Spannung von 5V für die Matrixanzeigen zu Verfügung steht. Die zusätzliche 5V Spannungsversorgung muss aber über eine Masse Leitung mit dem Arduino verbunden sein.



Als Bibliothek binden wir ledcontrol über den Bibliotheksverwalter ein:

Dann rufen wir das Beispiel [LCDemoMatrix.ino](#) auf.

Die Helligkeit sollte ohne zusätzliche Spannungsversorgung auch nicht zu hoch gestellt sein:

```
lc.setIntensity(modul,intensity); // intensity hat dabei einen Wertebereich: 0 ... 15;
```

Je größer der Wert, desto größer die Helligkeit und auch der Stromverbrauch !!

Die Verkabelung ist beim Erzeugen des **Objektes** mit anzugeben:

```
LedControl lc = LedControl(12,11,10,1); // LedControl(Data-Pin, CLK-Pin, CS/LOAD-Pin, Anzahl Module)
```

Helligkeit vordefinieren : **#define** brightness 5 // mal so zum Probieren

Einen Buchstaben in einem Array darstellen:

Spalte 1: B01111110,

Spalte 2: B00001001,

Spalte 3: B00001001,

Spalte 4: B00001001,

Spalte 5: B01111110,

0	1	1	1	0
1	0	0	0	1
1	0	0	0	1
1	1	1	1	1
1	0	0	0	1
1	0	0	0	1
1	0	0	0	1
0	0	0	0	0

Spalte1
Spalte2
Spalte3
Spalte4
Spalte5

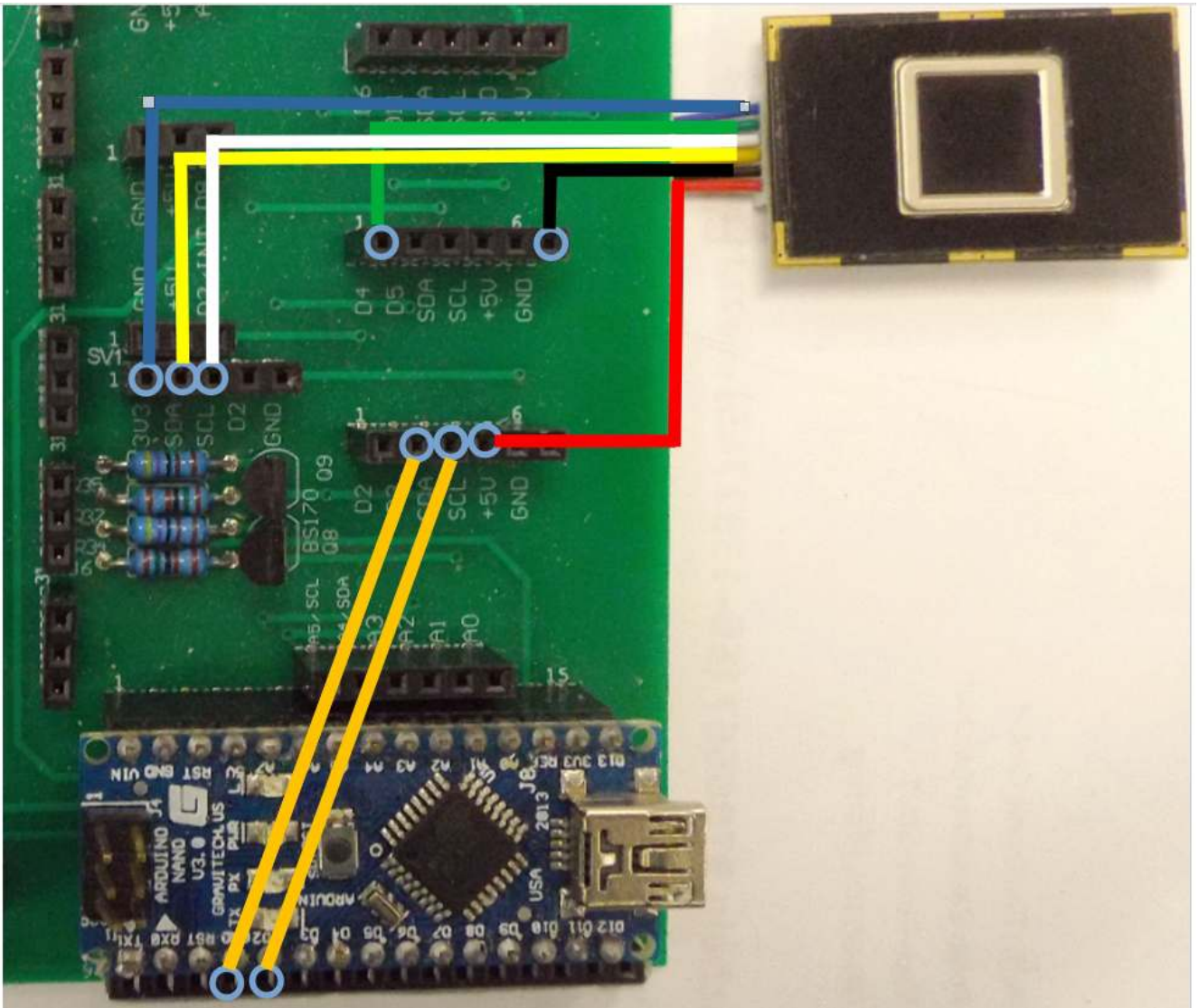
Im Beispiel soll das Prinzip verdeutlicht werden, wie eine Laufschrift

funktioniert. Natürlich gibt es da schon spezielle Bibliotheken, die Text auf

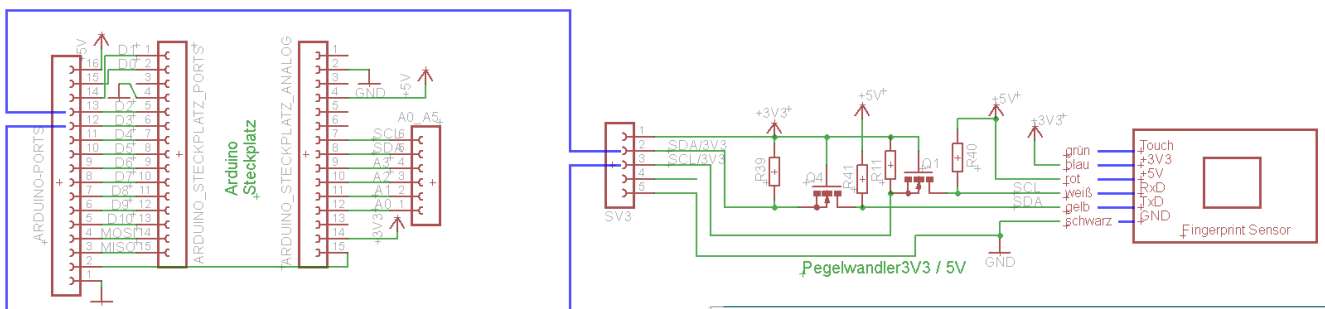
mehrere Matrixmodule ausgeben können. Aber hier soll das Prinzip verdeutlicht

werden. Alle Buchstaben des darzustellenden Textes sind im Array a[48] abgelegt. Dann wird im ersten Durchlauf das Array von 0 ... 7 ausgegeben. Im zweiten Durchlauf wird die Ausgabe bei a[1] begonnen. Mit jedem Durchlauf wird x erhöht, bis der gesamte Text durchlaufen ist. Dann wird x wieder auf 0 gesetzt.

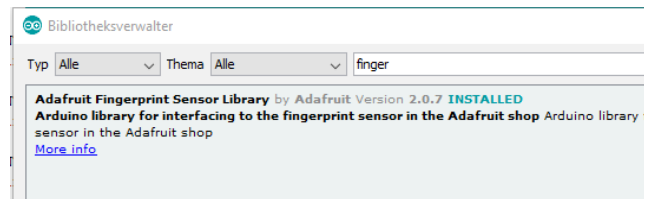
12.26. Fingerabdruck-Sensor (Artikelnummer 180132)



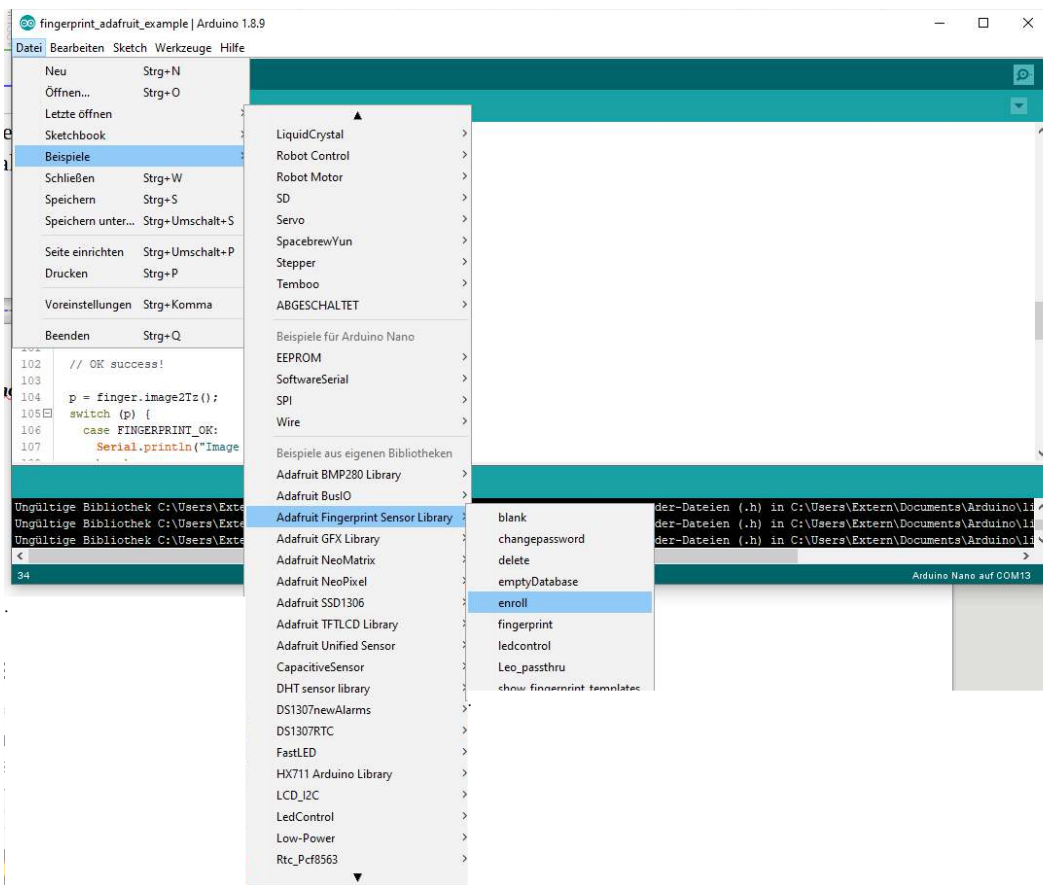
Verdrahtung des Fingerprintsensors:



Zunächst wird die Library mit dem Bibliotheksverwalter installiert: Als erstes wird das Beispiel „enroll“ aufgerufen, in den Arduino geflasht und über Werkzeuge → serieller Monitor mit „9600 Baud“ gestartet. Das Programm wird ausgeführt und dann legt man einen Finger auf den Sensor.



Anschließend muss man die ID wählen unter der das Bild abgespeichert wird. Dann kann z.B: Fingerprint geflasht werden. Nun sollte der Finger mit dem Modul erkannt werden.



Oben im Bild ist dargestellt, wie nach der Installation ein Beispiel aufgerufen und anschließend in den Arduino geladen werden kann. Wie bei allen Programmen, die als Beispiele geladen werden, ist zu beachten, dass die Verkabelung auch dem entspricht, wie die Ports oder Pins des Arduino belegt werden. Dies ist immer zu prüfen. Hier jedoch betrifft dies die Pins 2 und 3. Allerdings fällt auf, dass diese zwar kontaktiert werden, aber sie sind nicht direkt mit dem Modul verbunden. Das resultiert daraus, dass das Modul nur mit 3,3V Spannung arbeitet und die Eingänge nicht 5V tolerant sind. Deshalb der Umweg über SDA und SCL. Denn diese beiden Pins A4 und A5 des Arduino verfügen über eine Pegelwandlung, bestehend aus Q8 und Q9. So dass nach der obigen Verdrahtung, auch nur 3V3 Pegel am Modul anliegen werden.

12.27. Der Interrupt Befehl

Interrupt bedeutet unterbrechen. Das bedeutet, die laufende Programmroutine wird unterbrochen und dafür ein anderes Programm ausgeführt. Interrupts werden überwiegend verwendet, wenn bei einer Aktion, der Mikrocontroller schnell auf ein Ereignis reagieren soll. Mit einer Interruptroutine kann aber auch Rechenleistung eingespart werden. Wenn man z.B auf einen Tastendruck oder ein externes Signal von einem Sensor wartet, das aber unvorhergesehen eintreten kann, dann ist eine Interruptfunktion nützlich, weil es erspart, laufend im Programm den Status eines Pins abzufragen. Beim Interrupt wird sofort eine Aktion ausgeführt, wenn der Status sich ändert und nicht erst, wenn das Programm gerade zufällig in der Lage ist den Status abzufragen. Wenn z.B. der Airbag im Auto aktiviert werden soll, dann muss der Mikrocontroller sofort reagieren und nicht erst, wenn er seine laufende Programmroutine beendet hat. Sonst könnte es schon zu spät sein.

Ein externer Interrupt, wie er in diesem Kapitel behandelt wird, kann nur von bestimmten Anschlüssen des Mikrocontrollers generiert werden. Beim Arduino uno, mini und nano, die einen ATMEG328 verbaut haben, sind dies die Pins D2 und D3. Nur diese beiden Pins sind mit dem Interrupt-block im Mikrocontroller verbunden.

Um Interrupts im Programm nutzen zu können, ist die Funktion `attachInterrupt()` notwendig. Damit verbindet sich der zu wählende Pin mit der Interruptfunktion. Es sind beim Programmaufruf von `attachInterrupt()` noch einige Parameter zu definieren und dem Programm bei dessen Aufruf zu übergeben:

`attachInterrupt(digitalPinToInterrupt(PIN), Funktionsname, Modus)`

PIN ist der Portpin D2 oder D3, der als Interruptpin verwendet werden soll;

Funktionsname ist der Name der Programmroutine, die im Interruptfall ausgeführt werden soll.

