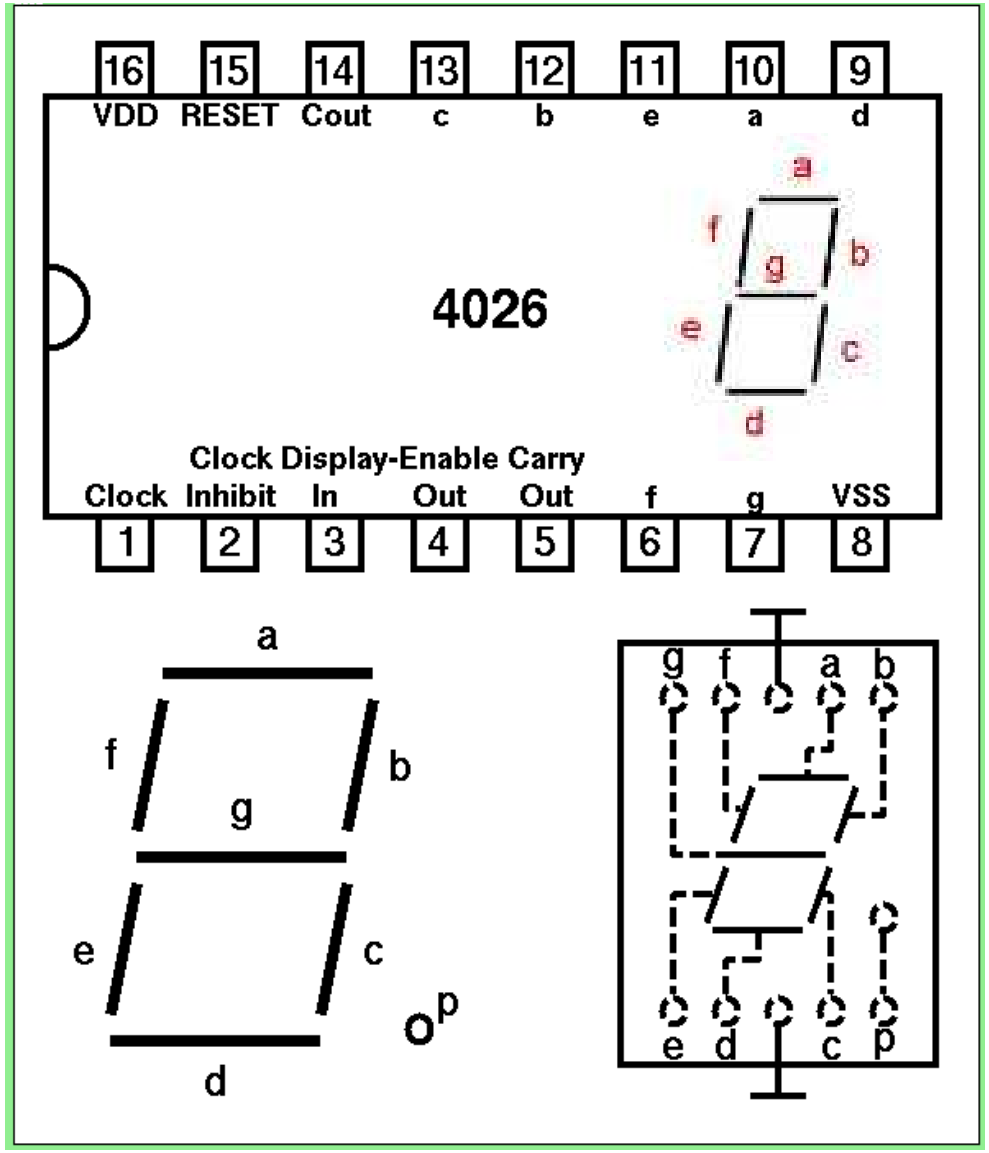
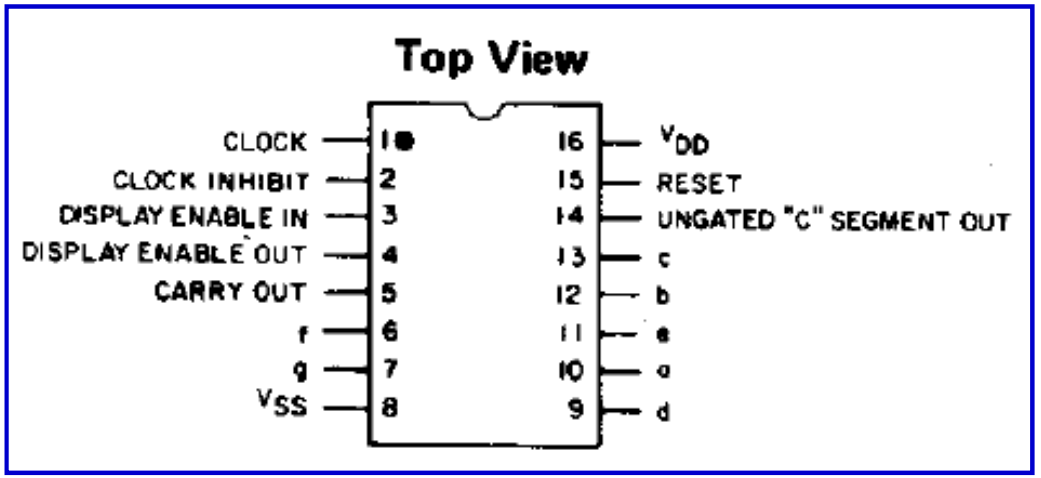
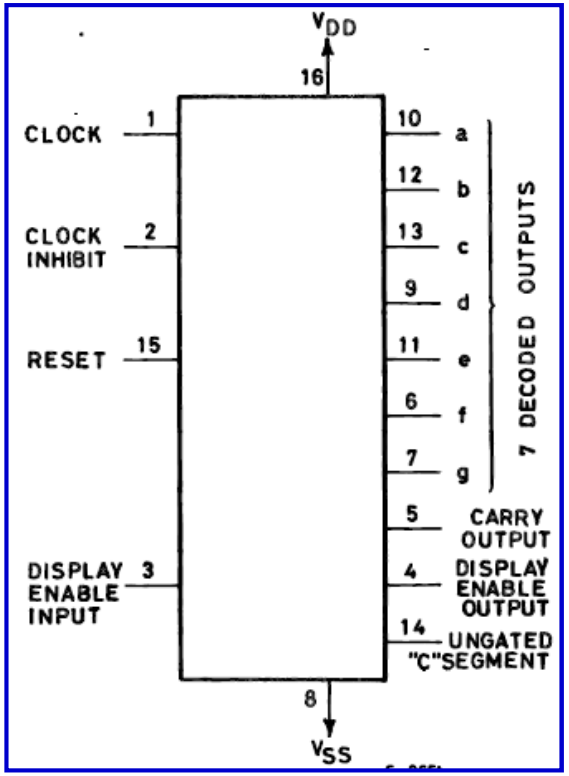
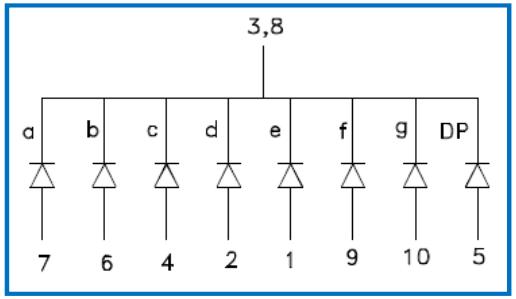
```
33 void loop()
34 {
35   tasterSumChange = LOW;                  // Initialwert keine Taste hat sich geändert
36   for(int i = 0; i < MAXLED; i++)         // Schleife über Anzahl MAXLED
37   {
38     if(analogRead(tasterPin[i]) < TST_SCHWELLE)   // Array tasterPin[i] lesen
39     {                                     // Wenn Pin == LOW, dann Taste gedrückt, da Taster auf pull up
40       tasterStatus[i] = HIGH;              // Tastenstatus in einem Array speichern
41       digitalWrite(ledPin[i], HIGH);       // Array ledPin[i] LOW, d.h. aus
42     }  // if
43     else                                  // else --> d.h. Pin == LOW, dann Taste gedrückt, da Taster auf pull down
44     {
45       tasterStatus[i] = LOW;             // Tastenstatus in einem Array speichern
46       digitalWrite(ledPin[i], LOW);      // Array ledPin[i] HIGH, d.h. ein
47     }  // else
48     if (tasterStatusOld[i] != tasterStatus[i])  // hat sich Tastenstatus geändert?
49     {
50       tasterStatusOld[i] = tasterStatus[i];     // dann neuen Status wieder als "Alt" merken
51       tasterSumChange = HIGH;                   // es hat sich an 1 - x Taster was geändert
52     }  //if
53   }  // for
54
55   if (tasterSumChange)                          // nur wenn sich eine der Tasten ändert, Status aller Tasten ausgeben
56   {
57     for(int i = 0; i < MAXLED; i++)             // Status aller MAXLED Tasten ausgeben
58     {
59       Serial.print("Taste ");                   // Einen Text in einer Zeile zusammenbauen
60       Serial.print(i);                          // Nummer der Taste [i]
61       Serial.print(" = Status: ");              // Text Status
62       Serial.print(tasterStatus[i]);            // Status der Taste [i]
63       Serial.print("\t");                       // Trennzeichen zur nächsten Taste
64     }  // for
65     Serial.println("");                        // new line für nächste Status Ausgabe
66     tasterSumChange = LOW;                     // Ausgabe erledigt, also Änderungsvariable wieder rücksetzen
67   } // if
68 }  // loop
```

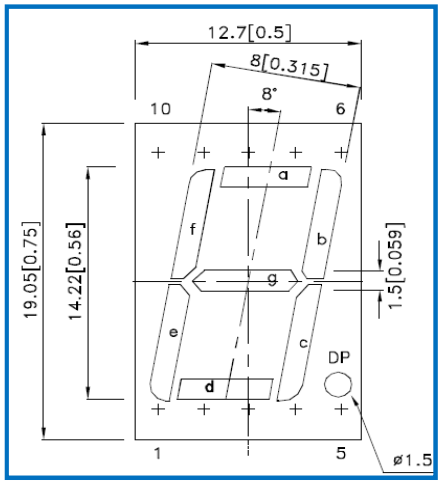
2. Teil: 7Segment (gemeinsame Kathode) mit einem externen BCD Zähler (CD4026) ansteuern.

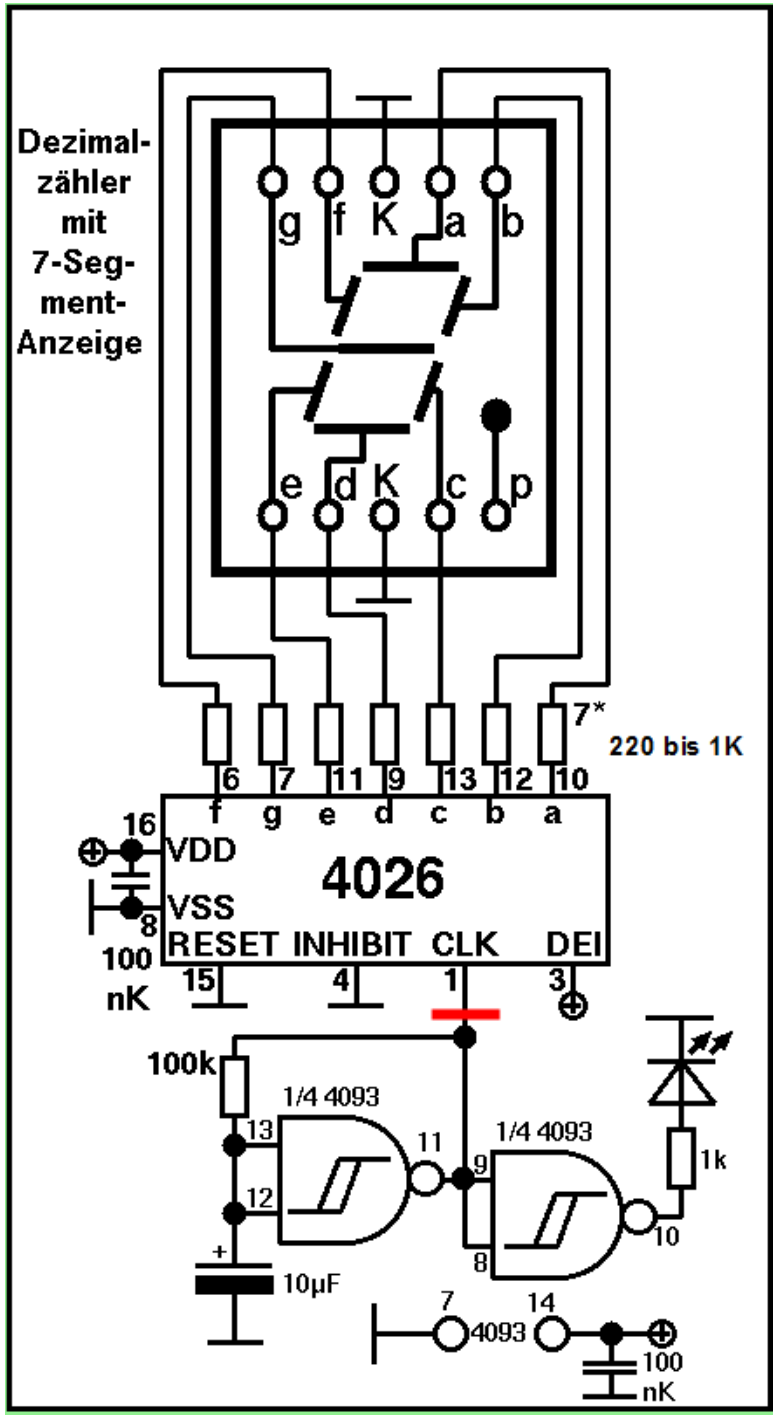












Neben den 7 Anzeige-Pins „a bis g“ sind noch vorhanden:

Clock, Pin 1: Takteingang des Zählers, zählt bei positiven Flanken.

Clock Inhibit, Pin 2: verhindert bei High, dass der Zähler weiterzählt, wenn Clock = High gehalten wird, kann dieser Eingang zum Zählen negativer Flanken benutzt werden.

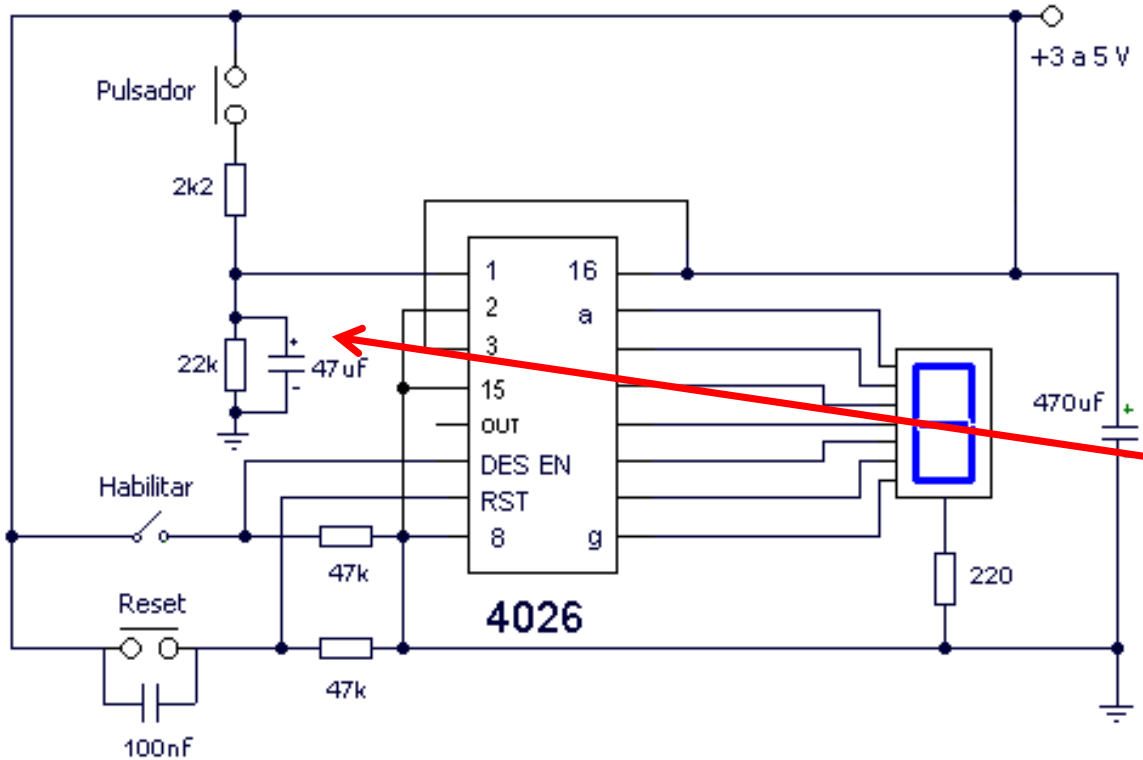
Display Enable In, Pin 3: schaltet die Anzeige aus, wenn Low und der Zählerstand Null ist,

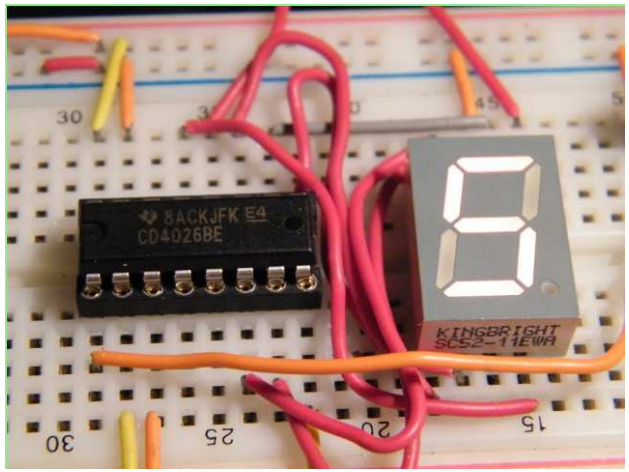
Display Enable Out, Pin 4: wird Low, wenn Display Enable = Low und der Zählerstand Null ist, gibt dies an das nächstniedrige 4026 weiter, wenn Zähler zu Kaskaden verschaltet werden (Unterdrückung führender Nullen).

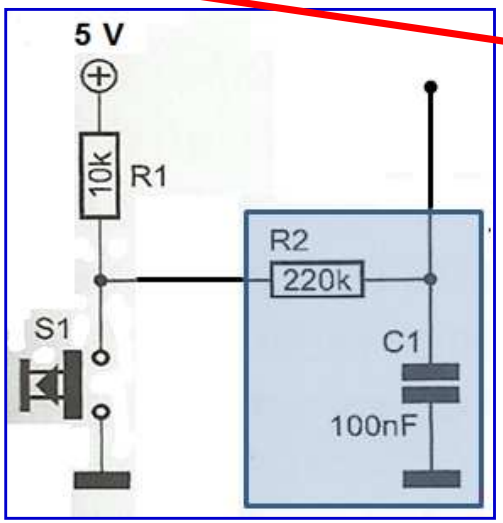
Carry Out, Pin 5: geht beim Zählerstand Fünf auf Low und bei Zählerstand Null auf High, kann an den Zählereingang des nächsthöheren Zählers bei Kaskaden geschaltet werden.

Cout, Pin 14: geht bei Zählerstand Zwei auf Low, kann bei Zählern bis 1,999 (beschränkte vorderste Anzeige) zur Erkennung von Überläufen benutzt werden,

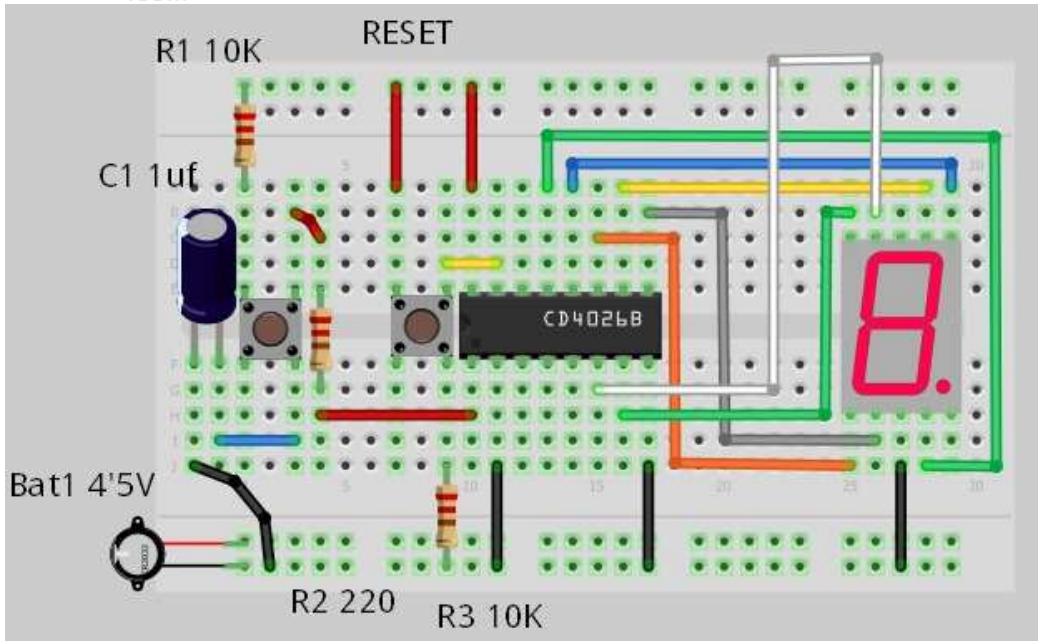
Reset, Pin 15: setzt bei High den Zähler auf Null.

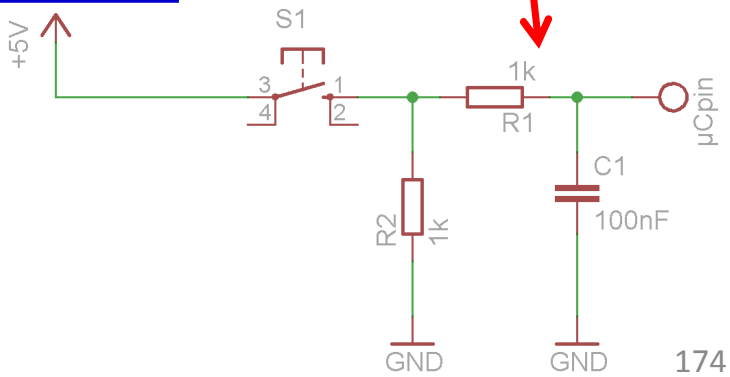






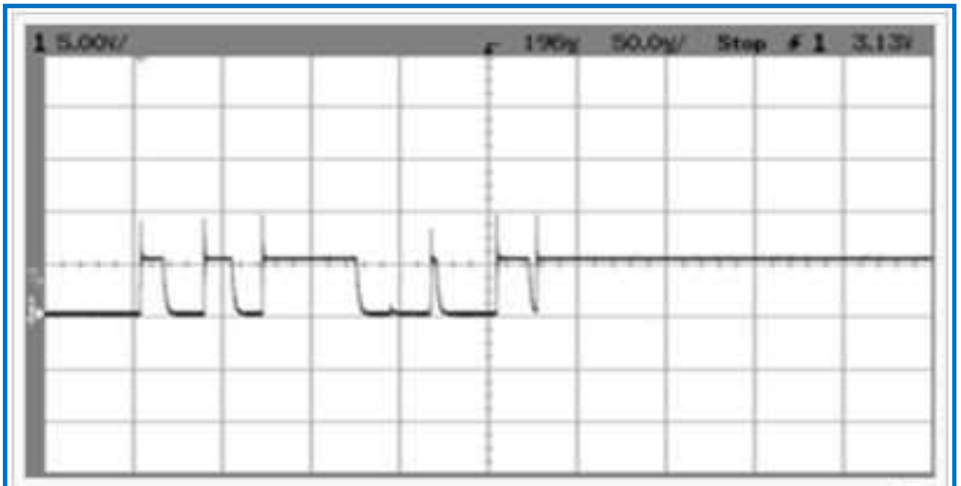
„R-C“ Glied (Tiefpaß) verhindert Schalterprellen





Was ist „Schalter-Prellen“

Als **Prellen** wird ein mechanischer Effekt bei Schaltern und Tastern bezeichnet: Statt des sofortigen elektrischen Kontaktes ruft die Betätigung des Schalters kurzzeitig ein mehrfaches Schließen und Öffnen des Kontakts hervor. Der Grund sind Federungen an Bauteilen der Schaltermechanik.



Zeitlicher Signalverlauf eines über etwa 250 µs prellenden Tasters bei einem Schließvorgang

Dieser Effekt des mehrfachen Schließen und Öffnen des Kontaktes führt bei schnellen elektronischen Schaltungen zu unerwünschten **Mehrfachereignissen**. Betroffen sind beispielsweise digitale Eingabegeräte wie eine Computertastatur, Eingabecontroller an Tastenfeldern oder elektronische Schaltungen, welche einen Relais- oder einen anderen mechanischen Kontakt erfassen. Durch die prellenden Vorgänge würde ohne **Entprellung** ein Tastenanschlag fehlerhafterweise als mehrfacher Anschlag registriert werden.

Massnahmen zum Entprellen erfolgen mittels eines Tiefpassfilters, einer **Verriegelungslogik** oder per **Software**.

BCD_4026_7SegZahlenTest

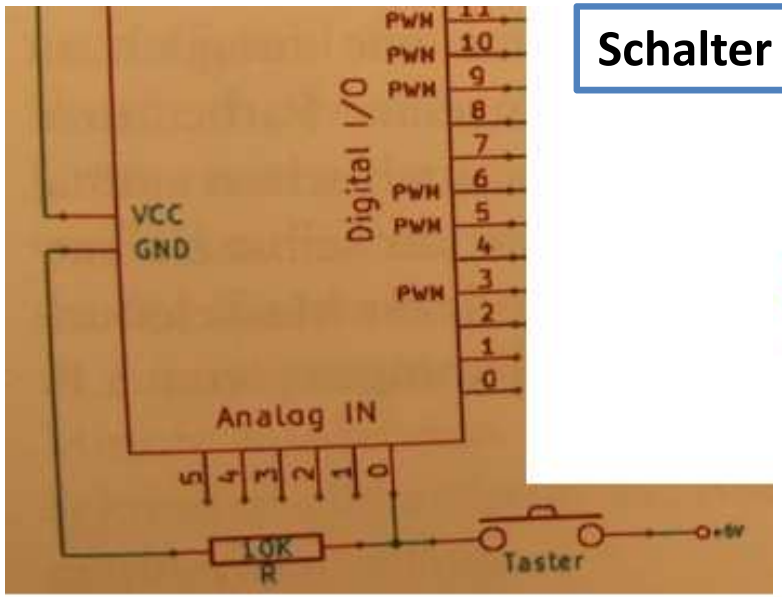
```
/* Counter Test:  Ausgabe einer Zahl durch "Reset" und "clocken" von 4026 Zaehler */

#define counterResetPin  7                // Counter 4026 Reset-Pin #15
#define counterClockPin  6                // Counter 4026 Clock-Pin #1
#define microSek 2                        // Pulsdauer für BCD 4026 Zaehler
#define anzDelay 2000                     // Anzeige Dauer in MilliSek

void setup()
{
  pinMode(counterResetPin, OUTPUT);       // Pin als Ausgang
  pinMode(counterClockPin, OUTPUT);       // Pin als Ausgang
} // fct setup

void showZahl(int value)                  //  zeigt Parameter "value" mit 7Seg-Ziffern an
{
  resetCounter();                         // Anzeige erst mal löschen
  for(int i = 0; i < value; i++)          // "value" mal einen Clock Puls ausgeben
  {
    digitalWrite(counterClockPin, HIGH);  // Clock-Pin auf HIGH setzen
    delayMicroseconds(microSek);          // warten bis HIGH-Pegel stabil ist
    digitalWrite(counterClockPin, LOW);   // Clock-Pin auf LOW setzen
    delayMicroseconds(microSek);          // warten bis LOW-Pegel stabil ist
  }  // for
}  // fct

void resetCounter()             // setzt Zaehlerbaustein auf "0" (Reset)
{
  digitalWrite(counterResetPin, HIGH);    // Reset-Pin auf HIGH setzen
  delayMicroseconds(microSek);            // warten bis HIGH-Pegel stabil ist
  digitalWrite(counterResetPin, LOW);     // Reset-Pin auf LOW setzen
  delayMicroseconds(microSek);            // warten bis LOW-Pegel stabil ist
}  // fct

void loop()
{
  showZahl(11);                           // mit Fkt "showZahl"  "11" anzeigen
  delay(anzDelay);                        // "x" milli-sec warten
  showZahl(88);                           // mit Fkt "showZahl"  "88" anzeigen
  delay(anzDelay);                        // "x" milli-sec warten
  showZahl(34);
  delay(anzDelay);
  resetCounter();                         // Zaehler auf "0"
  delay(anzDelay);                        // "x" milli-sec warten
}  // loop
```

BCD_4026_7SegZahlenTest_und_Counter

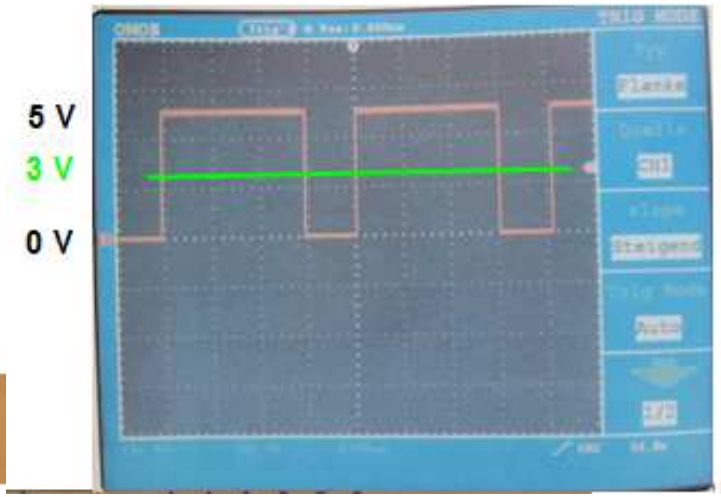
```
1  /* Counter Test:  Ausgabe von Zahlen durch "Reset" und "clocken" von 4026 Zaehler
2  und Hochzaehlern  von 0 bis 99 */
3
4  #define counterResetPin  7              // BCD_4026 Reset-Pin #15
5  #define counterClockPin  6              // BCD_4026 Clock-Pin #1
6  #define microSek 2                      // Pulsdauer für BCD 4026 Zaehler
7  #define ANZ_DELAY 2000                  // Anzeige Dauer in MilliSek
8  #define COUNTER_DELAY 100               // Anzeige Dauer in MilliSek
9
10
11 void setup()
12 {
13   pinMode(counterResetPin, OUTPUT);     // Pin als Ausgang
14   pinMode(counterClockPin, OUTPUT);     // Pin als Ausgang
15   Serial.begin(9600);
16 } // fct setup
17
18 void showZahl(int value)                // zeigt Parameter "value" mit 7Seg-Ziffern an
19 {
20   resetCounter();                       // Anzeige erst mal löschen
21   for(int i = 0; i < value; i++)        // "value" mal einen Clock Puls ausgeben
22   {
23     digitalWrite(counterClockPin, HIGH);  // Clock-Pin auf HIGH setzen
24     delayMicroseconds(microSek);          // warten bis HIGH-Pegel stabil ist
25     digitalWrite(counterClockPin, LOW);   // Clock-Pin auf LOW setzen
26     delayMicroseconds(microSek);          // warten bis LOW-Pegel stabil ist
27   }  // for
28 }  // fct
29
30 void resetCounter()            // setzt Zaehlerbaustein auf "0" (Reset)
31 {
32   digitalWrite(counterResetPin, HIGH);   // Reset-Pin auf HIGH setzen
33   delayMicroseconds(microSek);           // warten bis HIGH-Pegel stabil ist
34   digitalWrite(counterResetPin, LOW);    // Reset-Pin auf LOW setzen
35   delayMicroseconds(microSek);           // warten bis LOW-Pegel stabil ist
36 }  // fct
```

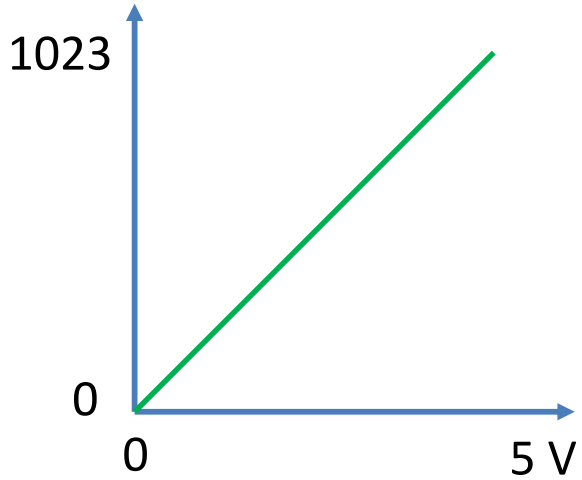
```
37
38 void loop()
39 {
40   Serial.print("Zahlen anzeigen:    ");
41   showZahl(11);                          // mit Fkt "showZahl"  "11" anzeigen
42   Serial.print("11;  ");
43   delay(ANZ_DELAY);                      // "x" milli-sec warten
44   showZahl(88);                          // mit Fkt "showZahl"  "88" anzeigen
45   Serial.print("88;  ");
46   delay(ANZ_DELAY);                      // "x" milli-sec warten
47   showZahl(34);
48   Serial.print("34;  ");
49   delay(ANZ_DELAY);
50   resetCounter();                        // Zaehler auf "0"
51   Serial.println("00");
52   delay(ANZ_DELAY);                      // "x" milli-sec warten
53   Serial.println("Von 0 nach 99 hochzaehlen:    ");
54   for (int i=0; i<100; i++)
55   {
56     showZahl(i);                          // mit Fkt "showZahl"  "11" anzeigen
57     Serial.print(i);
58     Serial.println("");
59     delay(COUNTER_DELAY);
60   }  // for
61   Serial.println("");
62 }  // loop
```

Zu wenige Digitale Ein- / Ausgänge ?



Schalter können auch mit einem „Analog-In“ Pin eingelesen werden:





int tasterPin = A1; // Analog Eingang 1

```
if (analogRead(tasterPin) > 900)   // Spannung > Schwelle
{                                  // dann ist Taste gedrückt
  if (!taster)                     // war taste noch nicht gedrückt
  {
    tasterNeu = true;              // dann ist taste neu gedrückt (Flanke)
    taster = true;                 // Status Taste setzen
  }
}
else                               // Spannung < Schwelle
{                                  // Taste nicht gedrückt
  taster = false;                  // Status Taste setzen
}
```

Im Beispiel mit «Pull-Down»

Geht auch mit «Pull-UP, dann:

analogRead(tasterPin < 900)

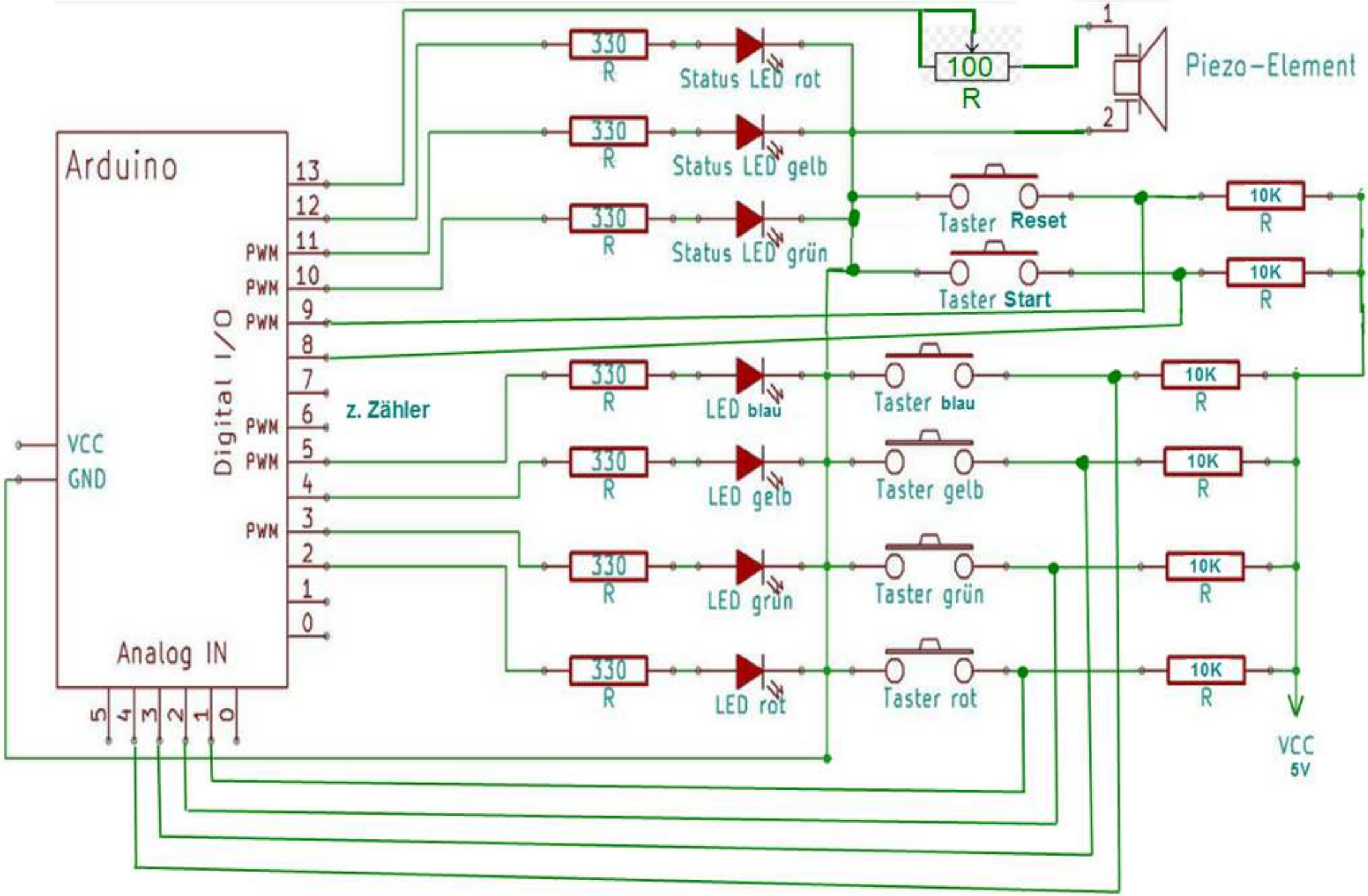
```
/* Farbensequenzspiel */

#define MAXARRAY 6                  // Vorgabe der Sequenzlänge
# define MAXLED 4                   // Anzahl der LEDs
int ledPin[] = {2, 3, 4, 5};        // LED-Array mit Pin-Werten
#define piezoPin 13                 // Piezo-Pin
#define tasterPinRot 6              // Taster Pin an roter LED
#define tasterPinGruen 7            // Taster-Pin an grüner LED
#define tasterPinGelb 8             // Taster-Pin an gelber LED
#define tasterPinWeiss 9            // Taster-Pin an weißer LED
#define ledStatusPinGruen 10        // Status LED grün
#define ledStatusPinGelb 11         // Status LED gelb
#define ledStatusPinRot 12          // Status LED rot

int colorArray[MAXARRAY];           // Enthält die Zahlenfolge für anzuzeigende Farben
int tones[] = {1047, 1175, 1319, 1397}; // Tonfrequenzen für die 4 Farben
int counter = 0;                    // Anzahl der gerade aufleuchtenden LEDs
boolean fail = false;
```

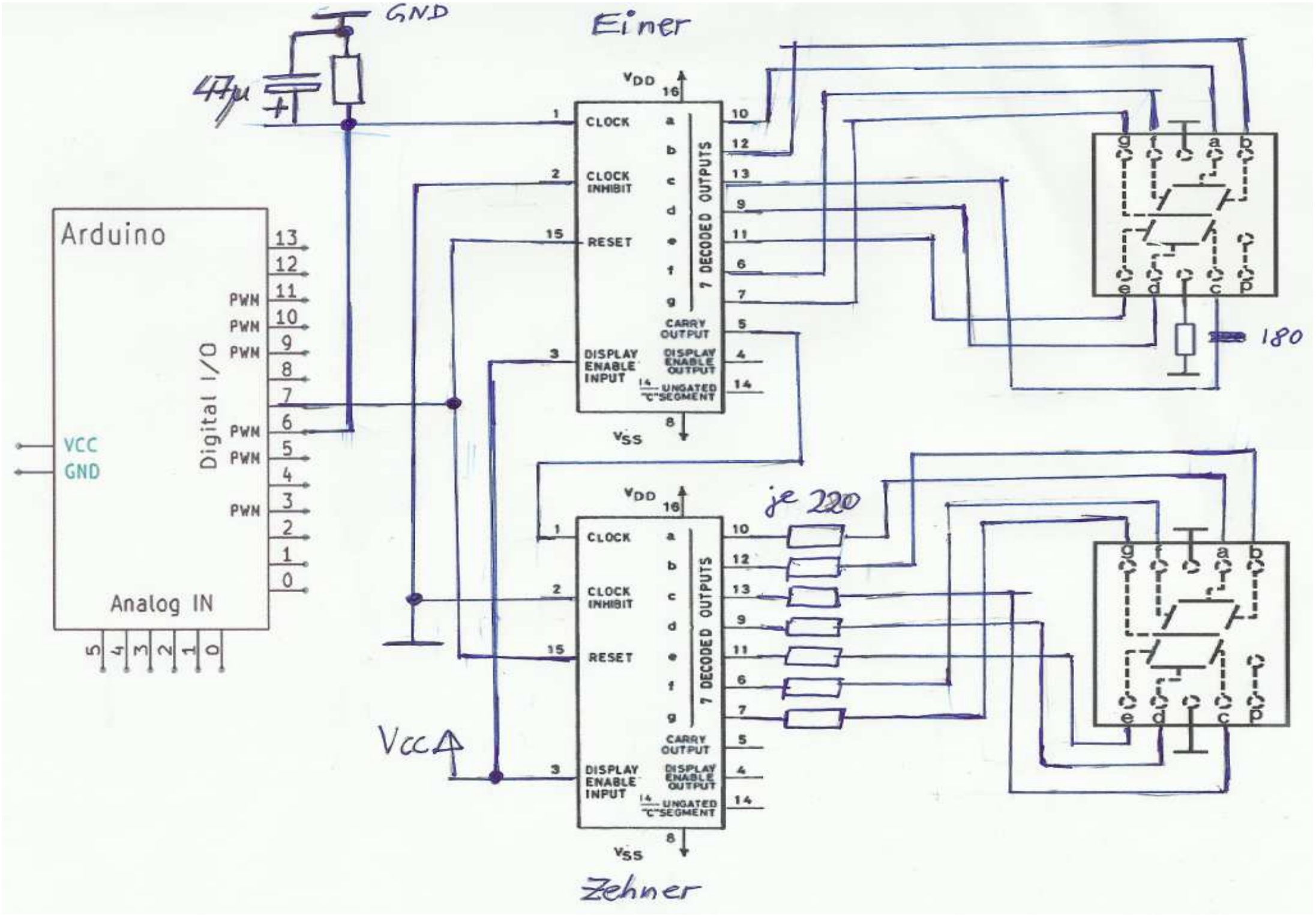
```
void setup(){
  Serial.begin(9600);
  for(int i = 0; i < MAXLED; i++)
    pinMode(ledPin[i], OUTPUT);        // LED-Pins als OUTPUT programmieren
  pinMode(tasterPinRot, INPUT);
  digitalWrite(tasterPinRot, HIGH);
  pinMode(tasterPinGruen, INPUT);
  digitalWrite(tasterPinGruen, HIGH);
  pinMode(tasterPinGelb, INPUT);
  digitalWrite(tasterPinGelb, HIGH);
  pinMode(tasterPinWeiss, INPUT);
  digitalWrite(tasterPinWeiss, HIGH);
  pinMode(ledStatusPinGruen, OUTPUT);
  pinMode(ledStatusPinGelb, OUTPUT);
  pinMode(ledStatusPinRot, OUTPUT);
}
```

```
void loop()
{
  digitalWrite(ledStatusPinGelb, HIGH);
  Serial.println("Spielstart");
  generateColors();
  int tasterCode;
  for(int i = 0; i <= counter; i++)    // Äußere Schleife
  {
    digitalWrite(ledStatusPinGelb, HIGH);
    giveSignalSequence(i);
    for(int k = 0; k <= i; k++)        // Innere Schleife
    {
      digitalWrite(ledStatusPinGelb, LOW);
      while(digitalRead(tasterPinRot) && digitalRead(tasterPinGruen)
            digitalRead(tasterPinGelb) && digitalRead(tasterPinWeiss
      Serial.println("Taste gedrückt!"); // Zur Kontrolle im Serial
      // Anzeigen der gerückten Farbe
      if(!digitalRead(tasterPinRot))
        tasterCode = 0;
      if(!digitalRead(tasterPinGruen))
        tasterCode = 1;
      if(!digitalRead(tasterPinGelb))
        tasterCode = 2;
      if(!digitalRead(tasterPinWeiss))
        tasterCode = 3;
      giveSignal(tasterCode);
      // Überprüfung ob richtige Farbe gedrückt wurde
      if(colorArray[k] != tasterCode)
      {
        fail = true;
        break; // Innere for-Schleife verlassen
      }  // if
    }    // innere for
    if(!fail)
      Serial.println("richtig"); // Zur Kontrolle im Serial Monitor
    else
    {
      digitalWrite(ledStatusPinRot, HIGH);
      for(int i = 3000; i > 500; i-=150)
      {
        tone(piezoPin, i, 10); delay(20);
      }  // for
      Serial.println("falsch"); // Zur Kontrolle im Serial Monitor
      delay(2000);
      digitalWrite(ledStatusPinRot, LOW);
      counter = 0; fail = false;
      break;      // for-Schleife verlassen
    }  // else
    delay(2000);
    if(counter + 1 == MAXARRAY)
    {
      digitalWrite(ledStatusPinGruen, HIGH);
      for(int i = 500; i < 3000; i+=150)
      {
        tone(piezoPin, i, 10); delay(20);
      }  // for
      Serial.println("Ende!"); // Zur Kontrolle im Serial Monitor
      delay(2000);
      digitalWrite(ledStatusPinGruen, LOW);
      counter = 0; fail = false;
      break;                    // Äußere for-Schleife verlassen
    }  // if
    counter++; // Zähler inkrementieren
  }  // for
}    // loop
```

```
void giveSignalSequence(int value)       // Anzeige LEDs
{
  for(int i = 0; i <= value; i++)
  {
    digitalWrite(2 + colorArray[i], HIGH);
    generateTone(colorArray[i]);
    delay(1000);
    digitalWrite(2 + colorArray[i], LOW);
    if (i < value)
      delay(1000);
  }
}

void generateTone(int value)
{
  tone(piezoPin, tones[value], 1000);
}

void giveSignal(int value)     // Anzeige LED + Tonsignal
{
  digitalWrite(2 + value, HIGH);
  generateTone(value);
  delay(200);
  digitalWrite(2 + value, LOW);
  delay(200);
}
```

```
void generateColors()
{
  randomSeed(analogRead(0));
  for(int i = 0; i < MAXARRAY; i++)
    colorArray[i] = random(4); // Zufallszahlen von 0 bis 3 generieren
  // 0 = Rot, 1 = Grün, 2 = Gelb, 3 = Weiss
  for(int i = 0; i < MAXARRAY; i++)
    Serial.println(colorArray[i]); // Zur Kontrolle im Serial Monitor
}
```


Arduino
Digital I/O
Analog IN
VCC
GND
13
12
11
10
9
8
7
6
5
4
3
2
1
0
PWM
z. Zähler
330 R
Status LED rot
Status LED gelb
Status LED grün
100 R
Piezo-Element
Taster Reset
Taster Start
10K R
LED blau
LED gelb
LED grün
LED rot
Taster blau
Taster gelb
Taster grün
Taster rot
VCC 5V


GND
Einer
47µ
Arduino
VCC
GND
Digital I/O
Analog IN
PWM
VDD
VSS
CLOCK
CLOCK INHIBIT
RESET
DISPLAY ENABLE INPUT
7 DECODED OUTPUTS
CARRY OUTPUT
DISPLAY ENABLE OUTPUT
UNGATED "C" SEGMENT
180
je 220
Vcc
Zehner

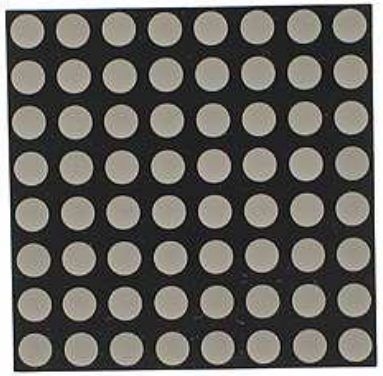
```
/* Farbensequenzspiel PLUS mit Anzeige und Reset*/

#define MAXARRAY 15                // Vorgabe der Sequenzlänge
#define MAXLED 4                   // Anzahl der LEDs
#define TST_SCHWELLE 800           // Analoge Taster Schwelle
// 0 = Rot, 1 = Grün, 2 = Gelb, 3 = Blau
#define LED_PIN_ROT 2
#define LED_PIN_GRUEN 3
#define LED_PIN_GELB 4
#define LED_PIN_BLAU 5
int ledPin[] = {LED_PIN_ROT, LED_PIN_GRUEN, LED_PIN_GELB, LED_PIN_BLAU}; // LED-Array mit Pin-Werten
#define tasterPinRot A1            // Taster Pin an roter LED
#define tasterPinGruen A2          // Taster-Pin an grüner LED
#define tasterPinGelb A3           // Taster-Pin an gelber LED
#define tasterPinBlau A4           // Taster-Pin an blau LED
#define tasterStartStopPin 8       // Start Stop Taster-Pin
#define tasterResetPin 9           // Reset Taster-Pin

#define ledStatusPinGruen 10       // Status LED grün
#define ledStatusPinGelb 11        // Status LED gelb
#define ledStatusPinRot 12         // Status LED rot
#define piezoPin 13                // Piezo-Pin

#define counterResetPin 7          // Counter 4026 Reset-Pin #15
#define counterClockPin 6          // Counter 4026 Clock-Pin #1

int colorArray[MAXARRAY];          // Enthält die Zahlenfolge für anzuzeigende Farben
int tones[] = {1047, 1175, 1319, 1397}; // Tonfrequenzen für die 4 Farben
int counter = 0;                   // Anzahl der gerade aufleuchtenden LEDs
int counterMax = 0;                // HighScore
int counterAktuell = 0;            // Aktuelle Score
boolean fail = false;
boolean spielStart = false;
```

```
void setup()
{
  Serial.begin(9600);

  for(int i = 0; i < MAXLED; i++)
    pinMode(ledPin[i], OUTPUT);           // LED-Pins als OUTPUT programmieren

  pinMode(ledStatusPinGruen, OUTPUT);     // Pin als Ausgang
  pinMode(ledStatusPinGelb, OUTPUT);
  pinMode(ledStatusPinRot, OUTPUT);
  pinMode(counterResetPin, OUTPUT);       // Pin als Ausgang
  pinMode(counterClockPin, OUTPUT);       // Pin als Ausgang

  pinMode(tasterResetPin, INPUT);         // Pin als Eingang
  digitalWrite(tasterResetPin, HIGH);     // Pull Up einschalten

  pinMode(tasterStartStopPin, INPUT);       // Pin als Eingang
  digitalWrite(tasterStartStopPin, HIGH);  // Pull Up einschalten
}
```

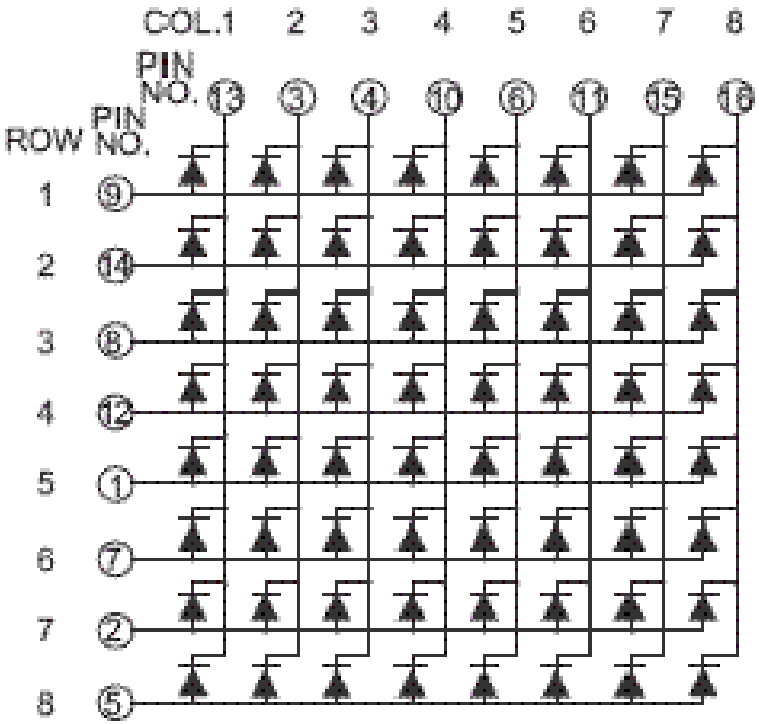
```
void loop()
{
  if (!spielStart)       // Spielbeginn auf Tastendruck
  {                      // Spiel noch nicht begonnen
    digitalWrite(ledStatusPinRot, LOW);
    digitalWrite(ledStatusPinGelb, LOW);
    digitalWrite(ledStatusPinGruen, HIGH);
    while(!spielStart)  // Tasten abfragen, bis Spiel Start
    {
      if (!digitalRead(tasterStartStopPin))     // Start-Spiel Taste gedrückt
        spielStart = true;                      // Spiel starten
      if (!digitalRead(tasterResetPin))         // Reset Taste gedrückt
      {
        showZahl(counterMax);                   // Zeige High Score
        delay(1500);
        if (!digitalRead(tasterResetPin))       // Reset nach 1,5sec immer noch gedrückt dann Reset
        {
          counterMax = 0;
          counterAktuell = 0;
          showZahl(counterMax);
          delay(1000);
        }
        else
        {
          showZahl(counterAktuell);             // Aktuelle Score wieder anzeigen
        }
      }  // if
    }  //while
  } // if
  else
  {
    Serial.println("Spielstart");
    generateColors();
    int tasterCode;
    for(int i = 0; i <= counter; i++)     // Äußere Schleife
    {
      digitalWrite(ledStatusPinRot, LOW);
      digitalWrite(ledStatusPinGelb, HIGH);
      digitalWrite(ledStatusPinGruen, LOW);
      giveSignalSequence(i);
```

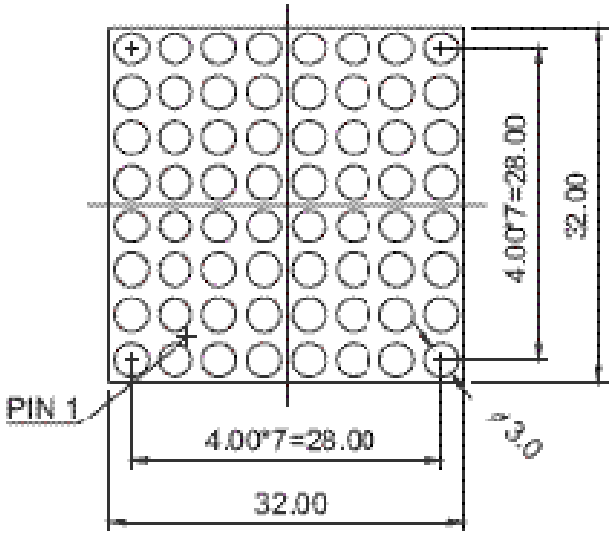
```
for(int k = 0; k <= i; k++)        // Innere Schleife
{
  digitalWrite(ledStatusPinGelb, LOW);
  // Taster mit PULL-UP
  while((analogRead(tasterPinRot) > TST_SCHWELLE) && (analogRead(tasterPinGruen) > TST_SCHWELLE) &&
        (analogRead(tasterPinGelb) > TST_SCHWELLE) && (analogRead(tasterPinBlau)) > TST_SCHWELLE);
  Serial.println("Taste gedrückt!"); // Zur Kontrolle im Serial Monitor
  // Anzeigen der gerückten Farbe
  if(analogRead(tasterPinRot) < TST_SCHWELLE)
    tasterCode = 0;
  if(analogRead(tasterPinGruen) < TST_SCHWELLE)
    tasterCode = 1;
  if(analogRead(tasterPinGelb) < TST_SCHWELLE)
    tasterCode = 2;
  if(analogRead(tasterPinBlau) < TST_SCHWELLE)
    tasterCode = 3;
  giveSignal(tasterCode);
  // Überprüfung ob richtige Farbe gedrückt wurde
  if(colorArray[k] != tasterCode)
  {
    fail = true;
    break; // Innere for-Schleife verlassen
  }  // if
}    // innere for
if(!fail)
{
  digitalWrite(ledStatusPinRot, LOW);
  digitalWrite(ledStatusPinGelb, LOW);
  digitalWrite(ledStatusPinGruen, HIGH);
  Serial.println("richtig");                          // Zur Kontrolle im Serial Monitor
  delay(500);
}
```

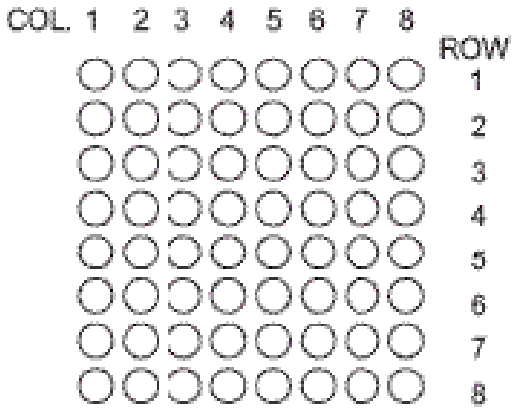
```
      else
      {
        digitalWrite(ledStatusPinRot, HIGH);
        digitalWrite(ledStatusPinGelb, LOW);
        digitalWrite(ledStatusPinGruen, LOW);
        for(int i = 3000; i > 500; i-=150)
        {
          tone(piezoPin, i, 10); delay(20);
        }  // for
        Serial.println("falsch"); // Zur Kontrolle im Serial Monitor
        showZahl(counter);
        counterAktuell = counter;
        if (counter > counterMax)
          counterMax = counter;
        counter = 0; fail = false;
        spielStart = false;
        delay(2000);
        break;        // for-Schleife verlassen
      }  // else
      delay(2000);
      if(counter + 1 == MAXARRAY)
      {
        showZahl(counter + 1);
        digitalWrite(ledStatusPinRot, LOW);
        digitalWrite(ledStatusPinGelb, LOW);
        digitalWrite(ledStatusPinGruen, HIGH);
        for(int i = 500; i < 3000; i+=150)
        {
          tone(piezoPin, i, 10); delay(20);
        }  // for
        spielStart = false;
        Serial.println("Ende!"); // Zur Kontrolle im Serial Monitor
        delay(2000);
        counter = 0; fail = false;
        break;                      // Äußere for-Schleife verlassen
      }  // if
      counter++; // Zähler inkrementieren
    }  // for
  }  // else
}    // loop
```

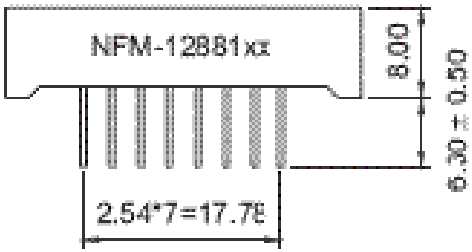


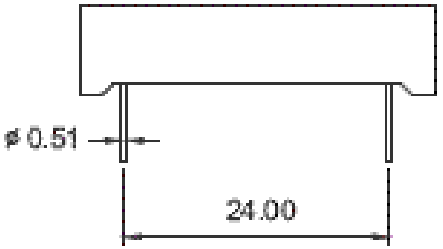
Ansteuerung eines 8x8 LED Displays
es lassen sich beliebige Muster erzeugen. Das Prinzip entspricht der Ansteuerung eines TV- / PC Monitors











_8x8LED_Seriell_Control §

```
/* Matrix Animation sketch
   per PC Monitor kann ein Herzschlag-Bild, ein Demo Lauflicht oder jede einzelne LED von 1-64 angesteuert werden
   Eingabe von:
   H     -->  Herzschlag
   D     -->  Demolauflicht
   1-64  -->  die entsprechende LED einschalten
Alle Zeichen werden empfangen & zur Kontrolle wieder an PC zurück geschickt. Buchstaben & Zahlen werden untersucht */

byte bigHeart[] = {            // Symbol für großes Herz. "1" bedeutet LED an
  B01100110,
  B11111111,
  B11111111,
  B11111111,
  B01111110,
  B00111100,
  B00011000,
  B00000000};

byte smallHeart[] = {
  B00000000,
  B00000000,
  B00010100,
  B00111110,
  B00111110,
  B00011100,
  B00001000,
  B00000000};

const int columnPins[] = {2, 3, 4, 5, 6, 7, 8, 9};
const int rowPins[]    = {10,11,12,15,16,17,18,19};
const int onTime = 250;      // Mikro Sekunden pro LED. Mal 64 ergibt Frame Zeit

char chSerial = 0;           // speichert die empfangenen Daten
char chLetter = 0;           // speichert den empfangenen Buchstaben
int empfZahl = 0;            // 0 - 64, als0 max. 2 Ziffern, default == 0,
int pulseDelay = 800 ;       // milliseconds to wait between beats
int zahl = -1;               // default == -1 ==> keime Ziffer
```

```
void setup()
{
  for (int i = 0; i < 8; i++)
  {
    pinMode(rowPins[i], OUTPUT);                    // make all the LED pins outputs
    pinMode(columnPins[i], OUTPUT);
  }  // for
  Serial.begin(9600);                               // öffnet die serielle Schnittstelle
}  // setup
```

```
void loop()
{
  if (Serial.available() > 0)                          // gibt es ein Zeichen
  {                                                    // read the incoming byte:
    chSerial = Serial.read();                          // Datenbyte lesen
    Serial.print("Arduino hat empfangen (ASCII: ");   // Text ausgeben
    Serial.print(chSerial);                            // empfangenes Datenbyte wieder ausgeben ASCII
    Serial.print(" = (DEC): ");
    Serial.println((int)chSerial);                     // nach interger wandeln und ausgeben

    if (isDigit(chSerial))                             // prüfen ob Zeichen eine Zahl ist
    {
      empfZahl = (empfZahl*10) + (chSerial - '0');     // Zahl ist "ASCII-Wert" - "ASCII-0"
      zahl = empfZahl;                                 // für weitere Anwendung speichern
      chLetter = 0;                                    // Da "zahl" empfangen wurde "chLetter" löschen
    }
    else
    {
      if ((chSerial < 91) && (chSerial > 64))          // prüfen ob Zeichen eine Buchstabe ist
      {
        chLetter = chSerial;                           // für weitere Anwendung speichern
        empfZahl = 0;                                  // Zifferneingabe zu Ende,  bereit für neue Zahlenumwandlung
        zahl = -1;                                     // Da "chLetter" empfangen wurde "zahl" löschen
      }
      else
      {
        if ((chSerial == 10) || (chSerial == 13))     // NL newline oder carriage return CR
        {
          empfZahl = 0;                               // Zifferneingabe zu Ende,  bereit für neue Zahlenumwandlung
        }
      }
    }  // else
  }        // if serial available
```

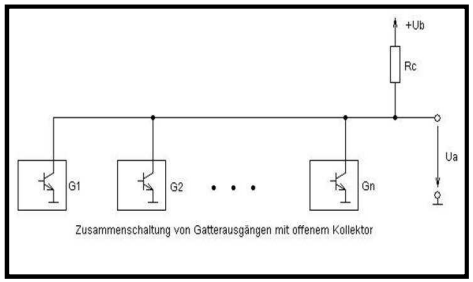
```
  if ((zahl >= 0) && (zahl < 65))                    // wenn eine Zahl zwischen 1 und 64
  {
    showPixel(zahl);                                 // diese LED einschalten
  }
  else
  {
    if (zahl > 64)                                    // wenn Zahl > 64
    {
        empfZahl = 0;                                 // Zifferneingabe zu groß,  bereit für neue Zahlenumwandlung
        zahl = -1;                                    // und "zahl" löschen
    }
  }
  if (chLetter == 'H')                                // Wenn Buchstabe 'H' Herzschlag anzeigen
  {
    show(smallHeart, 100);                            // 80 show the small heart image for 100 ms
    show(bigHeart, 200);                              // 160 followed by the big heart for 200ms
    delay(pulseDelay);                                // show nothing between beats
  }  // if

  if (chLetter == 'D')                                // wenn Buchstabe 'D' Demo Lauflicht anzeigen
  {
    showDemo();
  }  // if
}  // loop
```

```
// routine to show a frame of an image stored in the array pointed to
// by the image parameter.
// the frame is repeated for the given duration in milliseconds
void show( byte * image, unsigned long duration)
{
  unsigned long start = millis();             // begin timing the animation
  while (start + duration > millis())         // loop until duration period has passed
  {
    for(int row = 0; row < 8; row++)
    {
      digitalWrite(rowPins[row], HIGH);           // connect row to +5 volts
      for(int column = 0; column < 8; column++)
      {
        boolean pixel = bitRead(image[row],column);
        if(pixel == 1)
        {
          digitalWrite(columnPins[column], LOW);  // connect column to Gnd
        }
        delayMicroseconds(300);                   // a small delay for each LED
        digitalWrite(columnPins[column], HIGH);   // disconnect column from Gnd
      }  // for column
      digitalWrite(rowPins[row], LOW);            // disconnect LEDs
    }    // for row
  }     // while
}    // fct fct. show

void showDemo()
{
  for (int pixel = 0; pixel < 64; pixel++)
  {
    showPixel(pixel);
  }  // for pixel
  delay(100);
}  // fct. demo
```

```
void showPixel(int pixelNr)
{
  int columnLevel = pixelNr / 8;                    // map to the number of columns
  int rowLevel = pixelNr % 8;                       // get the fractional value
  for (int column = 0; column < 8; column++)
  {
    digitalWrite(columnPins[column], LOW);          // connect this column to Ground
    for(int row = 0; row < 8; row++)
    {
      if (columnLevel > column)
      {
        digitalWrite(rowPins[row], HIGH);           // connect all LEDs in row to +5 volts
      }  // if
      else
      {
        if (columnLevel == column && rowLevel > row)
        {
          digitalWrite(rowPins[row], HIGH);
        }  // if
        else
        {
          digitalWrite(columnPins[column], LOW); // turn off all LEDs in this row
        }  // else
      }  // else
      delayMicroseconds(onTime);                // gives frame time (64 LEDs mal 300 = 20ms)
      digitalWrite(rowPins[row], LOW);          // turn off LED
    }  // for "row"
    digitalWrite(columnPins[column], HIGH);   // disconnect this column from GND
  }  // for "column"
}  // fct, showPixel
```

Sensoren mit serieller Schnittstelle: One Wire, SPI oder I2C

Bisher haben wir Sensoren direkt, also analog eingelesen (Poti‘s, Schalter, Licht- oder Temperatur). Es gibt auch „intelligente“ Sensoren, die ihre Werte (oft mehrere) über diverse serielle Schnittstellen, also digital, übertragen, z.b.

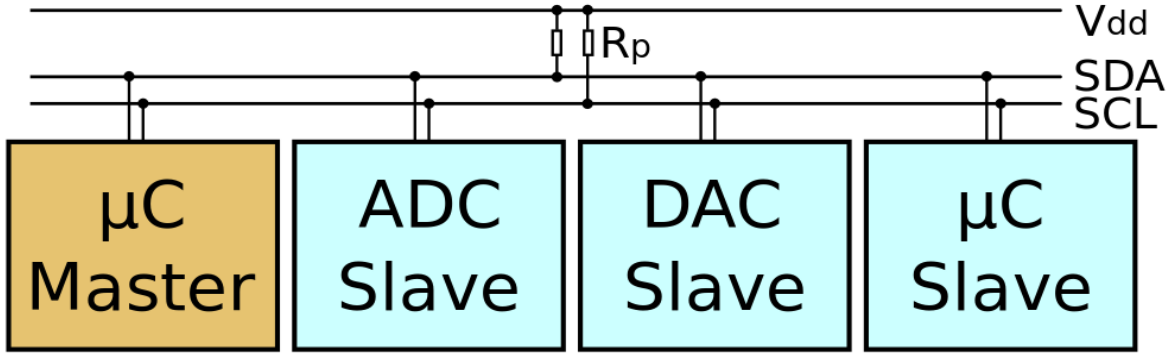


SPI = Serial Peripheral Interface
I²C = Inter-Integrated Circuit

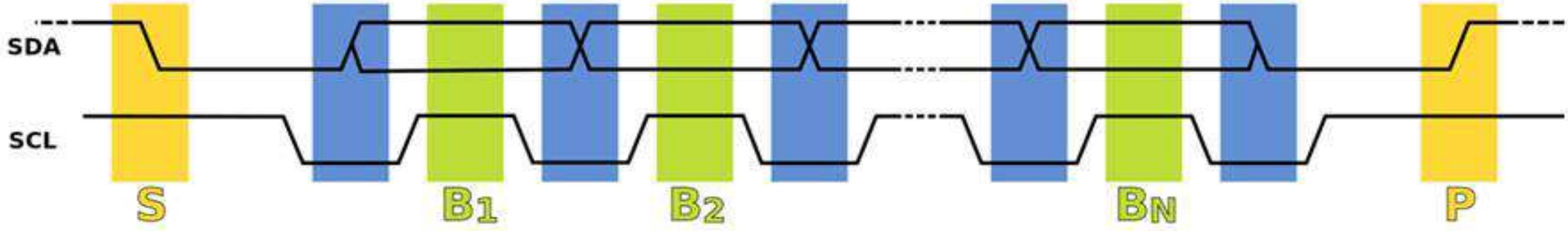
Vorteil: **nur ein bis drei I/O-Leitungen, viele Sensoren auf gleichen Leitungen gleichzeitig (Master - Slave Betrieb)**

Nachteil: **man muss das genaue digitale „Übertragungs-Protokoll“ kennen.**

I2C Bus: Schaltbild (Anschluß) bzw. Funktionsweise



I²C benötigt zwei Signalleitungen: Takt- (SCL) und Datenleitung (SDA). Beide liegen mit Pull-up-Widerständen R_P an der Versorgungsspannung V_{DD}. Alle angeschlossenen Geräte haben Open-Collector-Ausgänge, was zusammen mit den Pull-up-Widerständen eine Wired-AND-Schaltung ergibt. Der High-Pegel soll mindestens $0{,}7 \times V_{DD}$ betragen und der Low-Pegel soll höchstens bei $0{,}3 \times V_{DD}$ liegen. I²C-Bus arbeitet mit positiver Logik, d.h. ein HIGH-Pegel auf der Datenleitung entspricht einer logischen '1', der LOW-Pegel einer '0'



Bussystem

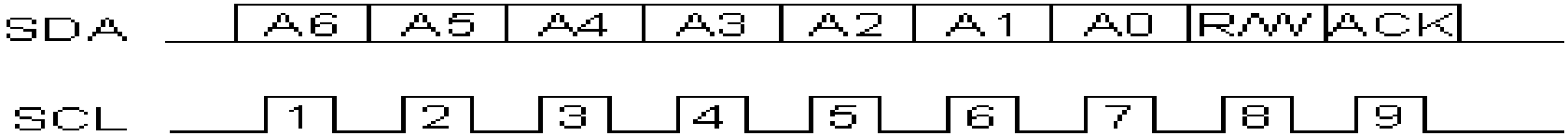
I²C ist als Master-Slave-Bus konzipiert. Ein Datentransfer wird vom Master initiiert; der über eine Adresse angesprochene Slave reagiert. Mehrere Master sind möglich (Multimaster-Mode).

Daten (Einzelbits) sind nur gültig, wenn sich ihr logischer Pegel während einer **Clock-High-Phase** nicht ändert. Ausnahmen sind das ***Start-, Stop-* und *Repeated-Start*-Signal**. Das *Start*-Signal ist eine fallende Flanke auf SDA, während SCL high ist, das *Stop*-Signal ist eine steigende Flanke auf SDA, während SCL high ist. *Repeated-Start* sieht genauso aus wie das *Start*-Signal.
Eine Dateneinheit besteht aus 8 Datenbits = 1 Oktett (wird als Wert oder Adresse interpretiert) und einem ACK-Bit. Dieses Bestätigungsbit (Acknowledge) wird vom Slave durch einen Low-Pegel auf der Datenleitung während der neunten Takt-High-Phase (welche nach wie vor vom Master generiert wird) und als NACK (für engl. *not acknowledge*) durch einen High-Pegel signalisiert. Der Slave muss den Low-Pegel an der Datenleitung anlegen *bevor* SCL auf high geht, andernfalls würden weitere eventuelle Teilnehmer ein *Start-Signal* lesen.
Wenn der Slave mehr Zeit benötigt, als durch den Master-Takt vorgegeben, kann er zwischen der Übertragung einzelner Bytes den Clock auf low halten (***Clock-Stretching***) und den Master bremsen.

Adressierung

Eine I²C-Adresse ist das erste vom Master gesendete Byte, wobei die ersten **sieben Bit die Adresse** darstellen und das **achte Bit (R/W-Bit)** dem Slave mitteilt, ob er Daten vom Master empfangen soll (LOW), oder Daten an den Master zu übertragen hat (HIGH). I²C nutzt daher einen Adressraum von7 Bit, was bis zu 112 Knoten auf einem Bus erlaubt (16 der 128 möglichen Adressen sind für Sonderzwecke reserviert).

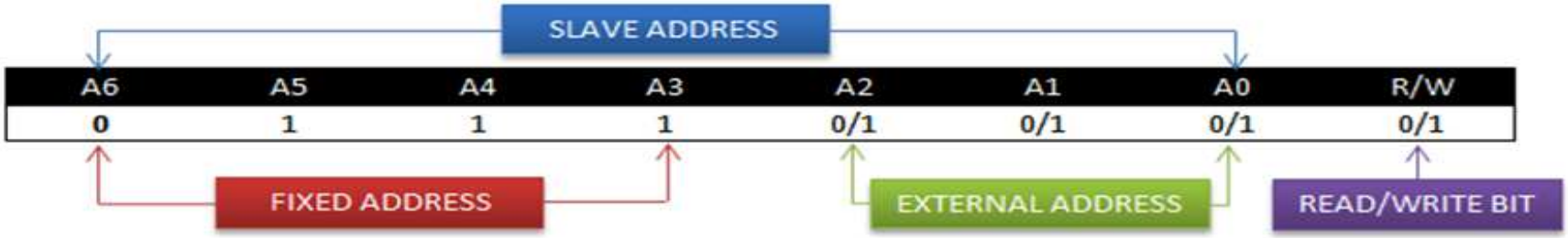
Jedes I²C-fähige IC (Sensor) hat eine vom Hersteller festgelegte Adresse, von der bisweilen die untersten drei Bits (*Subadresse* genannt) über drei Steuerpins festgelegt werden können. In diesem Fall können bis zu acht gleichartige ICs an einem I²C-Bus betrieben werden. Wenn nicht, müssen mehrere gleiche ICs mit getrennten I²C-Bussen angesteuert oder abgetrennt werden können.



Bsp.:

INTERFACE DEFINITION

BYTE	BIT							
	7 (MSB)	6	5	4	3	2	1	0 (LSB)
I²C slave address	L	H	L	L	A2	A1	AO	R/$\overline{W}$
I/O data bus	P7	P6	P5	P4	P3	P2	P1	P0



Übertragungsprotokoll

Der Beginn einer Übertragung wird mit dem ***Start*-Signal** vom **Master** angezeigt, dann folgt die **Adresse**. Diese wird durch das **ACK-Bit** vom entsprechenden **Slave** bestätigt. Abhängig vom **R/W-Bit** werden nun Daten byteweise geschrieben (Daten an Slave) oder gelesen (Daten vom Slave). Das ACK beim Schreiben wird vom Slave gesendet und beim Lesen vom Master. Das letzte Byte eines Lesezugriffs wird vom Master mit einem **NACK** quittiert, um das Ende der Übertragung anzuzeigen. Eine Übertragung wird durch das ***Stop*-Signal** beendet. Alternativ kann auch ein ***Repeated-Start*** am Beginn einer erneuten Übertragung gesendet werden, ohne die vorhergehende Übertragung mit einem *Stop*-Signal zu beenden.

Alle Bytes werden dabei „Most Significant Bit First“ übertragen. Hier ein Bsp.:

Wire Library für I2C Bus (TWI) → Wire.xxx()

Name	Bedeutung	Param.	Bedeutung	Return	Bedeutung
begin() begin(address)	Initiate the Wire library and join the I2C bus as a master or slave. This should normally be called only once	()	if not specified, join the bus as a master	none	
		address	7-bit slave address (optional)		
requestFrom(address, quantity) requestFrom(address, quantity, stop)	Used by the master to request bytes from a slave. The bytes then be retrieved with the available() and read() functions. accepts a boolean argument changing its behavior for compatibility with certain I2C devices. If true, requestFrom() sends a stop message after the request, releasing the I2C bus. If false, requestFrom() sends a restart message after the request. The bus will not be released, which prevents another master device from requesting between messages. This allows one master device to send multiple requests while in control. The default value is true.	address	7-bit address of the device to request bytes from	byte	number of bytes returned from the slave device
		quantity	the number of bytes to request		
		stop	boolean. true will send a stop message after the request, releasing the bus. false will continually send a restart after the request, keeping the connection active		
beginTransmission(address)	Begin a transmission to the I2C slave device with the given address. Subsequently, queue bytes for transmission with the write() function and transmit them by calling endTransmission()	address	7-bit address of the device to transmit to	none	
endTransmission() endTransmission(stop)	Ends a transmission to a slave device that was begun by beginTransmission() and transmits the bytes that were queued by write(). As of Arduino 1.0.1, endTransmission() accepts a boolean argument changing its behavior for compatibility with certain I2C devices. If true, endTransmission() sends a stop message after transmission, releasing the I2C bus. If false, endTransmission() sends a restart message after transmission. The bus will not be released, which prevents another master device from transmitting between messages. This allows one master device to send multiple transmissions while in control. The default value is true	()	The default value is true	byte	0:success 1:data too long to fit in transmit buffer 2:received NACK on transmit of address 3:received NACK on transmit of data 4:other error
		stop	boolean. true will send a stop message, releasing the bus after transmission. false will send a restart, keeping the connection active.		

Wire Library für I2C Bus (TWI) → Wire.xxx()

Name	Bedeutung	Param.	Bedeutung	Return	Bedeutung
write(value) write(string) write(data, length)	Writes data from a slave device in response to a request from a master, or queues bytes for transmission from a master to slave device (in-between calls to beginTransmission() and endTransmission()).	value	a value to send as a single byte	byte	will return the number of bytes written, though reading that number is optional
		string	a string to send as a series of		
		data	an array of data to send as		
		length	the number of bytes to transmit		
available()	Returns the number of bytes available for retrieval with read(). This should be called on a master device after a call to requestFrom() or on a slave inside the onReceive() handler	()	none	byte	number of bytes available for reading
read()	Reads a byte that was transmitted from a slave device to a master after a call to requestFrom() or was transmitted from a master to a slave	()	none	byte	The next byte received
onReceive(handler)	Registers a function to be called when a slave device receives a transmission from a master	handler	the function to be called when the slave receives data; this should take a single int parameter (the number of bytes read from the master) and return nothing, e.g.: void myHandler(int numBytes)	none	
onRequest(handler)	Register a function to be called when a master requests data from this slave device	handler	the function to be called, takes no parameters and returns nothing, e.g.: void myHandler()	none	

Board	I2C / TWI pins
Uno, Ethernet	A4 (SDA), A5 (SCL)
Mega2560	20 (SDA), 21 (SCL)
Leonardo	2 (SDA), 3 (SCL)
Due	20 (SDA), 21 (SCL), SDA1, SCL1

On Arduino boards with R3 layout (1.0 pinout), SDA (data line) and SCL (clock line) are on the pin headers close to the AREF pin.
Arduino Due has two I2C / TWI interfaces: SDA1 and SCL1 are near to the AREF pin and the additional one is on pins 20 and 21.