Modus ist unter welchen Umständen der Interrupt ausgeführt werden soll:

LOW bedeutet, wenn der Pin den Zustand von 0V erreicht;

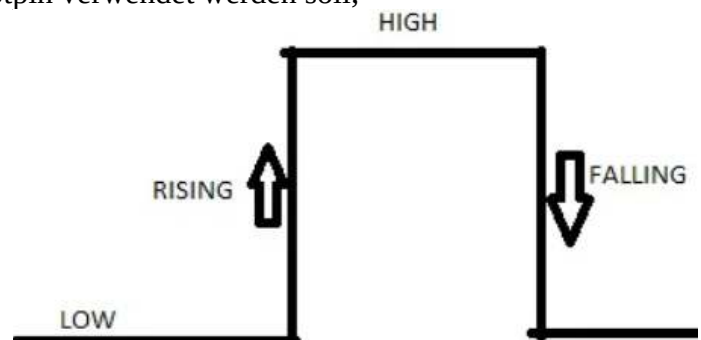
HIGH bedeutet, wenn der Pin den Zustand 5V erreicht

CHANGE bedeutet, jedes mal wenn entweder die Spannung für High-Pegel überschritten oder für LOW-Pegel unterschritten wird, wenn sich also der Zustand ändert.

FALLING bedeutet, wenn der Prozessor am PIN eine fallende Flanke des Signals erkennt

RISING bedeutet, wenn der Mikrocontroller eine steigende Flanke des Signalpegels detektiert.

Die Funktion liefert keinen Rückgabewert.



Es gibt einige Funktionen, die selber Interrupts benutzen.

Deshalb ist zu beachten, dass ein `delay(warte)` in einer Interruptfunktion nicht funktionieren wird. Ebenso wie die Funktionen: `millis()`, `micros()`

Die Funktion `delayMicroseconds()` jedoch kann verwendet werden, weil sie nur mit dem Dekrement arbeitet.

Das bedeutet mit jedem Taktzyklus wird eine Zahl so lange dekrementiert, bis sie 0 ist und dann arbeitet das Programm weiter.

Es gibt natürlich andere Möglichkeiten, Interrupts in den Arduino einzubinden, aber das ist ein sehr komplexes Thema. Für den Einstieg reicht es allemal aus sich auf diese Varianten zu konzentrieren.

Verwenden wir als Beispielcode eine blinkende LED:

Programmbeispiel:

```
unsigned char LED = 13;           // Setze den Pin für die LED auf 13 = interne LED
const unsigned char interruptPin = 2; // Setze den Interrupt Pin auf D2
```

```
volatile unsigned char status = LOW; // Definiere eine globale volatile Variable für den Status der LED
```

```
void setup()
```

```
{
  pinMode(LED, OUTPUT); // Lege den Pin für die LED als Outputpin fest
  pinMode(interruptPin, INPUT_PULLUP); // Lege den Interrupt Pin als Input mit mit internem
  Pullupwiderstand fest
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}
```

```
void loop()
```

```
{
  digitalWrite(ledPin, state); // gib einfach fortlaufend den Status der LED aus
}
```

```
void blink() // dies ist die Interruptfunktion, genannt Interrupt Service Routine ISR
```

```
{
  status = !status; // mit Änderung des Zustandes des Pins D2 ändert sich der Zustand von status
}
```


12.28. WiFi modul ESP12 (Artikelnummer 811152)



Grundverdrahtung des WLAN moduls ESP12F mit dem Arduino (Software Serial verwenden!):

Dieses Modul bringt den Vorteil, dass es mit 5V versorgt werden kann. Da das ESP-Modul nur mit 3,3V arbeitet, ist auf dem Board auch gleich ein Pegelwandler für die Signale (RX und TX) und ein Spannungswandler von 5V → 3,3V. Zu den beiden Schnittstellensignalen ist dann neben der Spannungsversorgung von 5V nur ein EN (enable) Signal zusätzlich noch notwendig. Dieser Pegel ist High-Aktiv, das bedeutet, wenn das Signal LOW ist, ist das Wifi-Modul deaktiviert. Wenn das Signal „HIGH“ ist, ist das Modul in Betrieb.

Grundlegendes zu AT-Befehlen:

Früher wurden zur Ansteuerung von einem Modem, die sogenannte AT-Befehle verwendet. Dabei steht die Abkürzung AT für attention. Der Befehlssatz wurde von der Firma Hayes entwickelt und zum Standard etabliert. Ein Modem wurde in den 1980er Jahren verwendet, damit Computer die in verschiedenen Orten standen, miteinander kommunizieren konnten. Auch das Internet lief in den 1990er Jahren über Modems. Dazu brauchte man dann die Befehle ATD0890815000 um mit dem Modem sich in einen anderen Computer einzuwählen. Diese Art der Befehle hat ich dann auch bei WiFi-Modulen etabliert und wurde durch die Hersteller erweitert.

Hier nun ein kleiner Überblick über die grundlegendsten Befehle:

AT\r\n liefert als Antwort **AT\r\r\n\r\nOK\r\n** zurück. \r\n steht für line feed (neue Zeile) und \r oder cr für carriage return (Wagenrücklauf, also an den Zeilenanfang zurück kehren) . Das stammt noch es den Zeiten der Fernschreiber. Der AT Befehl kann dazu verwendet werden, zu prüfen, ob die eingestellte Baudrate richtig ist. Standardmäßig sollte bei einem WiFi-Modul die Baudrate auf 115200 stehen. Diese kann aber vom Nutzer oftmals geändert werden, oder auch durch einen Reset eingestellt werden. Ist man sich nicht sicher, welche Baudrate derzeit gültig ist, kann diese mit dem AT-Befehl kontrolliert werden. Dazu wird einfach die Baudrate verändert und wenn sie richtig ist, dann bekommt man als Antwort auf den Befehl AT ein Echo des Befehls und ein OK zurück. Alle Befehle sollten mit \r\n abgeschlossen werden. Die Zeichenfolge AT alleine, wird nicht als Befehl erkannt und das Modem reagiert nicht auf diese Zeichenfolge alleine, oder sendet einen „Error“ als Text zurück.

ATE0\r\n und **ATE1\r\n**:

Mit dem Befehl ATE0 kann das Echo abgeschaltet werden und mit ATE1 kann es wieder angeschaltet werden. Da beim Arduino die Zeichen, die im Empfangspuffer stehen alle wieder ausgelesen werden müssen, kann es durchaus sinnvoll sein, das Echo abzuschalten. Dann kann sich der Programmierer und der Arduino rein auf die Nutzdaten konzentrieren.

AT+GMR \r\n:

damit kann die Firmware des Wifi-Moduls abgefragt werden:

Als Antwort könnte z.B. folgender Text empfangen werden:

version:1.2.0.0(Jul 1 2016 20:04:45) \r \n SDK version:1.5.4.1(39cb9a32) \r \n Ai-Thinker
Technology Co. Ltd. \r \n Dec 2 2016 14:21:16 \r \n OK \r \n

Um abzufragen, in welchem Modus sich das WiFi-Modul befindet sende: **AT+CWMODE?** \r \n
Die Antwort des Moduls ist dann 1: Client 2: Access Point oder 3: beides. Wenn man den Modus
änder möchte, sende einfach **AT+CWMODE=1** \r \n, wenn das Modul nur ein Client sein soll.

Wenn der Modus geändert wurde, muss das Modul neu gestartet werden. Dies geschieht am
Einfachsten mit der Befehlsfolge **AT+RST** \r \n.

Möchte man wissen, wie viele Access Points für das WiFi-Modul erreichbar sind, nutzt man dazu
die Befehlsfolge **AT+CWLAP** \r \n.

Die Antwort des Moduls könnte dann so ähnlich aussehen:

```
<\n>+CWLAP:(3,"WerkstattDSL",-66,"c8:0e:14:7b:a0:9f",1,1,0)<\r>
<\n>+CWLAP:(3,"Extern",-77,"00:1a:8c:c4:24:01",1,-31,0)<\r>
<\n>+CWLAP:(3,"Extern",-68,"00:1a:8c:c4:1e:e4",1,-34,0)<\r>
<\n>+CWLAP:(4,"Gast",-68,"00:1a:8c:c4:1e:e5",1,-32,0)<\r>
<\n>+CWLAP:(3,"Test-Wlan",-69,"00:1a:8c:c4:1e:e6",1,-34,0)<\r>
<\n>+CWLAP:(4,"Gast",-84,"00:1a:8c:c4:24:03",1,-31,0)<\r>
<\n>+CWLAP:(3,"Extern",-56,"00:1a:8c:c4:23:57",6,-26,0)<\r>
<\n>+CWLAP:(4,"Gast",-56,"00:1a:8c:c4:23:59",6,-26,0)<\r>
<\n>+CWLAP:(3,"Test-Wlan",-56,"00:1a:8c:c4:23:5a",6,-26,0)<\r>
<\n>+CWLAP:(4,"Gast",-88,"00:1a:8c:c4:23:9c",6,-26,0)<\r>
<\n>+CWLAP:(3,"Extern",-89,"00:1a:8c:c4:23:9b",6,-27,0)<\r>
<\n>+CWLAP:(3,"Extern",-77,"00:1a:8c:c4:23:68",7,-26,0)<\r>
<\n>+CWLAP:(4,"Gast",-76,"00:1a:8c:c4:23:6a",7,-26,0)<\r>
<\n>+CWLAP:(4,"Gast",-85,"00:1a:8c:c4:1e:f6",8,-31,0)<\r>
<\n>+CWLAP:(3,"Test-Wlan",-91,"00:1a:8c:c4:c4:ac",11,-19,0)<\r>
<\n>+CWLAP:(3,"LinksysWerkstatt",-61,"00:18:f8:6c:d8:97",11,0,0)<\r>
<\n>+CWLAP:(3,"Extern",-81,"00:1a:8c:87:ee:17",11,-26,0)<\r>
<\n>+CWLAP:(4,"Gast",-79,"00:1a:8c:87:ee:18",11,-26,0)<\r>
<\n>+CWLAP:(4,"Gast",-76,"00:1a:8c:87:ed:c3",11,-27,0)<\r>
<\n>+CWLAP:(4,"Gast",-87,"00:1a:8c:c4:c4:ab",11,-19,0)<\r>
<\n><\r>
<\n>OK<\r>
<\n>
```

Dabei ist die **erste Ziffer** in der Klammer die Art der Verschlüsselung:

- 0: keine
- 1: WEP
- 2: WPA_PSK
- 3: WPA2_PSK
- 4: WPA_WPA2_PSK
- 5: WPA2_ENTERPRISE
- 6: WPA3_PSK
- 7: WPA2_WPA3_PSK
- 8: WAPI_PSK

Der zweite zurückgegebene Parameter ist die **SSID**, der Name des Access Points.

Der dritte parameter ist die sogenannte **rsssi** (Received Signal Strength Indicator, also ein Messwert
der Empfangsfeldstärke. So kann festgestellt werden, welcher Acces Point am besten empfangen
werden kann. Zu beachten ist, dass der Wert negativ ist und je **kleiner** die Ziffer, desto **besser** das
empfangene Signal. Das liegt daran dass der Wert in dezibel gemessen wird und Werte von 0 ... -
120 haben kann. Dezibel ist dabei keine physikalische Größe, sondern ein Verhältniswert.

Der vierte Parameter ist die sogenannte **MAC-Adresse**. MAC steht in diesem Fall für **MediaAccessControl**-Adresse. Früher wurde festgelegt, dass in einem Rechnernetz jeder Teilnehmer eine Geräteadresse braucht. Diese Adresse wird in dieser 48Bit-Folge abgespeichert.

Der fünfte Parameter ist der **Kanal**. In Europa sind Kanäle von 1 ... 13 erlaubt.

Mit der Befehlsfolge **AT+CWJAP="SSID", "password"** kann sich das WiFi-Modul mit einem Access point verbinden.

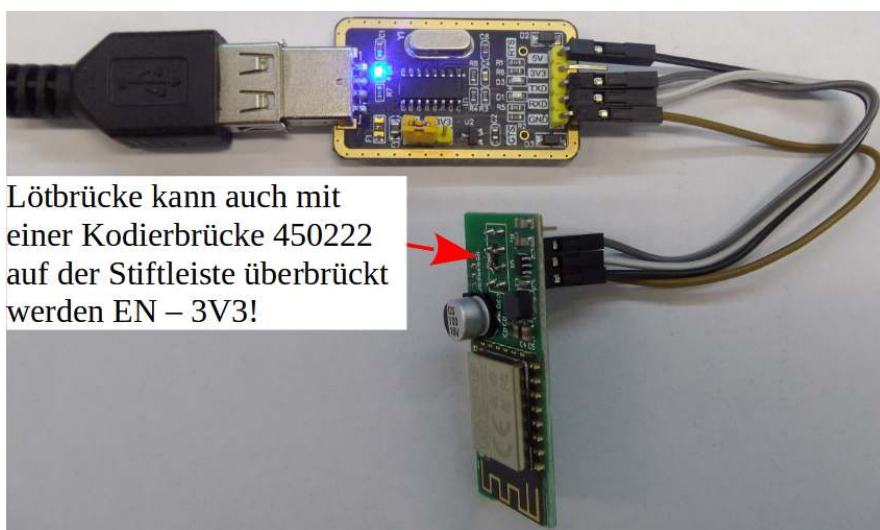
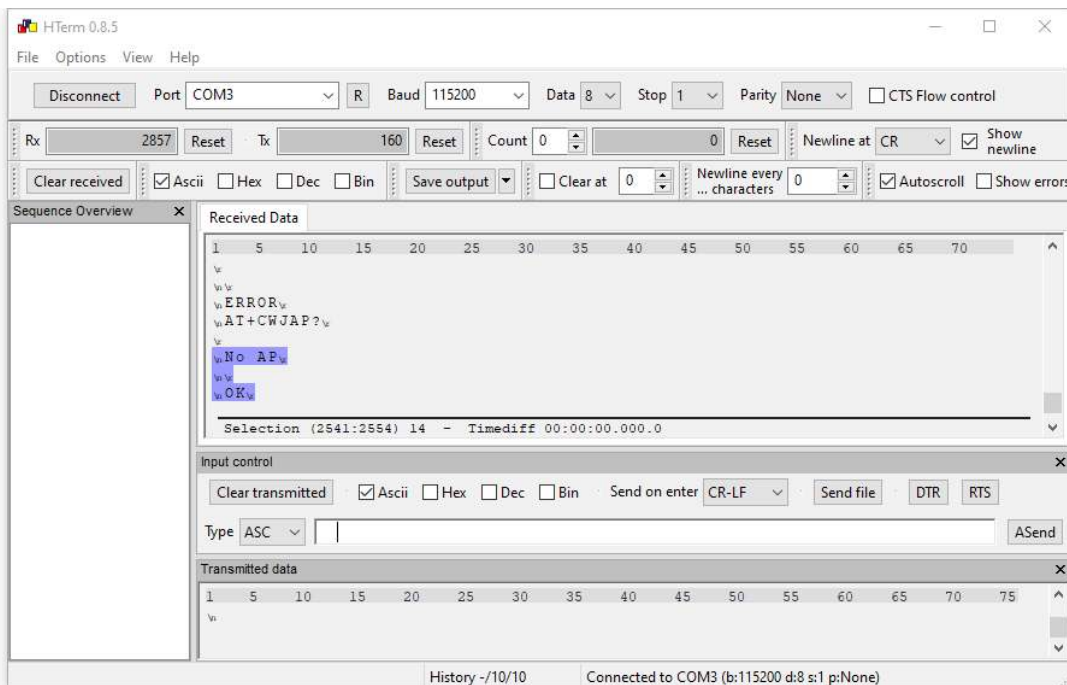
Mit der Befehlsfolge **AT+CWJAP=?** wird abgefragt, mit welchem Access Point das Modul gerade verbunden ist. Falls es nicht verbunden ist, erscheint folgende Antwort: `\n No AP\r\n\r\n OK \r\n`.