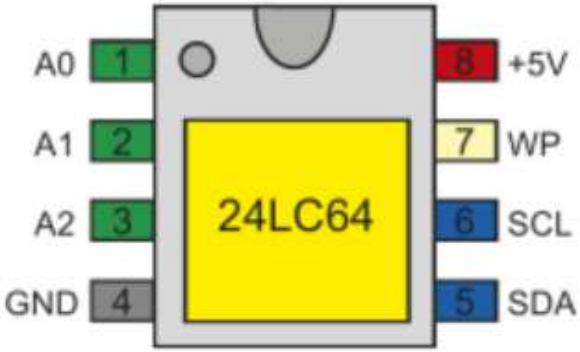
I2C-Bus: Typ. Bsp. zum Lesen von Daten

```
#include <Wire.h>                // „Wire“ Bibliothek muss eingebunden werden

void setup()
{
  Wire.begin();                  //  I2C initalisieren, kein Param.  (address) da Arduino Master ist
  Serial.begin(9600);            // „Serial Monitor“ starten um gelesene Daten auszugeben
}

void loop()
{
  Wire.requestFrom(44, 6);       //  6 bytes vom Slave mit Adresse  #44 (0x2c)  anfordern
  while(Wire.available())        // Sind Slave Daten da? Dann lesen (read) sonst weiter
  {
    char c = Wire.read();        // ein Byte vom Slave als Character (Zeichen) lesen
    Serial.print(c);             // dieses Zeichen über den Monitor ausgeben
  }
  delay(500);                    // warten, dann wieder von Vorne
}
```

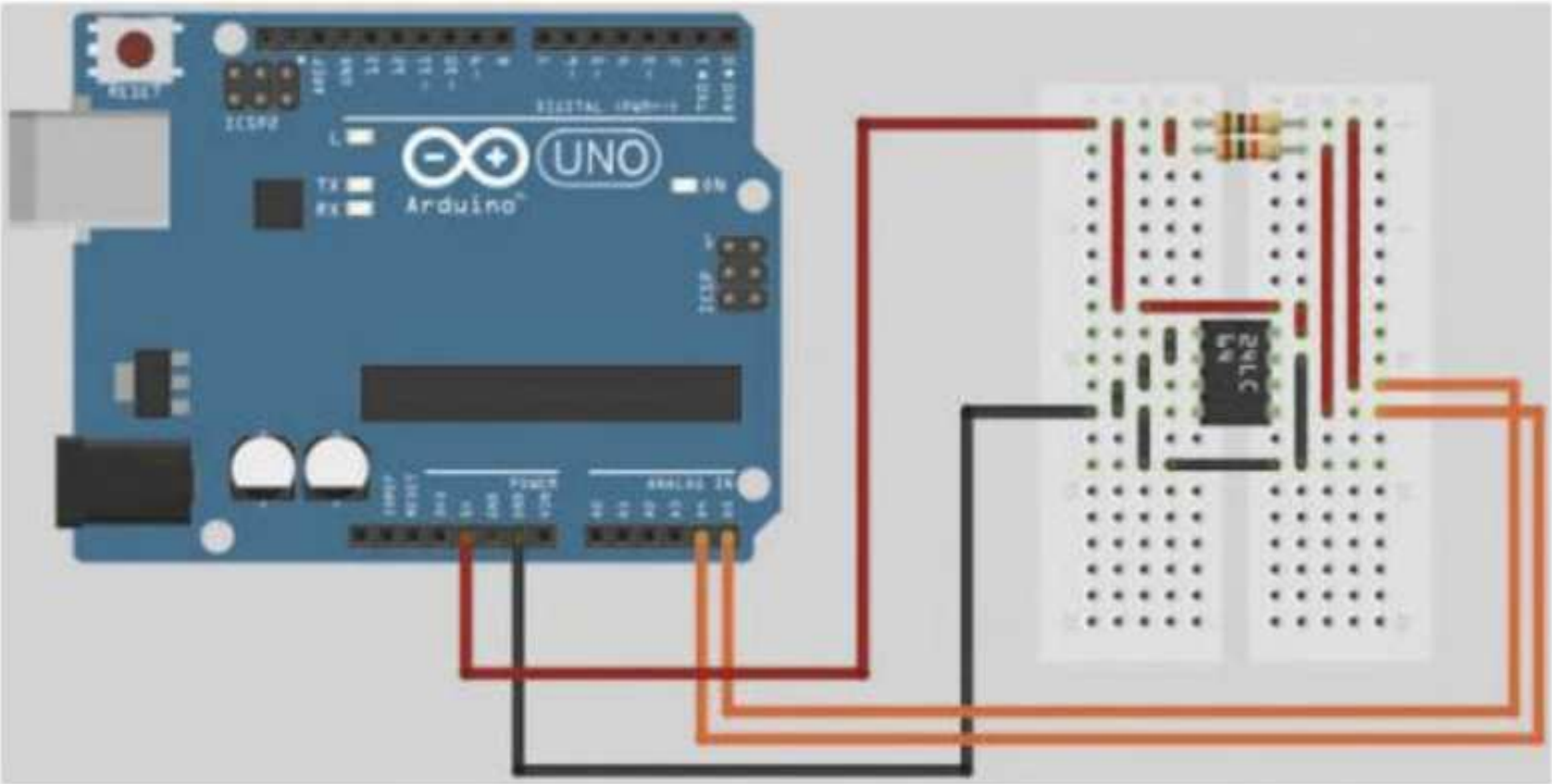
I2C-Bus: Typ. Bsp. zum Senden von Daten

```
#include <Wire.h>                          // „Wire“ Bibliothek muss eingebunden werden

byte wert = 0;

void setup()
{
  Wire.begin();                            // I2C initalisieren, kein Param. (address) da Arduino Master ist
}

void loop()
{
  Wire.beginTransmission(44);              // transmit to Slave mit Adresse #44 (0x2c)  (aus Datenblatt)
  Wire.write(wert);                        // byte „wert“ senden
  Wire.endTransmission();                  // stop transmitting
  wert++;                                  // „wert“ um eins erhöhen
  if(wert == 64)                           // if „wert“ hat maximum von 64 erreicht
  {
    val = 0;                               // „wert“ auf „0“ setzen und wieder beginnen
  }
  delay(500);                              // warten, dann wieder von Vorne
}
```

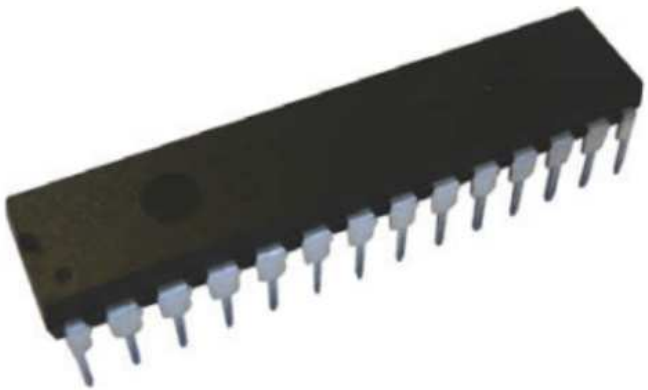
24LC64 I²C-Bus EEPROM



Pin	Bezeichnung	Bedeutung
1	A0	Bit 0 der Adresse
2	A1	Bit 1 der Adresse
3	A2	Bit 2 der Adresse
4	V_{SS} bzw. GND	Masse
5	SDA	Serial Data Line (wird mit analog Pin 4 verbunden)
6	SCL	Serial Clock Line (wird mit analog Pin 5 verbunden)
7	WP	Write Protect (GND: ermöglicht schreiben, +5V: nur lesen)
8	V_{CC} bzw. +5V	Spannungsversorgung

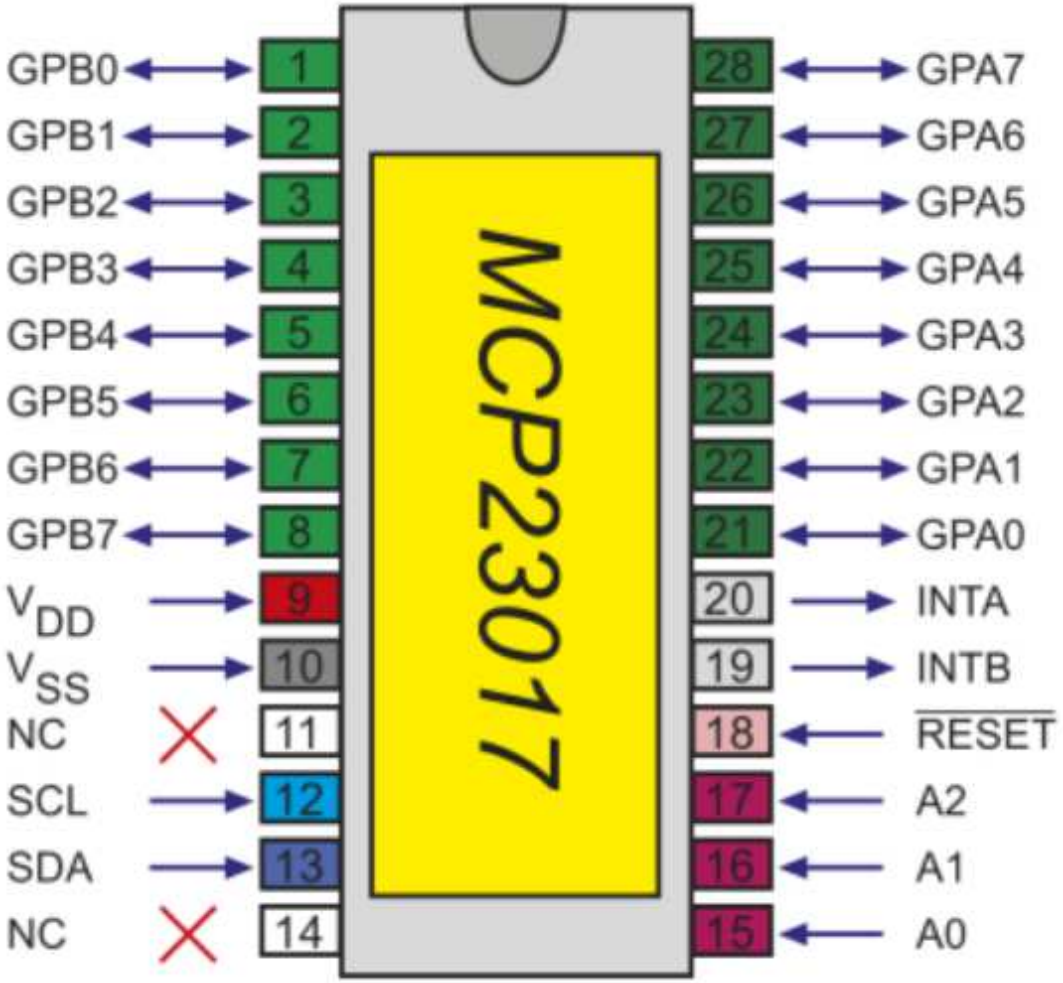


MCP23017 I²C-Bus Port Expander



Port B
GPIOB Register:*0x13*

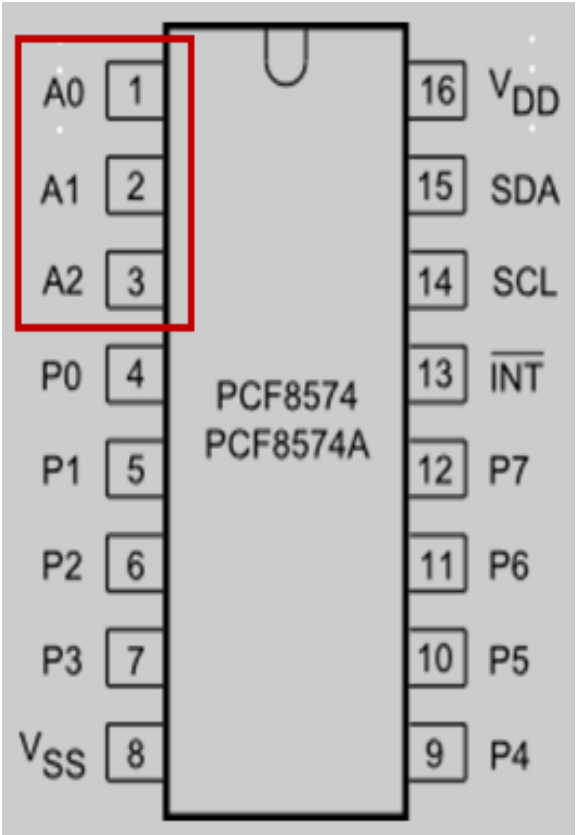
Port A
GPIOA Register:*0x12*

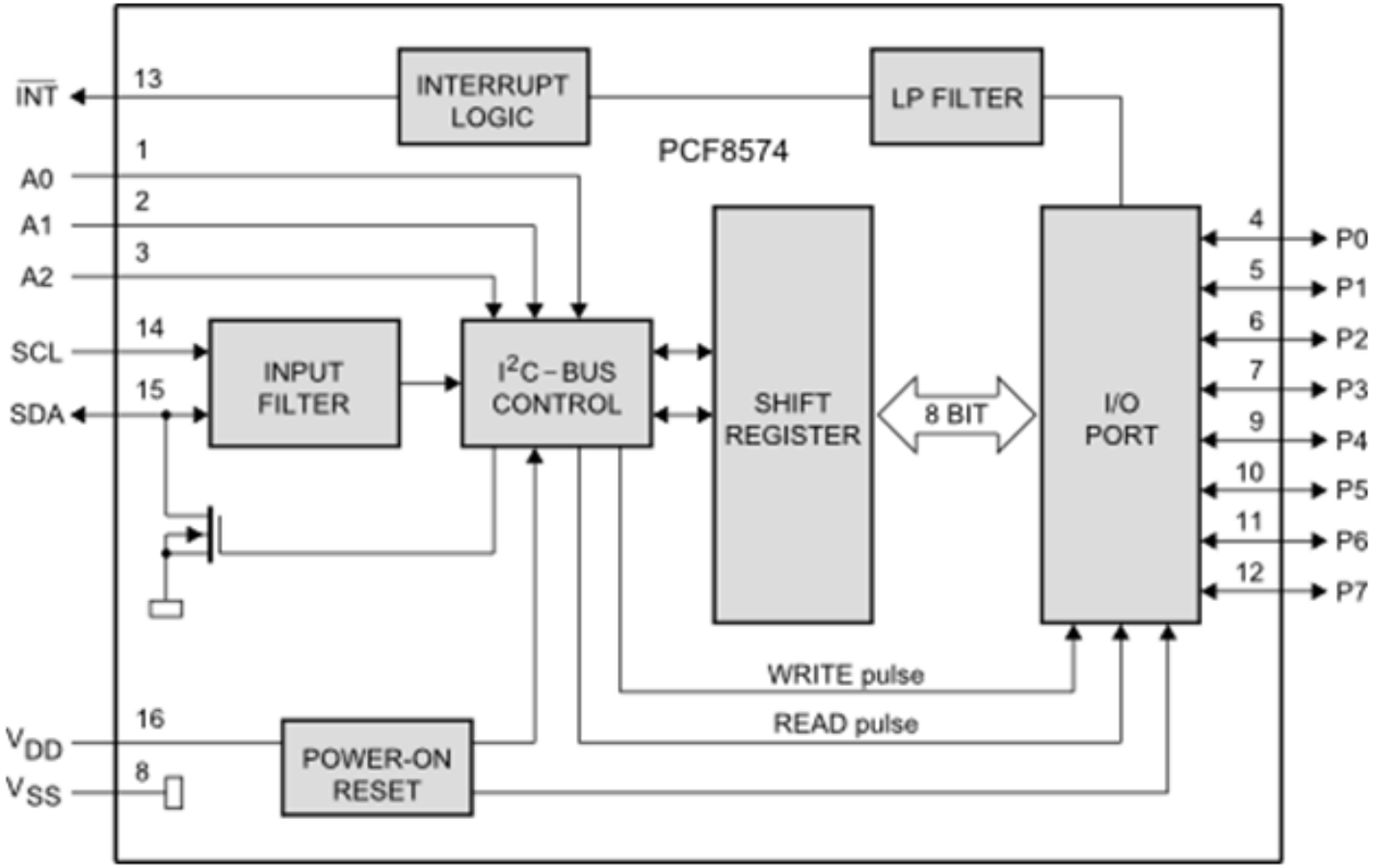


PCF8574 I²C-Bus Remote 8-Bit I/O Expander

8574 and **8574A** have **8 digital I/O** bits and can be addressed with **0x20-0x27** and **0x38-0x3F (A)**

All digital I/O ports can be configured as an input or output, and up to 8 of each type can be used at the same time on the same bus. Using 8574 and 8574A, you could have 16 independently addressable devices with 8 bits of I/O each - giving you a **total of 128 additional I/O lines**



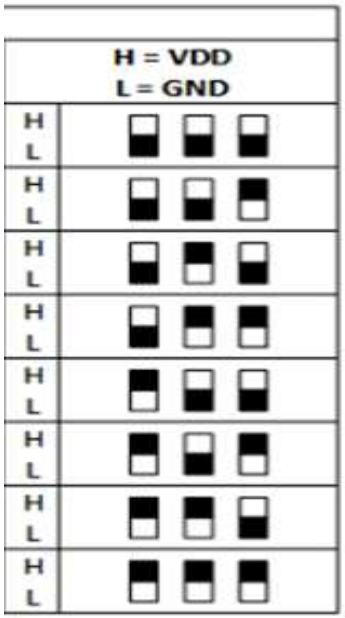


Die Versorgungsspannung kann zw. 2,5 und 6V variieren, i.d.R. Arduino mit 5V. Im Standby-Betrieb benötigt er maximal 10µA Strom. Er kann bis zu +/-25mA pro Ausgang liefern, damit können LEDs direkt geschaltet werden (natürlich über einen Vorwiderstand). Er verfügt über drei Adressleitungen, mit denen 8 unterschiedliche Adressen eingestellt werden können, so dass 8 PCF8574 direkt über den I²C-Bus adressiert werden können.
Seine Ports können **als Aus- und Eingänge konfiguriert** werden. Auf Pegeländerungen an den Eingängen kann er den Arduino über einen Open-Drain-Interrupt-Ausgang aufmerksam machen. Der PCF8574 kann einen **Interrupt auslösen**, woraufhin der Arduino die Eingänge abfragen kann, so ist er nicht ständig damit beschäftigt, die Eingangsleitungen zu überwachen (Pollen).

Bezeichnung	Beschreibung
A0	address input 0
A1	address input 1
A2	address input 2
P0	quasi-bidirectional I/O 0
P1	quasi-bidirectional I/O 1
P2	quasi-bidirectional I/O 2
P3	quasi-bidirectional I/O 3
V_{SS}	supply ground
P4	quasi-bidirectional I/O 4
P5	quasi-bidirectional I/O 5
P6	quasi-bidirectional I/O 6
P7	quasi-bidirectional I/O 7
$\overline{INT}$	interrupt output (active LOW)
SCL	serial clock line
SDA	serial data line
V_{DD}	supply voltage

ADDRESS REFERENCE

INPUTS			I²C-BUS SLAVE ADDRESS
A2	A1	A0	
L	L	L	32 (decimal), 20 (hexadecimal)
L	L	H	33 (decimal), 21 (hexadecimal)
L	H	L	34 (decimal), 22 (hexadecimal)
L	H	H	35 (decimal), 23 (hexadecimal)
H	L	L	36 (decimal), 24 (hexadecimal)
H	L	H	37 (decimal), 25 (hexadecimal)
H	H	L	38 (decimal), 26 (hexadecimal)
H	H	H	39 (decimal), 27 (hexadecimal)



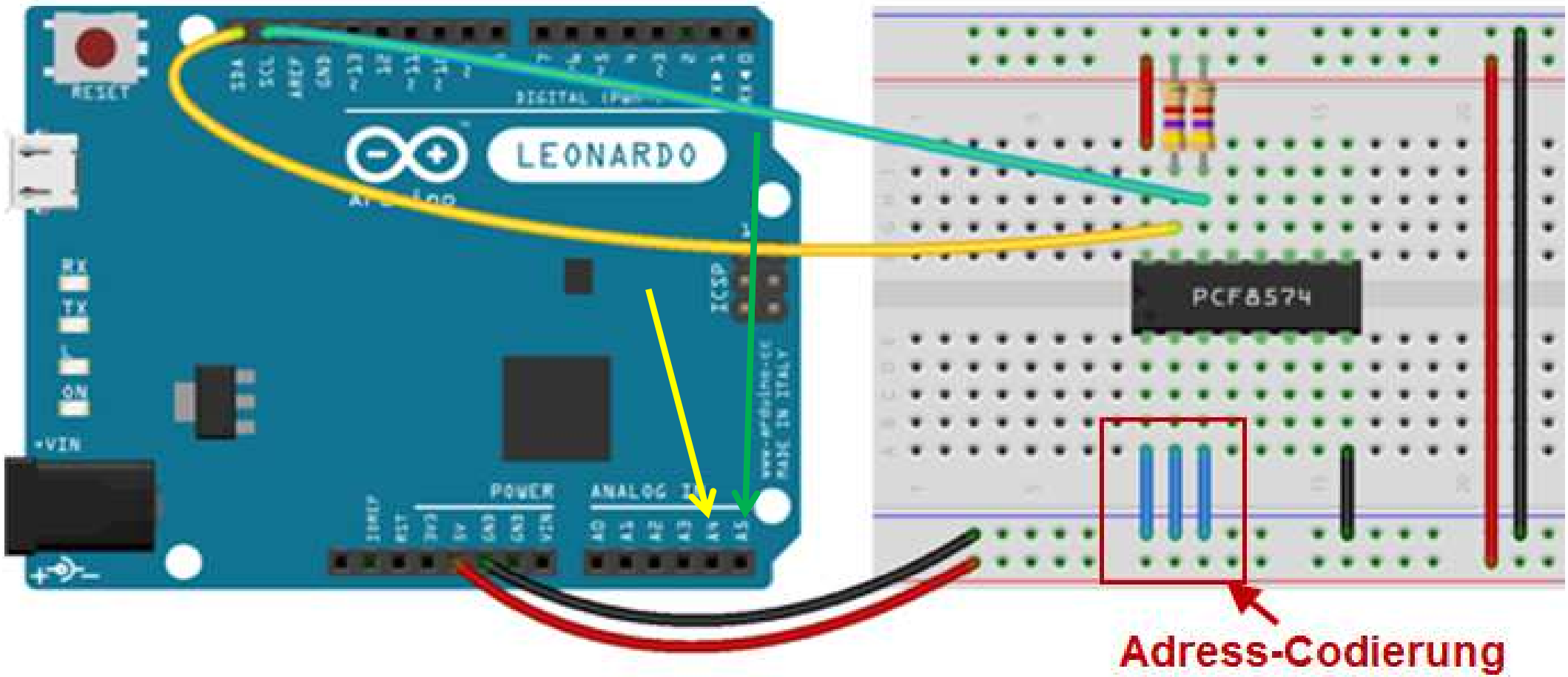
Adressen bei Bedarf einstellbar

Das Adressbyte für den PCF8574:

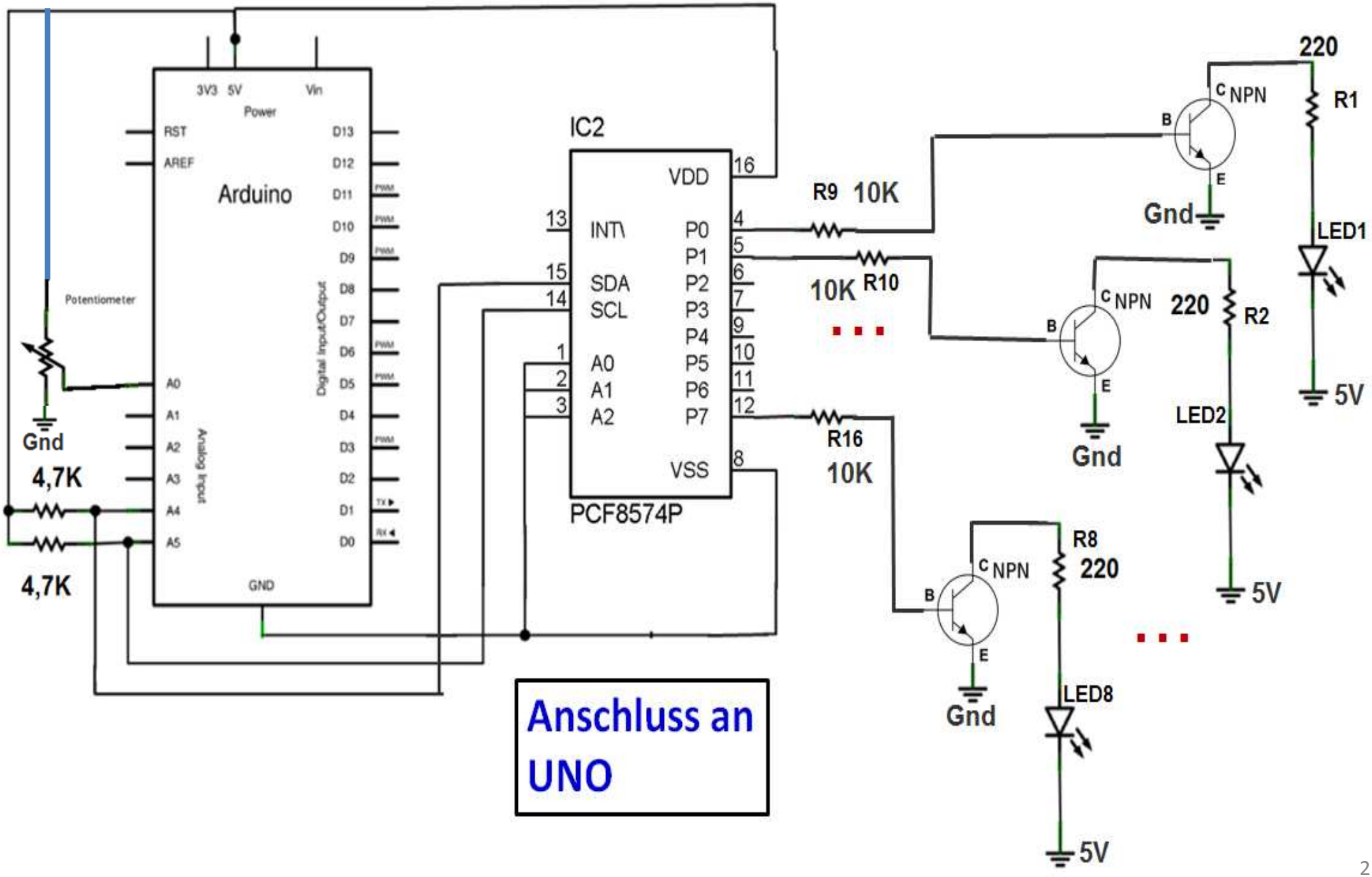
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Hexadezimal
0	1	0	0	A2	A1	A0	R/W	0x40 + A[2:0]*2 + R/W

Das Adressbyte für den PCF8574**A**:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Hexadezimal
0	1	1	1	A2	A1	A0	R/W	0x70 + A[2:0]*2 + R/W



Angeschlossen wird der PCF8574 an den I²C-Bus des Microcontrollers.
Leonardo: s. Bild; **Uno: SDA (Serial Data) = Pin A4; SCL (Serial Clock) = Pin A5**
Die beiden Leitungen SDA und SCL werden mit einem Pullup-Widerstand versehen. Je nach Leitungsführung und -länge ist ein Wert von ca. 1KΩ bis ca. 10KΩ angebracht.
(im Bild werden 4,7KΩ verwandt)

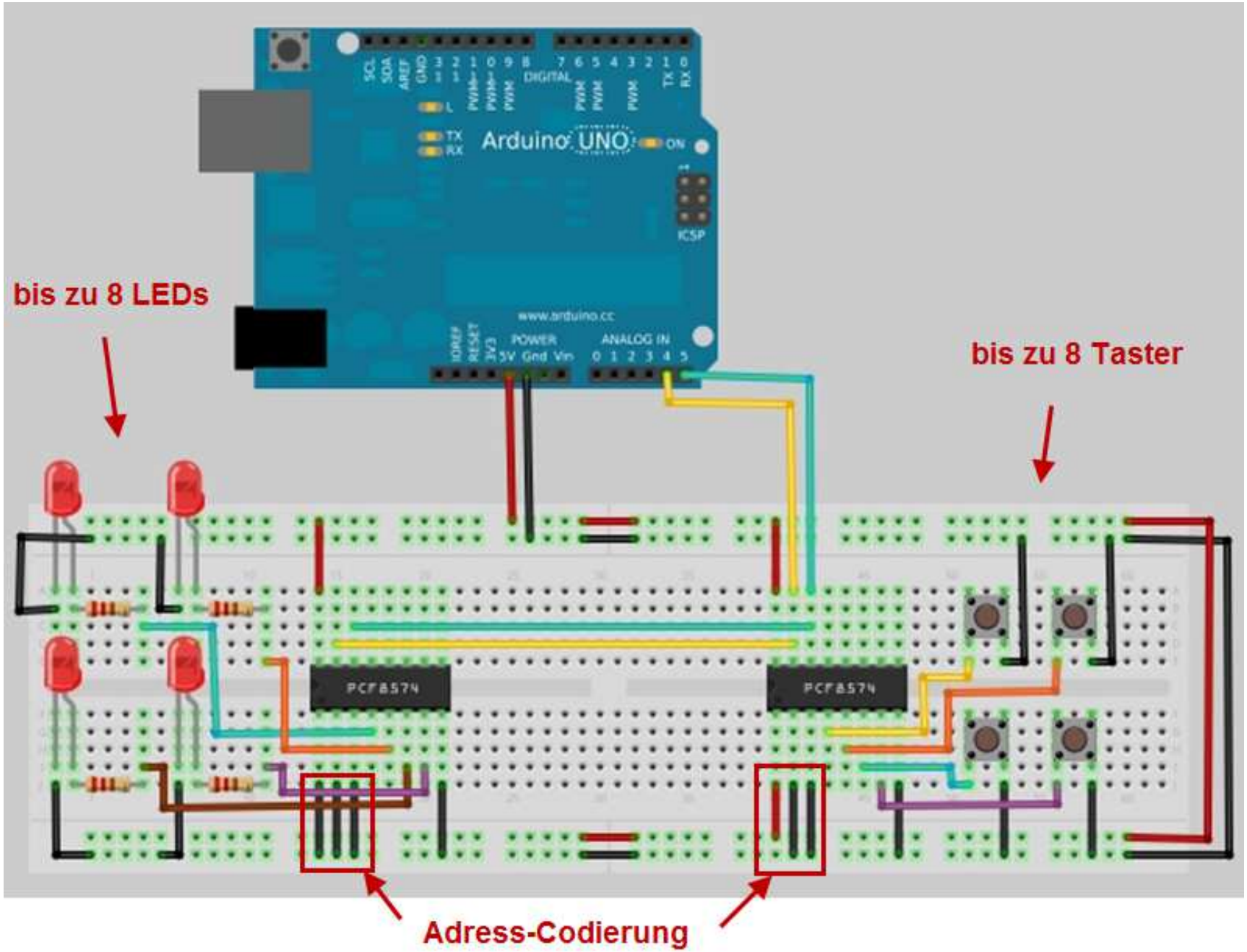

3V3 5V Vin
Power
RST
AREF
Arduino
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
PWM
Digital Input/Output
A0
A1
A2
A3
A4
A5
Analog Input
GND
Potentiometer
Gnd
4,7K
4,7K
IC2
VDD
INT\
SDA
SCL
A0
A1
A2
P0
P1
P2
P3
P4
P5
P6
P7
VSS
PCF8574P
R9 10K
10K R10
R16
10K
NPN
B
C
E
Gnd
220
R1
LED1
5V
R2
LED2
R8
220
LED8
Anschluss an
UNO

I2C_PortExp_PCF8574

```
/* Uses I2C port to drive a bar graph
 * Turns on a series of LEDs proportional to a value of an analog sensor. */

#include <Wire.h>

const int address = 0x38;                                   // Adresse fuer PCF8574
const int maxLEDs = 8;                                      // Anzahl LEDs

const int analogInPin = 0;                                  // Analog Eingangspin verbunden mit Poti

int sensorValue = 0;                                        // Wert vom Poti
int ledLevel = 0;                                           // Poti Wert in LED Level umwandeln
int ledBits = 0;                                            // bits fuer jede LED auf 0 (aus) oder 1 (ein)

void setup()
{
  Wire.begin();                                             // I2C Bus initialisieren
}

void loop()
{
  sensorValue = analogRead(analogInPin);                    // analog Wert einlesen
  ledLevel = map(sensorValue, 0, 1000, 0, maxLEDs);         // in Anzahl LEDs "mapen"
  for (int led = 0; led < maxLEDs; led++)
  {
    if (led < ledLevel)
    {
      bitWrite(ledBits,led, HIGH);                          // LED einschalten wenn kleiner Level
    }
    else
    {
      bitWrite(ledBits,led, LOW);                           // LED ausschalten wenn groeßer Level
    }
  }
  // send the value to I2C
  Wire.beginTransmission(address);                          // Byte an I2C Expander senden
  Wire.write(ledBits);
  Wire.endTransmission();
  delay(300);
}
```

```
/*****************************************************************
*
*  PCF8574 I²C-Bus-Expander als Eingabeport
*
*****************************************************************/

#include <Wire.h>

#define I2C_ADDR     0x40 >> 1 // I2C-Addresse


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

  // Für Leonardo
  while (!Serial)
    ;

  Wire.begin(I2C_ADDR);

  Serial.println("Testprogramm PCF8574 I2C-Bus-Expander");
  Serial.println("----------------------------------------");
  Serial.println();
}


void loop() {
  Wire.beginTransmission(I2C_ADDR);
  Wire.write(I2C_ADDR | 0x01);     // Alle Ports als Eingang
  Wire.requestFrom(I2C_ADDR, 1);   // Ein Byte (= 8 Bits) anfordern
  while(Wire.available() == 0)     // Warten, bis Daten verfügbar
    ;
  Serial.print("Eingelesen: ");
  Serial.println(Wire.read());
  Wire.endTransmission(true);

  delay(50);
}
```

```
/*****************************************************************
*
* PCF8574 I²C-Bus-Expander als Ausgabeport
*
*****************************************************************/

#include <Wire.h>

#define I2C_ADDR     0x40 >> 1 // I2C-Adresse


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

  // Für Leonardo
  while (!Serial)
    ;

  Wire.begin(I2C_ADDR);

  Serial.println("Testprogramm PCF8574 I2C-Bus-Expander");
  Serial.println("--------------------------------------");
  Serial.println();
}


void loop() {
  Serial.println("Pins HIGH schalten");
  Wire.beginTransmission(I2C_ADDR);
  Wire.write(I2C_ADDR & 0xFE);    // Alle Ports als Ausgang
  Wire.write(0xFF);               // Alle Pins HIGH
  Wire.endTransmission(true);

  delay(3000);

  Serial.println("Pins LOW schalten");
  Wire.beginTransmission(I2C_ADDR);
  Wire.write(I2C_ADDR & 0xFE);    // Alle Ports als Ausgang
  Wire.write(0x00);               // Alle Pins LOW
  Wire.endTransmission(true);

  delay(3000);
}
```



Sehr gutes Beispiel:

The left PCF8574 is a receiver, and it will control the LEDs.

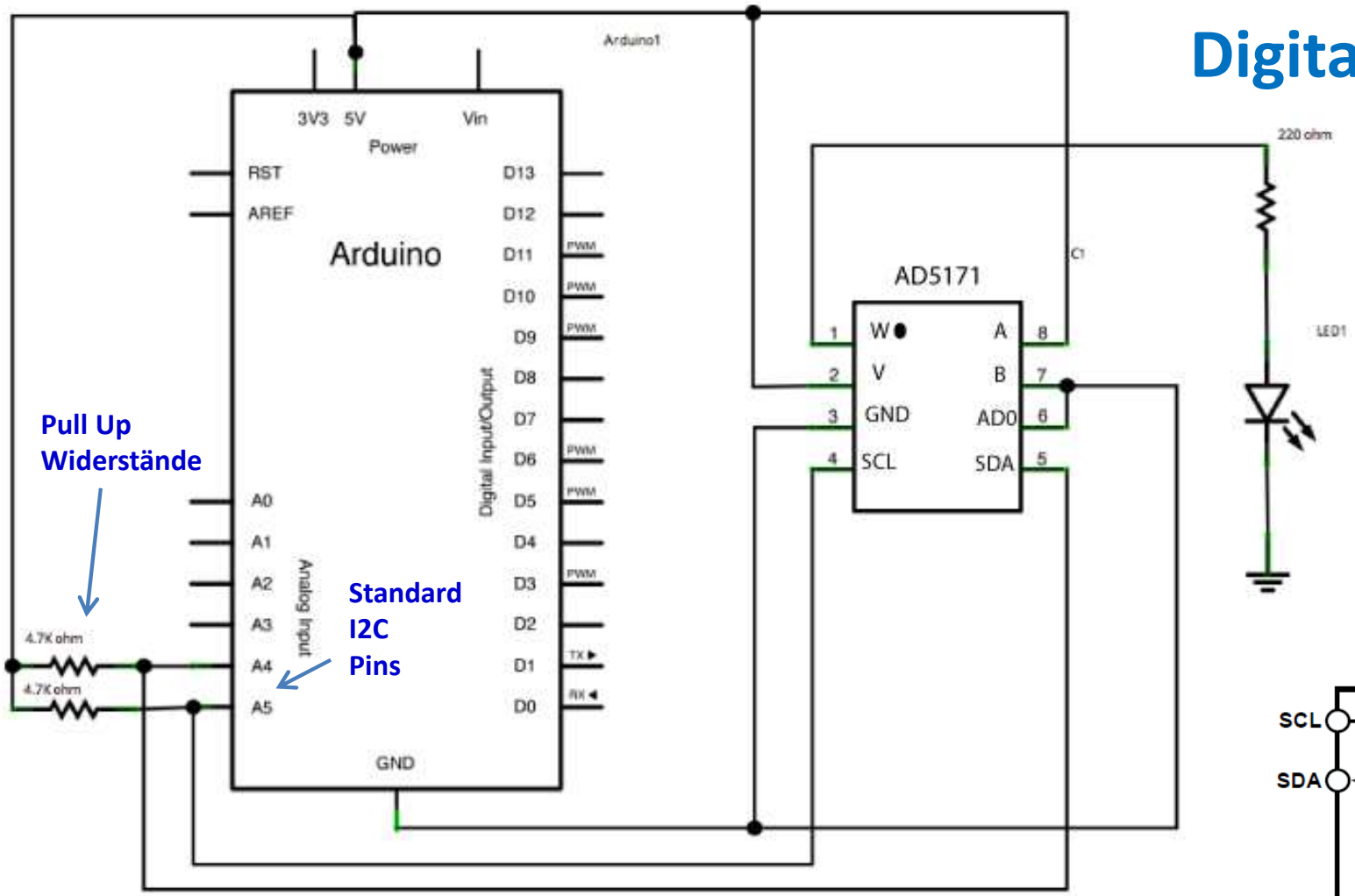
The right one will be a emitter and will transmit the button's status.

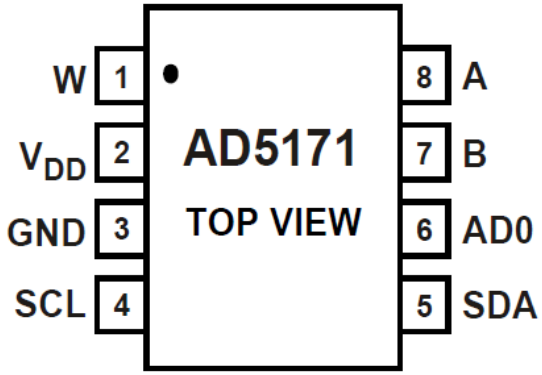
When you press a button, the Arduino will read it from the I2C interface and will command the left IC to turn the corresponding LED on.

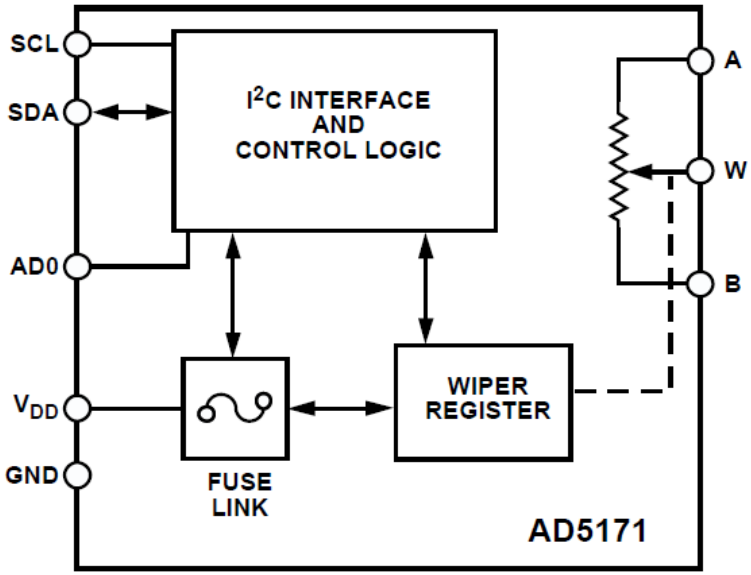
Für I2C gibt es fertige Bibliotheken. (#include <wire.h>)

→ Dennoch Datenblatt des Sensors und Beschreibung der Bibliothek lesen.

Digitales Potentiometer mit I2C Bus

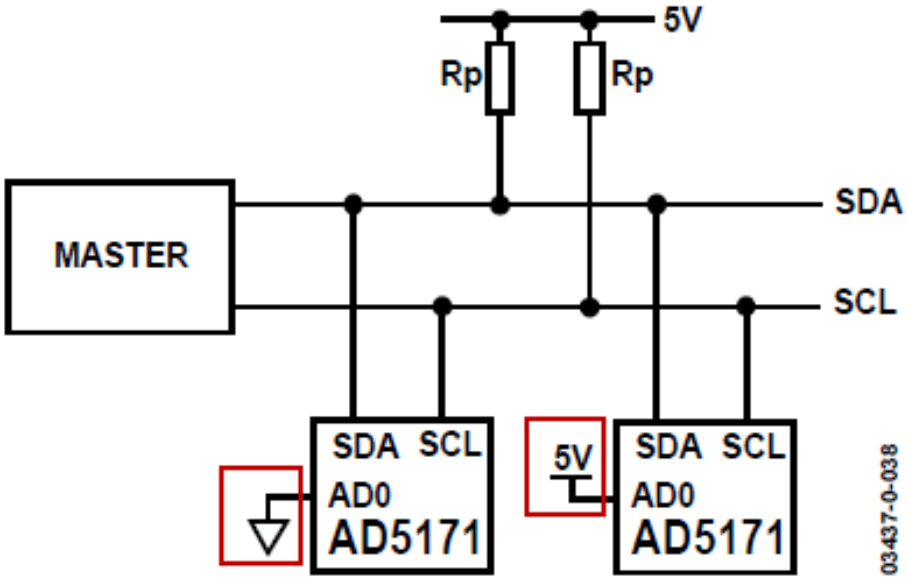






Functional Block Diagram

Baustein i.d.R. nur im SMD und leider nicht im DIL Gehäuse zu bekommen

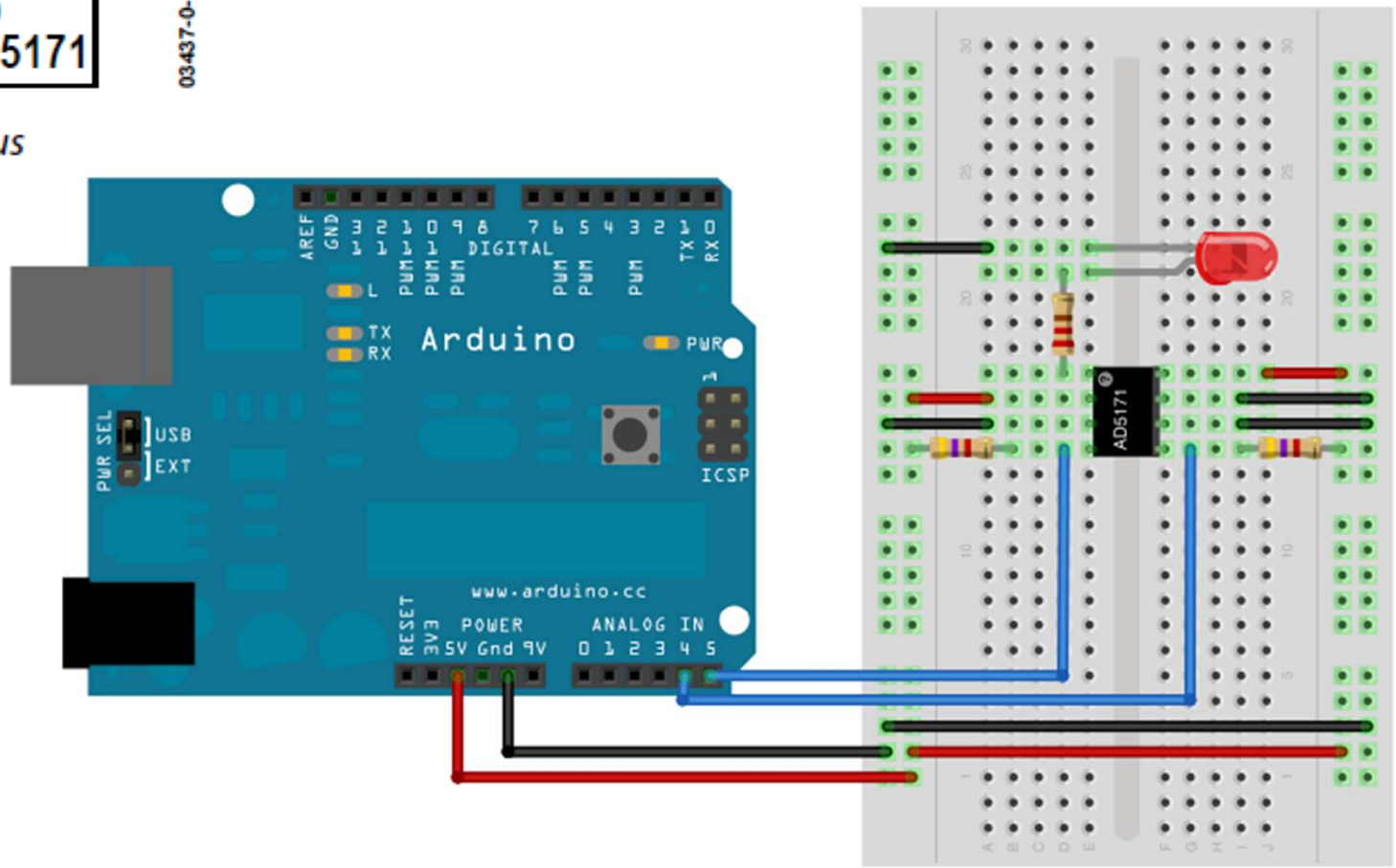


Two AD5171 Devices on One Bus

2 Bausteine max (Adressen):

Poti1	**2C HEX,**	**44 DEZ**
Poti2	**2D HEX,**	**45 DEZ**

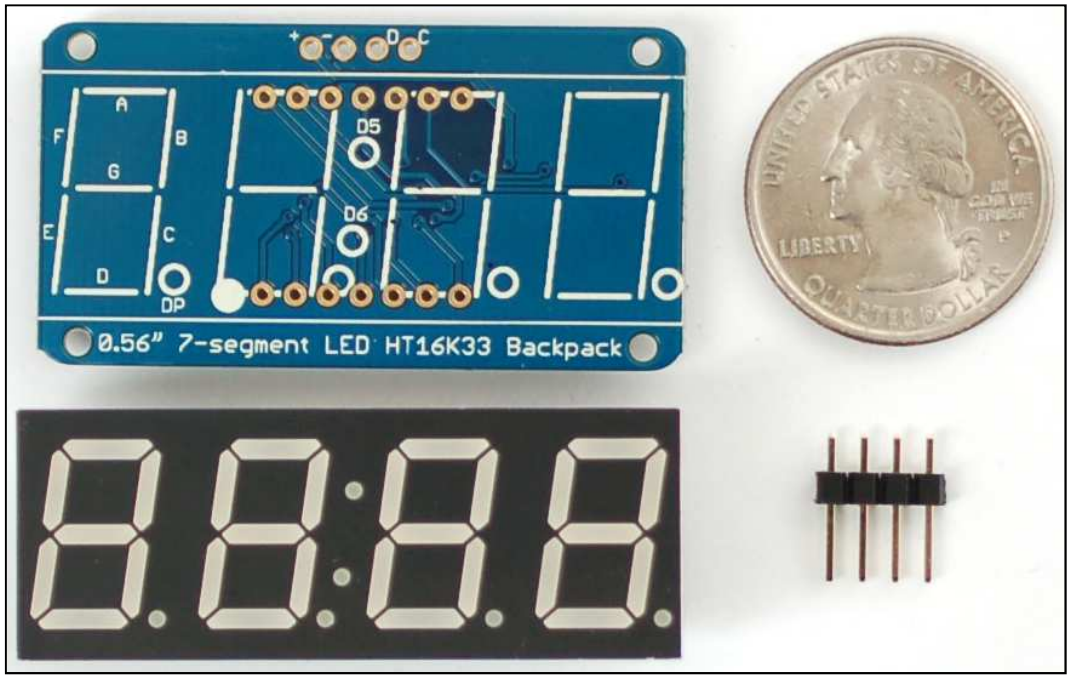
Dieses Bsp. dimmt eine LED

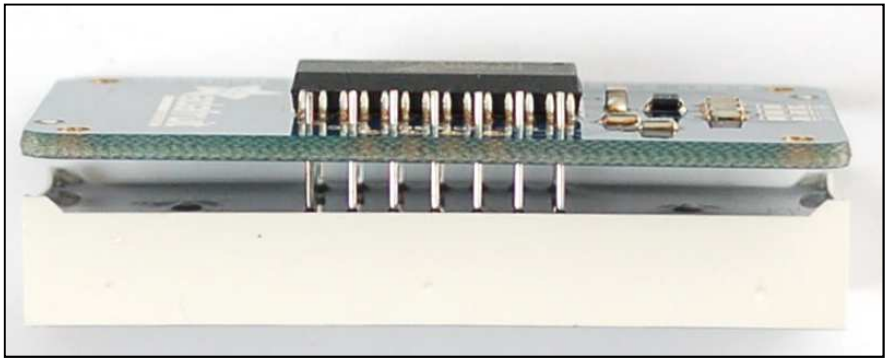


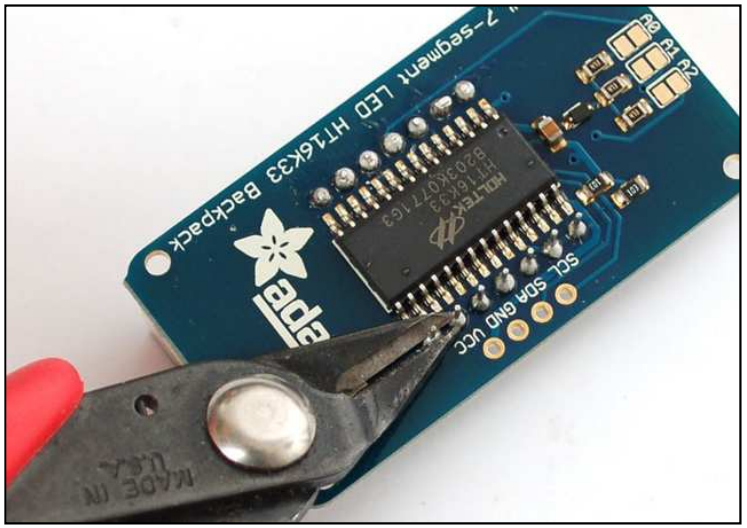
AD5171_I2C_digital_potentiometer

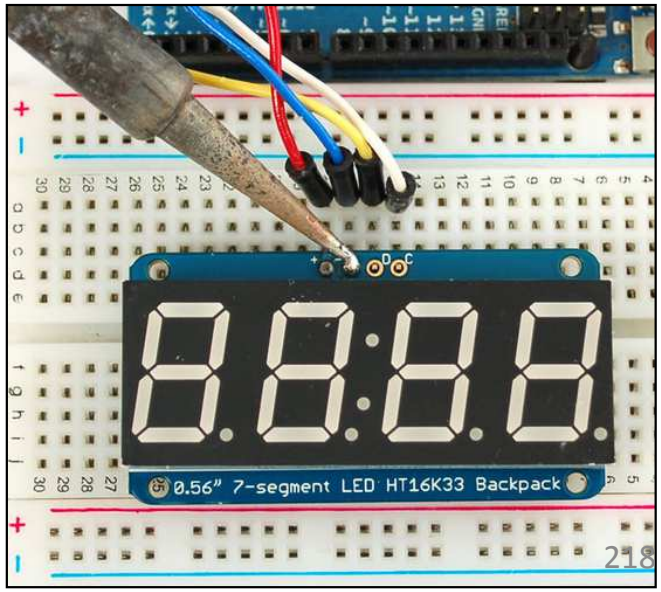
```
/* I2C Digital Potentiometer by Nicholas Zambetti and Shawn Bonkowski

Demonstrates use of the Wire library and controls AD5171 digital potentiometer
via I2C/TWI
Created 31 March 2006  */

#include <Wire.h>

byte val = 0;                        // Helligkeitswert für LED

void setup()
{
  Wire.begin();                      // Initialisliert (startet) den "I2C" Bus
}  // fct "setup"

void loop()
{
  Wire.beginTransmission(44);        // transmit to device with Adress #44 (0x2c)
  Wire.write(byte(0x00));            // sendet ein Befehlsbyte an Poti
  Wire.write(val);                   // sendet ein Datenbyte an Poti
  Wire.endTransmission();            // stop transmitting

  val++;                             // Dimmen: Helligkeitswert um 1 erhöhem
  if(val == 64)                      // if reached 64th position (max)
  {
    val = 0;                         // start over from lowest value
  }  // if
  delay(500);                        // Halbe Sekunde warten
}  // fct "loop"
```

I²C-Bus: 4fach 7-Segment Anzeige









I2C_4x7Segement

```
/*  This is a library for our I2C LED Backpacks
  Designed specifically to work with the Adafruit LED 7-Segment backpacks
  ----> http://www.adafruit.com/products/881
  ----> http://www.adafruit.com/products/880
  ----> http://www.adafruit.com/products/879
  ----> http://www.adafruit.com/products/878

  These displays use I2C to communicate, 2 pins are required to
  interface. There are multiple selectable I2C addresses. For backpacks
  with 2 Address Select pins: 0x70, 0x71, 0x72 or 0x73.
  For backpacks with 3 Address Select pins: 0x70 thru 0x77    */

#include <Wire.h>     // Enable this line if using Arduino Uno, Mega, etc.

#include "Adafruit_LEDBackpack.h"
#include "Adafruit_GFX.h"

Adafruit_7segment matrix = Adafruit_7segment();

void setup() {
  Serial.begin(9600);
  Serial.println("7 Segment Backpack Test");
//  matrix.begin(0x70);
  matrix.begin(0x71);
}  // setup
```

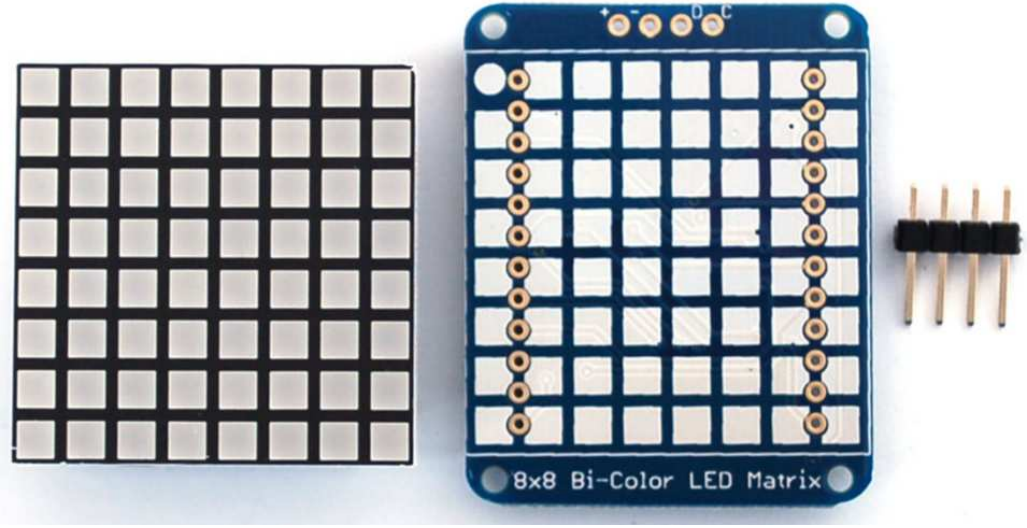
verschiedene Adressen möglich

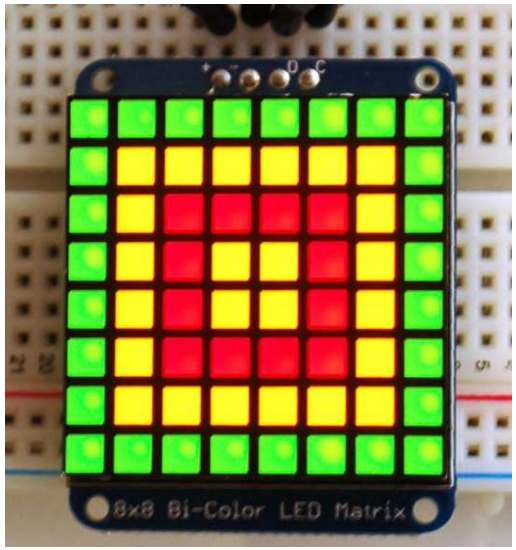
I2C_4x7Segement

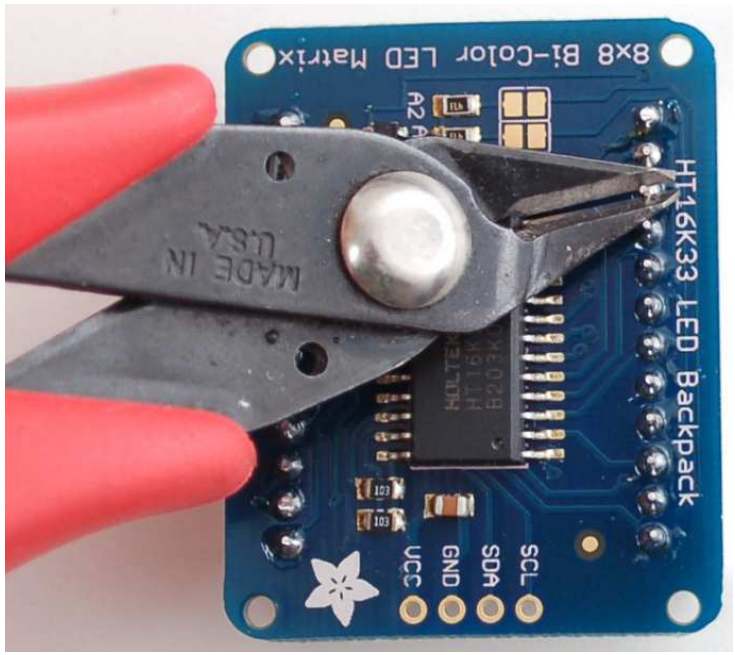
```
27  void loop()
28  {
29    // try to print a number thats too long
30    matrix.print(10000, DEC);
31    matrix.writeDisplay();
32    delay(500);
33
34    // print a hex number
35    matrix.print(0xBEEF, HEX);
36    matrix.writeDisplay();
37    delay(500);
38
39    // print a floating point
40    matrix.print(12.34);
41    matrix.writeDisplay();
42    delay(500);
43
44    // print with print/println
45    for (uint16_t counter = 0; counter < 9999; counter++)
46    {
47      matrix.println(counter);
48      matrix.writeDisplay();
49      delay(10);
50    }  // for
51    delay(1000);
```

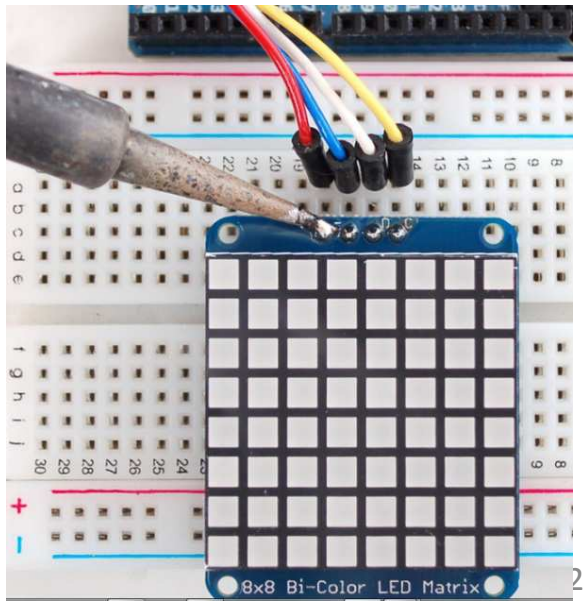
```
53    // method #2 - draw each digit
54    uint16_t blinkcounter = 0;
55    boolean drawDots = false;
56    for (uint16_t counter = 0; counter < 9999; counter ++)
57    {
58      matrix.writeDigitNum(0, (counter / 1000), drawDots);
59      matrix.writeDigitNum(1, (counter / 100) % 10, drawDots);
60      matrix.drawColon(drawDots);
61      matrix.writeDigitNum(3, (counter / 10) % 10, drawDots);
62      matrix.writeDigitNum(4, counter % 10, drawDots);
63
64      blinkcounter+=50;
65      if (blinkcounter < 500)
66      {
67        drawDots = false;
68      }  // if
69      else
70        if (blinkcounter < 1000)
71        {
72          drawDots = true;
73        }  // if
74        else
75        {
76          blinkcounter = 0;
77        }  // else
78      matrix.writeDisplay();
79      delay(10);
80    }  // for
81  }  // loop
```

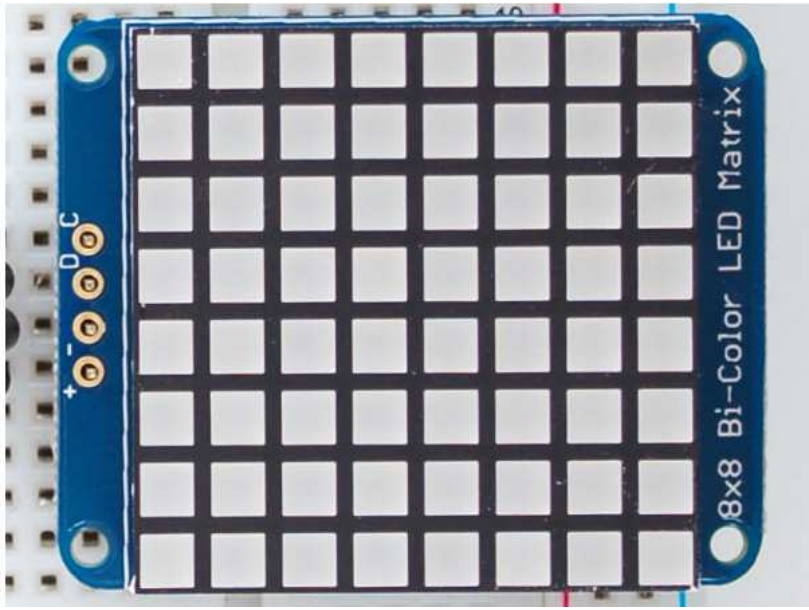
I²C-Bus: 8x8 Bi-Color Matrix Display am Beispiel einer BINÄR UHR





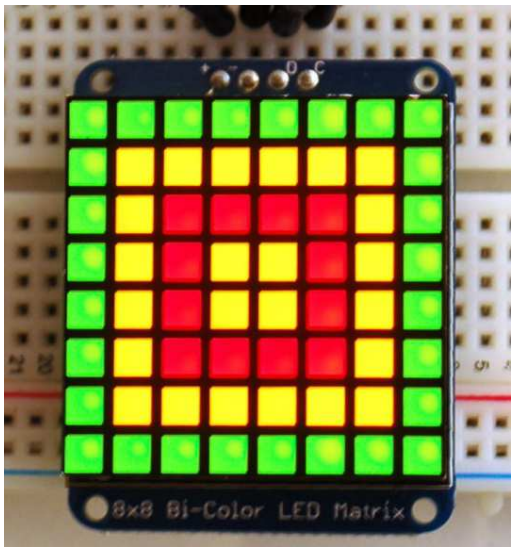


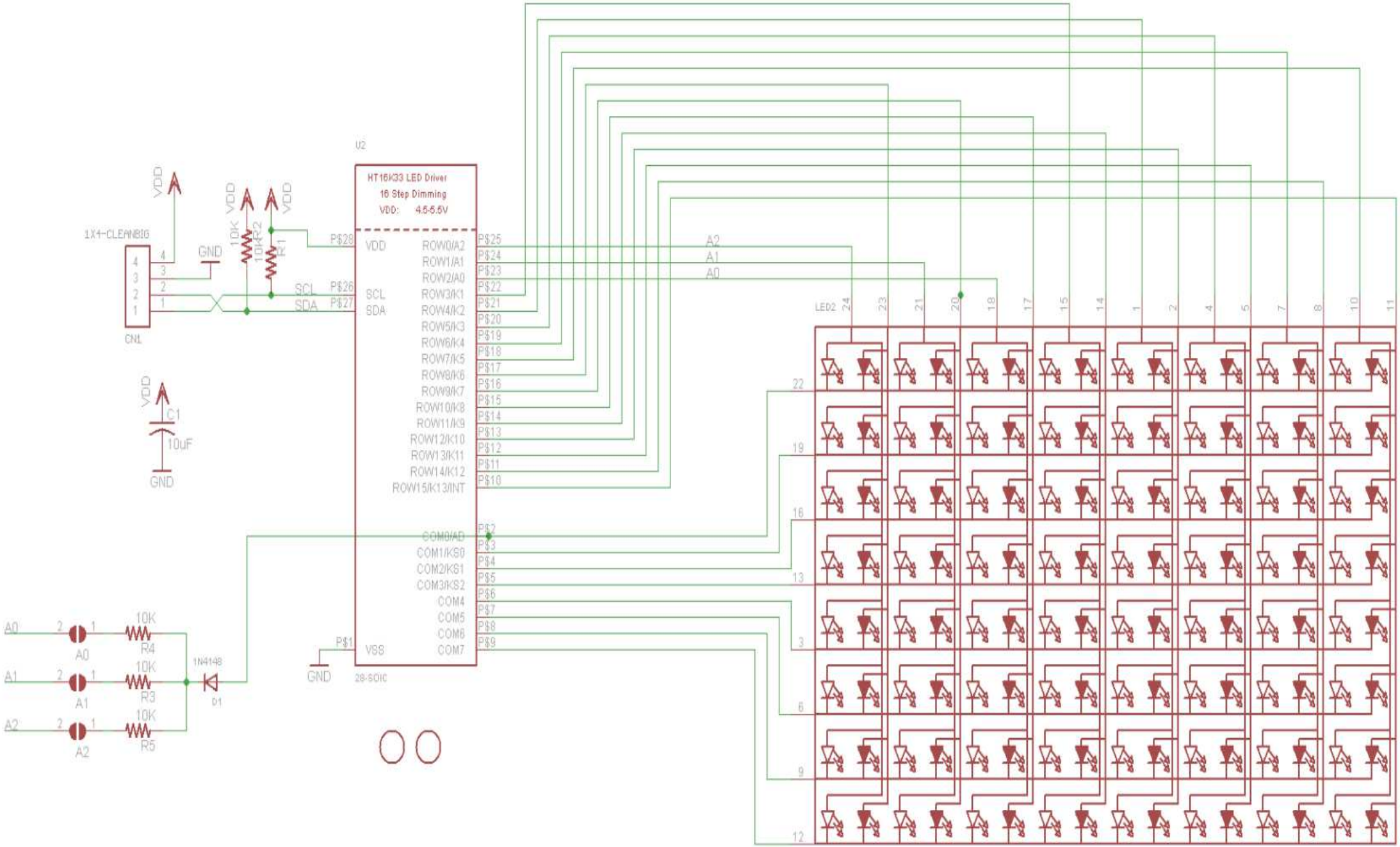




The default orientation for graphics commands on this display places pixel (0,0) at the top-left when the header is at the left and Adafruit logo at the right.

To use the matrix as shown below (header at top, logo at bottom), call **matrix.setRotation(3)** before issuing graphics commands.




U2
HT16K33 LED Driver
16 Step Dimming
VDD: 4.5-5.5V
VDD
SCL
SDA
VSS
28-SOIC
1X4-CLEANBIG
CN1
GND
10K
R1
R2
C1
10uF
LED2
A0
A1
A2
R3
R4
R5
1N4148
D1

I2C_binUhr

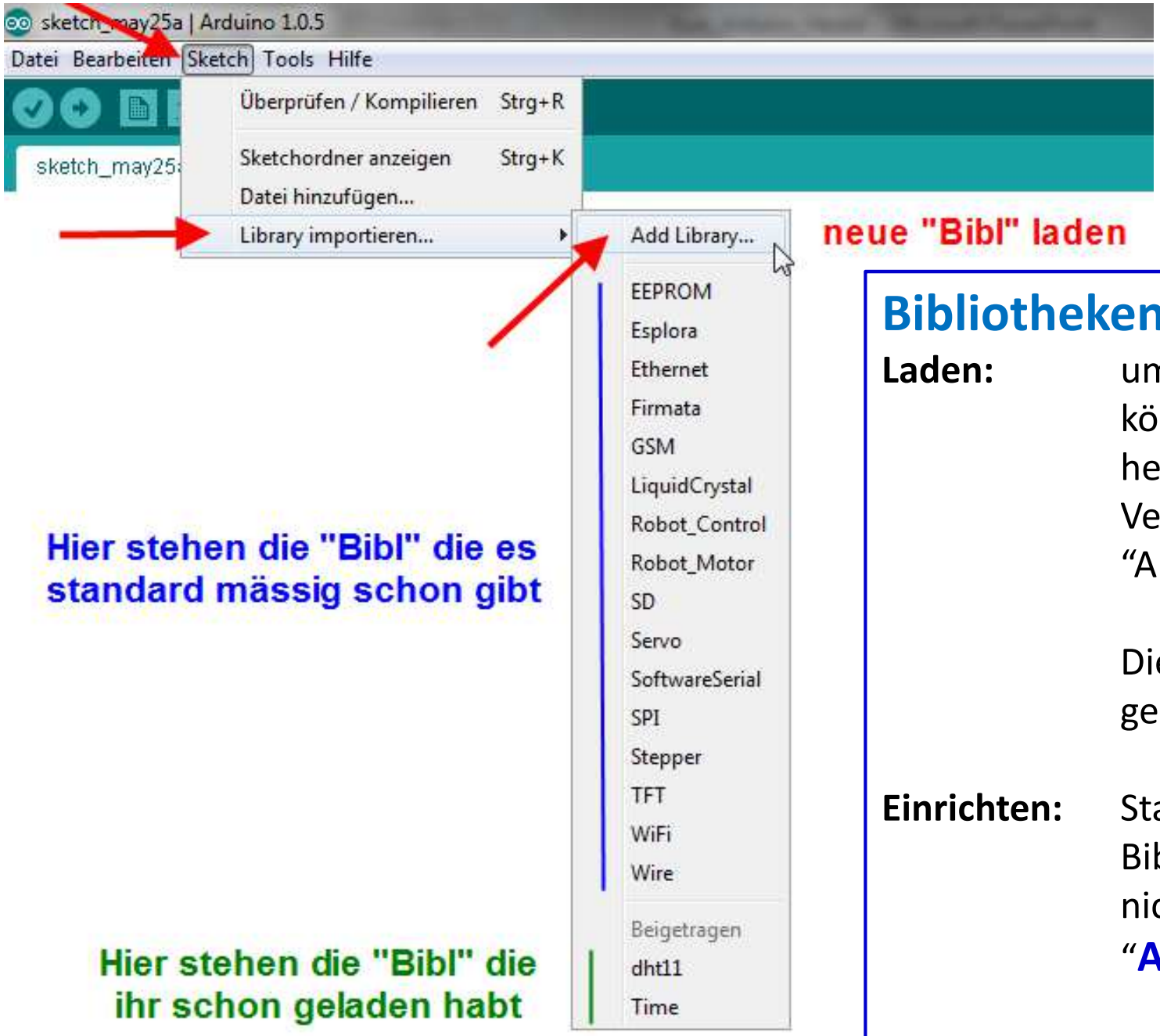
```
1  /*    This is a library for our I2C LED Backpacks
2   * Designed specifically to work with the Adafruit LED Matrix backpacks
3   * ----> http://www.adafruit.com/products/872
4   * ----> http://www.adafruit.com/products/871
5   * ----> http://www.adafruit.com/products/870
6   *
7   * These displays use I2C to communicate, 2 pins are required to
8   * interface. There are multiple selectable I2C addresses. For backpacks
9   * with 2 Address Select pins: 0x70, 0x71, 0x72 or 0x73. For backpacks
10  * with 3 Address Select pins: 0x70 thru 0x77     */
11
12 #include <Wire.h>
13 #include "Adafruit_LEDBackpack.h"
14 #include "Adafruit_GFX.h"
15
16 Adafruit_BicolorMatrix matrix = Adafruit_BicolorMatrix();
17 unsigned char hour = 12, minute = 37, second;
18 int setH, setM, setS;
19
20 void setup()
21 {
22   Serial.begin(9600);
23   Serial.println("8x8 LED Matrix Test");
24   matrix.begin(0x70);  // pass in the address
25 }  // setup
```

I2C_binUhr §

```
64  void loop()
65  {
66    matrix.clear();
67    for ( int i = 0; i <= 7; i++) {
68      if ( bitRead(hour / 10, i) ) {
69        matrix.drawPixel(0, 7 - i, LED_RED);
70      }
71    }
72    for ( int i = 0; i <= 7; i++) {
73      if ( bitRead(hour % 10, i) ) {
74        matrix.drawPixel(1, 7 - i, LED_RED);
75      }
76    }
77    for ( int i = 0; i <= 7; i++) {
78      if ( bitRead(minute / 10, i) ) {
79        matrix.drawPixel(3, 7 - i, LED_YELLOW);
80      }
81    }
82    for ( int i = 0; i <= 7; i++) {
83      if ( bitRead(minute % 10, i) ) {
84        matrix.drawPixel(4, 7 - i, LED_YELLOW);
85      }
86    }
87    for ( int i = 0; i <= 7; i++) {
88      if ( bitRead(second / 10, i) ) {
89        matrix.drawPixel(6, 7 - i, LED_GREEN);
90      }
91    }
92    for ( int i = 0; i <= 7; i++) {
93      if ( bitRead(second % 10, i) ) {
94        matrix.drawPixel(7, 7 - i, LED_GREEN);
95      }
96    }
97    matrix.writeDisplay();
```

```
98
99    second++;
100   if (second > 59) {
101     second = 0;
102     minute++;
103   }
104   if (minute > 59) {
105     minute = 0;
106     hour++;
107   }
108   if (hour > 23){
109     hour = 0;
110   }
111
112   setH = 255;
113   setM = 255;
114   setS = 255;
115
```

```
116     while( Serial.available() > 0) {
117       unsigned char code = Serial.read();
118       Serial.print("Char received: ");
119       Serial.println((String)code);
120       switch( code ){
121         case 'H':
122           setH = Serial.parseInt();
123           Serial.print("   H found: ");
124           Serial.println(setH);
125           break;
126         case 'M':
127           setM = Serial.parseInt();
128           Serial.print("   M found: ");
129           Serial.println(setM);
130           break;
131         case 'S':
132           setS= Serial.parseInt();
133           Serial.print("   S found: ");
134           Serial.println(setS);
135          break;
136         default:
137           Serial.println("   Wrong Char found!");
138           break;
139       }  // switch
140       if( Serial.read() == '\n') {
141         if( setH != 255 ) hour = setH;
142         if( setM != 255 ) minute = setM;
143         if( setS != 255 ) second = setS;
144       }  // if
145     }  // while
146     delay(1000);
147   }  // loop
```



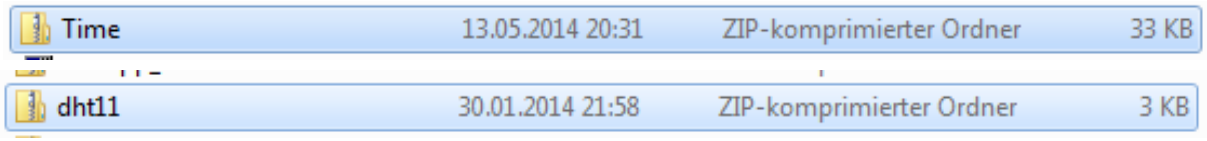
Bibliotheken laden und benutzen (I)

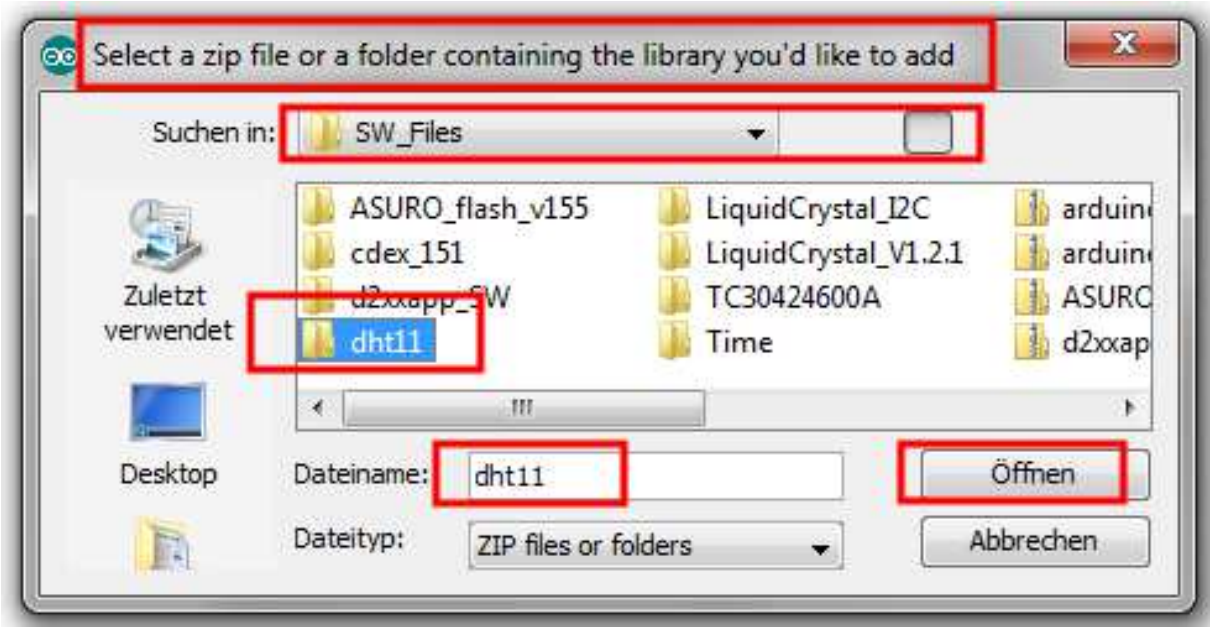
Laden: um eine Bibliothek verwenden zu können muss man diese im Internet herunterladen und speichern, das Verzeichnis ist egal, am besten "Arduino-Bibl" nennen.

Die Dateien im Internet sind meist gepackte "ZIP" Dateien.

Einrichten: Standardmässig hat Arduino schon Bibliotheken. Gibt es die "Bib" noch nicht dann so laden: "**Add Library...**"

Bsp.:





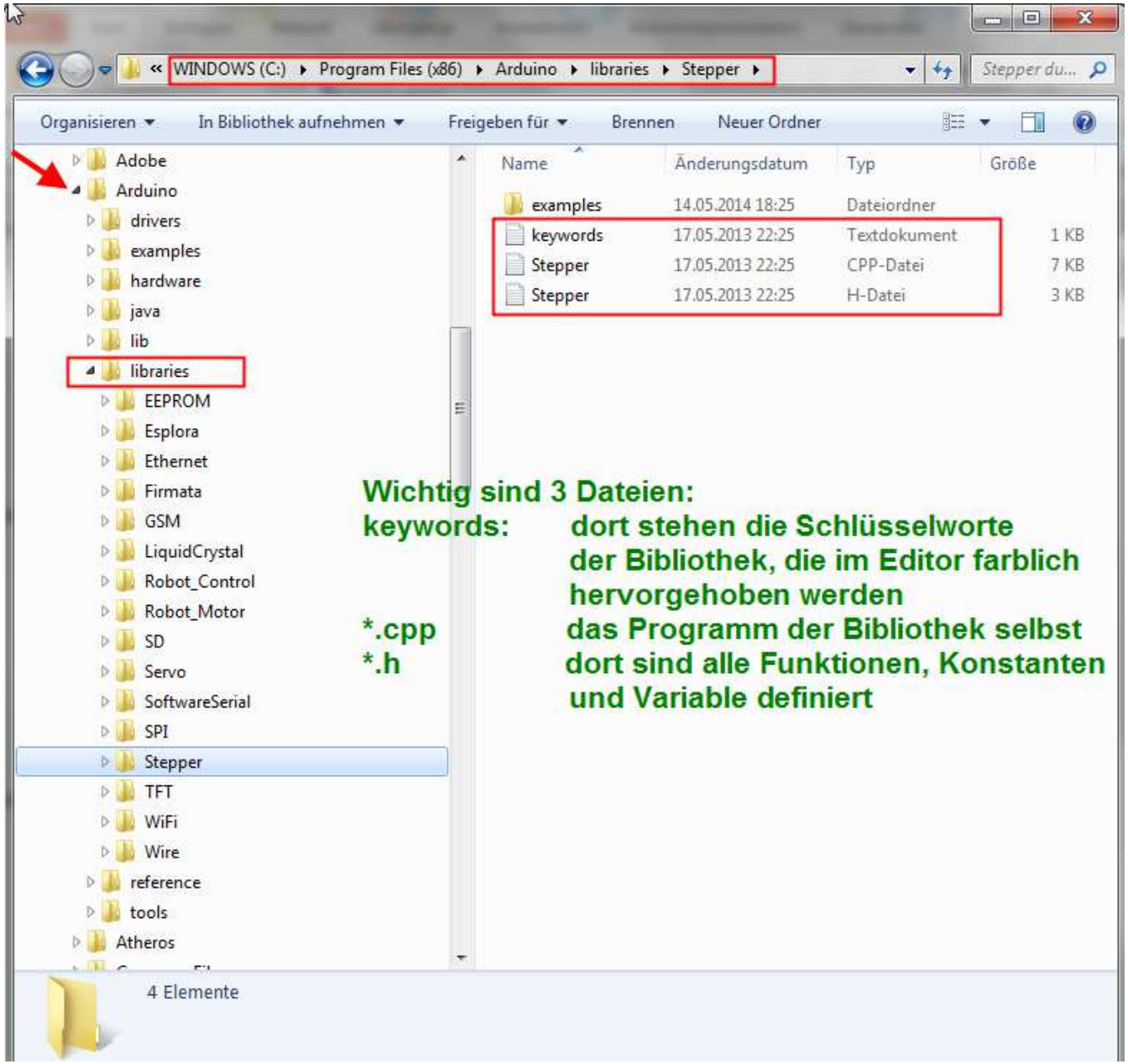
Bibliotheken laden und benutzen (II)

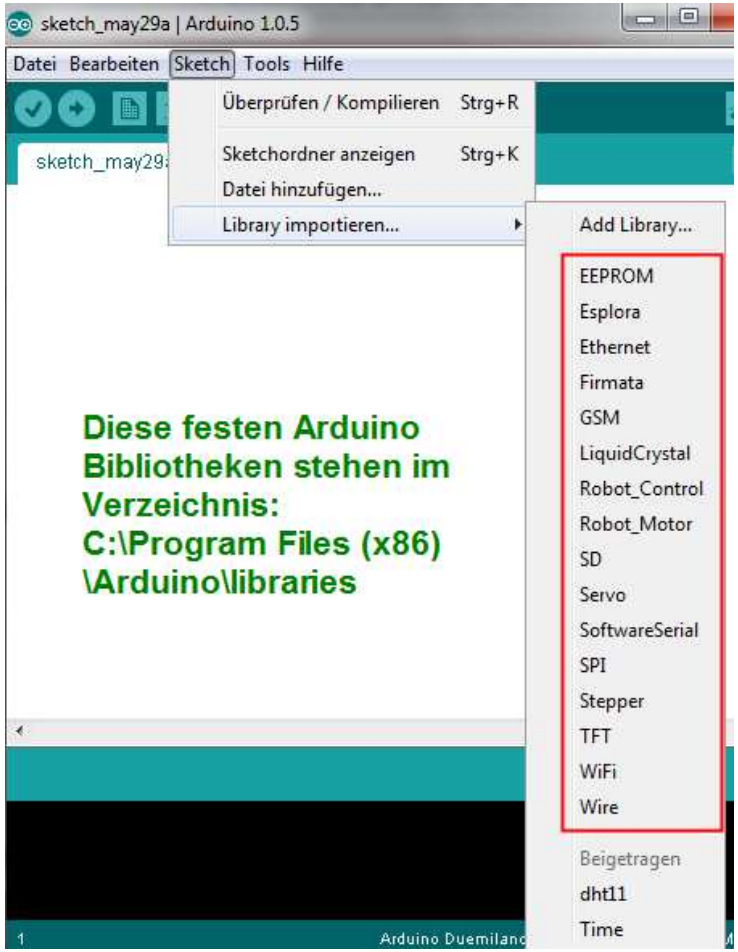
Öffnen: erst müsst ihr den Ort suchen wo eure (Internet) Bibliothek gespeichert ist, dann “Öffnen” klicken!

Achtung:

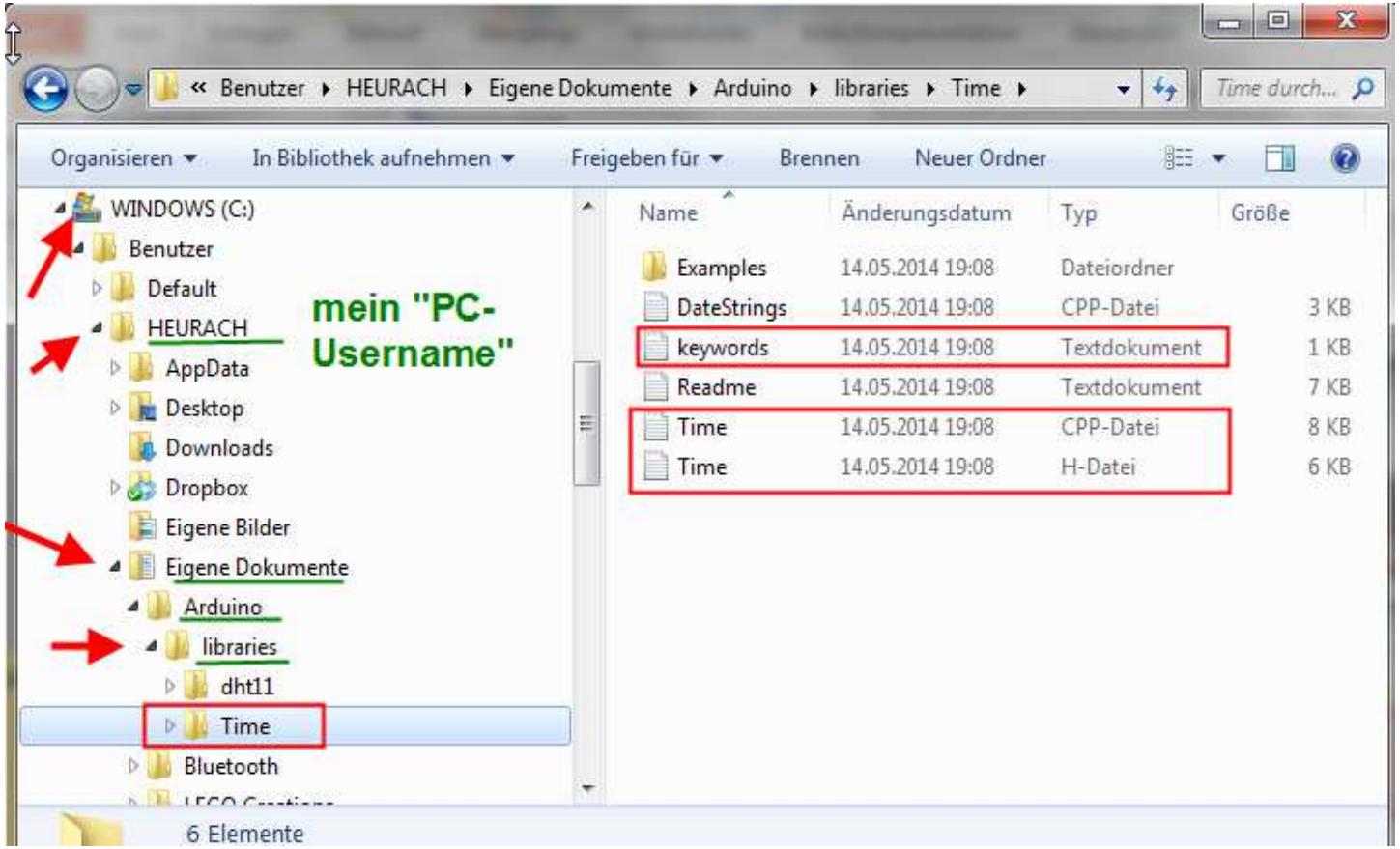
Arduino entpackt die “ZIP” Dateien normalerweise automatisch und alles funktioniert.
Es kann aber sein, dass das “ZIP file” aus dem Internet mehr enthält als nötig oder in einem falschen Format. Dann gibt es Probleme. Deshalb besser das ZIP File erst in einen Ordner auspacken und dann nur den “ORDNER” per Arduino installieren (“Öffnen”) den man haben möchte.