Es gibt natürlich eine Vielzahl weitere Befehle, die im Internet bei Espressif, dem Hersteller u.a. des ESP8266 und des ESP12 WiFi-Moduls. Aber alles zu erklären würde den Rahmen dieser Einführung sprengen. Verwiesen sei auch auf das nachfolgende Projekt im Kapitel über ein Bluetooth-Modul, weil dessen Ansteuerung auch über AT-Befehle erfolgt und auch über eine zweite serielle Schnittstelle, einer sogenannten seriellen Softwareschnittstelle. Denn zum Betrieb benötigt man zwei serielle Schnittstellen, eine zur Kommunikation mit dem PC und die zweite für die

Verbindung zum Modem / WiFi-Modul.

Ideal zum Test des WiFi-Moduls ist der Schnittstellen-Wandler mit der Artikelnummer 811109.

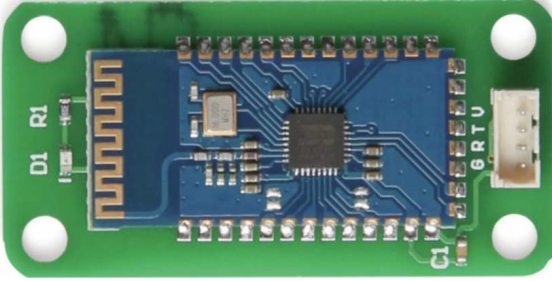
Dann lässt sich mit einem Terminalprogramm z.B: hterm das Modul testen.



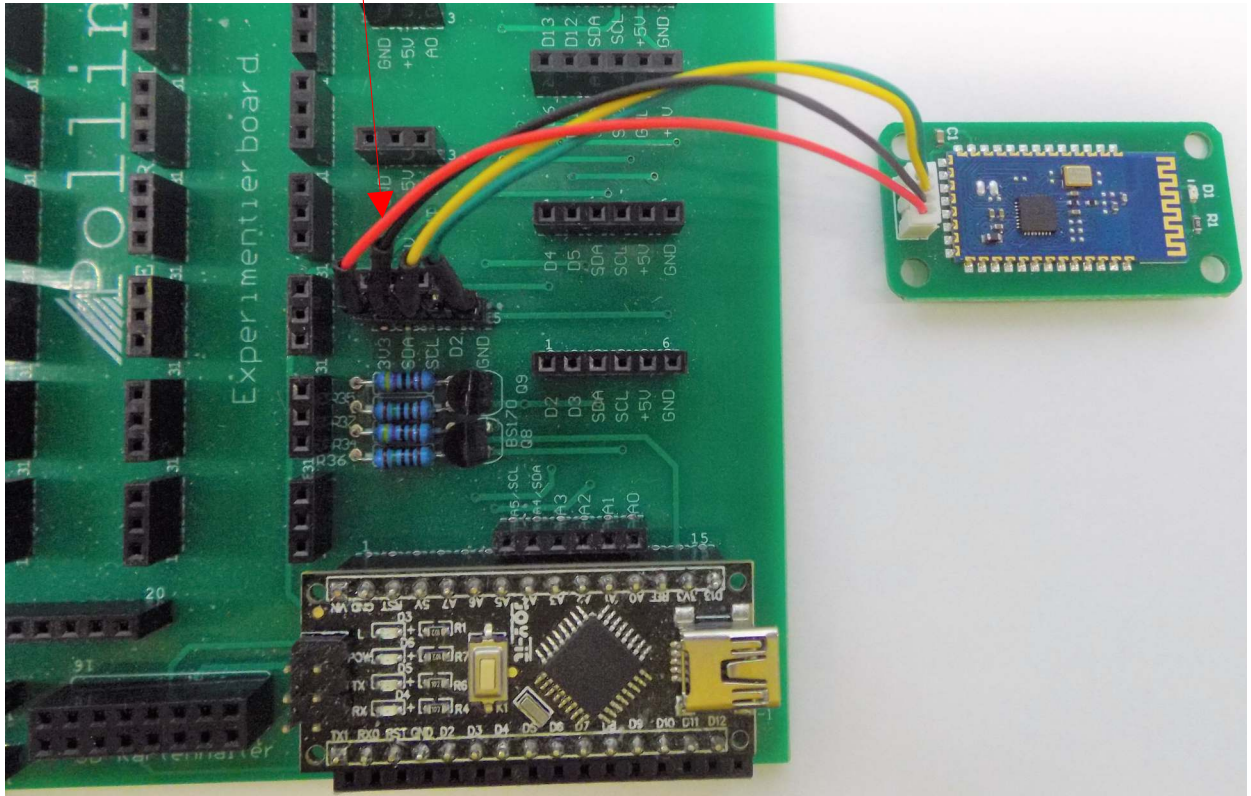
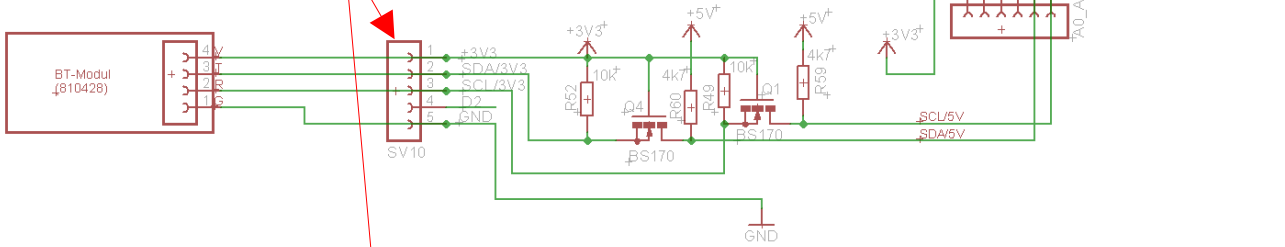
Lötbrücke kann auch mit einer Kodierbrücke 450222 auf der Stiftleiste überbrückt werden EN – 3V3!

12.29. Bluetooth-Modul (Artikelnummer 810928)

12.29.1. Aufbau: Datenübertragung und schalten einer LED



Kabelenden an Stiftleiste angelötet



Um das Bluetooth-Modul zu betreiben ist zuerst der Stecker an einem Ende der Leitung abzutrennen. Dann die Kabel abisolieren und in die entsprechenden Buchsen stecken. Es können die offenen Leitungen auch an eine 5 polige Stiftleiste angelötet werden. Dafür den Artikel 451550 mit der richtigen Polzahl ablängen und die losen Enden des Moduls in der richtigen Reihenfolge, wie im Schaltplan oben gezeigt, anlöten. Gegebenenfalls Schrumpfschlauch verwenden, wegen der Isolation und als Knickschutz der dünnen Litzen. Eine Lötstelle birgt auch immer die Gefahr einer Sollbruchstelle.

Anmerkungen zum Arduino Sketch:

Die Kommunikation vom Arduino zum Bluetooth-Modul läuft über eine serielle Schnittstelle. Die eingebaute Schnittstelle des Arduino wird verwendet, um mit dem PC zu kommunizieren. Also benutzen wir hier einen Software-UART.

So wird mit `#include <SoftwareSerial.h>`, die Bibliothek für die zusätzliche Schnittstelle eingebunden.

Mit der Definition der Instanz: `SoftwareSerial mySerial(2, 3);`

wird die neue Schnittstelle erstellt und Pin 2 der Empfängerpin und Pin 3 als der TX-Pin definiert. Beim Arduino Nano und Arduino UNO können nur diese beiden Pins für den SoftwareSerial-Betrieb genutzt werden, weil nur diese einen Hardware Interrupt anzeigen können. Bei anderen Prozessoren kann die Interruptfunktion auch an anderen Pins genutzt werden.

Um das UART-Interface des Bluetooth-Moduls jedoch mit dem Arduino verbinden zu können, ist ein Pegelwandler notwendig, weil auch dieses Modul mit 3,3V arbeitet. Das Bluetooth-Modul könnte zerstört werden, wenn es direkt mit den Arduino Pins zusammengeschaltet würde. Die Pins A4/SDA und A5/SCL sind im Normalfall als Eingang konfiguriert, weil sie als Eingang des AD-Wandlers verwendet werden. Nur wenn sie als I2C-Interface oder als normaler Port-Pin genutzt werden, werden sie als OUTPUT konfiguriert. Hier jedoch werden sie nur gebraucht, um den Pegelwandler mit Q8 und Q9 nutzen zu können.

Deshalb ist es in diesem Anwendungsfall notwendig, dass Pins A4 und A5 mit den Pins D2 und D3 verbunden sind.

Zuerst muss man die serielle Softwareschnittstelle noch konfigurieren:

Mit der Befehlszeile `mySerial.begin(BAUD);` wird die Schnittstelle für eine bestimmte BAUD-Rate konfiguriert. Mit `#define BAUD 9600` wird die genaue Baudrate festgelegt. Allerdings ist zuerst dann zu prüfen, ob es wirklich auf 9600 Baud eingestellt ist. Falls diese nicht funktioniert, das bedeutet, dass das Bluetooth-Modul nicht antwortet auf einen sogenannten „AT-Befehl“ dann sollte die Einstellung BAUD 115200 noch probiert werden. Dies sind die gebräuchlichsten BAUD-Raten für serielle Kommunikation.

Im Beispiel `BT_test01.ino` wird ein AT-Kommando zum Bluetooth-Modul gesendet und dann das Echo zum PC übertragen. Wird OK am PC empfangen, dann ist die Baudrate richtig eingestellt und die Verbindung zum Smartphone kann aufgenommen werden. Im Beispiel `BT_test02.ino` wird dann eine Kommunikation mit dem Smartphone aufgemacht.

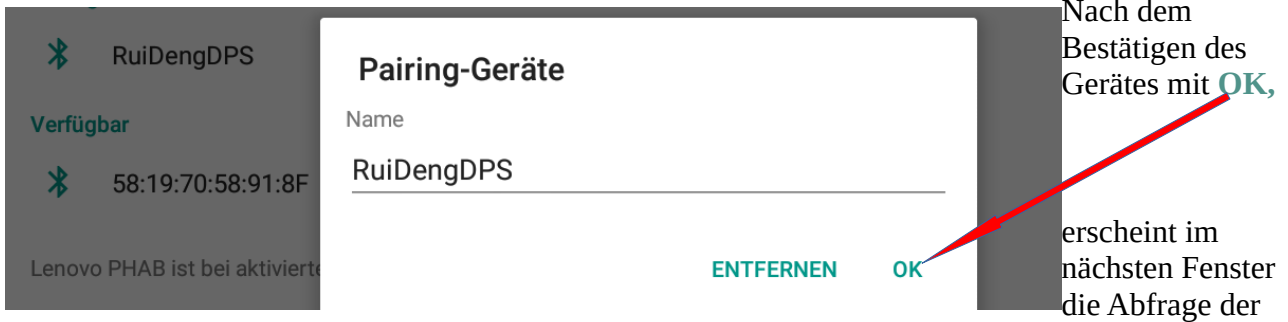
12.29.2.1. Kommunikation Smartphone ← → Arduino

Wenn das Bluetoothmodul an die Versorgungsspannung vom Arduino kit angeschlossen ist, dann beginnt die LED1 auf dem Modul zu blinken.

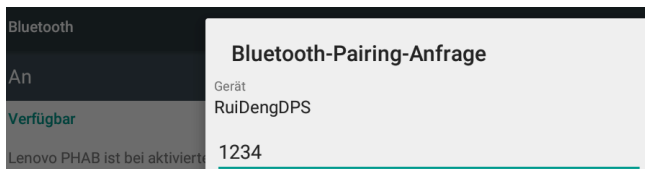
Nun auf dem Smartphone oder Tablet die Bluetooth Schnittstelle in den Systemeinstellungen aktivieren:

Dann muss das Bluetoothmodul: RuiDengDPS in der Liste Verfügbar angezeigt werden.

Wenn nun RuiDengDPS ausgewählt wird, erscheint das nachfolgende Fenster:

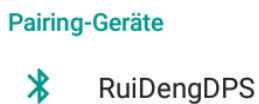


PIN:



Als PIN wird 1234 eingegeben

Wenn die PIN akzeptiert wurde, erscheint nach dem verbundenen Gerät ein Zahnradsymbol:



12.29.2.2. Installation und Konfiguration der Android - APP

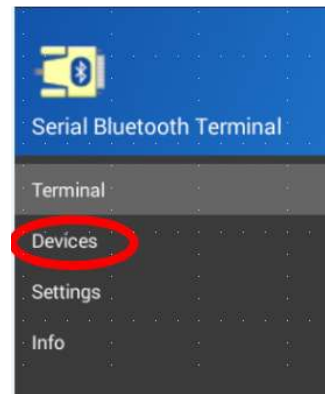
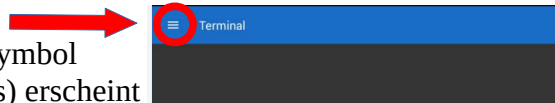
Die App für Android ist erhältlich zum Download:
beim **Google Play Store** und heißt **Serial Bluetooth Terminal**.

Nach der Installation befindet sie sich auf dem Homescreen des Smartphones:

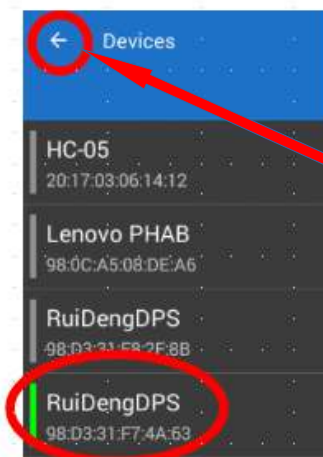


Nach dem Starten der APP, durch Antippen des Icon, erscheint folgender Bildschirm:

Durch Tippen auf das Symbol für Settings (roter Kreis) erscheint das rechte Fenster



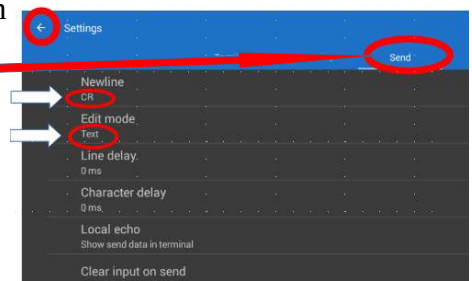
Hinter dem Submenue **„Devices“** verbirgt sich das unten dargestellte Fenster:



Es folgt eine Auflistung aller Bluetoothmodule, die bereits schon einmal verbunden waren. Dann wird das Modul ausgewählt, mit dem eine Kommunikation erfolgen soll. Dieses wird mit einem seitlichen grünen Balken markiert.