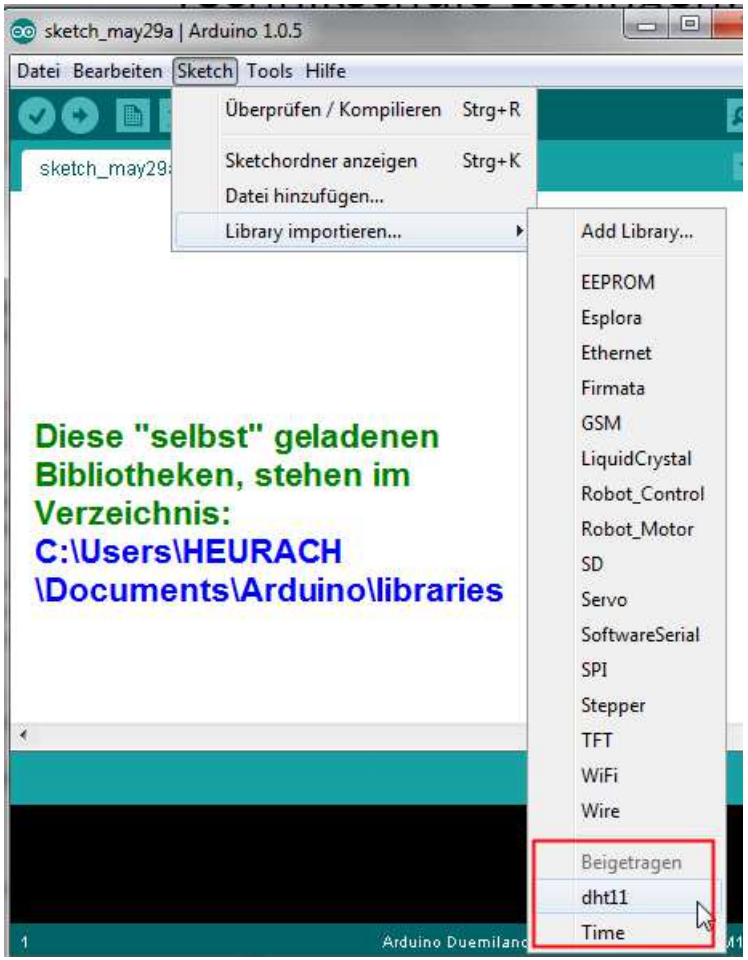
Bibliotheken laden und benutzen (Details)





Bibliotheken laden und benutzen (Details II)





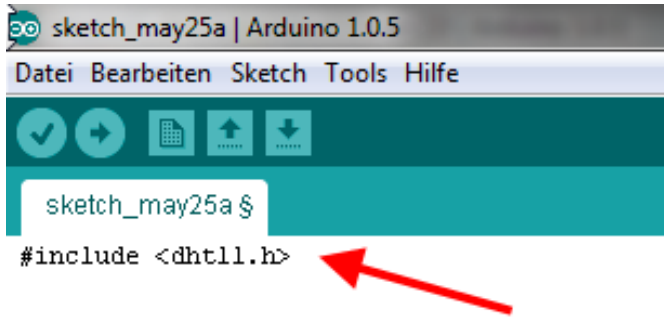
Achtung:

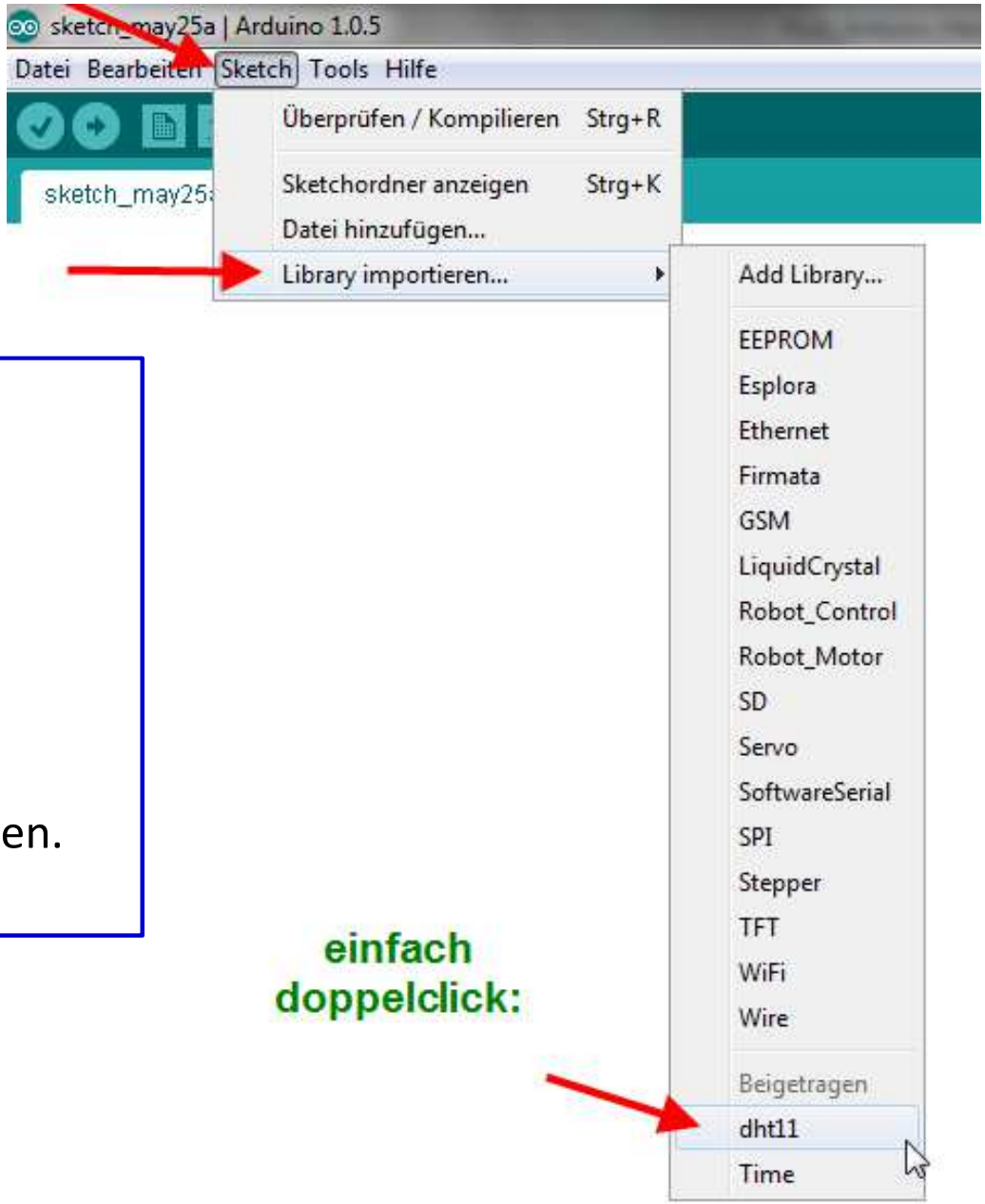
Die “*.h” und “*.cpp” Datei müssen den gleichen Namen haben und im gleichen Verzeichnis stehen. Und diese beiden Dateien müssen zueinander passen → also aus dem gleichen ZIP File stammen!

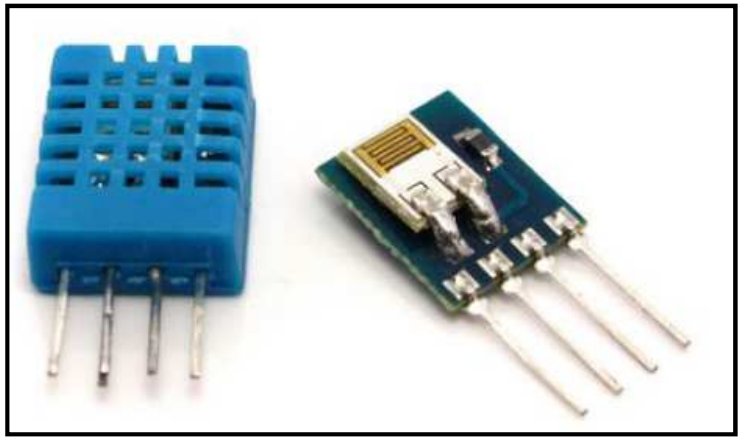
Bibliotheken laden und benutzen (III)

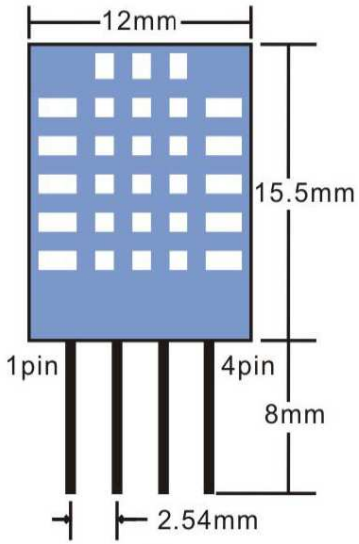
Benutzen: um eine Bibliothek im Programm zu nutzen, einfach diese anclicken:

Im Programm: muss eine Zeile wie z.B. diese auftauchen.
Oder Zeile von Hand eingeben









Digitaler Feuchte & Temperatur-Sensor

Single-bus (1 Wire) for communication between MCU & DHT11

Technical Specification:

Model:	DHT11
Power supply:	3-5.5V DC
Output signal:	digital signal via single-bus
Sensing element:	Polymer resistor
Measuring range	
Humidity:	20-90%RH;
Temperature:	0-50 Celsius
Accuracy	
humidity:	+-4%RH (Max +-5%RH);
Temperature:	+-2.0Celsius
Resolution or sensitivity	
humidity:	1%RH
Temperature:	0.1Celsius
Repeatability	
humidity:	+-1%RH
temperature	+-1Celsius
Humidity hysteresis:	+-1%RH
Long-term Stability:	+-0.5%RH/year
Sensing period Average:	2s
Dimensions size:	12*15.5*5.5mm

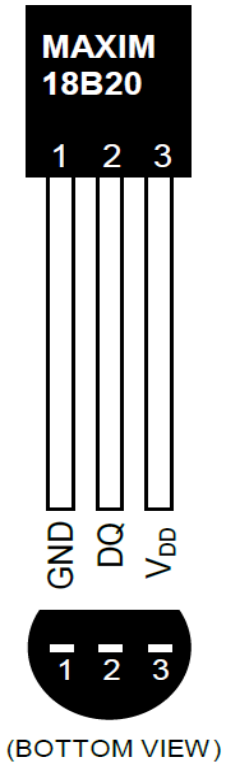
dht11

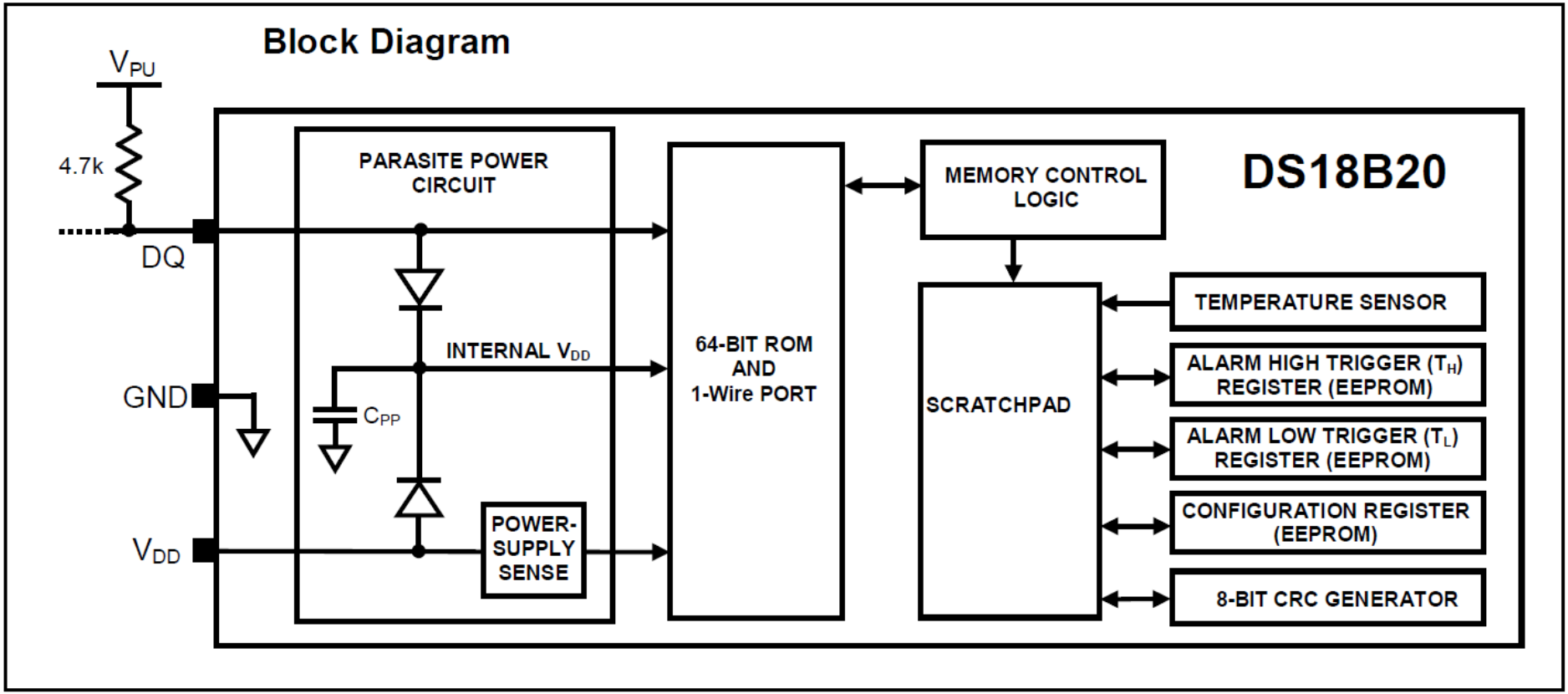
```
/* Programm steuert einen DHT11 Sensor an. Dieser misst Luft-Temperatur und -Feuchte
Dazu wird die Bibliothek dht11.h verwendet. Die Kommunikation mit dem Sensor findet seriell
über nur einen Pin statt  */

#include <dht11.h>                          // Bibliothek einbinden

dht11 DHT;                                  // Object DHT erzeugen
#define DHT11_PIN 12                        // Pin für serielle Kommunikation

void setup()
{
  Serial.begin(9600);                       // Seriellen Monitor öffnen
  Serial.println("DHT TEST PROGRAM ");
  Serial.print("LIBRARY VERSION: ");
  Serial.println(DHT11LIB_VERSION);
  Serial.println();
  Serial.println("Type,\tstatus,\tHumidity (%),\tTemperature (C)");  // Überschrift ausgeben
}  // fct "setup"

void loop()
{
  int chk;                                  // empfängt Zeichen vom Sensor
  Serial.print("DHT11, \t");
  chk = DHT.read(DHT11_PIN);                // Read Sensor Data
  switch (chk)                              // erstes Zeichen ist Sensor Status
  {
    case DHTLIB_OK:                         // Sensor ist OK
                Serial.print("OK,\t");
                break;
    case DHTLIB_ERROR_CHECKSUM:             // Übertragungsfehler
                Serial.print("Checksum error,\t");
                break;
    case DHTLIB_ERROR_TIMEOUT:              // Time Out Error
                Serial.print("Time out error,\t");
                break;
    default:                                // unbekannter Fehler
                Serial.print("Unknown error,\t");
                break;
  }  // switch
  Serial.print(DHT.humidity,1);             // Feuchte lesen mit eine Kommastelle und anzeigen
  Serial.print(",\t\t");                    // TAB ausgeben
  Serial.println(DHT.temperature,1);        // Temp lesen mit eine Kommastelle und anzeigen
  delay(3000);                              // alle 3 Sekunden anzeigen
}   // fct "loop"
```

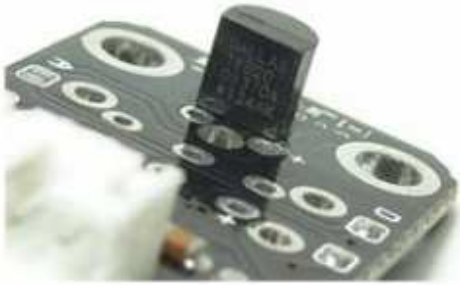
dht_xx

```
// Example testing sketch for various DHT humidity/temperature sensors

#include "DHT.h"                    // Library
#define DHTPIN 2                    // what pin DHT is connected to

// Uncomment whatever type you're using!
// #define DHTTYPE DHT11            // DHT 11
#define DHTTYPE DHT22               // DHT 22  (AM2302)
// #define DHTTYPE DHT21            // DHT 21 (AM2301)

// Connect pin 1 (on the left) of the sensor to +5V
// Connect pin 2 of the sensor to whatever your DHTPIN is
// Connect pin 4 (on the right) of the sensor to GROUND
// Connect a 4 to 10K resistor from pin 2 (data) to pin 1 (power) of the dht

DHT dht(DHTPIN, DHTTYPE);           // DHT Object initialisieren

void setup() {
  Serial.begin(9600);
  Serial.println("DHTxx test!");
  dht.begin();                      // Object bzw. Sensor initialisieren
}

void loop() {
  float h = dht.readHumidity();          // Reading temperature
  float t = dht.readTemperature();       // or humidity takes about 250ms to 2s
  if (isnan(t) || isnan(h))
  {            // check if h, t are valid, if NaN (not a number) then something wrong!
    Serial.println("Failed to read from DHT");
  }  // if
  else
  {
    Serial.print("Humidity: ");
    Serial.print(h);
    Serial.print(" %\t");
    Serial.print("Temperature: ");
    Serial.print(t);
    Serial.println(" *C");
  }  // else
}  // loop
```

Temperatur-Sensor **18B20**: -55 bis +125 °C, Resolution 0,5 °C, 9-12 bit **1Wire serial output**.

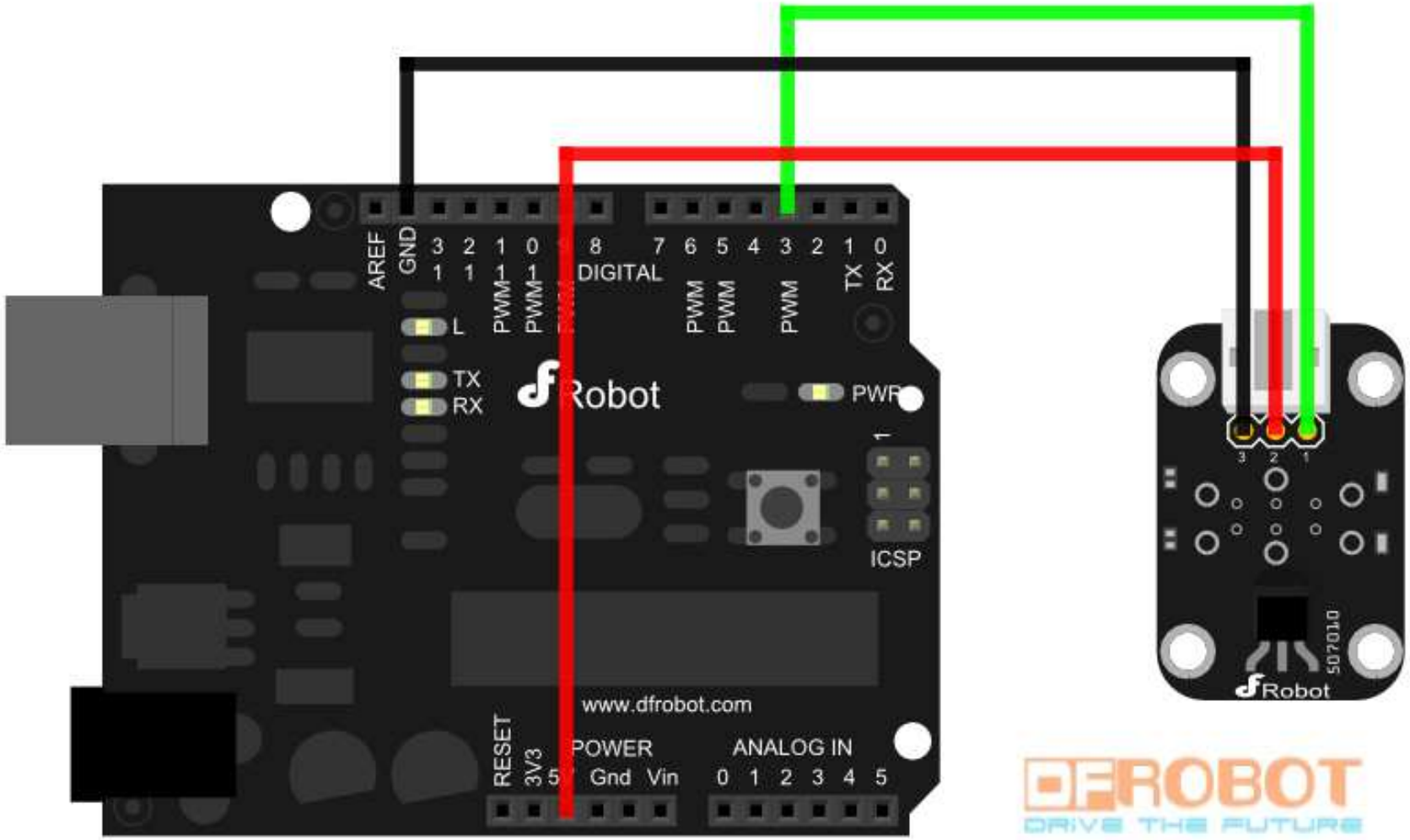




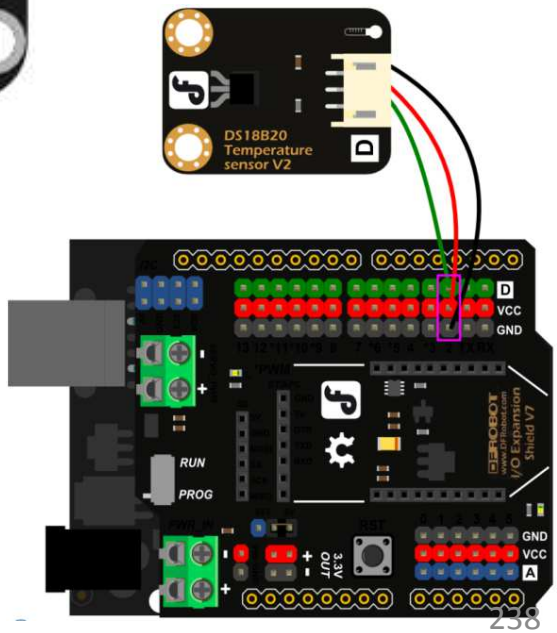


Specification

Voltage:	+5V
Interface:	Digital
Pin definition:	1: Output 2: Power 3: GND
Weight: 5g	


AREF
GND
3 2 1 0
1 1 1 1
PWM PWM
8 7 6 5 4 3 2 1 0
DIGITAL
PWM PWM PWM
TX RX
L
TX
RX
Robot
PWR
ICSP
www.dfrobot.com
RESET
3V3
5
POWER
Gnd Vin
ANALOG IN
0 1 2 3 4 5
Robot
DFROBOT
DRIVE THE FUTURE

DS18B20
Temperature
Sensor V2


DS18B20
Temperature
sensor V2
D
VCC
GND
RUN
PROG
GND
VCC
A

OneWire myWire(pin)

Create the OneWire object, using a specific pin. Even though you can connect many 1 wire devices to the same pin, if you have a large number, smaller groups each on their own pin can help isolate wiring problems. You can create multiple OneWire objects, one for each pin.

myWire.search(addrArray)

Search for the next device. The addrArray is an 8 byte array. If a device is found, addrArray is filled with the device's address and true is returned. If no more devices are found, false is returned.

myWire.reset_search()

Begin a new search. The next use of search will begin at the first device.

myWire.reset()

Reset the 1-wire bus. Usually this is needed before communicating with any device.

myWire.select(addrArray)

Select a device based on its address. After a reset, this is needed to choose which device you will use, and then all communication will be with that device, until another reset.

myWire.skip()

Skip the device selection. This only works if you have a single device, but you can avoid searching and use this to immediatly access your device.

myWire.write(num);

Write a byte.

myWire.write(num, 1);

Write a byte, and leave power applied to the 1 wire bus.

myWire.read()

Read a byte.

myWire.crc8(dataArray, length)

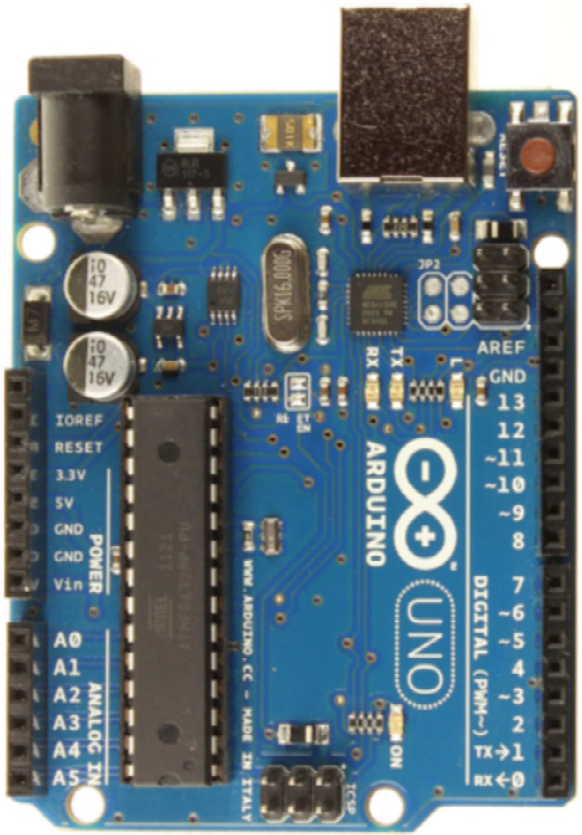
Compute a CRC check on an array of data.

Die “**OneWire.h**” Library

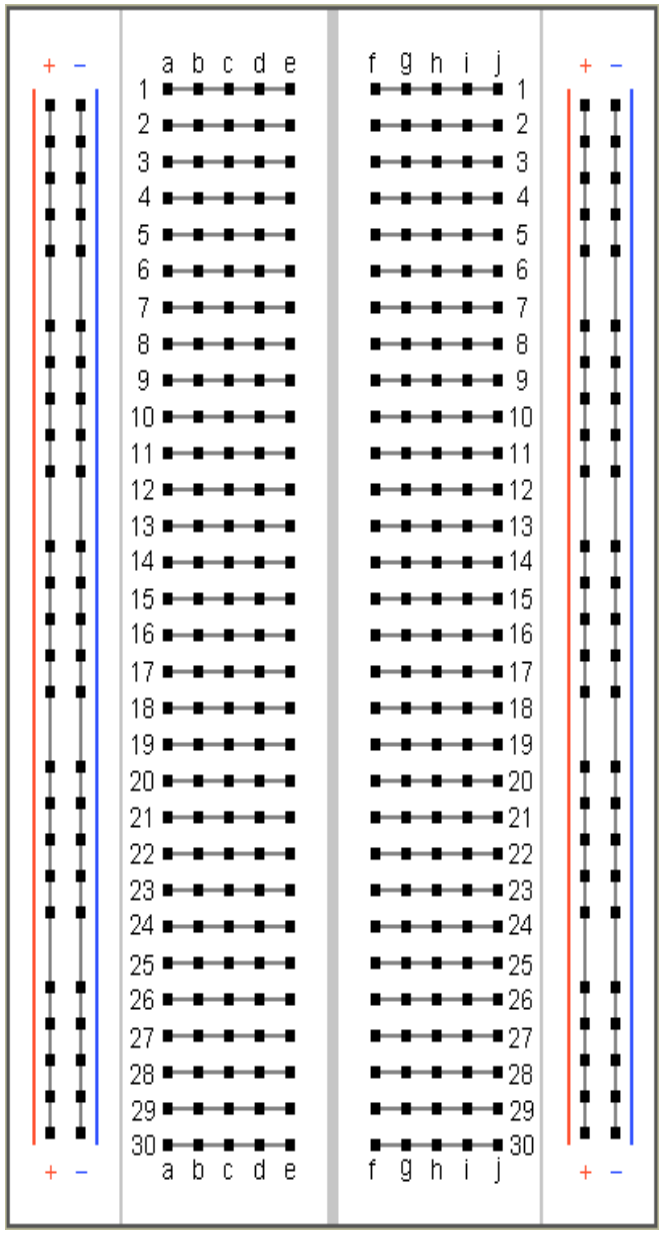
TempSensor_18B20_OneWire

```
#include <OneWire.h>

int DS18S20_Pin = 3;                    //DS18S20 Signal pin on digital 2

OneWire ds(DS18S20_Pin);                // Temperature chip i/o on digital pin 2

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

void loop(void) {
  float temperature = getTemp();
  Serial.println(temperature);
  delay(1000);                          // Delay für Anzeige, 1 mal pro Sekunde
}
```

```
float getTemp(){                        // Fct. returns the temperature from DS18S20 in DEG Celsius

  byte data[12];
  byte addr[8];

  if ( !ds.search(addr)) {              //no more sensors on chain, reset search
      ds.reset_search();
      return (-1000);
  }
  if ( OneWire::crc8( addr, 7) != addr[7]) {
      Serial.println("CRC is not valid!");
      return (-1000);
  }
  if ( addr[0] != 0x10 && addr[0] != 0x28) {
      Serial.print("Device is not recognized");
      return -1000;
  }
  ds.reset();
  ds.select(addr);
  ds.write(0x44,1);                     // start conversion, with parasite power on at the end

  byte present = ds.reset();
  ds.select(addr);
  ds.write(0xBE);                       // Read Scratchpad
  for (int i = 0; i < 9; i++){          // we need 9 bytes
    data[i] = ds.read();
  }  // for
  ds.reset_search();
  byte MSB = data[1];
  byte LSB = data[0];
  float tempRead = ((MSB << 8) | LSB);        //using two's compliment
  float TemperatureSum = tempRead / 16;
  return (TemperatureSum);
}  // fct
```



Arduino Uno R3 Front



SPI = Serial Peripheral Interface

MOSI (DI) → pin 11; MISO (DO) → pin 12; CLK → pin 13; CS → pin 4 oder 10; G → GND; + → 5V

SD class

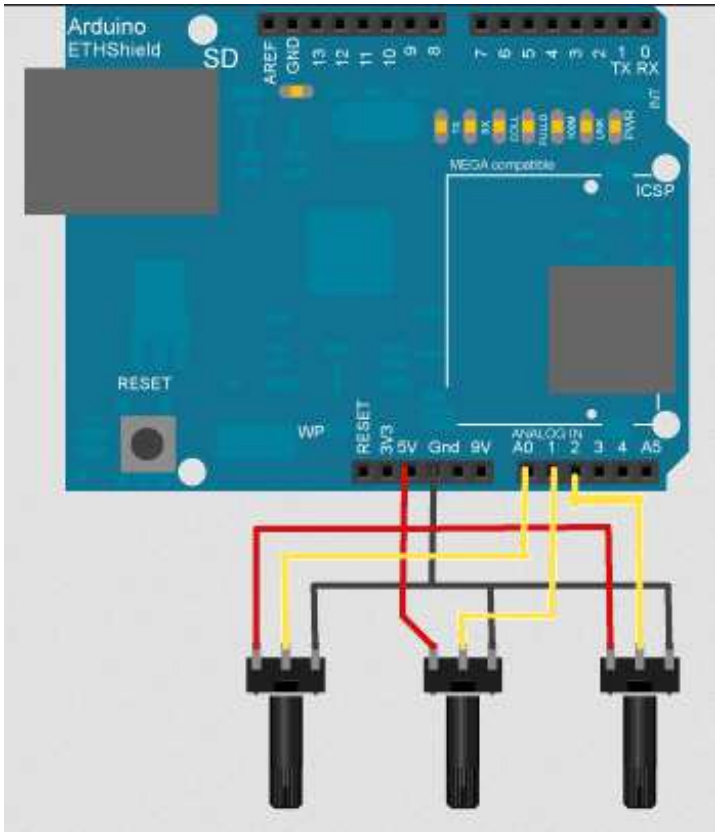
The SD class provides functions for accessing the SD card and manipulating its files and directories.

begin()
exists()
mkdir()
open()
remove()
rmdir()

File class

The File class allows for reading from and writing to individual files on the SD card.

available()
close()
flush()
peek()
position()
print()
println()
seek()
size()
read()
write()
isDirectory()
openNextFile()
rewindDirectory()



begin()

Description

Initializes the SD library and card. This use the SPI bus (digital pins 11, 12, and 13 on Arduino boards; 50, 51, and 52 on the Mega) and the chip select pin, which defaults to the hardware SS pin (pin 10 on Arduino boards, 53 on the Mega). Note that even if you use a different chip select pin, **the hardware SS pin must be kept as an output** or the SD library functions will not work.

Syntax

SD.begin()
SD.begin(cspin)

Parameters

cspin (*optional*): the pin connected to the chip select line of the SD card; defaults to the hardware SS line of the SPI bus

Returns

true on success; false on failure

exists()

Description

Tests whether a file or directory exists on the SD card.

Syntax

SD.exists(filename)

Parameters

filename: the name of the file to test for existence,
which can include directories (delimited by forward-slashes, /)

Returns

true if the file or directory exists, **false** if not

mkdir()

Description

Create a directory on the SD card. This will also create any intermediate directories that don't already exists; e.g. SD.mkdir("a/b/c") will create a, b, and c.

Syntax

SD.mkdir(filename)

Parameters

filename: name of directory to create, with sub-directories separated by forward-slashes, /

Returns

true if the creation of the directory succeeded, false if not

open()

Description
Opens a file on the SD card. If the file is opened for writing, it will be created if it doesn't already exist (but the directory containing it must already exist).

Syntax
SD.open(filepath)
SD.open(filepath, mode)

Parameters
filename: the name the file to open, which can include directories (delimited by forward slashes, /) - *char **
mode (*optional*): the mode in which to open the file, defaults to FILE_READ - *byte*. one of:
FILE_READ: open the file for reading, starting at the beginning of the file.
FILE_WRITE: open the file for reading and writing, starting at the end of the file.

Returns
a **File object** referring to the opened file; if the file couldn't be opened, this object will evaluate to false in a boolean context, i.e. you can test the return value with "if (f)".

remove()

Description

Remove a file from the SD card.

Syntax

SD.remove(filename)

Parameters

filename: the name of the file to remove, which can include directories (delimited by forward-slashes, /)

Returns

true if the removal of the file succeeded, false if not. (if the file didn't exist, the return value is unspecified)

```
if (SD.remove(myFileName) )
{
  Serial.print(F("Deleting File:  "));
  Serial.print(myFileName);
  Serial.println(F("   erfolgreich"));
}  // if
else
{
  Serial.print(F("Deleting File:  "));
  Serial.print(myFileName);
  Serial.println(F("   nicht erfolgreich"));
}  // else
```

available()

Description

Check if there are any bytes available for reading from the file.

Syntax

file.available() *file*: an instance of the File class (returned by SD.open())

Returns

the number of bytes available (*int*)

close()

Description

Close the file, and ensure that any data written to it is physically saved to the SD card.

Syntax

file.close() *file*: an instance of the File class (returned by SD.open())

Returns

none

flush()

Description

Ensures that any bytes written to the file are physically saved to the SD card. This is done automatically when the file is closed.

Syntax

file.flush() ***file***: an instance of the File class (returned by SD.open())

Returns

none

peek()

Description

Read a byte from the file without advancing to the next one. That is, successive calls to peek() will return the same value, as will the next call to read().

Syntax

file.peek() ***file***: an instance of the File class (returned by SD.open())

Returns

The next byte (or character), or -1 if none is available.

position()

Description

Get the current position within the file (i.e. the location to which the next byte will be read from or written to).

Syntax

file.position()

file: an instance of the File class (returned by SD.open())

Returns

the position within the file (*unsigned long*)

rmdir()

Description

Remove a directory from the SD card. The directory must be empty.

Syntax

SD.rmdir(filename)

Parameters

filename: name of directory to remove, with sub-directories separated by forward-slashes, /

Returns

true if the removal of the directory succeeded, false if not.

(if the directory didn't exist, the return value is unspecified)

print()

Description
Print data to the file, which must have been opened for writing. Prints numbers as a sequence of digits, each an ASCII character (e.g. the number 123 is sent as the three characters '1', '2', '3').

Syntax
file.print(data)
file.print(data, BASE)

Parameters
data: the data to print (char, byte, int, long, or string)
BASE (optional): the base in which to print numbers: BIN for binary (base 2), DEC for decimal (base 10), OCT for octal (base 8), HEX for hexadecimal (base 16).

Returns
byte
print() will return the number of bytes written, though reading that number is optional

println()

Description

Print data, followed by a carriage return and newline, to the File, which must have been opened for writing. Prints numbers as a sequence of digits, each an ASCII character (e.g. the number 123 is sent as the three characters '1', '2', '3').

Syntax

file.println()
file.println(data)
file.print(data, BASE)

Parameters

data (optional): the data to print (char, byte, int, long, or string)
BASE (optional): the base in which to print numbers: BIN for binary (base 2), DEC for decimal (base 10), OCT for octal (base 8), HEX for hexadecimal (base 16).

Returns

byte
println() will return the number of bytes written, though reading that number is optional

seek()

Description

Seek to a new position in the file, which must be between 0 and the size of the file (inclusive).

Syntax

file.seek(pos)

Parameters

pos: the position to which to seek (*unsigned long*)

Returns

true for success, false for failure (*boolean*)

size()

Description

Get the size of the file.

Syntax

file.size()

Returns

the size of the file in bytes (*unsigned long*)

write()

Description
Write data to the file.

Syntax
file.write(data)
file.write(buf, len)

Parameters
data: the byte, char, or string (char *) to write
buf: an array of characters or bytes
len: the number of elements in buf

Returns
byte
write() will return the number of bytes written, → reading that number is optional

read()

Description

Read a byte from the file.

Syntax

file.read()

Returns

The next byte (or character), or -1 if none is available.

isDirectory()

Description

Directories (or folders) are special kinds of files, this function reports if the current file is a directory or not.

Syntax

file.isDirectory()

Returns

boolean

openNextFile()

Description

Reports the next file or folder in a directory.

Syntax

file.openNextFile()

Returns

char : the next file or folder in the path

rewindDirectory()

Description

rewindDirectory() will bring you back to the first file in the directory, used in conjunction with openNextFile().

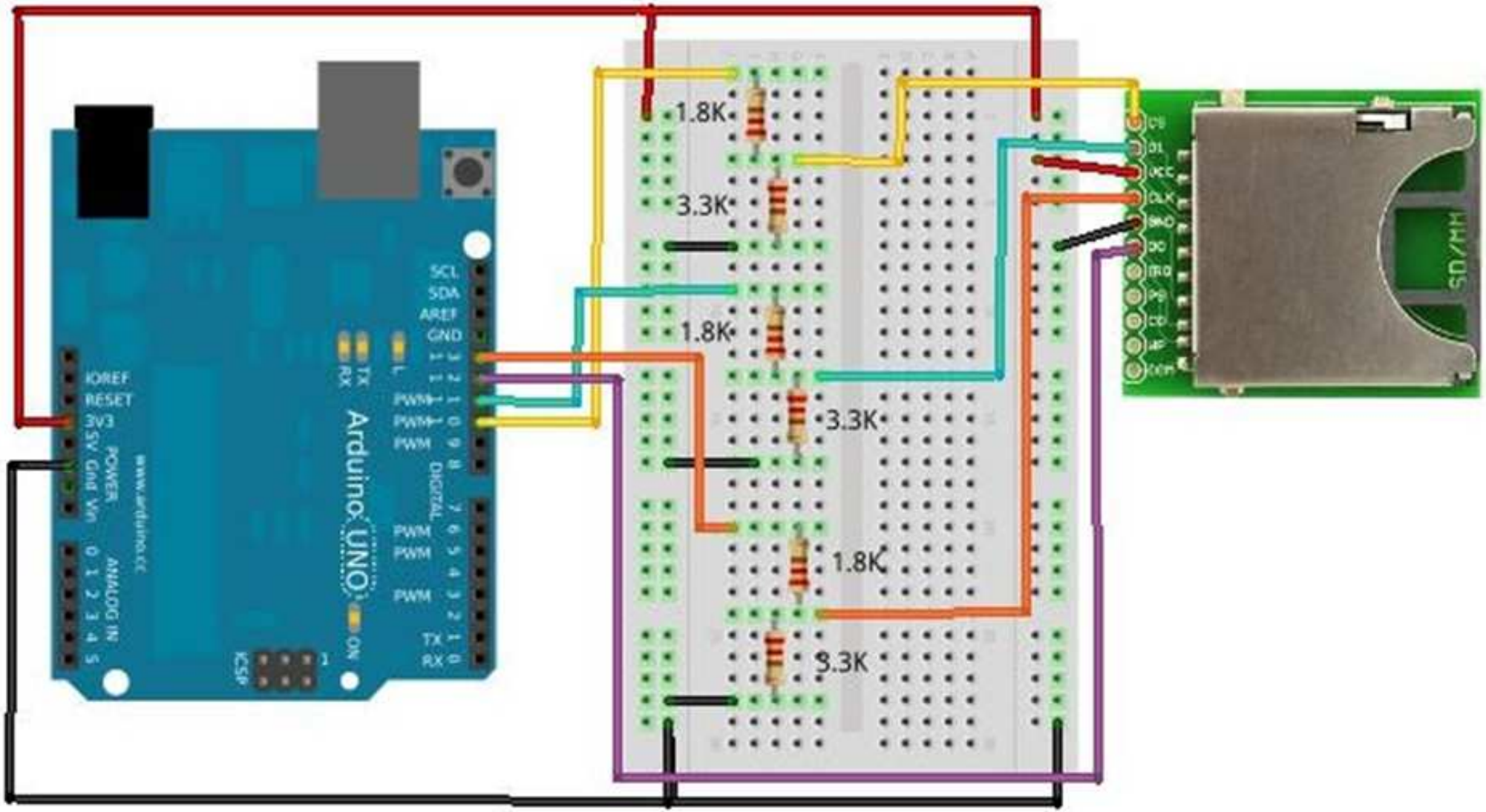
Syntax

file.rewindDirectory()

Returns

None

Alternative:
http://garagelab.com/profiles/blogs/tutorial-how-to-use-sd-card-with-arduino
http://code.google.com/p/sdfatlib/



Das Windows File System:

Verzeichnis → Unterverzeichnis → Unterverzeichnis **Baumstruktur**

Datei, Datei-Endung (Typ), Datum, Größe, ….

Computer ▸ (C:) System Volume ▸ daten ▸ PreForms ▸ Search PreForms

File Edit View Tools Help

Organize ▾ Include in library ▾ Share with ▾ New folder

- (C:) System Volume
  - ae-doors
    - addins
  - CFNM
  - daten
    - eforms
    - IEAddOns
    - MobilityOfflineDocs
    - PreForms
      - @
      - BU
      - content
      - Info
      - Startup
    - sap
    - users
      - DE_erh5lr
        - local settings
          - application data
            - microsoft
              - outlook
      - DE_gsh9ebh
      - DE_haz1lr
      - erh5lr
  - ibox.dir

Name	Date modified	Type	Size
@	24.03.2014 06:58	File folder	
BU	24.03.2014 06:55	File folder	
content	24.03.2014 06:58	File folder	
Info	24.03.2014 06:55	File folder	
Startup	06.05.2014 06:33	File folder	
_version.ini	12.05.2014 18:34	Configuration settings	1 KB
30x190_Ordnerruecken_Einschub.dot	17.09.2010 14:31	Microsoft Office Word...	407 KB
38x192_Ordnerruecken_selbstklebend.dot	17.09.2010 14:31	Microsoft Office Word...	403 KB
54x190_Ordnerruecken_Einschub.dot	17.09.2010 14:30	Microsoft Office Word...	402 KB
61x192_Odnerruecken_selbstklebend.dot	17.09.2010 14:31	Microsoft Office Word...	401 KB
15493_NamensschilderMultiTypoDe.dot	28.09.2005 10:31	Microsoft Office Word...	36 KB
15494_NameMultiTypoDe.dot	14.09.2006 12:19	Microsoft Office Word...	54 KB
15495_TischaufstellerNameDesk.dot	22.05.2012 12:32	Microsoft Office Word...	29 KB
15496_Tischnamensschilder.dot	08.06.2006 12:59	Microsoft Office Word...	32 KB
15496_Tischnamensschilder_Buerotisch.d...	29.08.2008 10:35	Microsoft Office Word...	31 KB
15769_VerschlussSache.dot	28.09.2005 10:32	Microsoft Office Word...	41 KB
15786_Ordnerruecken.dot	14.08.2006 13:14	Microsoft Office Word...	25 KB
15787_Ordnerruecken.dot	14.08.2006 13:14	Microsoft Office Word...	26 KB
15788_Ordnerruecken.dot	14.08.2006 13:14	Microsoft Office Word...	27 KB
15789_Ordnerruecken.dot	14.08.2006 13:14	Microsoft Office Word...	28 KB
15790_Ordnerruecken.dot	14.08.2006 13:14	Microsoft Office Word...	27 KB
15791_Ordnerruecken.dot	14.08.2006 13:14	Microsoft Office Word...	27 KB
Abgabeantrag.bmp	04.09.2006 10:22	Bitmap image	10 KB
Abgabeantrag.dot	18.12.2013 15:11	Microsoft Office Word...	168 KB

Das DOS File System:

Verzeichnis → Unterverzeichnis → Unterverzeichnis Baumstruktur

Datei, Datei-Endung (Typ), Datum, Größe, ….

```
Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

U:\>c:

C:\>_
```

```
Command Prompt
U:\>c:

C:\>Dir
 Volume in drive C is System Volume
 Volume Serial Number is 42A1-A2BE

 Directory of C:\

06.06.2013  16:47    <DIR>          ae-doors
24.05.2013  13:22    <DIR>          CFNM
24.03.2014  07:54    <DIR>          daten
24.05.2013  14:33             9.122 fraglist.luar
06.06.2013  16:41    <DIR>          ibox.dir
23.05.2013  23:35    <DIR>          Intel
01.12.2006  23:37           904.704 msdia80.dll
14.05.2014  17:15    <DIR>          peacy.sys
14.07.2009  05:20    <DIR>          PerfLogs
24.05.2013  13:31    <DIR>          Perl
11.02.2014  07:56    <DIR>          Program Files
24.04.2014  07:02    <DIR>          Program Files (x86)
24.05.2013  00:02               194 setup.log
23.05.2013  23:51    <DIR>          SWTOOLS
15.05.2014  06:48    <DIR>          temp
30.03.2014  19:48    <DIR>          Users
14.05.2014  06:59    <DIR>          Windows
               3 File(s)        914.020 bytes
              14 Dir(s)  199.031.599.104 bytes free

C:\>_
```

```
Command Prompt
C:\>cd users

C:\Users>Dir
 Volume in drive C is System Volume
 Volume Serial Number is 42A1-A2BE

 Directory of C:\Users

30.03.2014  19:48    <DIR>          .
30.03.2014  19:48    <DIR>          ..
13.05.2014  06:48    <DIR>          Administrator
15.05.2014  16:48    <DIR>          erh5lr
27.05.2013  08:51    <DIR>          gsh9ebh
24.05.2013  12:52    <DIR>          Public
12.06.2013  16:32    <DIR>          tm81lr
24.05.2013  15:03    <DIR>          xDEadmin
28.08.2013  07:12    <DIR>          xDEGisi
               0 File(s)              0 bytes
               9 Dir(s)  199.031.451.648 bytes free

C:\Users>_
```

```
Comm...
C:\>cd users

C:\Users>
```

FAT 16 bzw. **FAT 32**: „File Allocation Table“ 16 oder 32 bit

DOS 8.3: File / **Datei-Namen** immer nur 8 Zeichen
und 3 Zeichen für **Datei-Endung**

z.B. **luftdrk**.**txt**
raumtemp.**txt**

SD_ReadWriteFile

```
/*      SD card read/write a File "FnameX.txt"
  * SD card attached to SPI bus as follows:
 ** MOSI - pin 11
 ** MISO - pin 12
 ** CLK - pin 13
 ** CS - pin 4  oder 10    On the Ethernet Shield, CS is pin 4.

Note that even if standard CS is not used as the CS pin for SD, the hardware SS pin
(10 on most Arduino boards, 53 on Mega) must be left as an output or SD lib. fcts. will not work. */

#include <SD.h>                          // Library

const int chipSelect = 10;               // Select-Pin für SPI Bauteil (hier SD-Card)
char myFileName[] = "wetter11.txt";      // Filename, max 8 Zeichen + 3 Extension
File myFile;                             // File Object definieren
```

```
void setup()
{
  Serial.begin(9600);
  Serial.print("Initializing SD card...    ");
  pinMode(10, OUTPUT);
  if (!SD.begin(chipSelect))             // SD Karte (Object) initialisieren
  {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.\n");
```

```
void loop()
{
        // nothing happens after setup
}
```

```
  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.
  myFile = SD.open(myFileName, FILE_WRITE);     // or "FILE_READ"
  if (myFile)                                   // if file opened okay, write to it
  {
    Serial.print("Writing to:  \"");            // Ausgabe eines  "  mit \"
    Serial.print(myFileName);
    Serial.print("\" : ");
    myFile.println("testing 1, 2, 3.");
    myFile.close();                             // close the file
    Serial.println(" --> done.\n");
  }  // if
  else
  {
    Serial.print("error opening for write: "); // if file didn't open, print error
    Serial.println(myFileName);
  }  // else

  myFile = SD.open(myFileName);                 // re-open the file for reading
  if (myFile)                                   // if file opened okay, read it
  {
    Serial.print("Inhalt von :   \"");
    Serial.print(myFileName);
    Serial.println("\"");
    while (myFile.available())                  // read from file until there's nothing else in it:
    {
        Serial.write(myFile.read());            // read file
    }
    myFile.close();                             // close file:
  }  // if
  else
  {
    Serial.print("error opening for read:  "); // if file didn't open, print error
    Serial.println(myFileName);
  }  // else
}  // setup
```

Arduino Time library:

adds timekeeping functionality to Arduino with or without external timekeeping hardware. It allows a sketch to get the time and date as: **second, minute, hour, day, month and year**.
It also provides time as a standard C "**time_t**" so elapsed times can be easily calculated and time values shared across different platforms.

The download includes example sketches illustrating how similar sketch code can be used with: a Real Time Clock,Internet NTP time service, GPS time data, DCF77 radio signal, and Serial time messages from a computer. To use all of the features in the library, you'll need the UDPbitewise library, found here.
Additional documentation is included in a readme.txt file

Functional Overview:

```
hour();                 // The hour now  (0-23)
minute();               // The minute now (0-59)
second();               // The second now (0-59)

day();                  // The day now (1-31)
weekday();              // Day of the week, Sunday is day 1
month();                // The month now (1-12)
year();                 // The full four digit year: (2009, 2010 etc)

hourFormat12();         // The hour now in 12 hour format
isAM();                 // Returns true if time now is AM
isPM();                 // Returns true if time now is PM


now();                  // Returns current time as seconds since Jan 1 1970
```

The **time and date functions** can take an optional **parameter** for the time. This prevents errors if the time rolls over between elements. For example, if a new minute begins between getting the minute and second, the values will be inconsistent.
→ Using the following functions eliminates this problem:

```
time_t t = now();          // Store the current time in time variable "t"

hour(t);                   // Returns the hour for the given time "t"
minute(t);                 // Returns the minute for the given time "t"
second(t);                 // Returns the second for the given time "t"

day(t);                    // The day for the given time "t"
weekday(t);                // Day of the week for the given time "t"
month(t);                  // The month for the given time "t"
year(t);                   // The year for the given time "t"
```

Functions for managing the timer services are:

```
setTime(t);                          // Set the system time to the given time "t"
setTime(hr,min,sec,day,month,yr);    // Another way to set the time
adjustTime(adjustment);              // Adjust sys. time by adding "adjustment" value


timeStatus();                        // Indicates if time has been set and recently synchronized
                                     //  returns one of the following enumerations:
  * timeNotSet                       // time has never been set, clock started at Jan 1 1970
  * timeNeedsSync                    // time had been set, but a sync attempt did not succeed
  * timeSet                          // time is set and is synced
//  Time and Date values are not valid if the status is timeNotSet. Otherwise values can be used
but the returned time may have drifted if the status is timeNeedsSync.


setSyncProvider(getTimeFunction);    // Set the external time provider
setSyncInterval(interval);           // Set the number of seconds between re-sync
```

```
#include <Time.h>

#define TIME_MSG_LEN  11   // time sync to PC is HEADER followed by Unix time_t as ten ASCII digits
#define TIME_HEADER  'T'   // Header tag for serial time sync message
#define TIME_REQUEST  7    // ASCII bell character requests a time sync message
// T1262347200             // noon Jan 1 2010

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

void loop(){
  if (Serial.available())
      processSyncMessage();
  if (timeStatus() == timeNotSet)
      Serial.println("waiting for sync message");
  else
      digitalClockDisplay();
  delay(1000);
}   // loop()
```

```
void loop()
{
  //update the display only if the time has changed
  if( now() != prevDisplay)
  {
    prevDisplay = now();
    digitalClockDisplay();
  }
}
```

```
void processSyncMessage()
{         // if time sync available from serial port, update time
  while(Serial.available() >=  TIME_MSG_LEN )     // time message consists of header
  {                                                // & 10 ASCII digits
    char c = Serial.read();
    Serial.print(c);
    if (c == TIME_HEADER)
    {
      time_t pctime = 0;
      for(int i=0; i < TIME_MSG_LEN -1; i++)
      {
        c = Serial.read();
        if ( c >= '0' && c <= '9')
        {
          pctime = (10 * pctime) + (c - '0');                    // convert digits to a number
        }  // if
      }  // for
      setTime(pctime);                        // Sync Arduino clock to the time received on serial port
    }  // if
  }  // while
}  // fct()
```

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

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

Memory

There are **three pools of memory** in the microcontrollers used on Arduino boards:

1) **Flash memory** (program space) → where the Arduino sketch is stored.

2) **SRAM** (static random access memory) → where the sketch creates & manipulates variables when it runs.

3) **EEPROM** is memory space that programmers can use to store long-term information.

Flash & EEPROM memory are non-volatile (the information persists after the power is turned off).

SRAM is volatile and will be lost when the power is cycled.

Note: Flash (PROGMEM) memory can only be populated at program burn time.

→ You can't change the values in the flash after the program has started running.

	ATMega168	ATMega328P	ATmega1280	ATmega2560
Flash (1 Kbyte used for bootloader)	16 KBytes	32 KBytes	128 KBytes	256 KBytes
SRAM	1024 bytes	2048 bytes	8 KBytes	8 KBytes
EEPROM	512 bytes	1024 bytes	4 KBytes	4 KBytes

You notice that there is a lot more Flash (program) memory than SRAM available.

When you create variables with the Arduino language such as:

char message[] = "I support the Cape Wind project.";

You copy 33 bytes (1 char = 1 byte, plus terminating null) from program memory into SRAM before using it. If the sketch requires some large unchanging data structures - such as a large amount of text to send to a display, or a large lookup table, for example - using flash memory (program memory) directly for storage may be the only option. To do this, use the PROGMEM keyword.

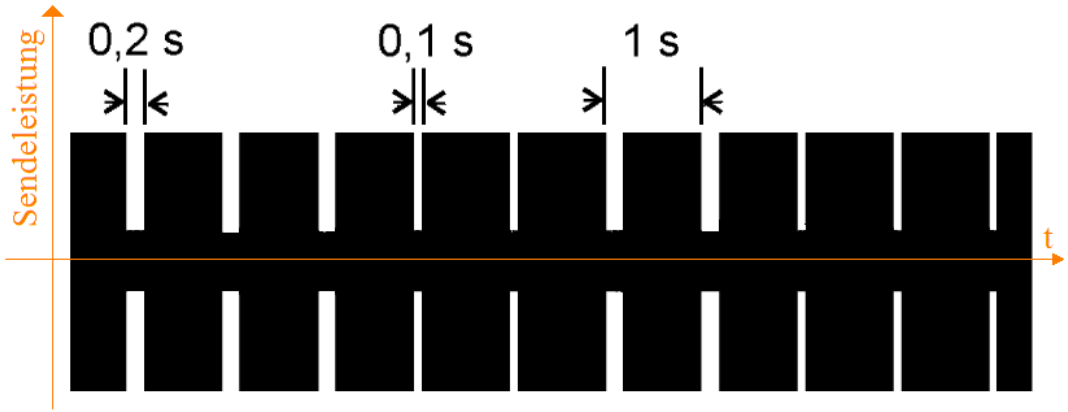
Version 1.0 of the Arduino introduced the **F() syntax** for storing **strings in flash memory** rather than RAM. e.g.

Serial.println(F("This string will be stored in flash memory"));

Der Sender steht in Mainflingen bei Frankfurt und arbeitet auf der Frequenz 77,5 kHz mit einer Leistung von 50 kW. Die Sendemasten kann man erkennen, wenn man auf der A3 bzw. A45 zum Seligenstädter Dreieck fährt.

Über DCF77 sendet die Physikalisch-Technische Bundesanstalt (PTB) in Braunschweig seit 1959 eine Normalfrequenz und seit 1973 zusätzlich dazu ein Datensignal für Datum und Uhrzeit.

Basis zur Erzeugung des Zeitsignals ist eine Steuereinrichtung mit drei Atomuhren. Diese wird mit den primären Atomuhren der PTB in Braunschweig (zwei Caesium-Uhren und zwei Caesium-Fontänen) synchronisiert. Das so gewonnene Signal hat nur einen Fehler von einer Sek. in 30.000 Jahren.



www.dcf77logs.de

Inhalt

Hauptseite
Über DCF77

Logdateien
Besondere Ereignisse
Web-Konsole

Logdateiarchiv
Software

Impressum
Kontakt

g+1 13

Web-Konsole

Auf dieser Seite ist die Konsole des DCF-Receivers zu sehen. Hier kann live verfolgt werden, wie die DCF-Daten empfangen und dekodiert werden. Die Ansicht wird jede Sekunde aktualisiert. (Hierzu ist aktiviertes Javascript erforderlich)

DCF-Receiver			
Impulszeit:	88ms	Pausenzeit:	907ms
Bit-Position:	09	Periodendauer:	995ms
Übertragungen:	11683	Erfolgreich	11620
		Abgebrochen:	63
Akt. DCF-Daten:	0-01111110_____-_____-_____-_____-_____-___-____-_____		
Letzte DCF-Daten:	0-11101100111001-001001-01100101-1010101-011000-101-01100-001010000		
Datum u. Uhrzeit:	06.06.2014 15:26	Wochentag:	Freitag
Rufbit:	Inaktiv	Zeitzonenwechsel:	Nicht bevorstehend
Schaltsekunde:	Wird nicht eingefügt	Zeitzone:	MESZ
Daten fehlerfrei:	Fehlerfrei	Synchronisierung:	Nein, Abweichung <1s
Dekodierte Wetterdaten:	100001001100010001100110		
Temperatur:	16°C	Prognose, Tag:	sonnig
Dekoderstatus:	Daten fehlerfrei	Nacht:	leicht bewölkt
Region:	15:23 Region: 7 (Tiefstwerte, 3. Tag) F - Dijon (Bourgogne), Bourgogne (Ostfrankreich / Burgund)		
2. Region:	--		
Letzte DCF-Daten:	0010010010100110101001011010101010111011 11 1100010010101000011000000110010100101000		

Letzter Logdateieintrag:
0 11101100111001 001001 01100101 1010101 011000 101 01100 001010000 Fr, 06.06.14 15:26:00, SZ

Das Trägersignal von 77,5 kHz ist in Frequenz und Phasenlage mit der steuernden primären Atomuhr synchronisiert und besitzt deshalb nur geringe Abweichungen von der Sollfrequenz. Es kann somit auch ohne Auswertung der Zeitinformation als Eichfrequenz für sehr genaue Hoch- und Niederfrequenzgeneratoren benutzt werden.

Ein Problem entsteht, wenn starker Wind die T-Antenne bewegt. Das äußert sich in einer messbaren Phasenmodulation des empfangenen Signals. Bei sehr starkem Sturm und dadurch stark bewegter Antenne, was zu einer Fehlanpassung der Antenne führt, muss der Sender vorübergehend außer Betrieb genommen werden.

Der Sender erzeugt eine nominelle Leistung von 50 kW, wovon etwa 30 bis 35 kW über die Antenne abgestrahlt werden.

Im Bereich bis etwa 600 km ist das Signal als Bodenwelle zu empfangen. Ab etwa 1100 km überwiegt der Raumwellenanteil. Im Abstand von 600 km bis 1100 km vom Sender kann es gelegentlich bei gleichen Feldstärken von Boden- und Raumwelle zur Auslöschung des Signals kommen (Fading von 15 min Dauer und mehr). Die Sollreichweite beträgt 2000 km (vgl. Empfangsgebiet).

Die Empfänger in den Funkuhren liefern am Ausgang die "Hüllkurve" des amplitudenmodulierten Signals.. Das Ausgangssignal der Empfänger kann direkt an einen Eingangs-Pin des Arduinos gelegt und von diesem weiter verarbeitet werden.

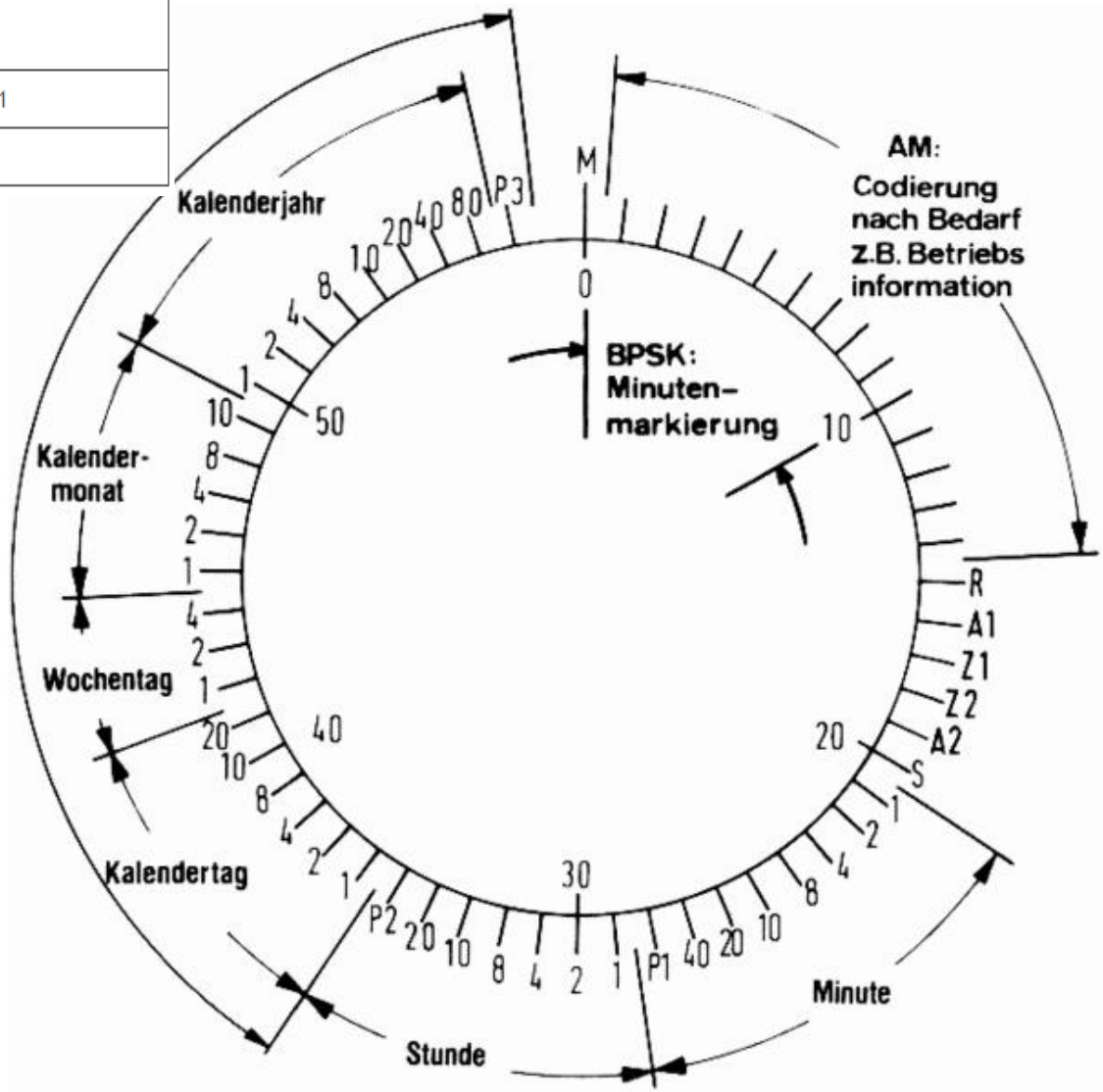
Die Zeitinformationen werden als digitales Signal zusätzlich zur Normalfrequenz (der Trägerfrequenz also 77,5 kHz) übertragen. Das geschieht durch negative Modulation des Signals (Absenken der Trägeramplitude auf etwa 15 %) im Sekundentakt.
Der Beginn der Absenkung liegt jeweils auf dem Beginn der Sekunden 0 bis 58 innerhalb einer Minute. In der letzten Sekunde erfolgt keine Absenkung, wodurch die nachfolgende Sekundenmarke den Beginn einer Minute kennzeichnet und der Empfänger synchronisiert werden kann.
Die Länge der Amplitudenabsenkungen am Beginn der Sekunden steht jeweils für den Wert eines binären Zeichens: **100 ms Absenkung stehen für den Wert „0“, 200 ms für „1“.**
Damit stehen innerhalb einer Minute 59 Bit für Informationen zur Verfügung.
Die Bits, deren Zählung bei 0 beginnt, werden wie folgt verwendet:

Bit	**Bedeutung der Werte**
0	Start einer neuen Minute (ist immer „0“)
1–14	Wetterinformationen der Firma MeteoTime sowie Informationen des Katastrophenschutzes
15	Rufbit (bis Mitte 2003: Reserveantenne)
16	„1“: Am Ende dieser Stunde wird MEZ/MESZ umgestellt.
17	„0“: MEZ, „1“: MESZ
18	„0“: MESZ, „1“: MEZ
19	„1“: Am Ende dieser Stunde wird eine Schaltsekunde eingefügt.

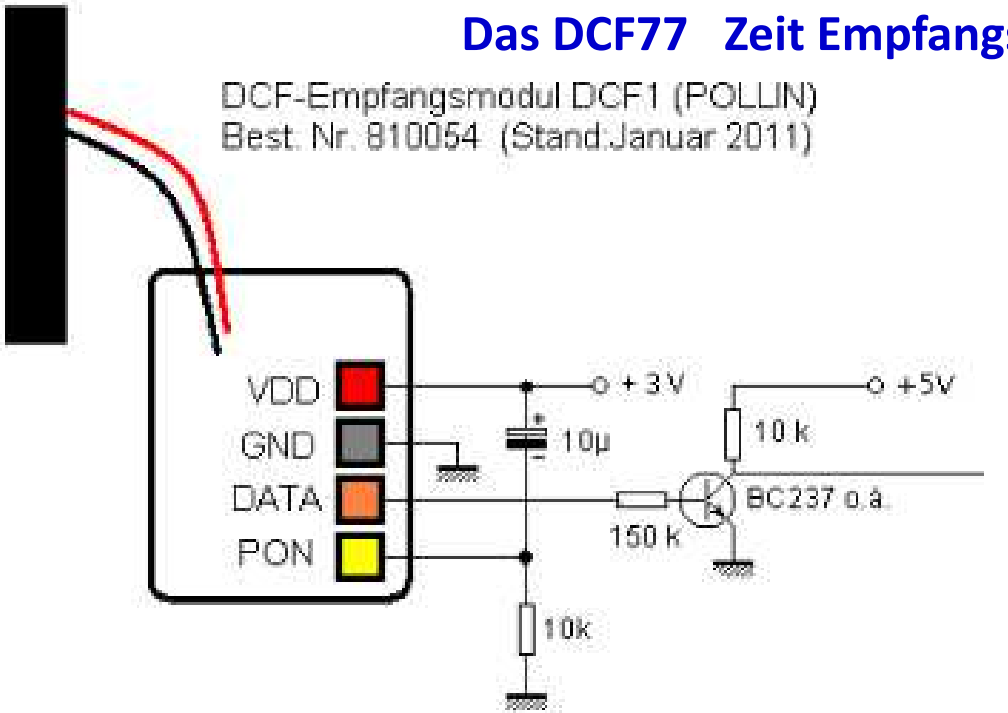
Bit	Bedeutung	
20	Beginn der Zeitinformation (ist immer „1“)	
21	Minute (Einer)	Bit für 1
22		Bit für 2
23		Bit für 4
24		Bit für 8
25	Minute (Zehner)	Bit für 10
26		Bit für 20
27		Bit für 40
28	Parität Minute	
29	Stunde (Einer)	Bit für 1
30		Bit für 2
31		Bit für 4
32		Bit für 8
33	Stunde (Zehner)	Bit für 10
34		Bit für 20
35	Parität Stunde	
36	Kalendertag (Einer)	Bit für 1
37		Bit für 2
38		Bit für 4
39		Bit für 8
40	Kalendertag (Zehner)	Bit für 10
41		Bit für 20

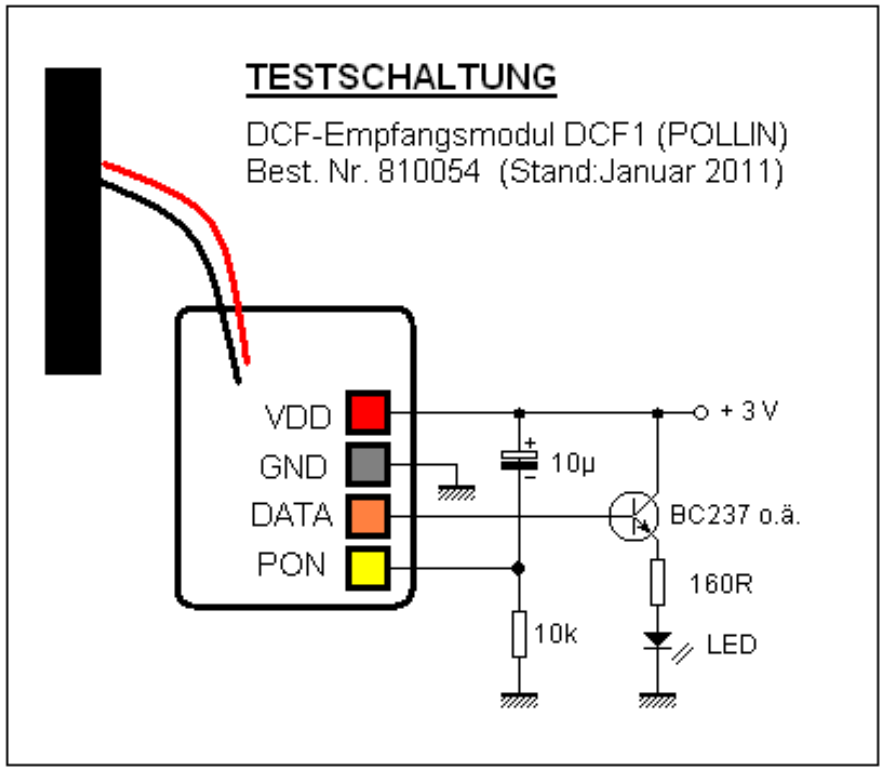
Bit	Bedeutung	
42	Wochentag	Bit für 1
43		Bit für 2
44		Bit für 4
45	Monatsnummer (Einer)	Bit für 1
46		Bit für 2
47		Bit für 4
48		Bit für 8
49	Monatsnummer (Zehner)	Bit für 10
50	Jahr (Einer)	Bit für 1
51		Bit für 2
52		Bit für 4
53		Bit für 8
54	Jahr (Zehner)	Bit für 10
55		Bit für 20
56		Bit für 40
57		Bit für 80
58	Parität Datum	
59	keine Sekundenmarke	

P1, P2, P3	Prüfbits
M	Minutenmarke
R	Aussendung über 0=Betriebsantenne, 1= Reserveantenne
A1	Ankündigung für Übergang auf Sommerzeit oder zurück, bevorstehender Zeitzonenwechsel, für die Dauer von 1 Stunde vor der Zeitzonenwechsel ist A1=1
Z1, Z2	Während der Sommerzeit 200 ms lang; sonst 100 ms Während der Normalzeit 200 ms lang; Sommerzeit 100 ms lang Aussendung von MEZ, dann ist Z1=0 und Z2=1; MESZ, dann Z1=1 und Z2=0
A2	Ankündigung einer Schaltsekunde, für die Dauer von 1 Stunde vor dem Einfügen der Schaltsekunde ist A2=1
S	Startbit der kodierten Zeitinformation (200 ms lang)



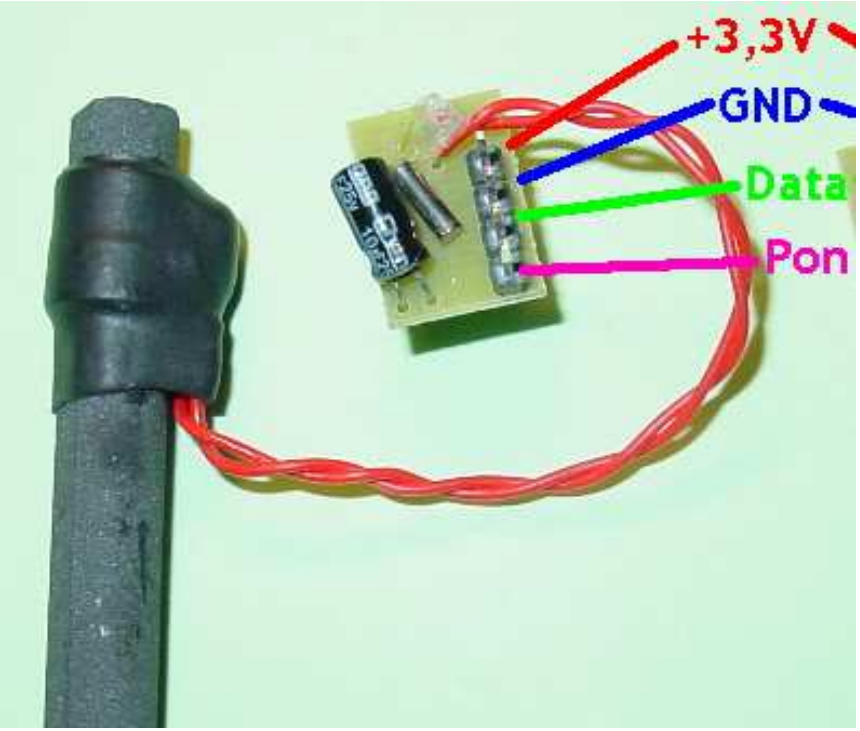
Das DCF77 Zeit Empfangs-Modul



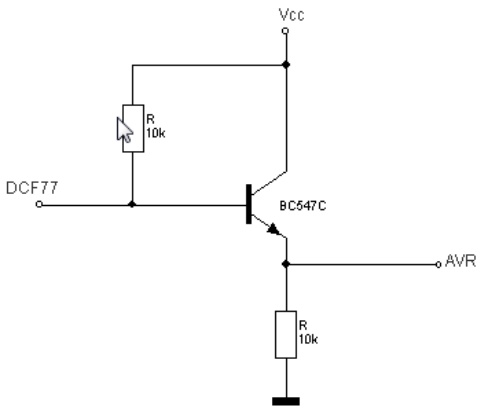


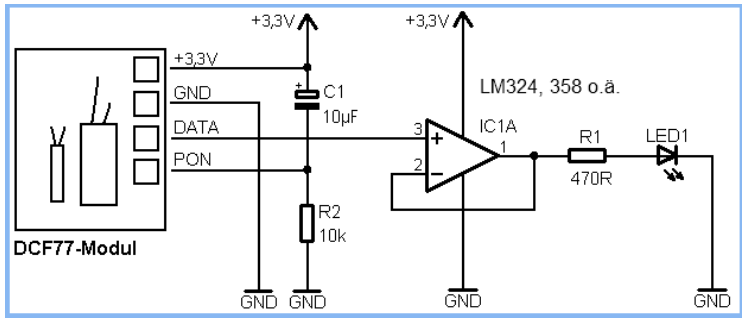
Da das DCF1 mit maximal 3,3 V betrieben werden darf, muss der Datenpegel für den Arduino angehoben werden. Das erledigt der Transistor T1. Allerdings wird dabei das Signal invertiert, so dass die Software entsprechend angepasst werden muss.
Der DATA-Ausgang des DCF1-Moduls sollte nur möglichst gering belastet werden, ein Basis-Vorwiderstand von 150 k sollte es sein.

Test: Der BC237 dient als Impedanzwandler. Das RC-Glied erzeugt nach dem Anlegen der Betriebsspannung eine negative Flanke am PON (PowerON) - Pin und schaltet das Modul ein. Bei guter Empfangsqualität blinkt die LED mit normaler Helligkeit im DCF77-Takt. Ist der Empfang durch Störungen überlagert, so bleibt zwar der Takt sichtbar, jedoch leuchtet die LED nur schwach



Das DCF77 Zeit Empfangs-Modul





http://www.b-redemann.de/sp-project-dcf77.shtml

DCF1 / Pollin	Arduino
3.3 V	3.3 V
GND	GND
Data	z.B. PIN 2
PON	GND

http://www.arduinoclub.de/2013/11/15/dcf77-dcf1-arduino-pollin/2/

http://www.mikrocontroller.net/articles/DCF77-Funkwecker_mit_AVR

http://arduino-hannover.de/2012/06/14/dcf77-empfanger-mit-arduino-betreiben/

InternalClockSync §

```
/*  This example shows how to fetch a DCF77 time and synchronize
    the internal clock. In order for this example to give clear output,
    make sure that you disable logging  from the DCF library. You can
    do this by commenting out   #define VERBOSE_DEBUG 1   in Utils.cpp. */

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

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

time_t time;
DCF77 DCF = DCF77(DCF_PIN, DCF_INTERRUPT);

void setup() {
  Serial.begin(9600);
  DCF.Start();
  Serial.println("Waiting for DCF77 time ... ");
  Serial.println("It will take up to 2 minutes until a first update can be processed.");
}

void loop() {
  delay(1000);
  time_t DCFtime = DCF.getTime();   // Check if new DCF77 time is available
  if (DCFtime!=0)
  {
    Serial.println("Time is updated");
    setTime(DCFtime);
  }
  digitalClockDisplay();
}
```

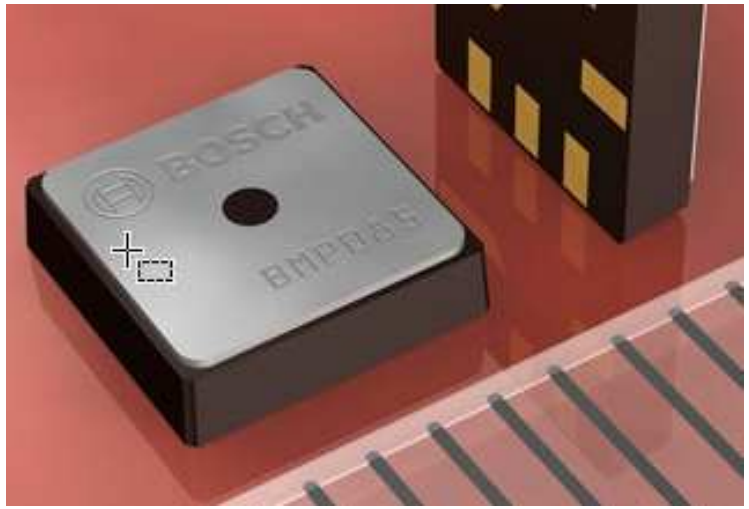
Luftdruck-Sensor mit I2C BUS

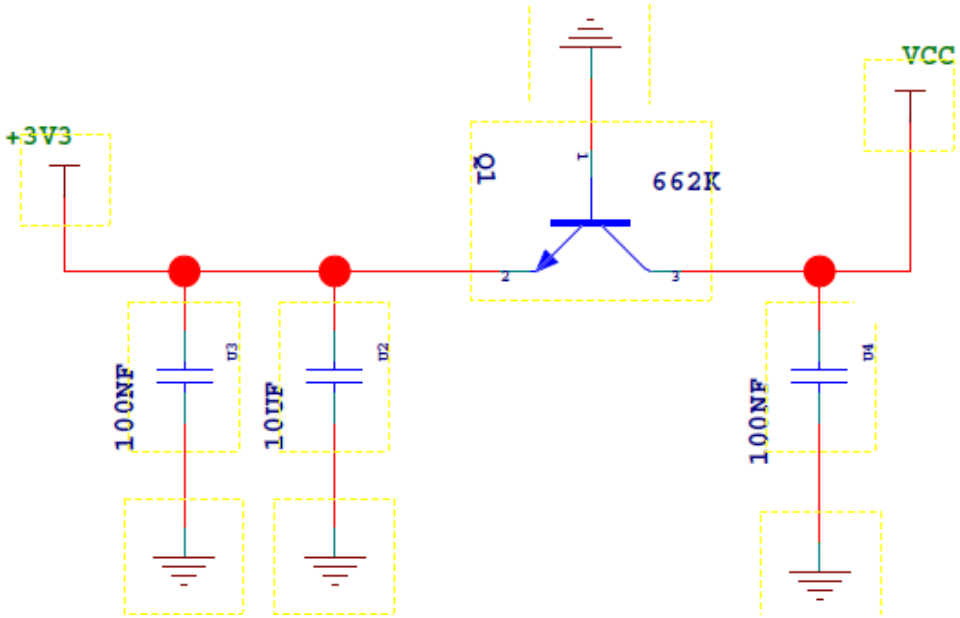
Luftdruck Sensor Breakout
(von ElecFreak Octopus Brick)
Messbereich 300 - 1100 hPa.
Anwendung z.B. als Barometer oder Höhenmesser.

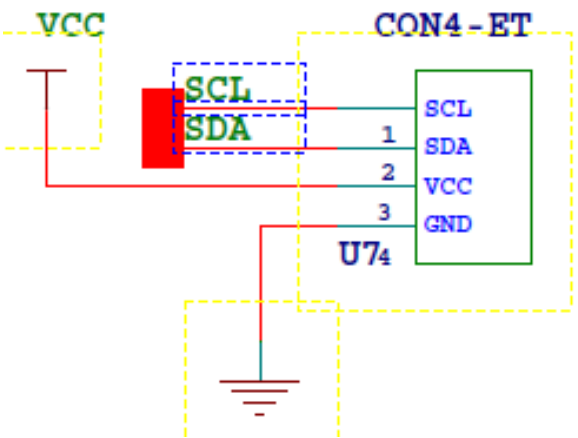
Board (UNO)	I2C / TWI pins
A4	(SDA),
A5	(SCL)
5V (VCC)	(VCC)
GND	]GND)

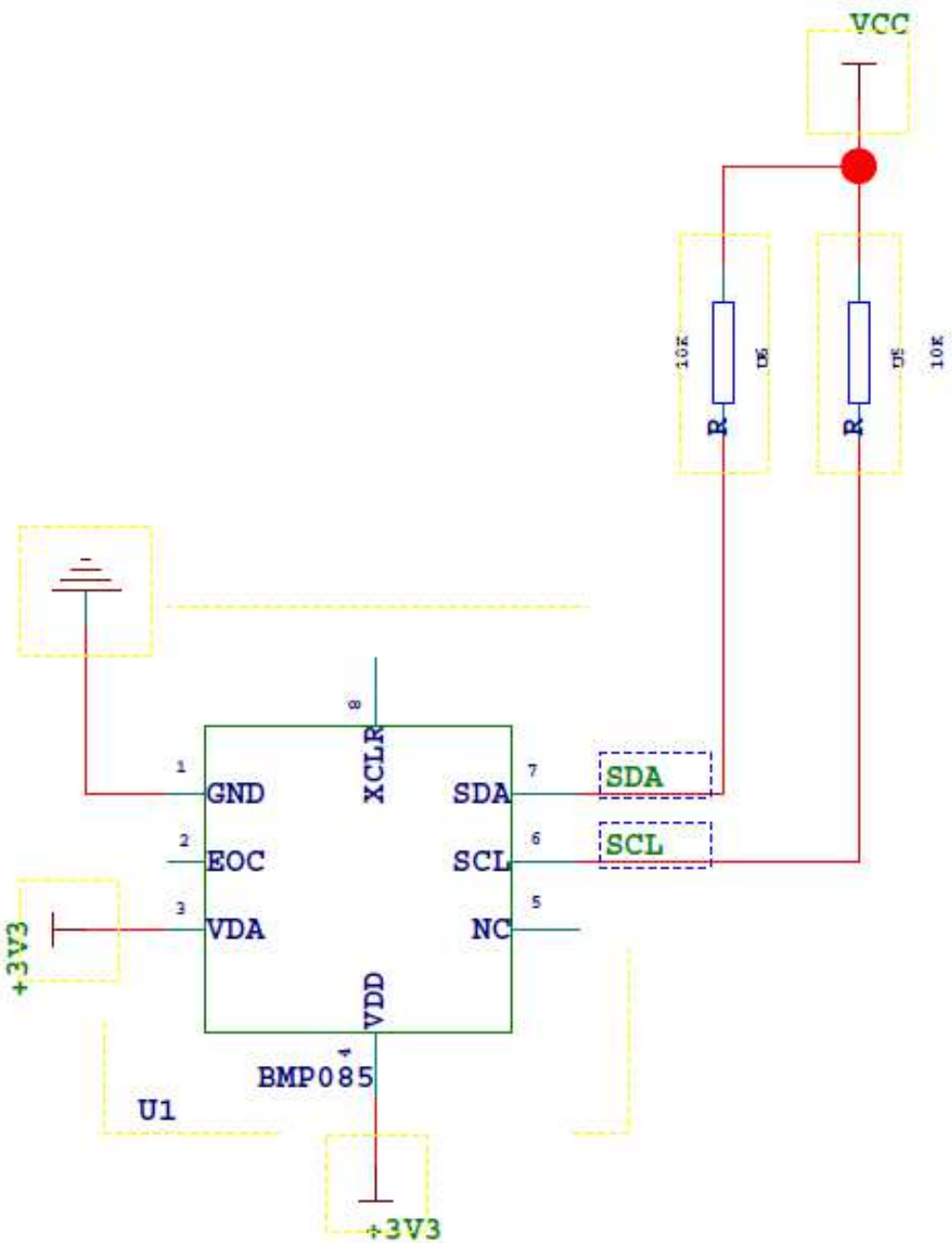
Versorgungsspannung 5V
(3,3V Stabi sitzt mit auf dem Modul)

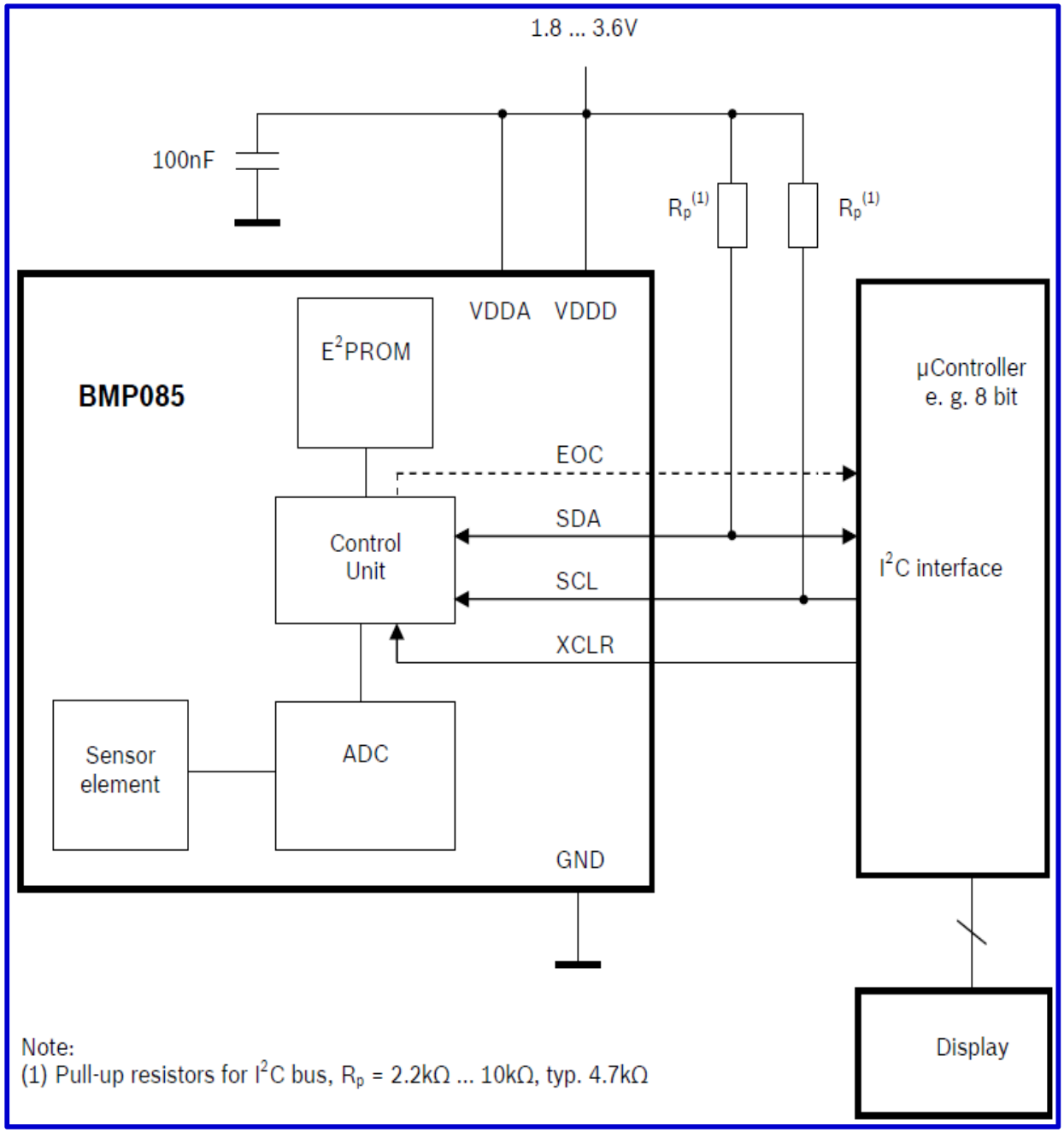
```
// Bibliothek die benötigt wird
#include <Wire.h>
// I2C address of BMP085
#define BMP085_ADDRESS 0x77
```