Durch Tippen auf den Pfeil (roter Kreis) wird wieder zurück ins Hauptmenue gewechselt. Jetzt den Pfeil im Settings menue antippen und den Untermenüpunkt Send auswählen.

Nun erscheint das rechts abgebildete Menü:



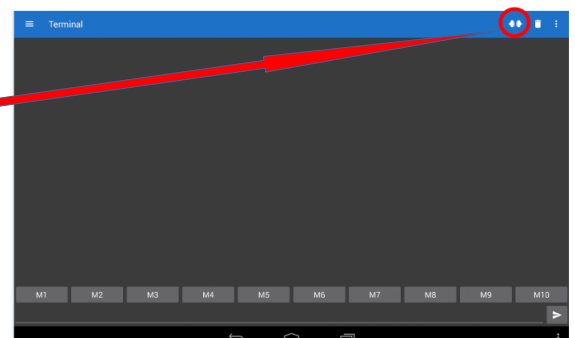
Für die Eingabe des **Beispieltextes** müssen im Reiter Send bei **Newline: CR** eingestellt und im **Edit mode** soll **Text** (weiße Pfeile) ausgewählt sein.

Für die Eingabe von Hexadezimalen Werten muss im Reiter Send bei **Newline: None** und im **Edit Mode: HEX** eingestellt sein !

Dann wieder den Button „Pfeil links“ (siehe roter Kreis neben „Settings“) betätigen.

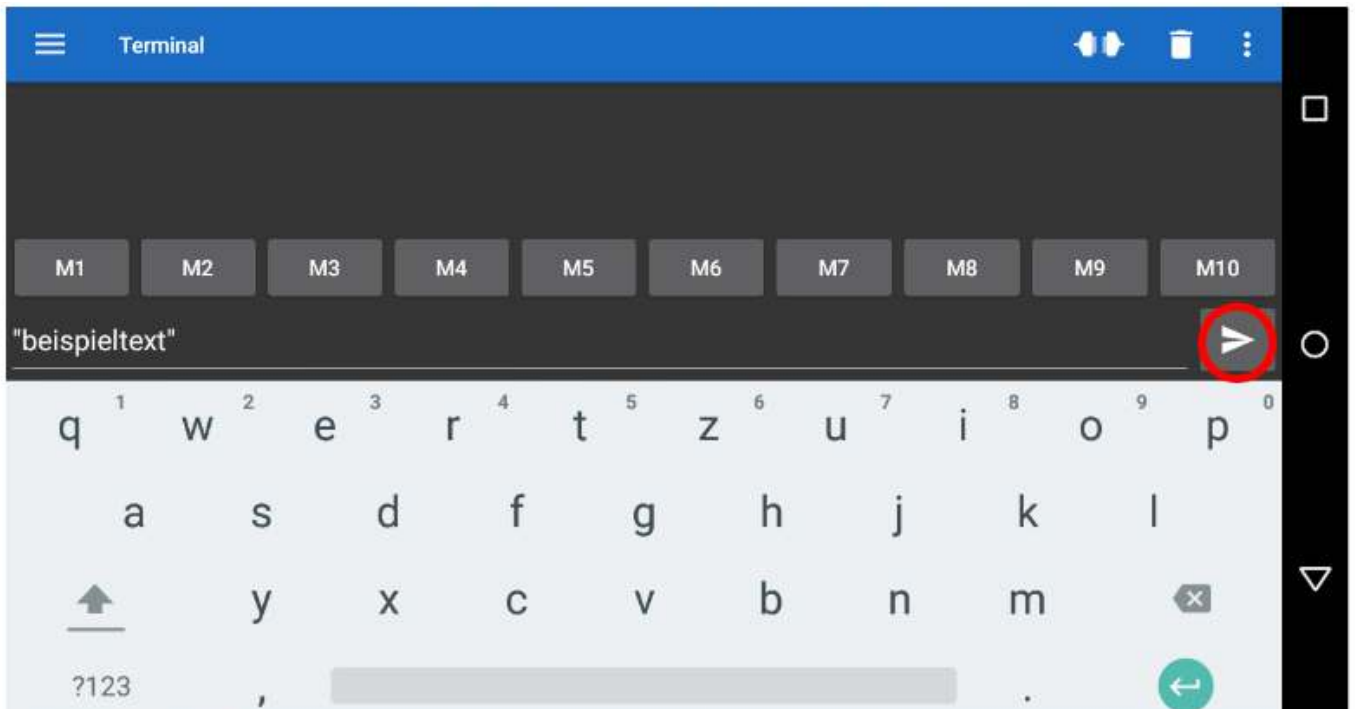
Jetzt ist wieder das Hauptmenü (Bild rechts) aktiv:

Mit dem Connect Symbol, wie er im linken Bild durch einen roten Kreis markiert, kann die Verbindung zwischen Bluetooth-modul und Smartphone geschlossen werden.



Durch Tippen auf das Symbol wird der connect angezeigt und durch ein erneutes antippen wird die Verbindung wieder getrennt.

12.29.2.3. Texte oder Werte übertragen:



Wie im Bild oben, zu sehen, wird der Beispieltext eingegeben und dann mit Betätigung des > Zeichen wird dieser Text übertragen.

Wenn im Reiter Send der Edit Mode auf Hex umgestellt wird, können Hexadezimale Werte übertragen werden. Dies reicht aus, um z.B. eine LED zu schalten. Dabei ist es aber eher die Geschmacksache des Programmierers, welchen Edit Mode er auswählt.

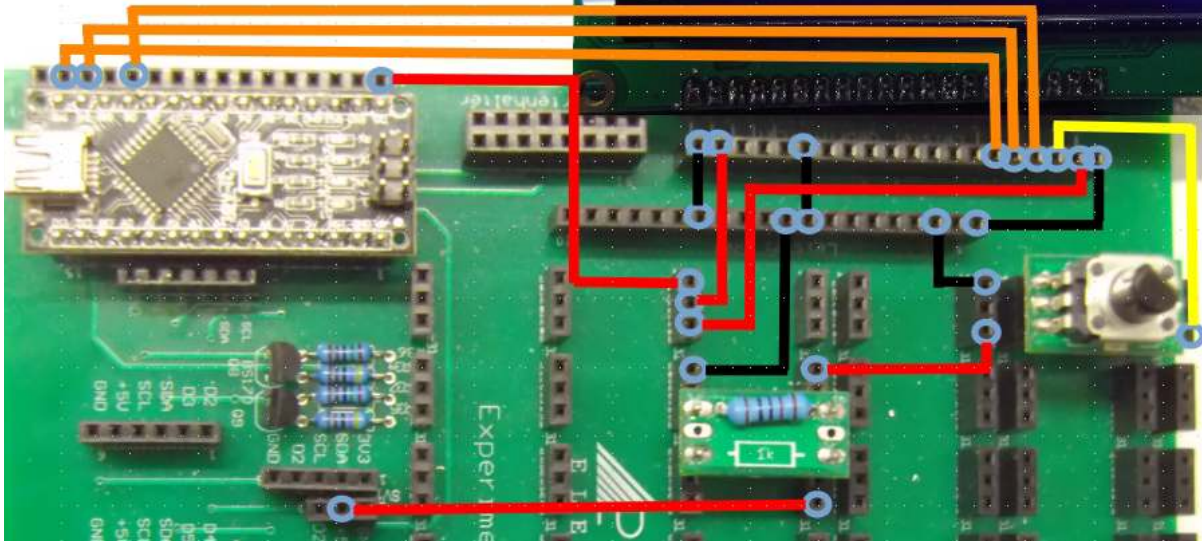
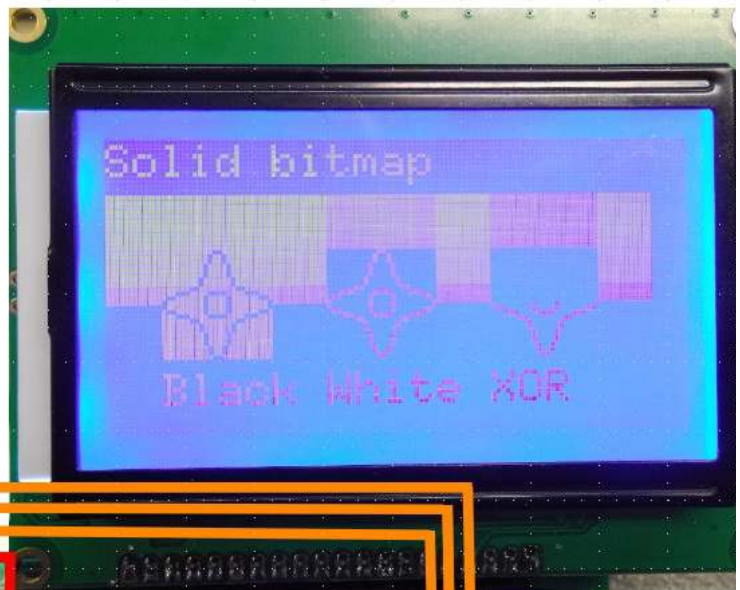
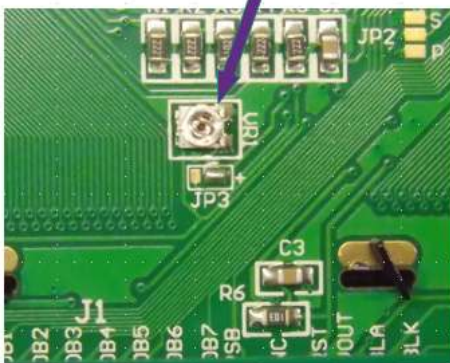
12.30. Ansteuerung eines Grafik-Display(Artikelnummer 121829)

Zuerst die Library u8g2 installieren und dann das Beispielprogramm aufrufen:

Im Menue **Datei** → **Beispiele** → **U8g2** → **full_buffer** → **GraphicsTest** öffnen.

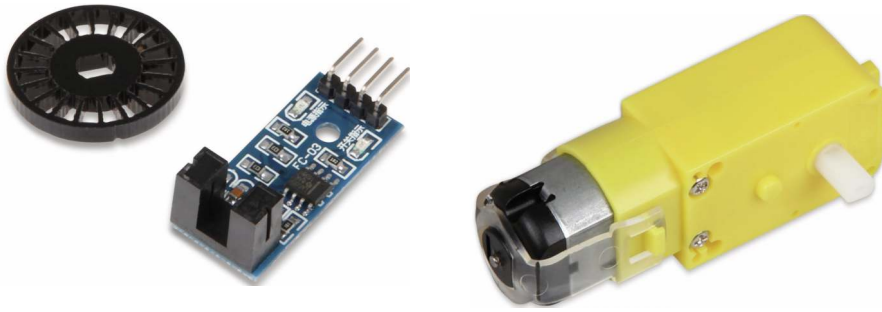
In der geöffneten Datei **GraphicsTest01.ino** nach der „constructor list“ z.B. in Zeile 304 die nachfolgende Zeile einfügen: `U8G2_ST7920_128X64_F_SW_SPI u8g2(U8G2_R0, 13, 11, 10);`
Damit wird die SPI-Schnittstelle konfiguriert. Denn im Gegensatz zu dem Display in Kapitel 12.18. kann dieses Display seriell angesteuert werden. Dies erspart ein paar Leitungen gegenüber dem Parallelbetrieb. Damit werden auch nicht so viele Pins am Arduino benötigt, was ein noch größerer Vorteil ist.

Mit dem Poti **VR1** lässt sich bei Bedarf der Kontrast einstellen, falls am Display nichts zu erkennen ist.

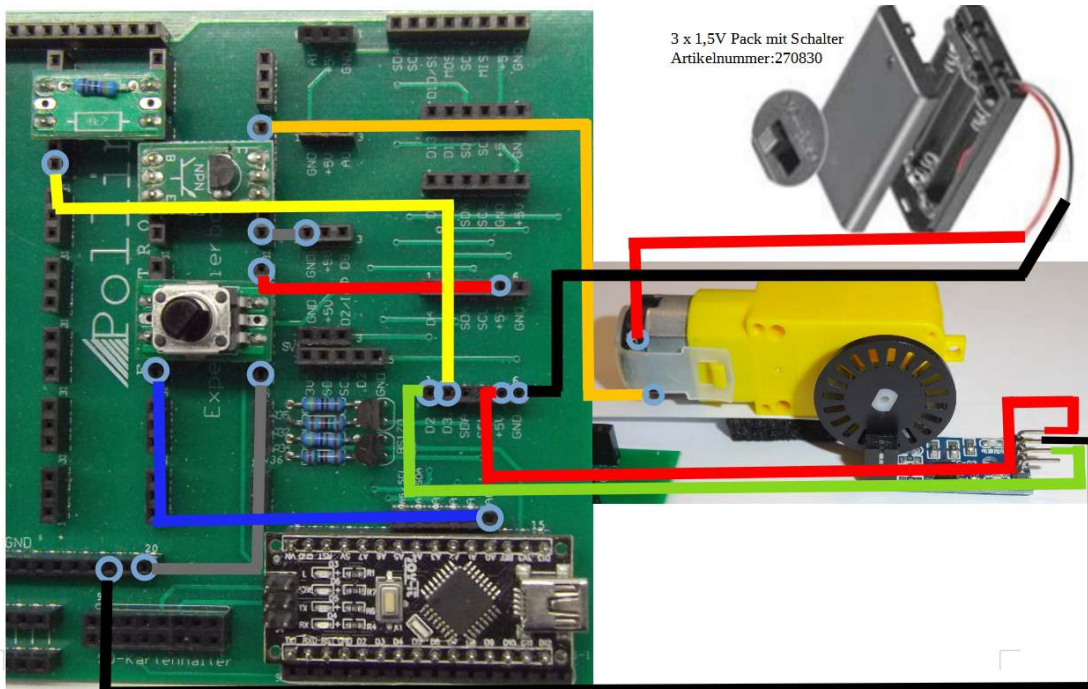
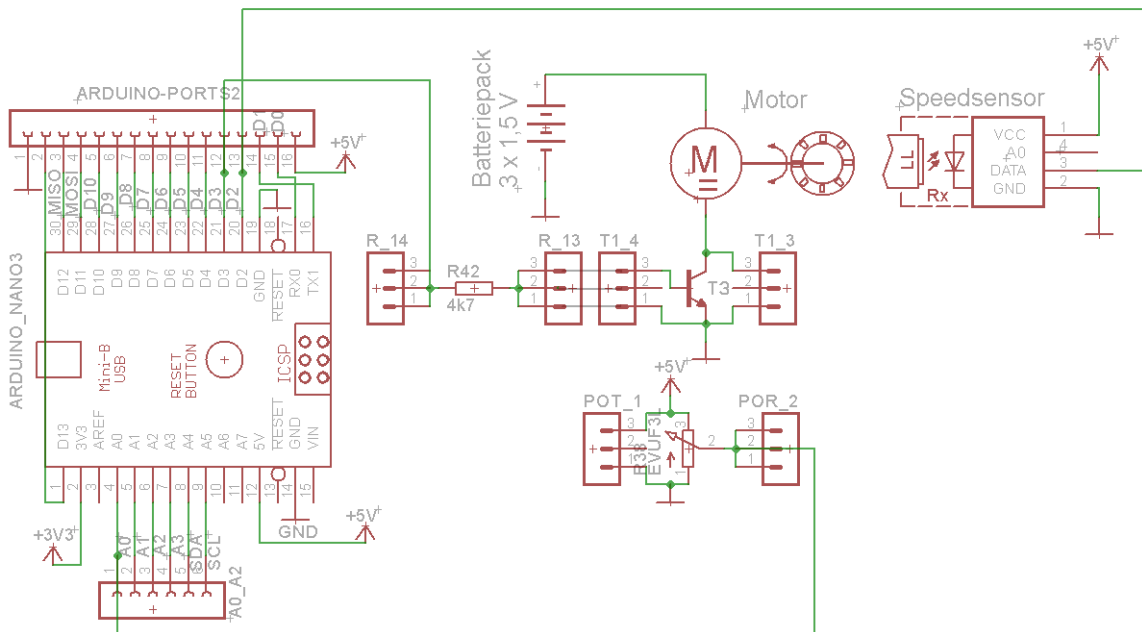


Achtung: beim Poti auf die Einbaulage achten !!