+3V3
VCC
Q1
662K
100NF
10UF
100NF


VCC
CON4-ET
SCL
SDA
SCL
SDA
VCC
GND
U74


VCC
R
R
10K
10K
GND
EOC
VDA
XCLR
SDA
SCL
NC
VDD
SDA
SCL
+3V3
BMP085
U1
+3V3



Parameter	Symbol	Condition	Min	Typ	Max	Units
Absolute accuracy pressure V_{DD} = 3.3V		700 … 1100 hPa 0 … +65 °C	-2.5	±1.0	+2.5	hPa
		300 … 700 hPa 0 … +65 °C	-3.0	±1.0	+3.0	hPa
		300 … 1100 hPa -20 … 0 °C	-4.0	±1.5	+4.0	hPa
Resolution of output data		pressure		0.01		hPa
		temperature		0.1		°C
Relative accuracy pressure V_{DD} = 3.3V		700 … 1100 hPa @ 25 °C		±0.2		hPa
		0 … 65 °C @ p const.		±0.5		hPa
Absolute accuracy temperature V_{DD} = 3.3V		@ 25 °C	-1.5	±0.5	+1.5	°C
		0 … +65 °C	-2.0	±1.0	+2.0	°C

Programm BMP085:

Wetter_Test | BMP85_fkts § | LCD_fkts | SD_Fkts | Time_Fkts | Utility

```
#include <Wire.h>                    // I2C Bus Library, z.b. für BMP085 Luftdrucksensor
bmp085Calibration();                 // BMP085  Kalibrier-Funktion
```

```
#define BMP085_ADDRESS 0x77          // I2C address of BMP085
const unsigned char OSS = 0;         // Oversampling Setting

// Calibration values
int ac1;
int ac2;
int ac3;
unsigned int ac4;
unsigned int ac5;
unsigned int ac6;
int b1;
int b2;
int mb;
int mc;
int md;

// b5 is calculated in bmp085GetTemperature(...),
// this variable is also used in bmp085GetPressure(...)
// so ...Temperature(...) must be called before ...Pressure(...).
long b5;
```

```
temperature = bmp085GetTemperature(bmp085ReadUT());
pressure = bmp085GetPressure(bmp085ReadUP());
Serial.print(F("BMP85 Temperature: "));
Serial.print(temperature, DEC);
Serial.println(F(" *0.1 deg C"));
Serial.print(F("BMP085 Pressure: "));
Serial.print(pressure, DEC);
Serial.println(F(" Pa"));
Serial.println();
```

```c
// Calculate pressure given up
// calibration values must be known
// b5 is also required so bmp085GetTemperature(...) must be called first.
// Value returned will be pressure in units of Pa.
long bmp085GetPressure(unsigned long up)
{
  long x1, x2, x3, b3, b6, p;
  unsigned long b4, b7;

  b6 = b5 - 4000;
  // Calculate B3
  x1 = (b2 * (b6 * b6)>>12)>>11;
  x2 = (ac2 * b6)>>11;
  x3 = x1 + x2;
  b3 = (((((long)ac1)*4 + x3)<<OSS) + 2)>>2;

  // Calculate B4
  x1 = (ac3 * b6)>>13;
  x2 = (b1 * ((b6 * b6)>>12))>>16;
  x3 = ((x1 + x2) + 2)>>2;
  b4 = (ac4 * (unsigned long)(x3 + 32768))>>15;

  b7 = ((unsigned long)(up - b3) * (50000>>OSS));
  if (b7 < 0x80000000)
    p = (b7<<1)/b4;
  else
    p = (b7/b4)<<1;

  x1 = (p>>8) * (p>>8);
  x1 = (x1 * 3038)>>16;
  x2 = (-7357 * p)>>16;
  p += (x1 + x2 + 3791)>>4;

  return p;
}
```

```c
// Calculate temperature given ut.
// Value returned will be in units of 0.1 deg C
short bmp085GetTemperature(unsigned int ut)
{
  long x1, x2;

  x1 = (((long)ut - (long)ac6)*(long)ac5) >> 15;
  x2 = ((long)mc << 11)/(x1 + md);
  b5 = x1 + x2;

  return ((b5 + 8)>>4);
}
```

```
// Read the uncompensated pressure value
unsigned long bmp085ReadUP()
{
  unsigned char msb, lsb, xlsb;
  unsigned long up = 0;

  // Write 0x34+(OSS<<6) into register 0xF4
  // Request a pressure reading w/ oversampling setting
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(0xF4);
  Wire.write(0x34 + (OSS<<6));
  Wire.endTransmission();

  // Wait for conversion, delay time dependent on OSS
  delay(2 + (3<<OSS));

  // Read register 0xF6 (MSB), 0xF7 (LSB), and 0xF8 (XLSB)
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(0xF6);
  Wire.endTransmission();
  Wire.requestFrom(BMP085_ADDRESS, 3);

  // Wait for data to become available
  while(Wire.available() < 3)
    ;
  msb = Wire.read();
  lsb = Wire.read();
  xlsb = Wire.read();

  up = (((unsigned long) msb << 16) | ((unsigned long) lsb << 8) | (unsigned long) xlsb) >> (8-OSS);

  return up;
}
```

```
// Read the uncompensated temperature value
unsigned int bmp085ReadUT()
{
  unsigned int ut;

  // Write 0x2E into Register 0xF4
  // This requests a temperature reading
  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(0xF4);
  Wire.write(0x2E);
  Wire.endTransmission();

  // Wait at least 4.5ms
  delay(5);

  // Read two bytes from registers 0xF6 and 0xF7
  ut = bmp085ReadInt(0xF6);
  return ut;
}
```

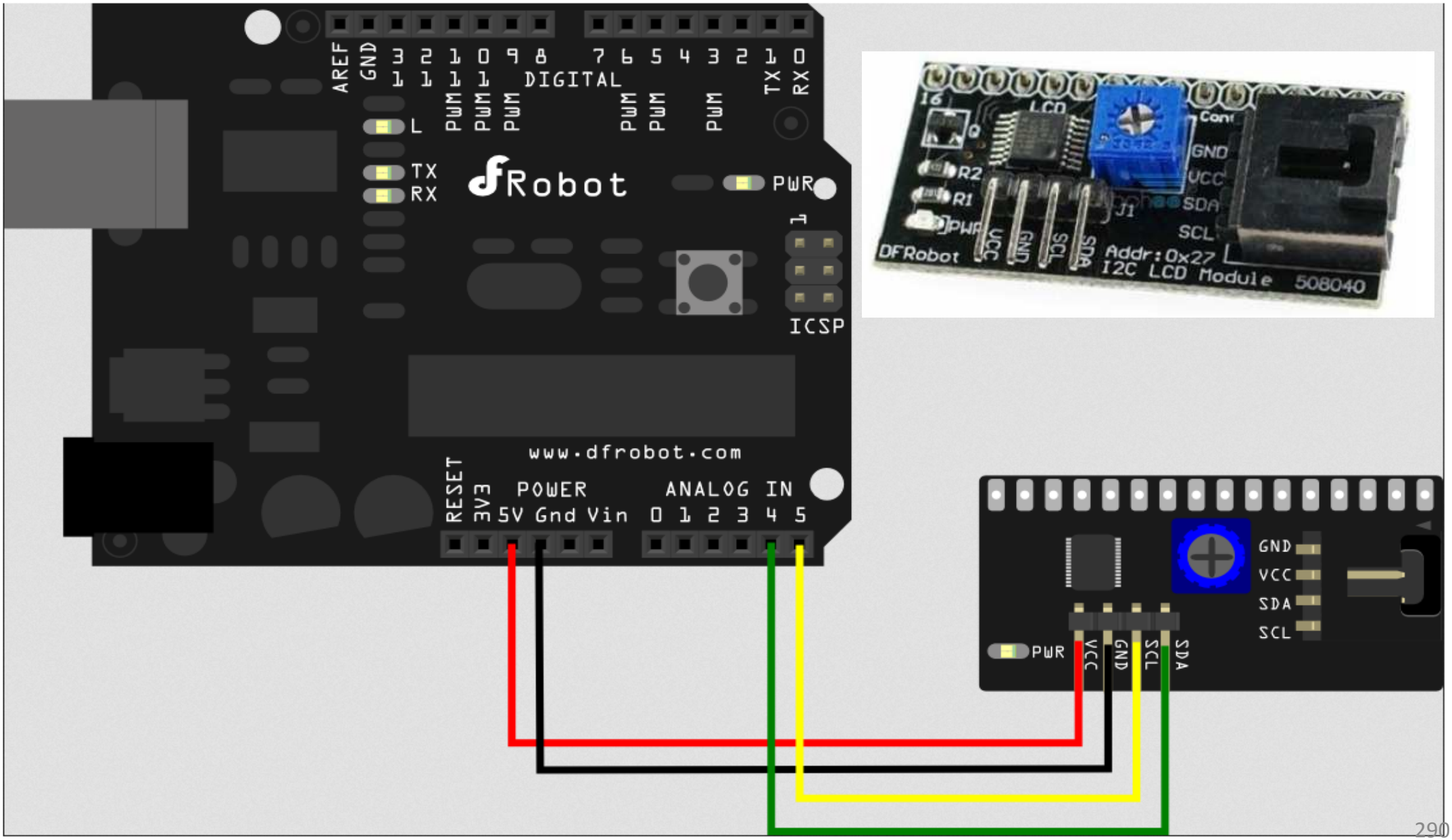
```
// Read 1 byte from the BMP085 at 'address'
char bmp085Read(unsigned char address)
{
  unsigned char data;

  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(address);
  Wire.endTransmission();

  Wire.requestFrom(BMP085_ADDRESS, 1);
  while(!Wire.available())
    ;

  return Wire.read();
}
```

```
// Stores all of the bmp085's calibration values into global variables
// Calibration values are required to calculate temp and pressure
// This function should be called at the beginning of the program
void bmp085Calibration()
{
  ac1 = bmp085ReadInt(0xAA);
  ac2 = bmp085ReadInt(0xAC);
  ac3 = bmp085ReadInt(0xAE);
  ac4 = bmp085ReadInt(0xB0);
  ac5 = bmp085ReadInt(0xB2);
  ac6 = bmp085ReadInt(0xB4);
  b1 = bmp085ReadInt(0xB6);
  b2 = bmp085ReadInt(0xB8);
  mb = bmp085ReadInt(0xBA);
  mc = bmp085ReadInt(0xBC);
  md = bmp085ReadInt(0xBE);
}
```

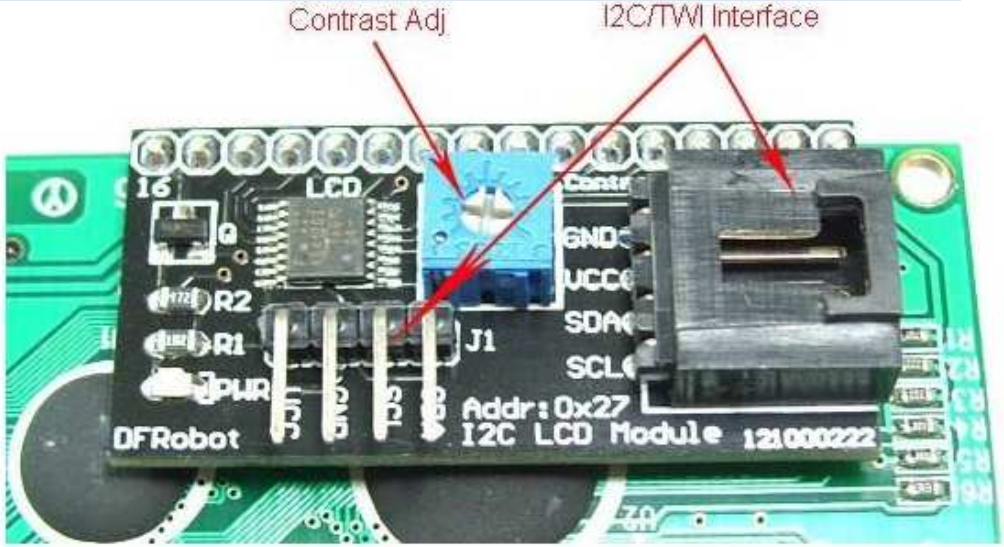
```
// Read 2 bytes from the BMP085
// First byte will be from 'address'
// Second byte will be from 'address'+1
int bmp085ReadInt(unsigned char address)
{
  unsigned char msb, lsb;

  Wire.beginTransmission(BMP085_ADDRESS);
  Wire.write(address);
  Wire.endTransmission();

  Wire.requestFrom(BMP085_ADDRESS, 2);
  while(Wire.available()<2)
    ;
  msb = Wire.read();
  lsb = Wire.read();

  return (int) msb<<8 | lsb;
}
```

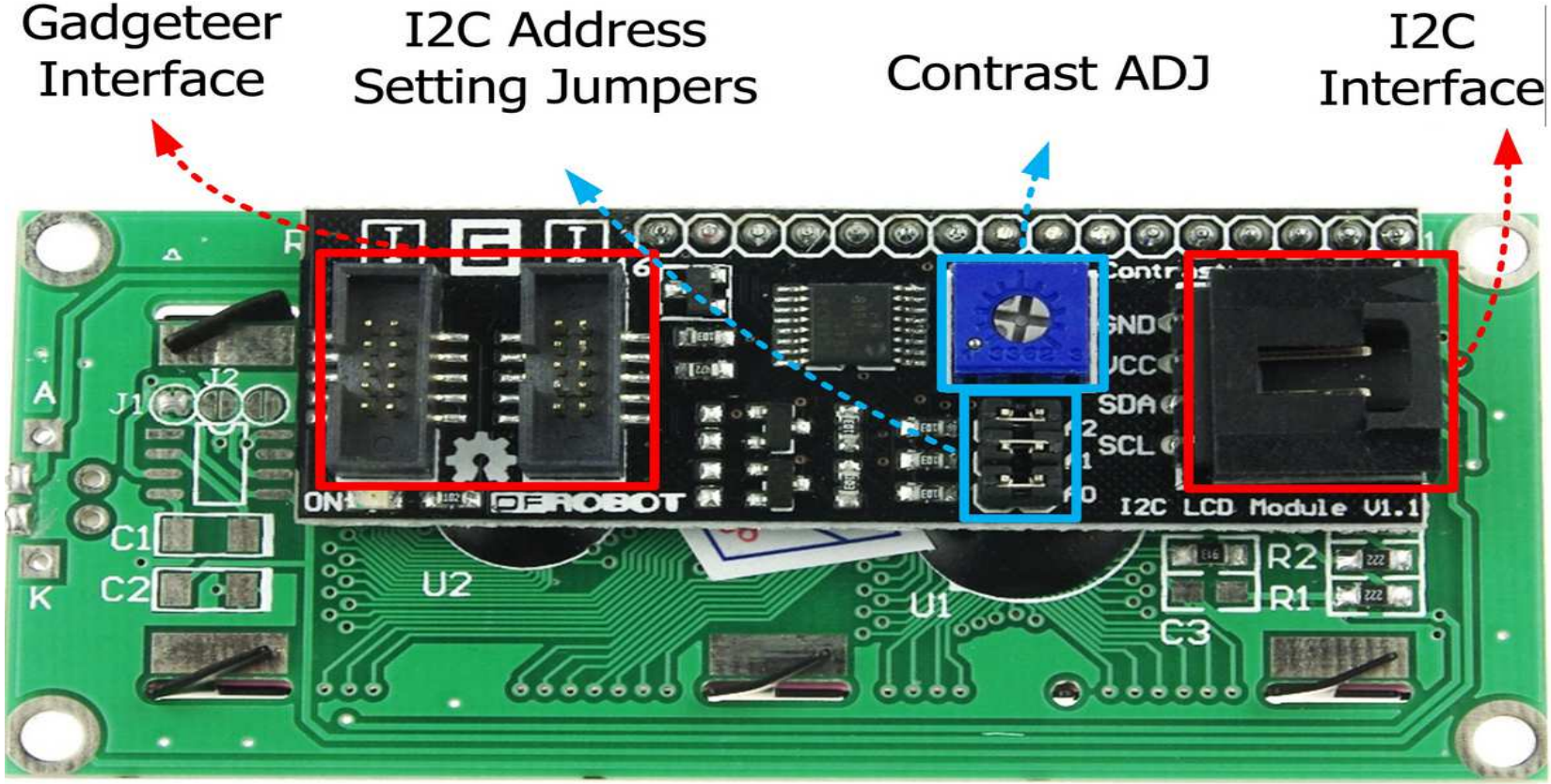
Bsp aus Wetter Programm

LCD Display mit I2C BUS: (2 statt “x” Leitungen)



Bsp: I2C - LCD Display von DFROBOT:





Library Support Functions:

LiquidCrystal_I2C()	// set the LCD “address” for a “x” chars and “y” line display
init()	// Initialization for the LCD
clear()	// clear display, set cursor position to zero
home()	// set cursor position to zero
createChar()	// Fill the first 8 CGRAM locations with custom characters
setCursor()	// set the position of the cursor
cursor()	// Turns the underline cursor on
noCursor()	// Turns the underline cursor off
blink()	// Turn on the blinking cursor
noBlink()	// Turn off the blinking cursor
display()	// Turn the display on(quickly)
noDisplay()	// Turn the display 0ff(quickly)
backlight()	// Turn the backlight on
noBacklight()	// Turn the backlight off
scrollDisplayLeft()	// Make the display scroll left without changing the RAM
scrollDisplayRight()	// Make the display scroll right without changing the RAM
autoscroll()	// This will 'right justify' text from the cursor
noAutoscroll()	// This will 'left justify' text from the cursor
leftToRight()	// This is for text that flows Left to Right
rightToLeft()	// This is for text that flows Right to Left

```
#include <Wire.h>                                   // Programm gibt „Text“ auf I2C LCD aus und lässt die
#include <LiquidCrystal_I2C.h>                      // Hintergrund Beleuchtung blinken

LiquidCrystal_I2C lcd(0x27,16,2);                   // set the LCD address to 0x27 for a 16 chars and 2 line display

void setup() {
  lcd.init();                                       // initialize the lcd
  lcd.backlight();
  lcd.home();
  lcd.print("Hello world...");
  lcd.setCursor(0, 1);
  lcd.print("dfrobot.com");
}

int backlightState = LOW;
long previousMillis = 0;
long interval = 1000;

void loop()
{
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis > interval) {
    previousMillis = currentMillis;
    if (backlightState == LOW)
    {
      backlightState = HIGH;
      lcd.backlight();
    }
    else
    {
      backlightState = LOW;
      lcd.noBacklight();
    }
  }
}
```

```
// Please download the Arduino library!
// The link:http://www.dfrobot.com/image/data/DFR0154/LiquidCrystal_I2Cv1-1.rar
// DFRobot.com
// Compatible with the Arduino IDE 1.0
// Library version:1.1

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

#if defined(ARDUINO) && ARDUINO >= 100
#define printByte(args)  write(args);
#else
#define printByte(args)  print(args,BYTE);
#endif

uint8_t bell[8]  = {0x4,0xe,0xe,0xe,0x1f,0x0,0x4};
uint8_t note[8]  = {0x2,0x3,0x2,0xe,0x1e,0xc,0x0};
uint8_t clock[8] = {0x0,0xe,0x15,0x17,0x11,0xe,0x0};
uint8_t heart[8] = {0x0,0xa,0x1f,0x1f,0xe,0x4,0x0};
uint8_t duck[8]  = {0x0,0xc,0x1d,0xf,0xf,0x6,0x0};
uint8_t check[8] = {0x0,0x1,0x3,0x16,0x1c,0x8,0x0};
uint8_t cross[8] = {0x0,0x1b,0xe,0x4,0xe,0x1b,0x0};
uint8_t retarrow[8] = { 0x1,0x1,0x5,0x9,0x1f,0x8,0x4};

LiquidCrystal_I2C lcd(0x27,20,4);     // set the LCD address to 0x27 for a 20 chars and 4 line display
```

```
void setup()
{
  lcd.init();                                   // initialize the lcd
  lcd.backlight();
  lcd.createChar(0, bell);
  lcd.createChar(1, note);
  lcd.createChar(2, clock);
  lcd.createChar(3, heart);
  lcd.createChar(4, duck);
  lcd.createChar(5, check);
  lcd.createChar(6, cross);
  lcd.createChar(7, retarrow);
  lcd.home();
  lcd.setCursor(0, 0);
  for(int i = 0;i < 20; i++)
       lcd.printByte(6);
  lcd.setCursor(0, 1);
  lcd.printByte(6);
  lcd.print("   Hello world    ");
  lcd.printByte(6);
  lcd.setCursor(0, 2);
  lcd.printByte(6);
  lcd.print("  ihr ");
  lcd.printByte(3);
  lcd.print(" arduinos!   ");
  lcd.printByte(6);
  lcd.setCursor(0, 3);
  for(int i = 0;i < 20; i++)
      lcd.printByte(6);
//  lcd.clear();
}   // setup
```

Technikschule Esslingen;
Kurs: Arduino

```
void loop()
{
}
```

Bunte LED Lichterkette **RGB WS 2812B**

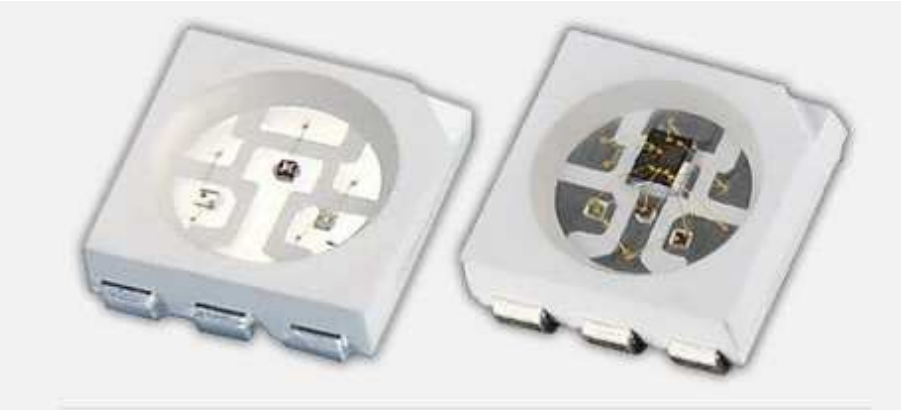
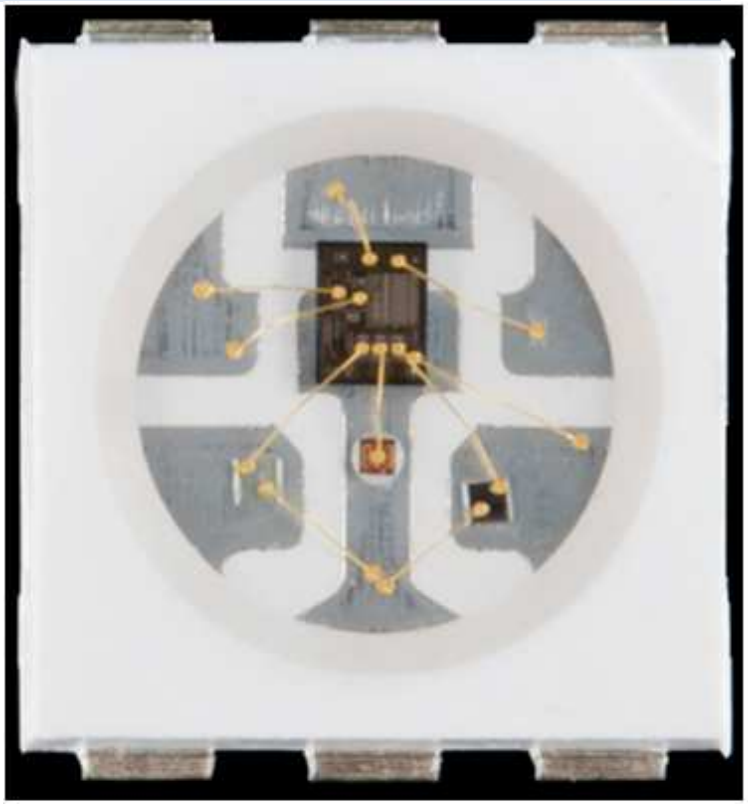
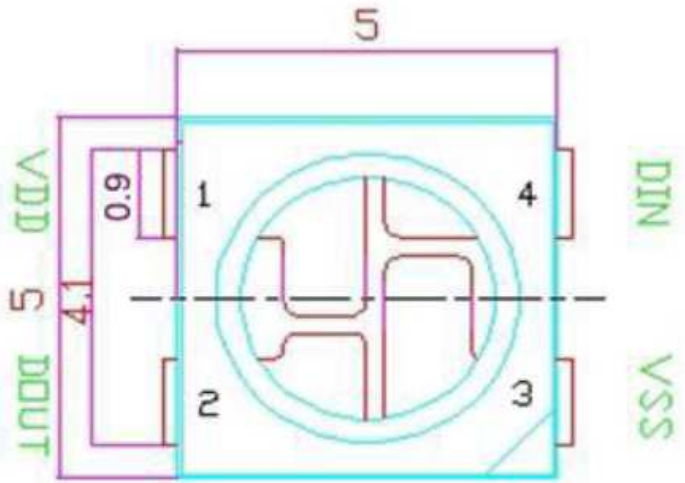


figure 1: 5050 RGB LED (left) and WS2812 (right)

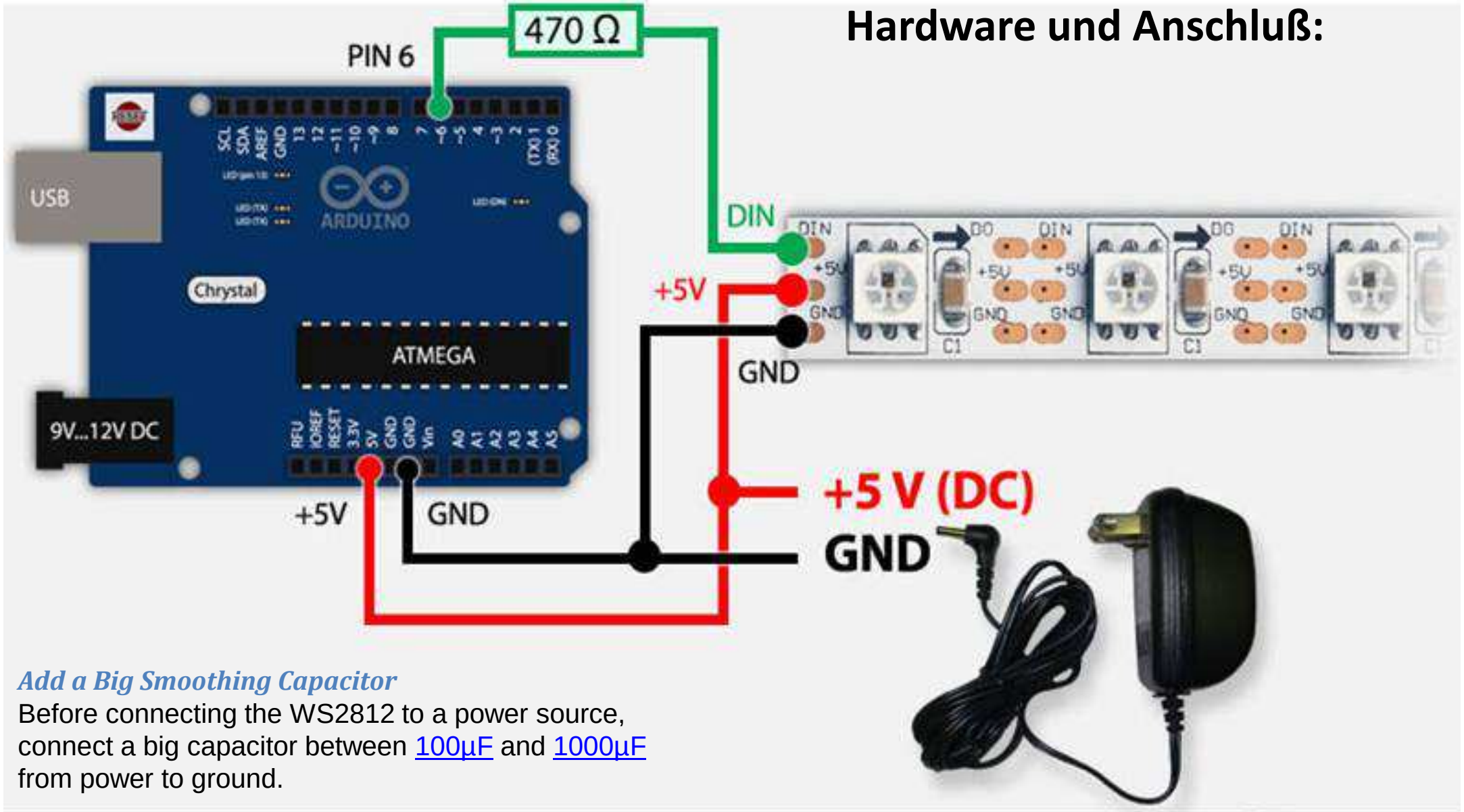


Pretty nifty view at the guts of the WS2812.

NO.	Symbol	Function description
1	VDD	Power supply LED
2	DOUT	Control data signal output
3	VSS	Ground
4	DIN	Control data signal input



Hardware und Anschluß:

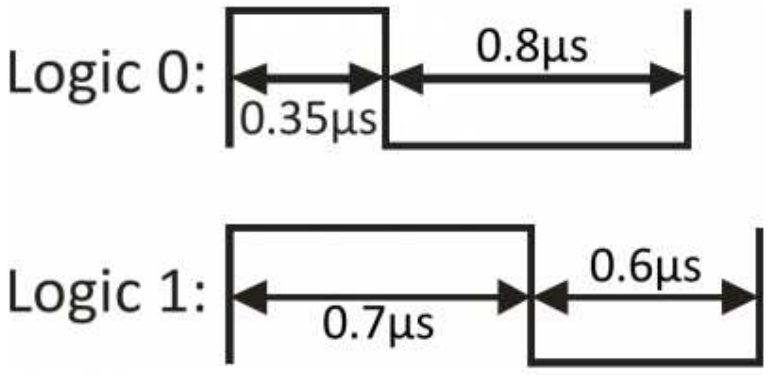


Add a Big Smoothing Capacitor

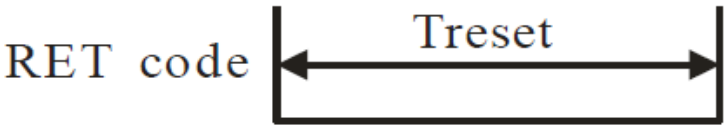
Before connecting the WS2812 to a power source, connect a big capacitor between 100µF and 1000µF from power to ground.

Arduino & WS2812 - Only running on external power supply

Prinzip der Ansteuerung, Schnittstelle und SW:



Timing diagram for a single bit of value 0 or 1.



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

Eine zeitgesteuerte 24 bit Folge ergibt eine Farbe, d.h. ein Pixel. Jede FarveA sequence of 24 timed-bits sets the color for each channel. 8-bits per channel. Green first, then red, then blue.

The larger the value of a specific color is, the brighter it will be. If every color is set to 0, the LED will be off. If every color is set to max – 255 – the LED will be as bright and white as can be.

This is all to say that the interface is **very time-specific**. To run the LEDs you'll need a real-time processor, like an Arduino; microprocessors like those on the Raspberry Pi or pcDuino can't give you a reliably-timed pulse. Even if one bit is less than a microsecond off, that could mean the difference between purple and maroon

Programming the Arduino for WS2811/WS2812

Now that we know how to connect the strip to our Arduino, time to get some cool effects going.

We *could* of course dig into all the timing details of the WS2811/WS2812 but there are already very good libraries out there that do the difficult work for us.

The most used at this moment are **FastLED** (successor of FastSPI_LED and FastSPI_LED2) and **NeoPixel** (by AdaFruit). I've added some demo videos, and even though it might give you the impression that NeoPixel could be slower, rest assure, it's not.

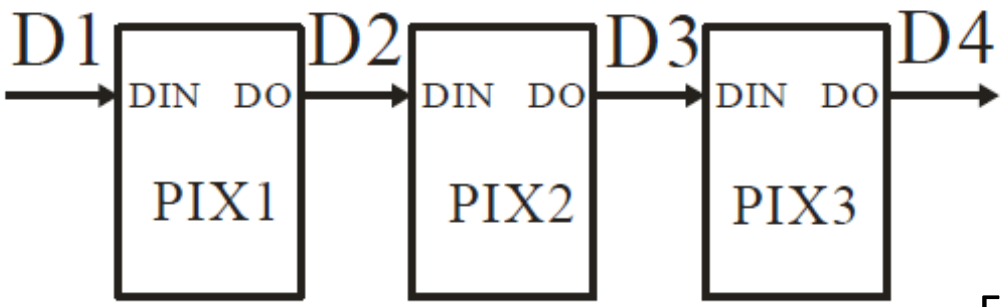
Both Libraries are very good and very fast.

•AdaFruit NeoPixel

•FastLED (FastSPI_LED)

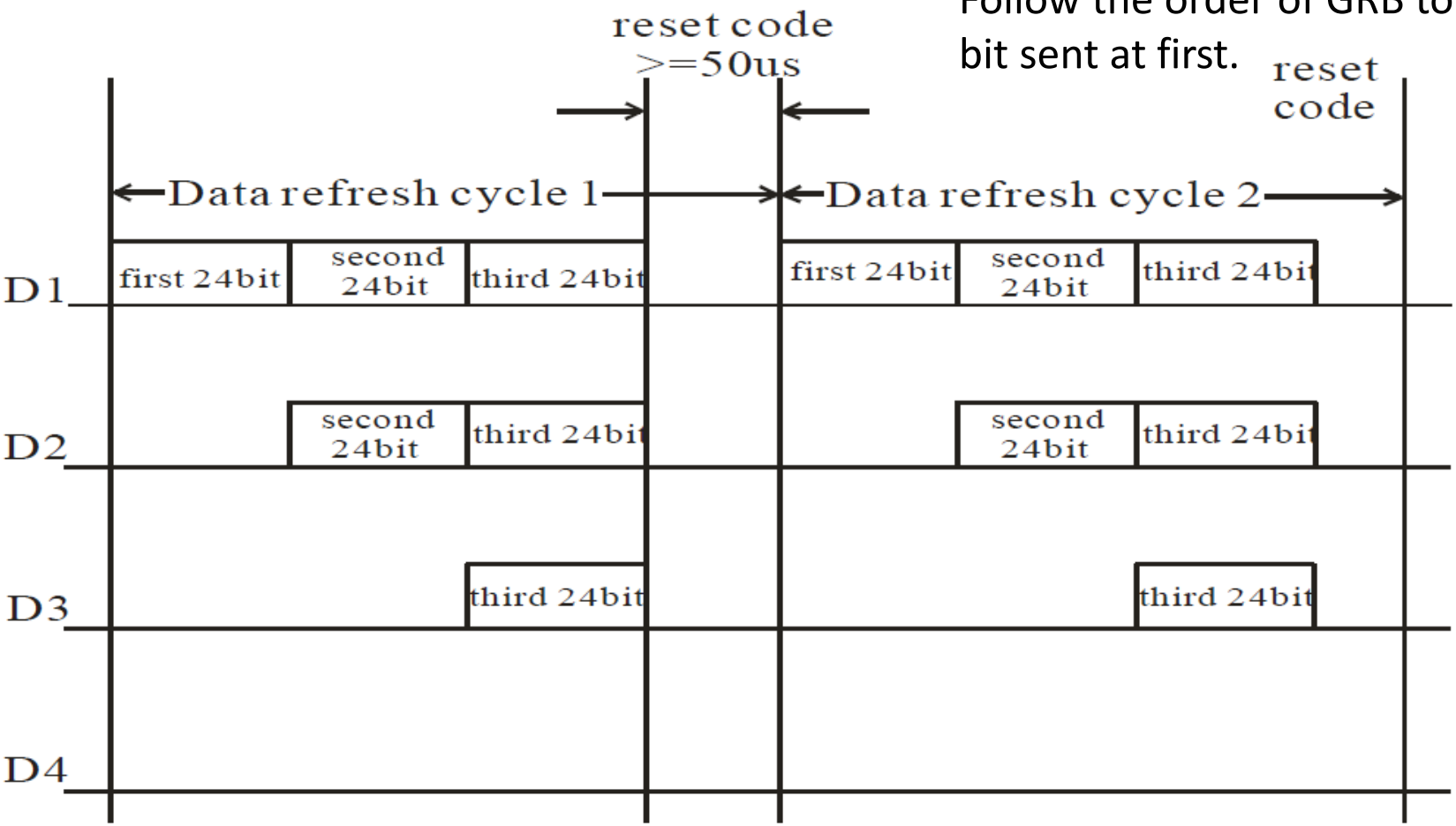
```
AliceBlue = 0xF0F8FF
Amethyst = 0x9966CC
AntiqueWhite = 0xFAEBD7
Aqua = 0x00FFFF
Aquamarine = 0x7FFFD4
Azure = 0xF0FFFF
Beige = 0xF5F5DC
Bisque = 0xFFE4C4
Black = 0x000000
BlanchedAlmond = 0xFFEBCD
Blue = 0x0000FF
BlueViolet = 0x8A2BE2
Brown = 0xA52A2A
BurlyWood = 0xDEB887
CadetBlue = 0x5F9EA0
Chartreuse = 0x7FFF00
Chocolate = 0xD2691E
Coral = 0xFF7F50
CornflowerBlue = 0x6495ED
Cornsilk = 0xFFF8DC
Crimson = 0xDC143C
Cyan = 0x00FFFF
DarkBlue = 0x00008B
DarkCyan = 0x008B8B
DarkGoldenrod = 0xB8860B
DarkGray = 0xA9A9A9
DarkGreen = 0x006400
DarkKhaki = 0xBDB76B
DarkMagenta = 0x8B008B
DarkOliveGreen = 0x556B2F
DarkOrange = 0xFF8C00
DarkOrchid = 0x9932CC
DarkRed = 0x8B0000
DarkSalmon = 0xE9967A
DarkSeaGreen = 0x8FBC8F
DarkSlateBlue = 0x483D8B
DarkSlateGray = 0x2F4F4F
DarkTurquoise = 0x00CED1
DarkViolet = 0x9400D3
DeepPink = 0xFF1493
DeepSkyBlue = 0x00BFFF
DimGray = 0x696969
DodgerBlue = 0x1E90FF
FireBrick = 0xB22222
FloralWhite = 0xFFFAF0
ForestGreen = 0x228B22
Fuchsia = 0xFF00FF
Gainsboro = 0xDCDCDC
GhostWhite = 0xF8F8FF
Gold = 0xFFD700
Goldenrod = 0xDAA520
Gray = 0x808080
Green = 0x008000
GreenYellow = 0xADFF2F
Honeydew = 0xF0FFF0
HotPink = 0xFF69B4
IndianRed = 0xCD5C5C
Indigo = 0x4B0082
Ivory = 0xFFFFF0
Khaki = 0xF0E68C
Lavender = 0xE6E6FA
LavenderBlush = 0xFFF0F5
LawnGreen = 0x7CFC00
LemonChiffon = 0xFFFACD
LightBlue = 0xADD8E6
LightCoral = 0xF08080
LightCyan = 0xE0FFFF
LightGoldenrodYellow = 0xFAFAD2
LightGreen = 0x90EE90
LightGrey = 0xD3D3D3
LightPink = 0xFFB6C1
LightSalmon = 0xFFA07A
LightSeaGreen = 0x20B2AA
LightSkyBlue = 0x87CEFA
LightSlateGray = 0x778899
LightSteelBlue = 0xB0C4DE
Lime = 0x00FF00
LimeGreen = 0x32CD32
Linen = 0xFAF0E6
Magenta = 0xFF00FF
Maroon = 0x800000
MediumAquamarine = 0x66CDAA
MediumBlue = 0x0000CD
MediumOrchid = 0xBA55D3
MediumPurple = 0x9370DB
MediumSeaGreen = 0x3CB371
MediumSlateBlue = 0x7B68EE
MediumSpringGreen = 0x00FA9A
MediumTurquoise = 0x48D1CC
MediumVioletRed = 0xC71585
MidnightBlue = 0x191970
MintCream = 0xF5FFFA
MistyRose = 0xFFE4E1
Moccasin = 0xFFE4B5
NavajoWhite = 0xFFDEAD
Navy = 0x000080
OldLace = 0xFDF5E6
Olive = 0x808000
OliveDrab = 0x6B8E23
Orange = 0xFFA500
OrangeRed = 0xFF4500
Orchid = 0xDA70D6
PaleGoldenrod = 0xEEE8AA
PaleGreen = 0x98FB98
PaleTurquoise = 0xAFEEEE
PaleVioletRed = 0xDB7093
PapayaWhip = 0xFFEFD5
PeachPuff = 0xFFDAB9
Peru = 0xCD853F
Pink = 0xFFC0CB
Plaid = 0xCC5533
Plum = 0xDDA0DD
PowderBlue = 0xB0E0E6
Purple = 0x800080
Red = 0xFF0000
RosyBrown = 0xBC8F8F
RoyalBlue = 0x4169E1
SaddleBrown = 0x8B4513
Salmon = 0xFA8072
SandyBrown = 0xF4A460
SeaGreen = 0x2E8B57
Seashell = 0xFFF5EE
Sienna = 0xA0522D
Silver = 0xC0C0C0
SkyBlue = 0x87CEEB
SlateBlue = 0x6A5ACD
SlateGray = 0x708090
Snow = 0xFFFAFA
SpringGreen = 0x00FF7F
SteelBlue = 0x4682B4
Tan = 0xD2B48C
Teal = 0x008080
Thistle = 0xD8BFD8
Tomato = 0xFF6347
Turquoise = 0x40E0D0
Violet = 0xEE82EE
Wheat = 0xF5DEB3
White = 0xFFFFFF
WhiteSmoke = 0xF5F5F5
Yellow = 0xFFFF00
YellowGreen = 0x9ACD32
```

Cascade method:



The data of D1 is send by MCU, and D2, D3, D4 through pixel internal reshaping amplification to transmit.

Follow the order of GRB to sent data and the high bit sent at first.