12.31. Speedsensor (Artikelnummer 810878) mit Motor (820 580)

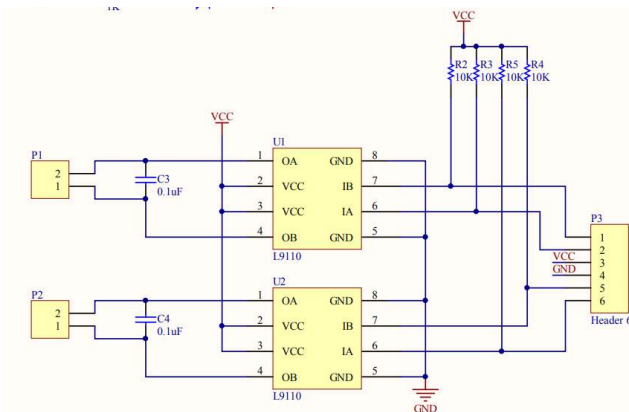


Das Programm ist unter den Namen [PWM_n_mess.ino](#) im Beispiele-Ordner abgespeichert. Durch die Interruptroutine count werden die Impulse für eine bestimmte Zeit gezählt. Daraus errechnet sich die Drehzahl. Diese wird über die PWM mittels des Poti eingestellt.

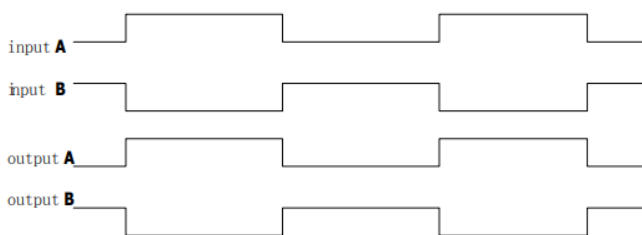
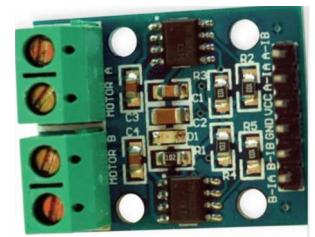


3 x 1,5V Pack mit Schalter
Artikelnummer:270830

12.32. Motorantriebsmodul L9110 (Artikelnummer 810 572)



Im Bild links ist der Schaltplan und rechts ist die tatsächlich Platine dargestellt. VCC kann bis zu 12V betragen. An den Eingängen Pin 1, 2, 5, 6 reichen 5V. Das bedeutet unabhängig von der Versorgungsspannung kann der Motortreiber mit einem Arduino angesteuert werden.



Links im Bild ist dargestellt, wie sich die Ausgänge an einem IC L9110 ändern, wenn sich die Pegel am Eingang ändern. Der IC dient nur als „Schalter“.

Wenn **ein** Motor (z.B. hier **310764**) angeschlossen werden soll, dann wird dieser an P1:2 und P1:1 angeklemt. Die Polarität spielt dabei erstmals eine Nebenrolle, weil diese mit den Steuereingängen vom Arduino festgelegt wird. Mit dem Pin A-1A wird der Ausgang an P1:2 geschaltet. Mit B-1B wird P1:1 geschaltet. Wie im Signaldiagramm zu sehen ist, dreht bei Input A „High“ und Input B „low“ der Motor in die eine Richtung und wenn sich beide Eingänge gleichzeitig umpolen in die andere Richtung. Der maximale Strom den der Motor aufnehmen darf, liegt dabei bei 800mA. Leider ist aus den Datenblättern nicht die maximale Schaltfrequenz zu erkennen. Trotzdem ist es möglich die PWM des Arduino mit AnalogWrite(0 ... 255) auf diesen Baustein zu schalten und so die Drehzahl eines Motors zu regeln.

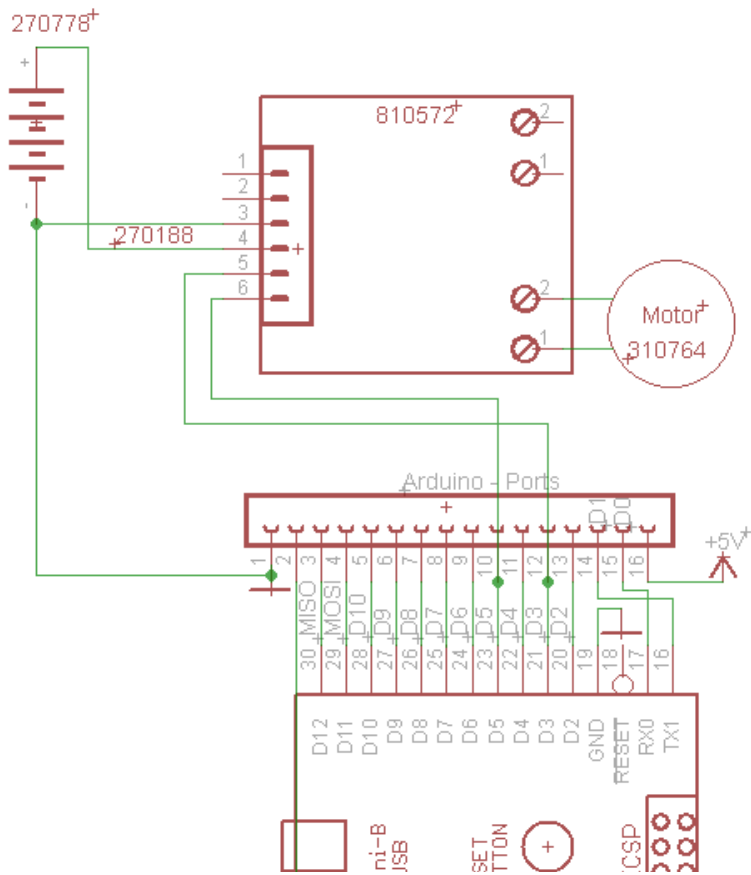
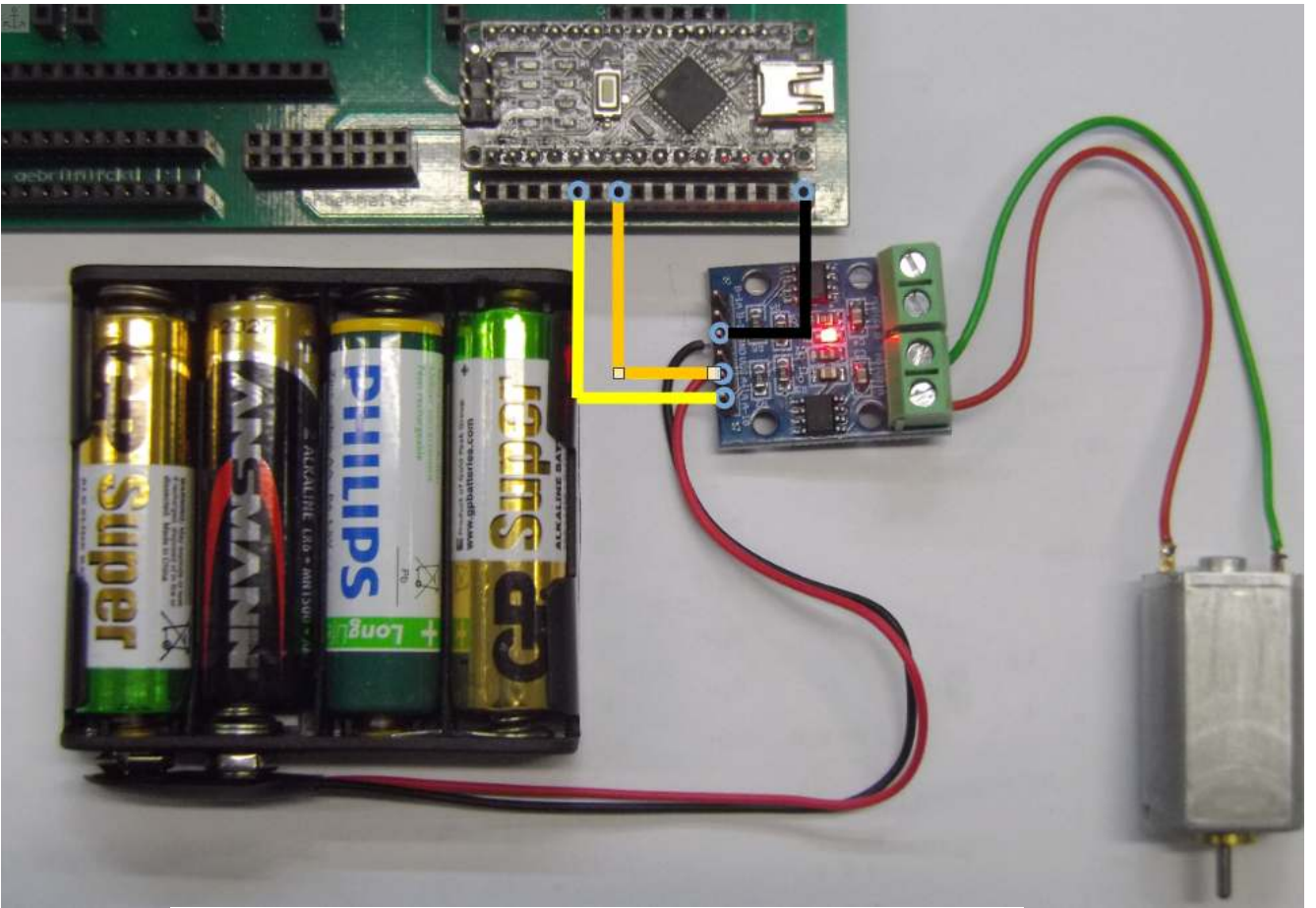
Ein **zweiter** Motor könnte an P2:2 und P2:1 angeklemt werden. Mit dem Pin P3:5 würde der Ausgang an P2:1 geschaltet. Mit einem Signal an Pin P3:6 würde P2:2 geschaltet.

Werden die Pins P1:1 und P2:1 sowie P1:2 und P2:2 gebrückt, so kann ein Motor mit der **doppelten** Stromaufnahme (1600mA) mit diesem Modul betrieben werden. Die Eingangspins A-1A und A-1B sowie B-1A und B-1B müssen ebenso zusammenschaltbar sein, oder gleichzeitig vom Arduino angesteuert werden. Mit den Pin A-1A und A-1B wird der Ausgang an P1:2 geschaltet. Mit B-1A und B-1B wird P2:1 geschaltet.

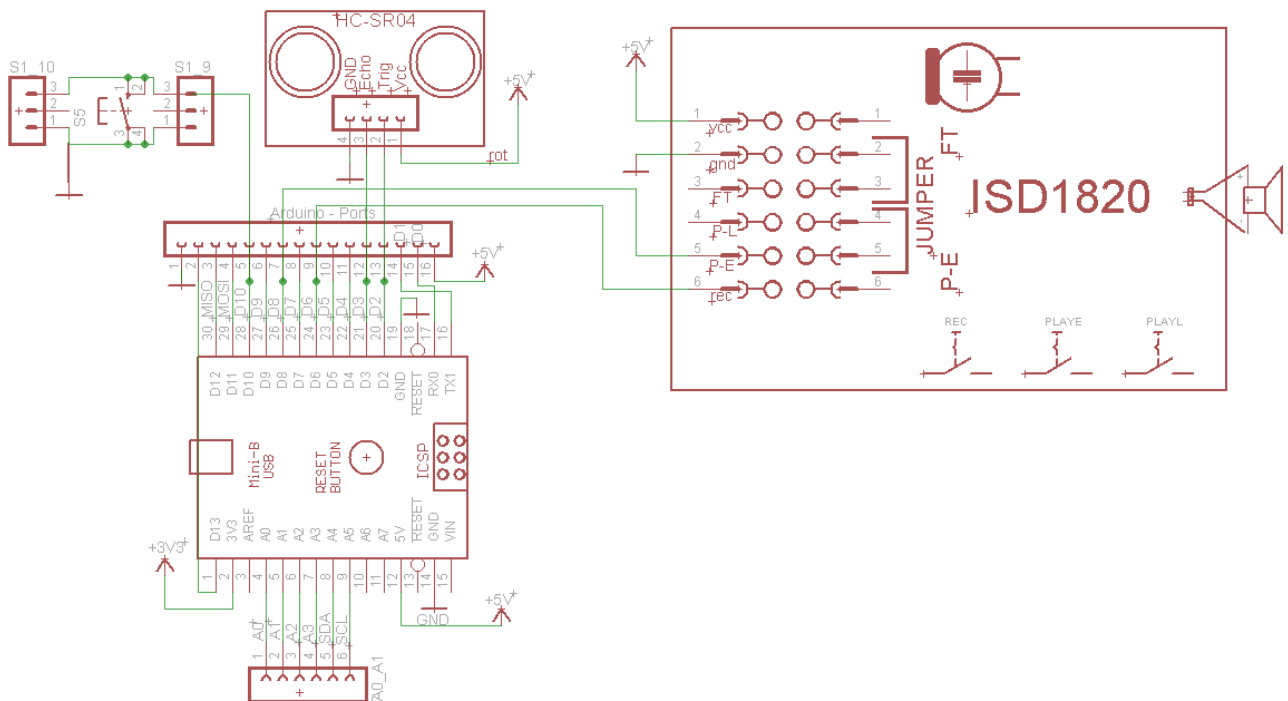
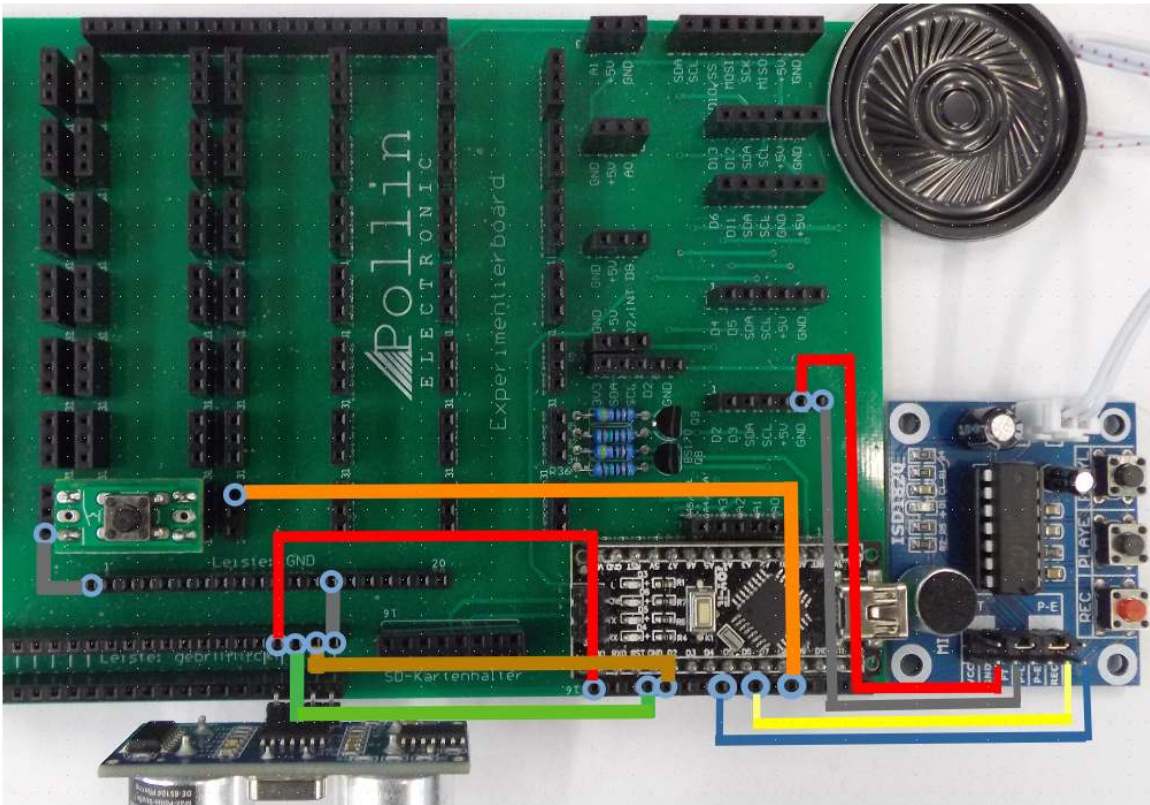
Im Beispiel [L9110Mot1.ino](#) ist gezeigt, wie eine Ansteuerung eines Gleichstrommotors über einen Arduino aussehen könnte: Mit den Seriellen Zeichen ‚s‘ wird der Motor gestoppt, mit ‚l‘ dreht er in die eine und mit ‚r‘ in die andere Richtung. Wichtig ist, dass während der Drehung nicht abrupt die Polarität geändert wird. Besser ist, erst zu stoppen, dann mit umgekehrter Polarität wieder einschalten! Realisiere das mit der Programmänderung: zuerst immer beide Eingänge auf low, delay(wartezeit), dann erst schalte einen der Ausgänge auf HIGH!

Die Spannungsversorgung notfalls über den Arduino (5V am Steckbrett, bei einem kleinen Motor mit einer Stromaufnahme von ca. 200mA), aber besser aus Batteriepack (Batteriehalter 270778 mit Mignonzellen oder Powerbank).

Im Beispiel [L9110Mot2.ino](#) ist gezeigt, wie eine Drehzahlsteuerung über die serielle Schnittstelle realisiert werden kann. Je nach Güte des Motors kann der PWM-Wert für 10% Leistung auch kleiner gewählt werden. Zuerst soll dabei gewählt werden, ob der Motor links oder rechts dreht. Dies wird in einer separaten Variablen gespeichert. Dann kann der Wert eingegeben werden, mit dem das PWM-Verhältnis eingestellt wird. Die Verdrahtung ist auf der nächsten Seite dargestellt.



12.33. Joy-IT Soundrecorder (Artikelnummer 810679)



Für die Ansteuerung benötigt man lediglich zwei Pins des Arduino, einen zum Starten der Aufnahme und den anderen für die Wiedergabe. Am Pin 10 des ICD1820 ist ein 100k Widerstand mit Masse verbunden. Das bedeutet, die Aufnahmezeit beträgt in etwa 10s. Je größer der Widerstand, desto kleiner die Abtastfrequenz und desto länger die Aufnahmezeit, aber eben auch um so schlechter die Klangqualität des aufgenommenen Signals. (siehe Tabelle unten rechts)

Gemäß Abtasttheorem sollte die Abtastfrequenz doppelt so groß sein, wie das abzuspeichernde Signal. Das Soundmodul kann zum Test auch manuell über die Tasten aktiviert werden: Mit Betätigung der Taste **REC** wird die Aufnahme gestartet. Diese wird beendet, wenn die Taste wieder losgelassen wird, oder wenn der Speicher im IC voll ist. **PLAYL** (bei Betätigung von **PLAYL** wird die Wiedergabe gespielt, bis die Aufzeichnung zu Ende ist, oder wenn die Taste losgelassen wird. Bei kurzer Betätigung der Taste **PLAYE** (= etch triggered = flankengetriggert) wird das aufgenommene Geräusch bis zum Ende abgespielt.

Die beiden Jumper sind im Auslieferungszustand nicht aktiv. Der **Jumper** auf **FT** („feed through“) gebrückt bedeutet, dass auf den Lautsprecher, das Signal vom Mikrofon direkt geschaltet wird.

Somit ist am Lautsprecher zu hören, was gerade am Mikrofon an Signalen ankommt.

Der **Jumper** auf **P-E** gesteckt, bedeutet, dass die Wiedergabe des aufgenommenen Geräusches in einer Endlosschleife erfolgt.

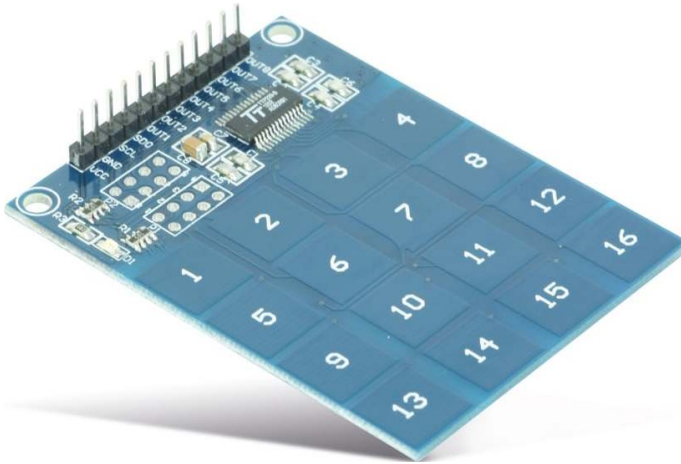
Die Spannungsversorgung des Moduls erfolgt mit 5V. Als Startsignal der Wiedergabe, kann z.B. ein PIR-Sensor, eine Lichtschranke oder ein Ultraschallsensor dienen. Der Start der Aufnahme erfolgt durch die Betätigung mit einem Taster.

Widerstand	Aufnahmezeit	Abtastrate
82 KΩ	ca. 8s	8 kHz
100 KΩ	ca. 10s	6.4 kHz
120 KΩ	ca. 12s	5.3 kHz
160 KΩ	ca. 16s	4.0 kHz
20 KΩ	ca. 20s	3.2 kHz

Im Beispiel [ISD1820_bsp01.ino](#) ist eine Lösungsmöglichkeit gezeigt, das Soundmodul mit einem Arduino zu betreiben. Den Taster könnte man sich natürlich auch ersparen, weil ja auch der auf der ISD1820 Platine benutzt werden kann. Es ist aber gleich eine zusätzliche Übungsmöglichkeit die Benutzung des Tasters in das Arduino-Beispiel mit einzubringen.

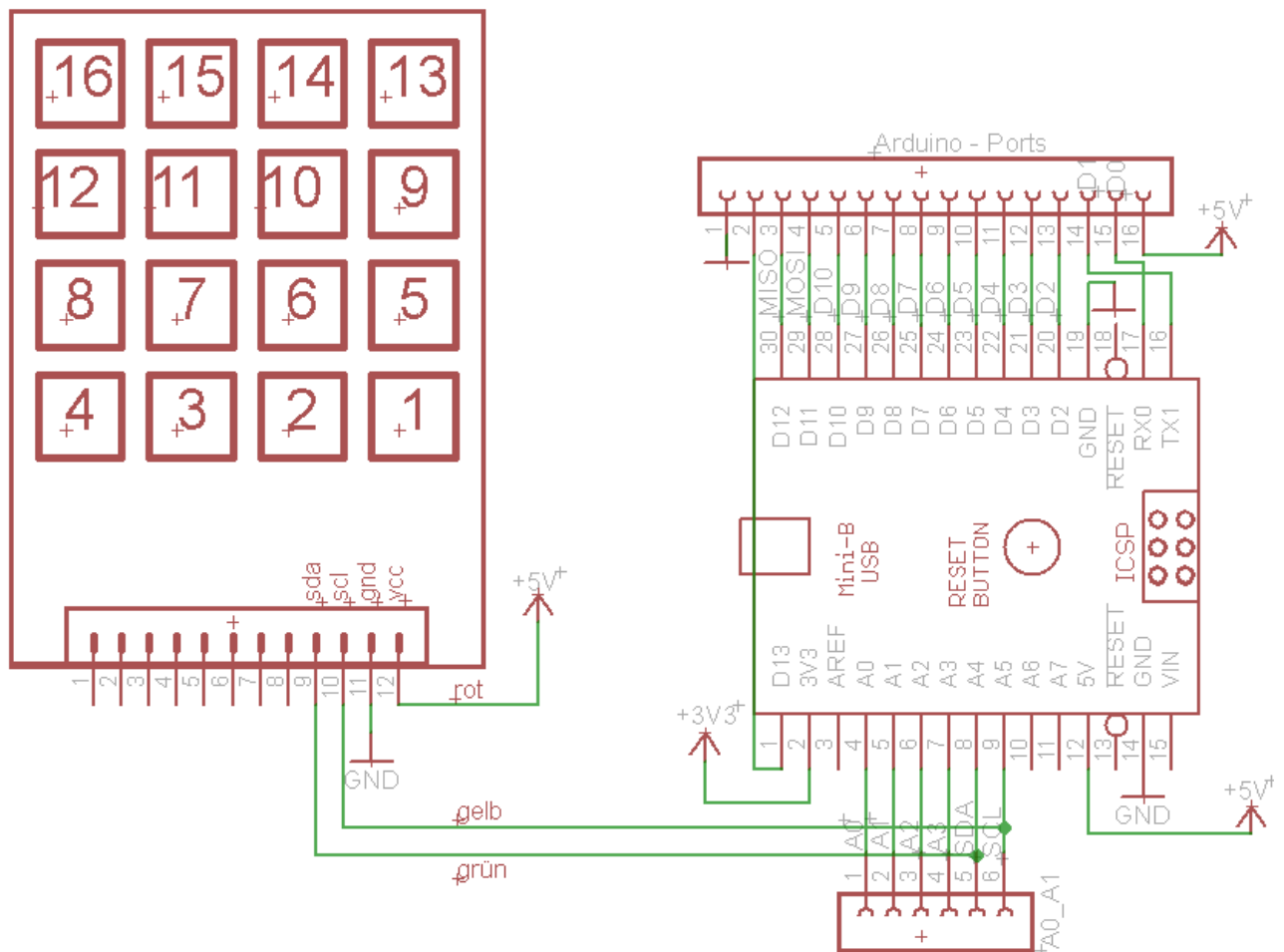
12.34. Ein touchmodul am Arduino

12.34.1 DAYCOM Tastaturmodul TTP229 mit 16 Tasten (Artikelnummer 810271)



Dieses funktioniert sowohl als I/O, als auch über Abfrage mit I2C-Bus. Dabei sind die Tasten 1..8 nur an die Lötungen von P1 und P2 geführt. Die Tastenfelder 9... 16 sind an die 12-polige Stiftleiste geführt. Aber in unserem Fall lesen wir die betätigten Tasten sowieso mit dem Arduino über die I2C-Schnittstelle aus. Als Bibliothek für den Arduino benötigt man eine für den TTP229: `capacitive touch keys`. Diese kann über den Bibliotheksverwalter ganz einfach installiert werden. Suche nach TTP229 und nach der

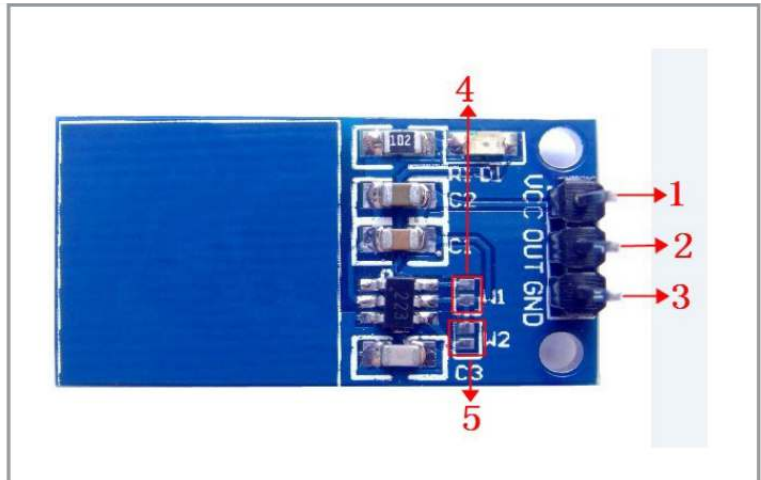
Installation einfach unter Datei → Beispiele → TTP229 das Beispiel [TTP229.ino](#) öffnen und ausführen. Aber nicht vergessen die Platine mit dem Arduino zu verdrahten, so wie im Bild unten gezeigt:



12.34.2. Kapazitives Berührungsschalter Modul DAYPOWER TTP-223 (Artikelnummer 810592)

Betrieb als statischer Schalter (Lötbrücke W1 offen)

Der kapazitive Berührungssensor arbeitet im Auslieferungszustand mit positiver Schaltlogik (W2 (5) ist offen) als statischer Schalter. Der Ausgang „OUT“ (2) schaltet nach Anlegen der Betriebsspannung nach GND (Masse) (3); die LED ist aus. Bei dauerhafter Betätigung mit einem Finger schaltet er zur positiven Betriebsspannung (VCC) (1) durch, solange der Finger die Sensorfläche berührt; LED ist ein. Nach Wegnahme des Fingers erfolgt ein Umschalten nach GND (3); LED erlischt wieder. Wenn die beiden Lötunkte von W2 (5) mit einem Zinnklecks überbrückt werden, kehrt sich der Wirkungssinn des Schalters um. Der Ausgang schaltet bei anlegen der Betriebsspannung zuerst nach VCC (1); die LED ist an. Bei Fingerberührung erfolgt ein Umschalten nach GND (3); die LED erlischt wieder.



Betrieb als Wechselschalter (Lötbrücke W1 geschlossen)

Durch Kurzschließen der beiden Lötunkte von W1 (4) ändert der Sensor seine Arbeitsweise von statisch nach dynamisch (Wechselschalter oder Toggleschalter). Der Schaltzustand ändert sich nach jedem Antippen mit dem Finger. Der Anfangszustand nach Anlegen der Betriebsspannung wird hierbei genau so wie im statischen Betrieb mit der Beschaltung von W2 (5) eingestellt.

Strombelastbarkeit des Ausganges

Der Ausgang kann beim Durchschalten nach GND (3) 8 mA Strom ziehen (Stromsenke), bzw. beim Durchschalten nach VCC (1) 4mA ausgeben (Stromquelle).

Programmtechnisch ist dieser Schalter genauso zu behandeln, wie ein mechanischer Taster. Er besitzt keine Schnittstelle wie der TTP229. Die Abfrage des Status erfolgt mit `digitalRead()`.

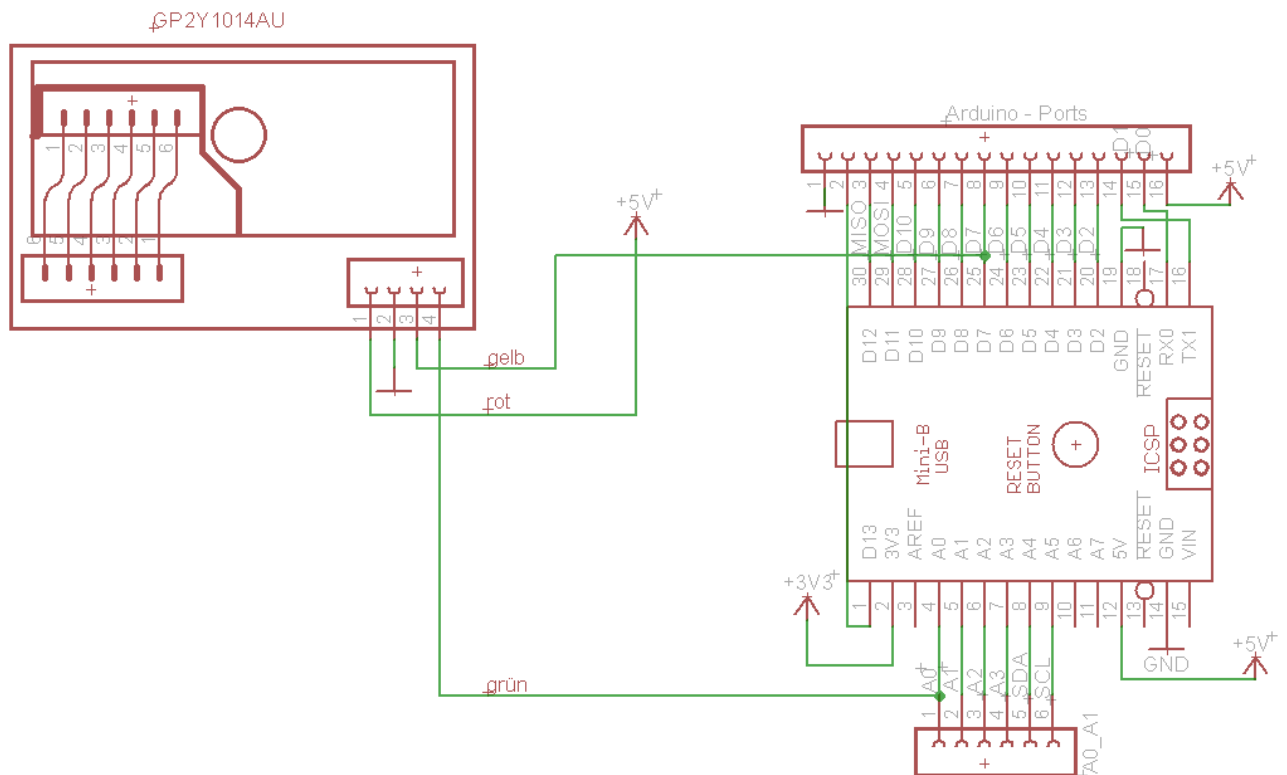
12.35. Staubsensor-Modul GP2Y1014AU (Artikel-Nr.: 811028)



Das Herzstück dieses Sensors ist eine Infrarotlichtschranke. Aus der Menge an Staubpartikeln, die zwischen Sendediode und Empfängertransistor sind, lässt sich ein Analogwert generieren. Mit zunehmender Anzahl der Staubpartikel wird immer mehr Licht reflektiert. Dies dient als Gradmesser für die Verschmutzung.

Im Beispiel Staubsensor01.ino ist angerissen, wie der Staubsensor ausgelesen und der Grad der Verschmutzung errechnet werden kann. Dazu wird zuerst die Spannung gemessen. Je 0.5V Ausgangsspannung nimmt die Verschmutzung der Luft um 0,1mg / m³ zu. Die Ausgangsspannung

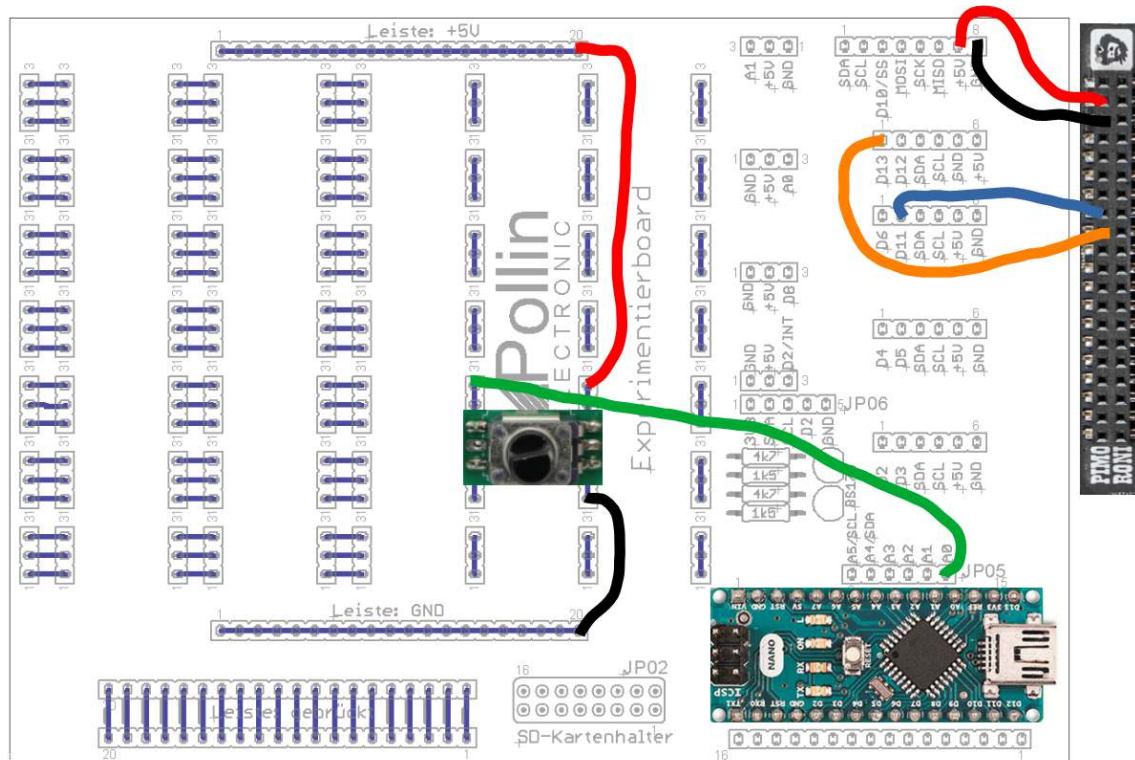
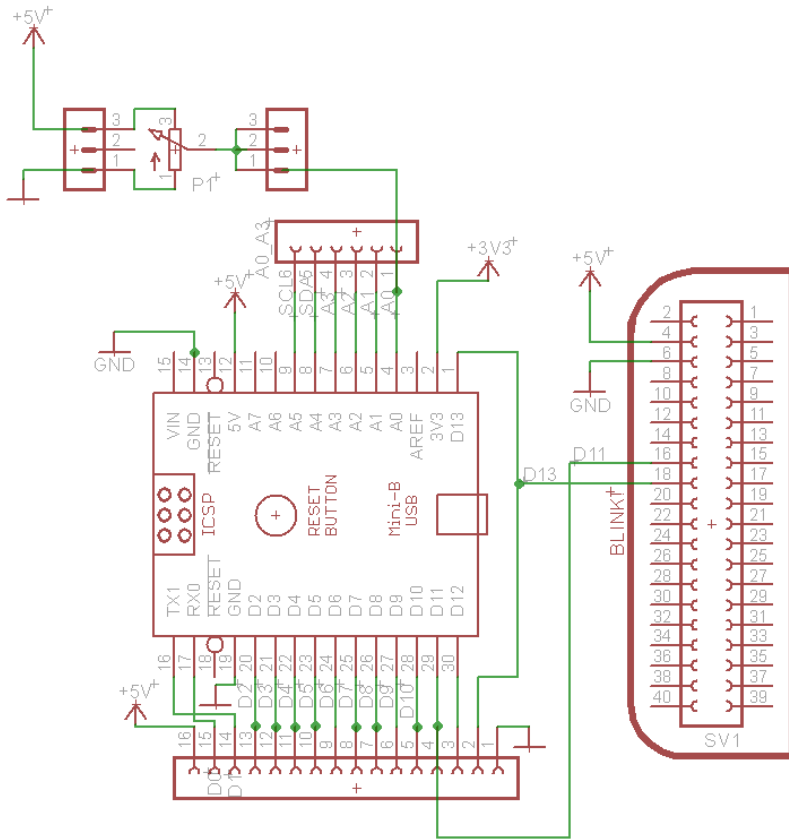
liegt im Bereich 0 ... 3,7V. Es soll nicht öfter als alle 10ms gemessen werden. Dabei soll die Sendediode nur für ca. 300us eingeschaltet sein. Dies alles wurde im Beispiel [Staubsensor01.ino](#) berücksichtigt. Die Verdrahtung des Sensors mit dem Arduino ist relativ einfach. Der Signalausgang des Sensors wird an A0 gelegt und die Infrarotdiode mit durch Pin D7 mit Spannung versorgt. Durch Schalten des Pins nach Masse (LOW), wird die Diode eingeschaltet.



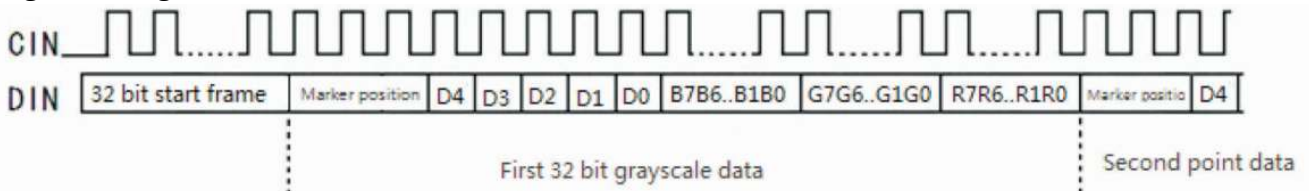
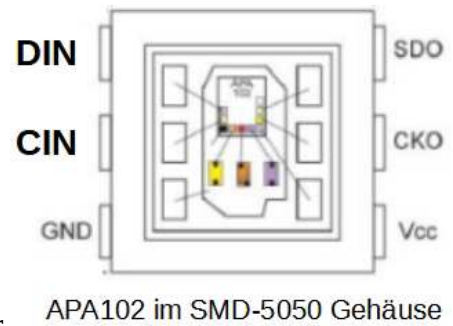
12.37. Ansteuerung des Raspberry Pi Blink! Moduls von pimoroni



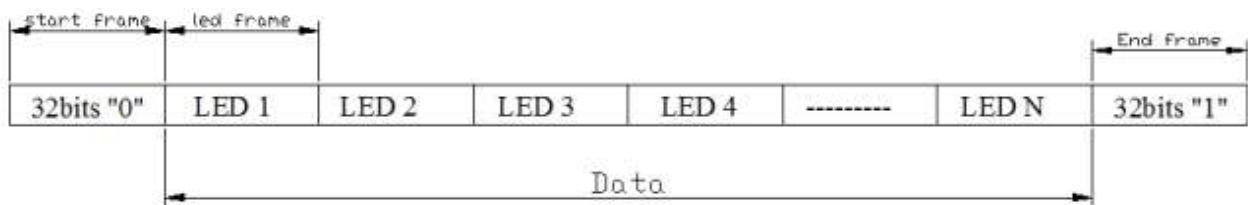
Artikelnummer: 810607



Ähnlich wie die LED im Kapitel 11.6.1. die LED WS2812 werden auch die LEDs APA102 auf dem Blink!-Modul seriell angesteuert. Der Unterschied besteht darin, dass zusätzlich zum Datensignal noch das Clock-Signal benötigt wird, so wie auch beim I2C- und SPI-Bus. Nur bei diesem und den folgenden Programmbeispielen benutzen wir weder eine vorgefertigte Bibliothek noch die Hardware-Schnittstelle des Arduino. Wir lernen hier, wie mittels Software die Schnittstelle nachgebildet werden kann. Deshalb sehen wir uns hier die Beschaffenheit der Signale mal genauer an:

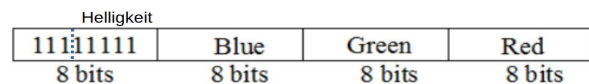


Das Bild oben ist dem Originaldatenblatt des Bausteins APA102 von der Firma ipixelleds entnommen. Dabei ist gut zu erkennen, wie der Takt (CIN) synchron mit den Daten (DIN) seine Flanken ändert. Bei jeder steigenden Flanke liest die LED den am Pin DIN anliegenden Pegel ein. Welche Daten erwartet nun die LED, um korrekt die gewünschten Farben wieder zu geben?



Die Datenübertragung beginnt mit einem Startframe, bestehend aus 32 low-Pegel (0); daran folgen die Frames für alle angeschlossenen LEDs:

Rechts im Bild ist ein einzelner Frame einer LED dargestellt. Das erste Byte ist aufgeteilt in drei High-Bits, darauf folgend sind 5 Bits, die ein Wert für die Helligkeit sind. Dies entspricht einem Zahlenwert von 0 (AUS) ... 32 (maximale Helligkeit). Abgeschlossen wird der End-Frame mit 32 High-Bits. Wie auch im Kapitel 11.6.1. beschrieben, kann mit den entsprechenden Werten für blau, grün und rot eine fast beliebige Farbe erzeugt werden (theoretisch: 2^{24} Farben also ca. 16 Millionen).



Doch nun zur Erzeugung der Signale mittels Software. Das Beispiel [LED_Blink_BSP1.ino](#): enthält die nachfolgend abgebildete Funktion:

```

159 void spiWrite(byte LEDs) // gib ein Datenbyte aus
160 {
161     uint8_t i;
162     for (i=0; i<8 ;i++)
163     {
164         if (LEDs<=0x80) // schaue, ob das Datenbit HIGH ist
165             { digitalWrite(LED_data, HIGH); }
166         else
167             { digitalWrite(LED_data, LOW); }
168         // erzeuge nun das CLK Signal, damit das Datenbit vom Empfänger eingelesen wird
169         digitalWrite(LED_clk, HIGH);
170         delayMicroseconds(1);
171         digitalWrite(LED_clk, LOW);
172         delayMicroseconds(1);
173         LEDs <<= 1; // einmal links schieben und nächstes MSB Bit prüfen
174     }
175     digitalWrite(LED_data, HIGH);
176 }

```

Die for-Schleife in Zeile 162 wird acht Mal durchlaufen. Damit werden acht Taktzyklen generiert und somit die 8 Datenbits des gewünschten Datenbytes übertragen.

In der Schleife wird mit der Abfrage `if(LEDs & 0x80)` geprüft, ob das Höchstwertigste Bit gesetzt (High) oder Low ist. Wenn es gesetzt ist, ergibt der Befehl `LEDs & 0x80` als Ergebnis 1. Wenn das MSB == 0 ist, dann ergibt die `&`-Verknüpfung als Ergebnis eine 0. Wenn das Bit 1 ist, wird der Ausgangszustand von `LED_data = HIGH`, wenn das Ergebnis Null ist, wird `LED_data = LOW`. Dann erfolgt der LOW -> HIGH Wechsel am `LED_clk`-Ausgang.

Mit dem Befehl `LEDs <<= 1;` wird der Wert vom Byte LED nach rechts geschiftet. So ist nun das Byte, das zuvor an Stelle 6 war, an Stelle 7 und somit nun das Höchstwertige Bit. So beginnt nun die Abfrage nach dem Zustand von neuem. Sobald alle 8 Bits geprüft wurden und die entsprechenden Zustände an den `LED_clk` und `LED_data` angelegen haben, wird die Unterfunktion beendet und zum aufrufenden Programm (in diesem Fall die `loop()`) zurückgekehrt. In der `loop` wird nacheinander jede im Modul enthaltene LED mit den entsprechenden Daten angesteuert. Dann wird nach einer Wartezeit `delay(warte_EIN)`, wieder jede LED abschaltet. Nach der Wartezeit `delay(warte_AUS)` beginnt der Blinkvorgang von Neuem.

Beim Beispiel [LED_Blink_BSP2.ino](#) wird ein Potiwert eingelesen und zur Steuerung der Helligkeit verwendet. Zur genaueren Erläuterung dient wieder ein Programmauszug:

```
38 | poti=analogRead(ADC); // dieser Wert ist 10 Bit lang
39 | poti >>= 5;           // jetzt wurden unteren 5 Bit abgeschnitten
40 | v_brightness= (byte) poti;// integer Variable in 5 Bit Variable umwandeln
41 | v_brightness = v_brightness | 0b11100000; // die ersten drei Bit auf HIGH setzen
42 | Serial.println(v_brightness,BIN);
```

In Zeile 38 wird der Wert des ADC der Variablen `poti` zugewiesen. Dieser Wert ist als Integer-Wert festgelegt, hat aber effektiv nur eine Länge von 10 Bit, so wie die Auflösung des ADC. Für `v_brightness` darf die Variable aber nicht länger als 8 Bit sein, weil diese als `byte` definiert wurde. Wie bereits oben im Abschnitt erwähnt, sind für den Wert für die Helligkeit nur ein kleiner Wertebereich von 5 Bit vorgesehen. Also schneiden wir in Zeile 39 durch den Schiebepfeil die niederwertigsten Bits ab. Nach dieser Schiebe-Operation bleiben die fünf höher wertigsten Bits übrig. In Zeile 40 wird der sogenannte CAST-Operator, oder auch Typecast genannt, verwendet. Das bedeutet, es wird mit `(byte)` vor der Variable `poti`, die Variable von integer (16 Bit) nach `byte` (8 Bit) gecastet. Auf Deutsch heißt `cast` so viel wie gießen und bedeutet nichts anderes als umwandeln.

In der Zeile 42 wird der Wert von `v_brightness` mit dem Wert `0b11100000` verodert. Dabei wird nichts anderes gemacht, als die drei höchstwertigen Bits auf 1 gesetzt. Dies ist notwendig, weil der Frameaufbau, wie er im Datenblatt beschrieben ist, dies fordert.

Im Beispiel [LED_Blink_BSP3.ino](#) wird der Poti-wert verwendet, um die Anzahl der angesteuerten LEDs zu verändern. Dabei sollen entweder (k)eine oder alle LEDs eingeschaltet werden können.

Bei diesem Beispiel ist lediglich der Befehl:

```
53 | anzahl=map(poti, 0,1023,1,8); // die Eingangsdaten des ADC
```

aus Zeile 53 zu erwähnen.

Der Befehl `map()` bewirkt, dass der Wert von `poti = 0 ... 1023` umgewandelt wird, in einen Wert von 1 bis 8, der dann der Variablen `anzahl` zugewiesen wird.

Im Beispiel [LED_Blink_BSP3a.ino](#) wurden daraufhin die Zeilen 126, 127, 128, 148, 149 und 150 auskommentiert. Dies ist laut Beschreibung aus dem Datenblatt falsch. Aber es bewirkte, dass nicht zusätzlich eine LED unerwünscht leuchtete.

Programmstruktur:

Syntaxbefehle:

vordefinierte Konstanten:

```
void setup() // Befehle werden einmalig abgearbeitet
void loop()  // Befehle werden in einer Endlosschleife abgearbeitet
```

```
// (Kommentar einer Zeile)
/* Kommentar für Block */
#define LED 13 // definiert LED an Pin D13
#include <avr/pgmspace.h> // zum Einbinden
```

```
HIGH / LOW      als Zustand der Ein- oder Ausgangspins
HIGH = 1 und LOW = 0
INPUT           Zustände der PORT-Pins, die mit pinMode() definiert werden können
INPUT_PULLUP   für Variablen die als Boolean definiert sind:true=1 und false =0
OUTPUT
true / false
LED_BUILTIN    damit wird die LED an Pin 13 Angesprochen
```

Schleifen und Kontrollstrukturen:

if-Bedingung

```
if (Bedingung)
{ Befehle; }
Else { Befehle }
```

Die Befehle werden so lange abgearbeitet, bis die Bedingung falsch ist; Sobald eine Bedingung nicht mehr erfüllt ist wird die Befehlsfolge nach else ausgeführt; Jedoch ist **else** nur eine Option.

```
if (x<5)
{ Befehle;}
```

die Befehle in den geschweiften Klammern werden nur ausgeführt, wenn zum Zeitpunkt der Abfrage x kleiner 5 ist.

switch case

```
switch (variable)
```

Die Abfrage mehrere Zustände einer Variablen, kann elegant mit switch case gelöst werden und nicht als Abfolge von if-Statements

```
{
case val1: { Befehle; }
break;
case val2: { Befehle; }
break;
default: { Befehle; }
}
```

Falls die Variable den Wert val einnimmt, dann werden die Befehle nach case val: abgearbeitet. Dabei ist der Befehl break jeweils ans Ende der Befehlskette zu stellen. Ohne break würden alle Befehle nach case val: ausgeführt werden bis zum Ende von switch. Default ist dabei auch nur optional, um dem Programm zu befehlen, was zu tun ist, wenn variable einen Wert annimmt, der nicht als case definiert ist.

for - Schleife

for (**Startwert** ; **Abbruchbedingung** ; **Änderung der Laufvariablen**)

Jede for-Schleife enthält eine Laufvariable; Diese benötigt einen Startwert; dazu eine Abbruchbedingung, also einen Endwert und eine Änderung des Wertes bei jedem Schleifendurchlauf: + - * oder /Dabei kann sich der Wert jedoch nicht nur um 1 verändern.

```
for (int i=0; i <= 255; i++)
{ Befehle; }
```

Bei jedem Durchlauf der Schleife werden die Befehle ausgeführt

while Schleifen

```
while (x<5)
{ }
```

die Schleife wird vorab geprüft und so lange durchlaufen bis x > oder gleich 5 ist

```
do { } while (x<5);
```

die Schleife wird mindestens einmal durchlaufen

Kontrollstruktur:

```
break;
```

// zum Verlassen einer Schleife

```
continue;
```

//springt zum Beginn der Anweisungen in einer Schleife; die Anweisungen dahinter werden dann nicht ausgeführt.

```
return x;
```

// beendet das Unterprogramm mit einem Rückgabe wert

```
return;
```

//springt zum aufrufenden Programm, ohne einen Rückgabewert

```
goto Name
```

// springt an eine mit Name bezeichnete Sprungadresse im Programm. Sollte allerdings vermieden werden.

```
sizeof(variable)
```

berechnet den Speicherbedarf einer Variablen (z.B. Array) in Bytes

```
PROGMEM
```

ermöglicht den Zugriff auf Daten im Programmflash

Variablen:

boolean: (0, 1, false, true)
char: -128 ... 127
unsigned char: 0 ... 255
byte: 0 ... 255
int: -32768 ... 32767

unsigned int: 0 ... 65535
Long: -2 147 483 648 ... 2 147 483 647
unsigned long 0 ... 4 294 967 295
Float (-3.4028235E+38 to 3.4028235E+38)

Strings:

char s2[8]={ 'a','r','d','u','i','n','o'};
char s3[8]={ 'a','r','d','u','i','n','o','\0'};
char s4[] = "arduino";
char s5[8] = "arduino";
char s6[15] = "arduino";

Arrays:

int i[8];
int arduinoPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2, 320};

Reserviert 16 Datenbytes für acht Zahlen
Array aus fünf arduino Pins
Array aus sechs Integer mit Wertzuweisung

reserviert acht Datenbytes für 7 Zeichen das achte Byte signalisiert Datenende: Nullcharacter = '\0'
hier wird Nullcharacter automatisch angefügt
hier wird Nullcharacter automatisch angefügt
hier wird mehr Speicher reserviert als Benötigt; Nullcharacter automatisch angefügt

Vergleichsoperatoren:

größer oder gleich: >=
kleiner oder gleich: <=
kleiner als: <
größer als: >
Gleichheit: ==
Ungleich !=
logisches UND: &&
logisches Oder: ||
logisches Nicht !

Zuweisungsoperatoren:

Wertzuweisung: =
Addition: +
Subtraktion: -
Multiplikation: *
Division: /
Modulo Division %

Bit- Operatoren:

Und: &
Oder: |
XOR: ^
Nicht: ~
Shift links: <<
shift rechts >>

Verbundoperatoren:

++ (Inkrement)
-- (dekrement)
-= (Subtraktion)
|= (bitweises oder)
&= (bitweises und)
/= (Division)

Ein-Ausgabe:

pinMode(Pin, Modus);
digitalWrite(Pin, Wert);
digitalRead(Pin);
analogRead(Pin);

analogWrite(Pin);

tone(Pin, Frequenz_Hz, Dauer_ms)
noTone();
shiftOut()
shiftIn()
pulseIn()
pulseInLong()

mathematische Funktionen:

min(x,y) welcher Zahl ist kleiner x oder y
max(x,y) welcher Zahl ist größer x oder y
abs() Absolutwert
constrain(x,u,o) beschränkt den Wertebereich von x auf einen u nteren und einen o beren Wert

map(value, fromLow, fromHigh, toLow, toHigh);

pow(x,y) Potenzfunktion x^y
sqrt() Wurzelfunktion
sin() Sinusfunktion
cos() Cosinusfunktion
tan() Tangensfunktion
sq() Quadratfunktion

random Zufallszahl
randomSeed initialisiert
Zufahlszahlengenerator

Bit-Operationen:

lowByte()
highByte()
bitRead()
bitSet()

bitClear()

bitWrite()
bit()

Zeigeroperatoren

& (Referenzoperator) // gibt die Adresse zurück
* (Dereferenzierungsoperator) // gibt den Wert zurück

Serielle Schnittstelle:

Serial.begin(Baud);
Serial.available()
Serial.read();
Serial.flush();
Serial.print();
Serial.println();
Serial.write();
SoftwareSerial mySerial (rxPin, txPin);
mySerial.write()
mySerial.read
mySerial.available()

Stringoperationen:

isAlpha
isAlphaNumeric
isAscii
isControl
isDigit
isGraph
isHexadecimalDigit()
isLowerCase()
isPrintable()
isPunct()
isSpace()
isUpperCase()
isWhitespace()

EEPROM:

EEPROM.Write(addr,val)
EEPROM.Read(addr)

Servomotor:

Servo.attach()
Servo.write()
Servo.read()
Servo.detach()

Wire:

begin()
end()
requestFrom()
beginTransmission()
write()
available()
read()
setClock()
onReceive()
onRequest()
setWireTimeout()
clearWireTimeoutFlag()
getWireTimeoutFlag()

SPI:

SPISettings
begin()
beginTransaction()
end()
setBitOrder()
setClockDivider()
setDataMode()
transfer()
usingInterrupt()

Zeitfunktionen:

millis();
micros();
delay(ms);
delayMicroseconds(us);

externer Interrupt:

interrupts()
noInterrupts()
attachInterrupt(digitalPinToInterrupt(pin), ISR_name, mode)
mode: LOW, HIGH, CHANGE, RISING, FALLING
detachInterrupt(digitalPinToInterrupt(pin));