RGB_LEDs_WS2812B_AdafruitDemo

```
#include <Adafruit_NeoPixel.h>

#define PIN 6

// Parameter 1 = number of pixels in strip
// Parameter 2 = pin number (most are valid)
// Parameter 3 = pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
// Adafruit_NeoPixel strip = Adafruit_NeoPixel(60, PIN, NEO_GRB + NEO_KHZ800);
Adafruit_NeoPixel strip = Adafruit_NeoPixel(3, PIN, NEO_GRB + NEO_KHZ800);

void setup() {
  strip.begin();
  strip.show(); // Initialize all pixels to 'off'
}

void loop() {
  // Some example procedures showing how to display to the pixels:
  colorWipe(strip.Color(255, 0, 0), 50); // Red
  colorWipe(strip.Color(0, 255, 0), 50); // Green
  colorWipe(strip.Color(0, 0, 255), 50); // Blue
  // Send a theater pixel chase in...
  theaterChase(strip.Color(127, 127, 127), 50); // White
  theaterChase(strip.Color(127,   0,   0), 50); // Red
  theaterChase(strip.Color(  0,   0, 127), 50); // Blue

  rainbow(20);
  rainbowCycle(20);
  theaterChaseRainbow(50);
}

// Fill the dots one after the other with a color
void colorWipe(uint32_t c, uint8_t wait) {
  for(uint16_t i=0; i<strip.numPixels(); i++) {
      strip.setPixelColor(i, c);
      strip.show();
      delay(wait);
  }
}
```

```
void rainbow(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256; j++) {
    for(i=0; i<strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel((i+j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

// Slightly different, this makes the rainbow equally distributed throughout
void rainbowCycle(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256*5; j++) { // 5 cycles of all colors on wheel
    for(i=0; i< strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel(((i * 256 / strip.numPixels()) + j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

//Theatre-style crawling lights.
void theaterChase(uint32_t c, uint8_t wait) {
  for (int j=0; j<10; j++) {  //do 10 cycles of chasing
    for (int q=0; q < 3; q++) {
      for (int i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, c);    //turn every third pixel on
      }
      strip.show();

      delay(wait);

      for (int i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, 0);        //turn every third pixel off
      }
    }
  }
}
```

```
//Theatre-style crawling lights with rainbow effect
void theaterChaseRainbow(uint8_t wait) {
  for (int j=0; j < 256; j++) {     // cycle all 256 colors in the wheel
    for (int q=0; q < 3; q++) {
        for (int i=0; i < strip.numPixels(); i=i+3) {
          strip.setPixelColor(i+q, Wheel( (i+j) % 255));    //turn every third pixel on
        }
        strip.show();

        delay(wait);

        for (int i=0; i < strip.numPixels(); i=i+3) {
          strip.setPixelColor(i+q, 0);        //turn every third pixel off
        }
    }
  }
}

// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
  if(WheelPos < 85) {
   return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
  } else if(WheelPos < 170) {
   WheelPos -= 85;
   return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  } else {
   WheelPos -= 170;
   return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
}
```

Interrupt Steuerung

Es gibt Hardware- und Timer-Interrupts → Ereignis
Wenn eines dieser Ereignisse eintritt unterbricht der Arduino sein Haupt-Programm, springt in eine sogenannte „Interrupt-Routine“ (ISR) und setzt danach mit seinem Hauptprogramm (loop) fort.
Hardwareinterrupts können Impulse bzw. Flanken also Wechsel HIGH/LOW bzw. LOW/HIGH an Eingängen sein.
Timerinterrupts treten immer nach einer bestimmten Zeit auf wenn z.B. ein Zähler auf „0“ springt
Beide Arten können programmiert werden.

Sinn und Zweck:

Typischer Anwendungsfall sind Taster, Tastatur oder Mausclicks und die Abfrage von Sensoren. Statt diese im Hauptprogramm ständig abzufragen (Polling) und damit den Arduino zu belasten, er kann solange keine anderen Sachen machen, ist es sinnvoller Tasten und Sensoren nur dann abzufragen, wenn sich ihr Zustand geändert hat. Und genau in nur diesem Fall wird automatisch ein Interrupt ausgelöst. Es wird kurz die Interrupt-Routine abgearbeitet und danach geht es sofort im Hauptprogramm weiter.

Liste aller Arduino Interrupt Quellen

```
 1  Reset
 2  External Interrupt Request 0  (pin D2)          (INT0_vect)
 3  External Interrupt Request 1  (pin D3)          (INT1_vect)
 4  Pin Change Interrupt Request 0 (pins D8 to D13) (PCINT0_vect)
 5  Pin Change Interrupt Request 1 (pins A0 to A5)  (PCINT1_vect)
 6  Pin Change Interrupt Request 2 (pins D0 to D7)  (PCINT2_vect)
 7  Watchdog Time-out Interrupt                     (WDT_vect)
 8  Timer/Counter2 Compare Match A                  (TIMER2_COMPA_vect)
 9  Timer/Counter2 Compare Match B                  (TIMER2_COMPB_vect)
10  Timer/Counter2 Overflow                         (TIMER2_OVF_vect)
11  Timer/Counter1 Capture Event                    (TIMER1_CAPT_vect)
12  Timer/Counter1 Compare Match A                  (TIMER1_COMPA_vect)
13  Timer/Counter1 Compare Match B                  (TIMER1_COMPB_vect)
14  Timer/Counter1 Overflow                         (TIMER1_OVF_vect)
15  Timer/Counter0 Compare Match A                  (TIMER0_COMPA_vect)
16  Timer/Counter0 Compare Match B                  (TIMER0_COMPB_vect)
17  Timer/Counter0 Overflow                         (TIMER0_OVF_vect)
18  SPI Serial Transfer Complete                    (SPI_STC_vect)
19  USART Rx Complete                               (USART_RX_vect)
20  USART, Data Register Empty                      (USART_UDRE_vect)
21  USART, Tx Complete                              (USART_TX_vect)
22  ADC Conversion Complete                         (ADC_vect)
23  EEPROM Ready                                    (EE_READY_vect)
24  Analog Comparator                               (ANALOG_COMP_vect)
25  2-wire Serial Interface  (I2C)                  (TWI_vect)
26  Store Program Memory Ready                      (SPM_READY_vect)
```

Pin D2 und D3:

Wir betrachten hier exemplarisch die zwei externen **Hardwareinterrupts** → Pin **D2** **&** **D3**.

4 Interrupt Arten gibt es:
LOW → trigger the interrupt whenever the pin is low,
CHANGE → trigger the interrupt whenever the pin changes value
RISING → trigger when the pin goes from low to high,
FALLING → trigger when the pin goes from high to low.

attachInterrupt(interrupt, ISR, mode) **legt fest welcher Interrupt scharf geschaltet werden soll, auf welches Ereignis (Modus) er reagieren und welche Interrupt Routine (ISR) gestartet werden soll (interrupt= 0 oder 1, ISR=„Name“ der Interruptroutine mode=LOW, CHANGE, RISING, FALLING)**

detachInterrupt(interrupt) **legt fest welcher Interrupt gestoppt werden soll (interrupt= 0 oder 1)**

Interrupt Service Routine(s) ISR:

ISRs haben Eigenarten die normale Funktionen nicht haben:

→ sie dürfen **keine Parameter** haben → ISR_name()
→ Sie können **keinen Wert zurückgeben**
→ Sie müssen möglichst **kurz** sein
→ Es kann immer **nur eine ISR gleichzeitig** laufen, die anderen werden ignoriert, da die Interrupts für die Dauer der ISR abgeschaltet ist.
→ Deshalb funktionieren Functionen wie delay() und millis() in einer ISR nicht, da beide auch Interrupt gesteuert sind. Zeichen über die USB Schnittstelle können verloren gehen, wenn die ISR zu lange dauert
→ **delayMicroseconds()** arbeitet korrekt da nicht von Interrupt abhängig
→ Werte vom Hauptprogramm zur ISR und zurück müssen über **globale Variablen** ausgetauscht werden. Diese müssen als “**volatile**“ (nicht flüchtig) deklariert werden.

Interrupt0_Taster_steuertLED_mVolatileVar

```
1  /*   Taster per Interrupt lesen und LED ein- ausschaltem
2  Board                Interrupt-Nr  int.0    int.1
3  Uno, Ethernet        Pin-Nr        D2       D3
4
5  attachInterrupt(interrupt, ISR, mode)
6    interrupt:     0 oder 1
7    ISR:           Interrupt Service Routine
8    mode: - LOW trigger interrupt whenever pin is low,
9          - CHANGE trigger interrupt whenever pin changes
10         - RISING trigger when pin goes from low to high,
11         - FALLING trigger when pin goes from high to low. */
12
13 const int ledPin = 13;              // extra oder eingebaute LED
14 const int tasterPin = 2;            // Taster am int0
15 volatile int state = LOW;           // nicht flüchtige Variable speziell für ISR
16
17 void setup()
18 {
19   pinMode(ledPin, OUTPUT);          // LED Pin daher OUTPUT
20   attachInterrupt(0, ISRtastenStatus, CHANGE); // Interrupt=0, ISR-Name,
21 }  // setup                            Interrupt bei jedem Flankenwechsel
22
23 void loop()                         //  beliebiger komplexter Code
24 {
25   digitalWrite(ledPin, state);      // LED ein bzw ausschalten, je nach "state"
26 }  // loop
27
28 void ISRtastenStatus()              // die Interrupt Fkt ISR
29 {                            // Immer wenn sich TST-Zustand ändert wird Fkt aufgerufen
30   state = !state;
31 } // fct ISR
```

Interrupt0_Taster_steuertLED §

```
1  /*   Taster per Interrupt lesen und LED ein- ausschaltem
2  Board                Interrupt-Nr  int.0    int.1
3  Uno, Ethernet        Pin-Nr        D2       D3
4
5  attachInterrupt(interrupt, ISR, mode)
6    interrupt:     0 oder 1
7    ISR:           Interrupt Service Routine
8    mode:  - LOW trigger interrupt whenever pin is low,
9           - CHANGE trigger interrupt whenever pin changes
10          - RISING trigger when pin goes from low to high,
11          - FALLING trigger when pin goes from high to low. */
12
13 const int ledPin = 13;              // extra oder eingebaute LED
14 const int tasterPin = 2;            // Taster am int0
15
16 void setup()
17 {
18   pinMode(ledPin, OUTPUT);          // LED Pin daher OUTPUT
19   attachInterrupt(0, ISRledSchalten, CHANGE); // Interrupt=0, ISR-Name,
20 }  // setup                          Interrupt bei jedem Flankenwechsel
21
22 void loop()
23 {
24                                     //  beliebiger komplexter Code
25 }  // loop
27 void ISRledSchalten()                // die Interrupt Fkt ISR
28 {                                    // Immer wenn sich Tastenzustand ändert wird Fkt aufgerufen
29   digitalWrite(ledPin, digitalRead(tasterPin)); // Gedrückt = EIN
30 } // fct ISR                        sonst = AUS
31
```

Interrupt0_Taster_steuertLED_Toggle

```
1  /*   Taster per Interrupt lesen und LED toggeln lassen
2  Diese SW fumtioniert nur zuverlässig mit HW Entprellung! */
3
4  const int ledPin = 13;                  // extra oder eingebaute LED
5  const int tasterPin = 2;                // Taster am int0
6  volatile byte ledStatus = 0;            // Led Status wird von ISR getoggelt
7  volatile int  ISRCounter = 0;           // zählt alle ISR Aufrufe
8  int ISRCounterOld = 0;                  // alten Zaehlerstand merken
9
10 void setup()
11 {
12   Serial.begin(9600);                   // Serielle Schnittstelle öffnen
13   pinMode(ledPin, OUTPUT);              // LED Pin daher OUTPUT
14   attachInterrupt(0, ISRledToggeln, RISING); // Interrupt=0, ISR-Name,
15 }  // setup                              Interrupt bei steigender Flanke
16
17 void loop()
18 {
19   //  beliebiger komplexter Code
20   if (ISRCounter != ISRCounterOld) // ISR Zaehler geaendert ?
21   {
22     Serial.println(ISRCounter);    // Dann Zaehler anzeigen
23     ISRCounterOld = ISRCounter;    // Alter gleich aktueller Zaehlerstand
24   }  // If
25 }  // loop
26
27 void ISRledToggeln()                // die Interrupt Fkt ISR
28 { // Immer wenn die Taste gedrückt wird, wird diese Fkt aufgerufen
29   ISRCounter++;                    // Jeder ISR Aufruf wird gezaehlt
30   digitalWrite(ledPin, ledStatus);
31   ledStatus = !ledStatus;          // Toggeln
32 } // fct ISR
```

Interrupt0_Taster_steuertLED_Toggle_SWEntpr

```
1  /*   Taster per Interrupt lesen und LED toggeln lassen   */
2
3  #define BOUND 20                          // Entprell Zeit
4  const int ledPin = 13;                    // extra oder eingebaute LED
5  const int tasterPin = 2;                  // Taster am int0
6  volatile byte ledStatus = 0;              // Led Status wird von ISR getoggelt
7
8  void setup()
9  {
10   pinMode(ledPin, OUTPUT);                // LED Pin daher OUTPUT
11   attachInterrupt(0, ISRledToggeln, RISING); // Interrupt=0, ISR-Name,
12 }  // setup                                Interrupt bei steigender Flanke
13
14 void loop()
15 {
16                                           //  beliebiger komplexter Code
17 }  // loop
18
19 void ISRledToggeln()                      // die Interrupt Fkt ISR
20 { // Immer wenn die Taste gedrückt wird, wird diese Fkt aufgerufen
21   for (int i=0;i<BOUND; i++)              // Schalter entprellen
22   {                                       // indem kurz gewartet wird
23     for (int j=0;j<BOUND; j++);           // bis wieder eine Flanke
24   }                                       // zugelassen wird
25   digitalWrite(ledPin, ledStatus);
26   ledStatus = !ledStatus;                 // Toggeln
27 } // fct ISR
```

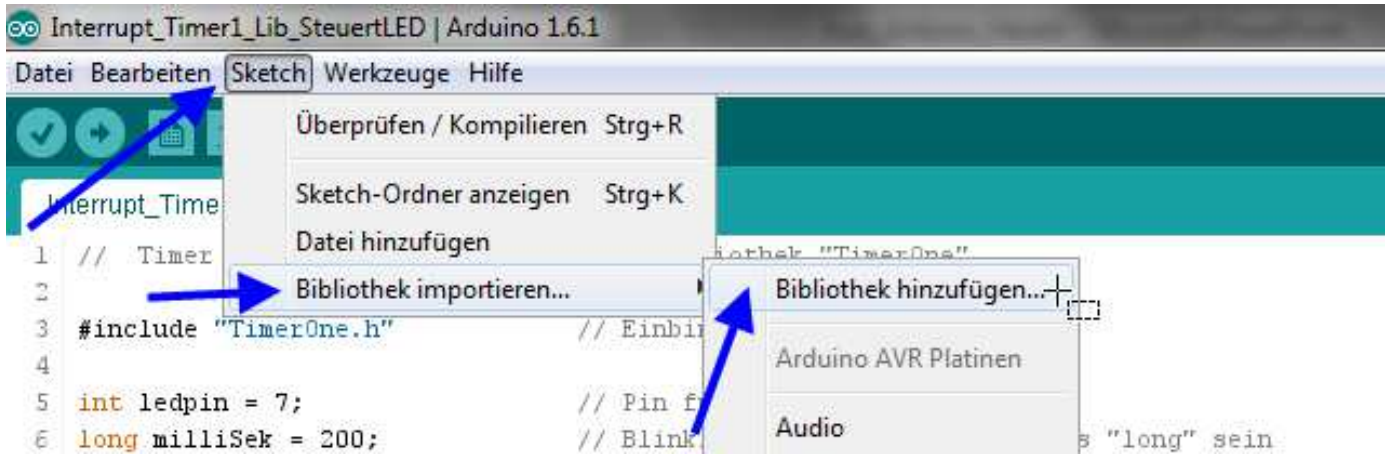
Timer Interrupts:

Hiermit ist es möglich zeitgesteuerte Abläufe automatisch immer wieder auszuführen. Dazu wird abhängig von eingebauten Arduino Timern immer wieder eine Interrupt Service Routine (ISR) aufgerufen. (Bei Erreichen der „0“ bzw. „MAX Wert“

Das kann man selber programmieren, aber besser man verwendet fertige Bibliotheken. Diese können in der Regel noch mehr, z.b auch flexible PWM Signale ansteuern.

Im Internet dieses ZIP File runterladen und über

TimerOne	10.04.2015 22:23	ZIP-komprimierte...	3 KB
TimerOne-r11	10.04.2015 23:11	ZIP-komprimierte...	12 KB



Timer Interrupts

mit eingebautem Arduino „Timer1“ & „TimerOne.h“ Bibliothek

Auszug aus dem Include File:

links ältere **rechts neuere Version**

Board	TimerOne PWM Pins	TimerThree PWM Pins
Teensy 3.1	3, 4	25, 32
Teensy 3.0	3, 4	
Teensy 2.0	4, 14, 15	9
Teensy++ 2.0	25, 26, 27	14, 15, 16
Arduino Uno	9, 10	
Arduino Leonardo	9, 10, 11	5
Arduino Mega	11, 12, 13	2, 3, 5
Wiring-S	4, 5	
Sanguino	12, 13	

```
class TimerOne
{
  public:

    // properties
    unsigned int pwmPeriod;
    unsigned char clockSelectBits;

    // methods
    void initialize(long microseconds=1000000);
    void start();
    void stop();
    void restart();
    void pwm(char pin, int duty, long microseconds=-1);
    void disablePwm(char pin);
    void attachInterrupt(void (*isr)(), long microseconds
    void detachInterrupt();
    void setPeriod(long microseconds);
    void setPwmDuty(char pin, int duty);
    void (*isrCallback)();
};

extern TimerOne Timer1;
```

```
class TimerOne
{
  public:

    // properties
    unsigned int pwmPeriod;
    unsigned char clockSelectBits;
    char oldSREG;                    // To hold Status Register while ints disabled

    // methods
    void initialize(long microseconds=1000000);
    void start();
    void stop();
    void restart();
    void resume();
    unsigned long read();
    void pwm(char pin, int duty, long microseconds=-1);
    void disablePwm(char pin);
    void attachInterrupt(void (*isr)(), long microseconds=-1);
    void detachInterrupt();
    void setPeriod(long microseconds);
    void setPwmDuty(char pin, int duty);
    void (*isrCallback)();
};

extern TimerOne Timer1;
```

Configuration

Timer1.initialize(microseconds);
Begin using the timer. This function must be called first. "microseconds" is the period of time the timer takes.

Timer1.setPeriod(microseconds);
Set a new period after the library is already initialized.

Run Control

Timer1.start();
Start the timer, beginning a new period.

Timer1.stop();
Stop the timer.

Timer1.restart();
Restart the timer, from the beginning of a new period.

Timer1.resume();
Resume running a stopped timer. A new period is not begun.

Interrupt Function

Timer1.attachInterrupt(function);
Run a function each time the timer period finishes. The function is run as an interrupt, so special care is needed to share any variables between the interrupt function and your main program.

Timer1.detachInterrupt();
Disable the interrupt, so the function no longer runs.

PWM Signal Output

Timer1.pwm(pin, duty);
Configure one of the timer's PWM pins. "duty" is from 0 to 1023, where 0 makes the pin always LOW and 1023 makes the pin always HIGH.

Timer1.setPwmDuty(pin, duty);
Set a new PWM, without reconfiguring the pin. This is slightly faster than pwm(), but pwm() must be used at least once to configure the pin.

Timer1.disablePwm(pin);
Stop using PWM on a pin. The pin reverts to being controlled by digitalWrite().

Interrupt_Timer1_Lib_SteuertLED

```
1  //  Timer Interrupt-Steuerung mit der Bibliothek "TimerOne"
2  //  wird eine LED geblinkt. Parallel werden die ISR Aufrufe gezählt
3  //  und bei Änderung angezeigt, jedoch max alle "x" Sekunden
4
5  #include "TimerOne.h"                          // Einbinden der Bibliothek
6  #define  ANZEIGE_ZEIT 1000
7
8  int ledpin = 7;                                // Pin für die Blink LED
9  long milliSek = 200;                           // Blinkintervall in mSek muss "long" sein
10 volatile int  ISRCounter = 0;                  // zählt alle ISR Aufrufe
11 int ISRCounterOld = 0;                         // alten Zaehlerstand merken
12
13 void setup()
14 {
15   Serial.begin(9600);                          // Serielle Schnittstelle öffnen
16   pinMode(ledpin, OUTPUT);
17   Timer1.initialize(milliSek*1000);            // Timer1 in MicroSek. einstellen
18   Timer1.attachInterrupt(blinken);             // ISR Funktion zuordnen
19 }  // setup
20
21 void blinken()
22 {
23   ISRCounter++;                                // Jeder ISR Aufruf wird gezaehlt
24   digitalWrite(ledpin, digitalRead(ledpin) ^ 1);  // Toggeln mit bitwise EXOR
25 }  // ISR
26
27 void loop()
28 {
29   // Hier kann nun ganz normal euer Programm stehen
30   // Die Led wird nun unabhaengig von eurem Programmcode
31   // jede Sekunde einmal aufblinken. Hier wird z.B. nur alle "x" Sekunden
32   // angezeigt wie oft die USR aufgerufen wurde
33
34   if (ISRCounter != ISRCounterOld)             // ISR Zaehler geaendert ?
35   {
36     Serial.println(ISRCounter);                // Dann Zaehler anzeigen
37     ISRCounterOld = ISRCounter;                // Alter gleich aktueller Zaehlerstand
38   }  // If
39   delay(ANZEIGE_ZEIT);                         // Trotz Delay funktioniert Interrupt
40 }  // loop
```

Interrupt_HW0_Pin2_und_TimerInterrupt_Lib

```
1  /*  1) Tastenflanke (Änderung)  per HW Interrupt0 am Pin2 erkennen, dann Taster Status lesen
2  und LED ein- bzw. ausschalten.
3  2) Ferner eine LED am beliebigen Pin (hier Pin7) per "Timer Interrupt" blinken lassen.
4  Dazu wird die Library --> "TimerOne.h" verwendet.
5  Beide Funktionen laufen wegen Interrupts unabhängig vom Hauptprogramm "loop"
6  Ist bei zeitkritischen Funktionen wichtig bzw. wenn Hauptprgramm mit anderen Dingen beschäftigt ist */
7
8  #include "TimerOne.h"                          // Bibliothek einbinden
9
10 int ledPinBlinken = 7;                         // Pin für Blink LED per Timer Interrupt
11 long milliSek = 200;                           // Blinkintervall muu "long" sein
12
13 const int ledPinTaster = 13;                   // 13 ist auch eingebaute LED
14 const int tasterPin = 2;                       // Tasteränderung wird durch "int0" == Pin2 erkannt
15
16 void setup()
17 {
18   pinMode(ledPinBlinken, OUTPUT);              // Pin als Ausgang setzen
19   Timer1.initialize(milliSek*1000);            // Blink Dauer einstellen, Wert in Micro Sekunden, daher * 1000
20   Timer1.attachInterrupt(ISR_LEDblinken);      // ISR übergeben. Fkt wird autom. bei jedem Interrupt aufgerufen
21   pinMode(ledPinTaster, OUTPUT);               // Pin als Ausgang setzen
22   attachInterrupt(0, ISR_LEDschalten, CHANGE); // Interrupt Nr, ISR und Flankenart übergeben.
23 }  // setup
24
25 void ISR_LEDblinken()                                  // ISR für Timeer Interrupt
26 {
27   digitalWrite(ledPinBlinken, digitalRead(ledPinBlinken) ^ 1);   // Toggeln der LED mit bitwise EXOR
28 }  // ISR Timer Interrupt
29
30 void ISR_LEDschalten()                                 // ISR für HW Interrupt0
31 {
32   digitalWrite(ledPinTaster, digitalRead(tasterPin));  // Taster Status lesen, LED EIN- oder AUS schalten
33 }  // ISR Hardware Interrupt
34
35 void loop()                                     // beliebiger komplexter Code
36 {                                               // Hier kann normal euer Programm stehen. Die Led wird
37                                                 // nun unabhaengig von eurem "Loop Code" alle "x" Milli-Sek.
38 }  // loop                                     //  einmal aufblinken. Und der Taster steuert die 2. LED
```

Timer1_Lib_PWM_Control

```
1  // Bsp erzeugt ein PWM signal mit variabler Frequenz und variablem Tastverhältnis.
2  // Arduino's analogWrite() gives you PWM output, but no control over the
3  // carrier frequency. The default frequency is low, typically 490 or
4  // 3920 Hz. Hier wird gezeigt wie man beides ändern kann
5  #include <TimerOne.h>
6
7  #define  REPEAT_TIME  2000          // Wartezeit für erneuter Programm Start
8  #define  DUTYCYCLE_LOOP_TIME  500   // Wartezeit für PWM Duty Cycle Schleife
9
10 const int LedPin = 9;               // PWM Pin 9 oder 10 beim Arduino
11 float dutyCycle = 0;                // Gleitkommavariable für Duty Cycle Berechnung
12
13 void setup(void)
14 {
15   Timer1.initialize(2000000);       // Angabe in MicroSek, 1/Sek = Frequenz
16   Serial.begin(9600);
17 }  // setup
18
19 void loop(void)
20 {
21   for (dutyCycle = 1.0; dutyCycle < 100.0; dutyCycle++)  // Tastverhältnis von 1-100 hochzählen
22   {
23     Serial.print("PWM LED, Duty Cycle = ");
24     Serial.println(dutyCycle);
25     Timer1.pwm(LedPin, (dutyCycle * 1023.0) / 100);      // Fct ".pwm" setzt PinNr & DutyCycle 0-1023
26     delay(DUTYCYCLE_LOOP_TIME);                          // kurz warten, damit man was sieht
27   }  // for
28   delay(REPEAT_TIME);                                    // Warten, dann beginnt Programm von vorne
29 }  // loop
```

Technikschule Esslingen; Kurs: Arduino
ZUSATZ
Elektronik - Grundlagen

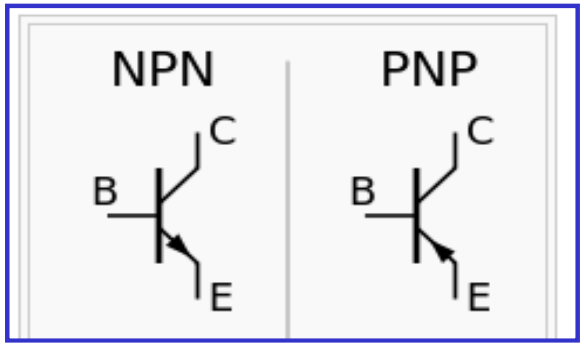
Eine Auswahl an diskreten Transistoren in verschiedenen Gehäuseformen

Ein **Transistor** ist ein elektronisches Bauelement zum Schalten und Verstärken von elektrischen Signalen, ohne dabei mechanische Bewegungen auszuführen. Transistoren sind die wichtigsten „aktiven“ Bestandteile elektronischer Schaltungen. Besondere Bedeutung haben Transistoren in integrierten Schaltkreisen, in der Mikroelektronik.

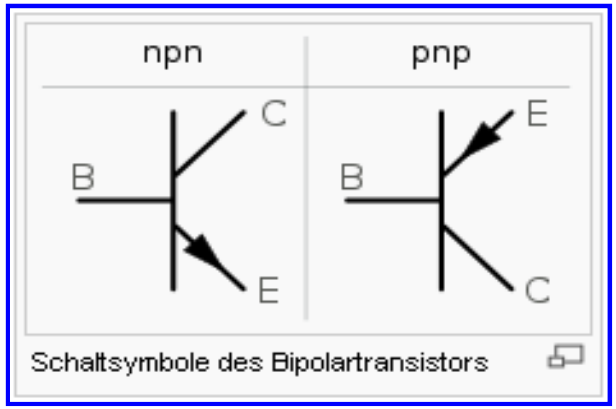
Der Begriff „Transistor“ ist eine Kurzform des englischen ***TRANS**FER RES**ISTOR***, was in der Funktion einem durch eine angelegte elektrische Spannung oder elektrischen Strom steuerbaren elektrischen Widerstand entspricht.

Der erste funktionierende Bipolartransistor wurde in Form eines Spitzentransistors bei den Bell Laboratories entwickelt und am 23. Dezember 1947 bei einer internen Demonstration erstmals präsentiert. Die an der Erfindung des Bipolartransistors beteiligten W. Shockley, J. Bardeen und W. Brattain erhielten dafür 1956 den Nobelpreis für Physik

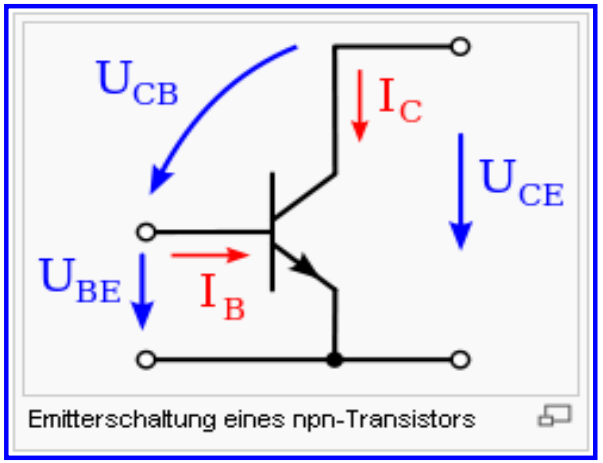
Moderne integrierte Schaltungen, wie die in Personal Computern eingesetzten Mikroprozessoren, bestehen aus vielen Millionen bis Milliarden Transistoren.



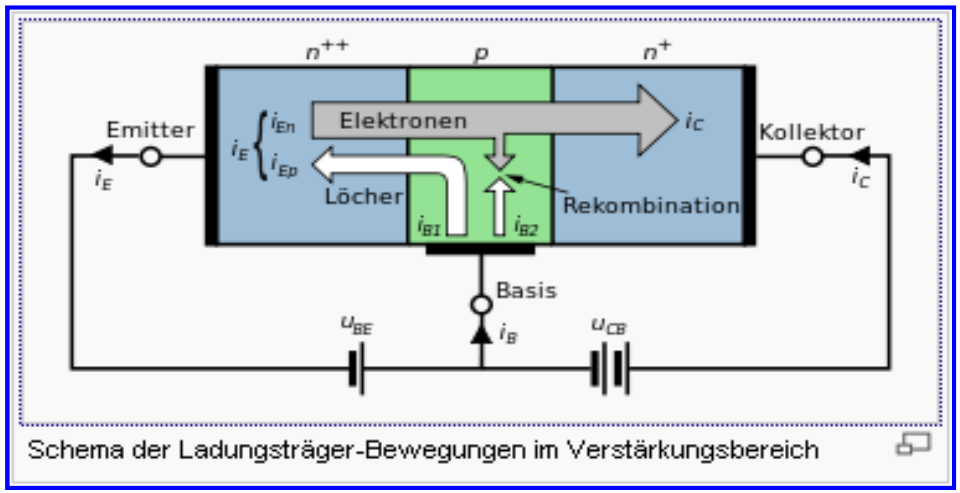
Es gibt zwei wichtige Gruppen von Transistoren, nämlich **Bipolartransistoren (NPN) und (PNP)** und **Feldeffekttransistoren (FET)**, die sich durch die Art der Ansteuerung voneinander unterscheiden.



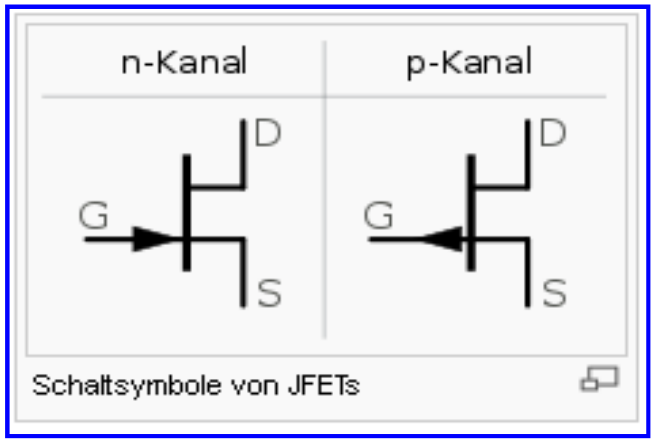
Schaltsymbole des Bipolartransistors



Emitterschaltung eines npn-Transistors

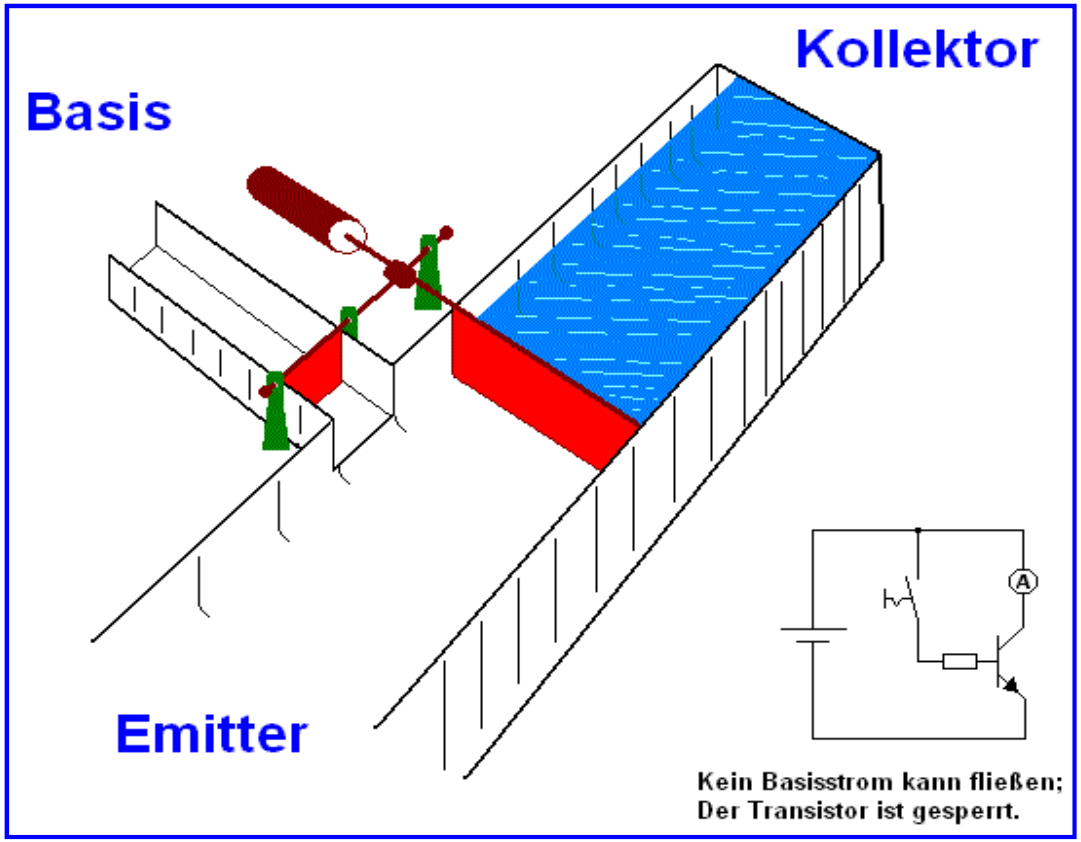


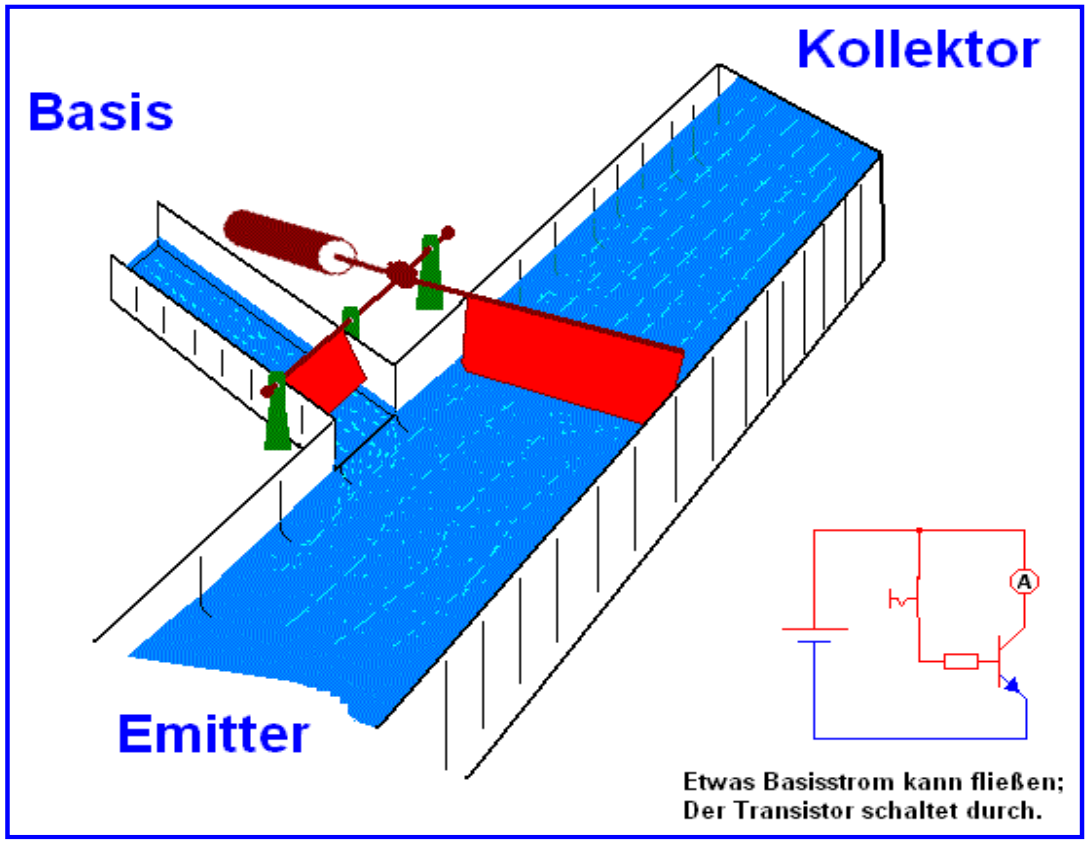
Schema der Ladungsträger-Bewegungen im Verstärkungsbereich



Schaltsymbole von JFETs

Prozessortyp	Anzahl der Transistoren	Technologieknoten	Entwicklungsjahr
Intel 4004	2.300	10.000 nm	1971
Intel Pentium (P5)	3.100.000	800 nm	1993
Intel Core 2 (Yorkfield) pro Die	410.000.000	45 nm	2007
Intel Itanium 2 Tukwila	2.046.000.000	65 nm	2010
AMD Tahiti XT[27]	4.312.711.873	28 nm	2011
Nvidia Kepler GK110[28]	7.100.000.000	28 nm	2012





Beim Bipolartransistor steuert ein elektrischer Strom I_{BE} zwischen Basis und Emitter einen stärkerer Strom I_{CE} zwischen Kollektor und Emitter. Das Verhältnis der beiden Ströme, das im Bereich von 4 bis 1000 liegt, ist vom Transistortyp und vom Kollektorstrom abhängig und wird als statischer Stromverstärkungsfaktor *B* bezeichnet.

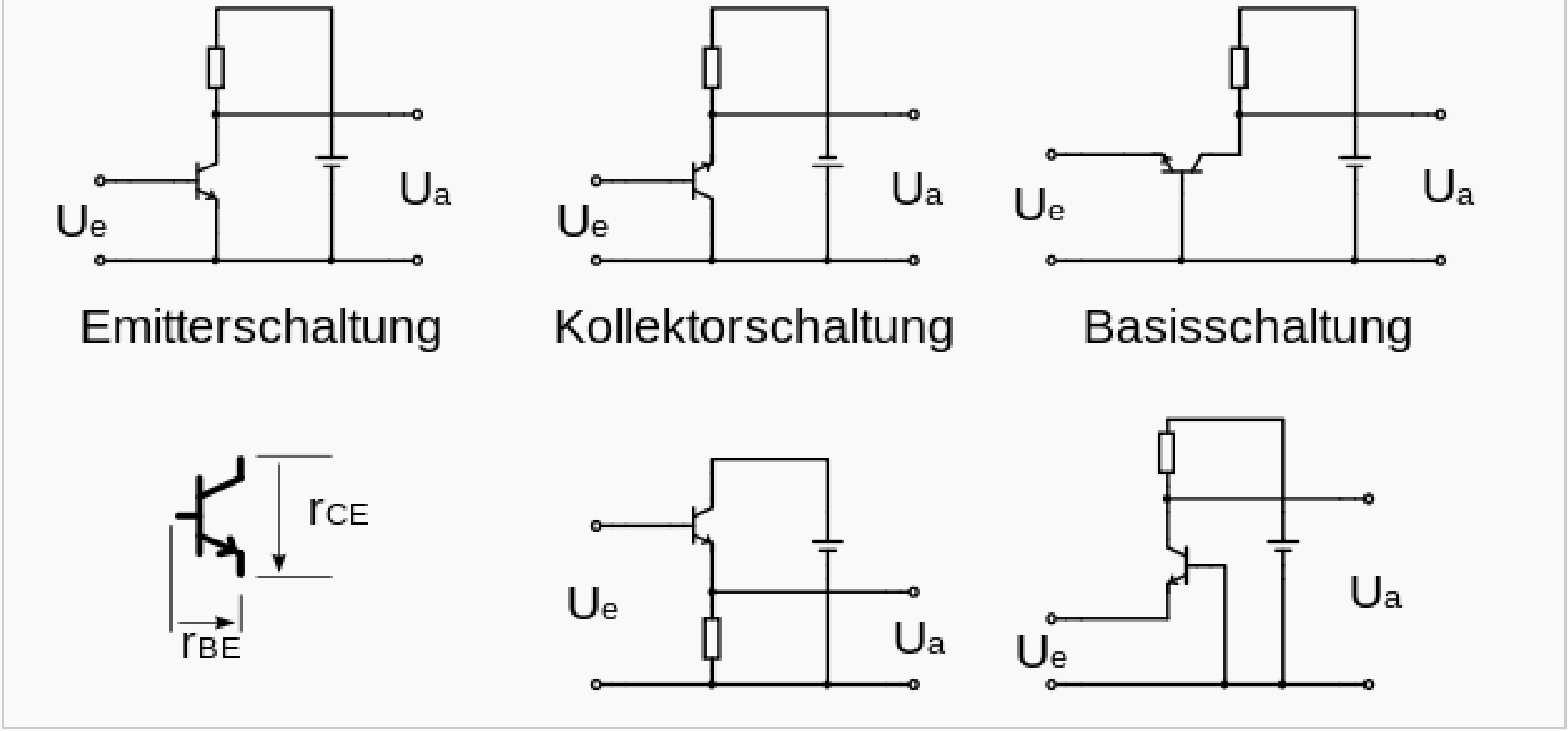
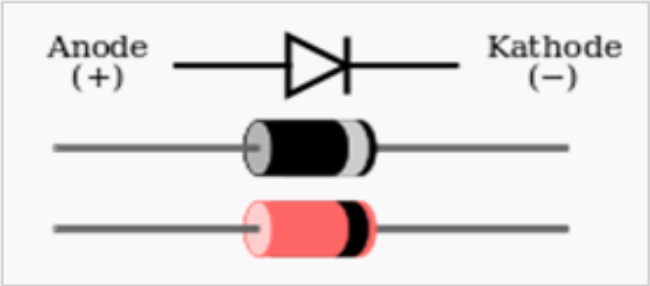


Abb. 1: Transistorgrundschaltungen (Emitter, Kollektor, Basis-Grundschaltung)

TO-92-Gehäuse (bedrahtet): ***BC547(B)* (npn-Transistor)** / ***BC557(B)* (pnp-Transistor)**: Verlustleistung Pmax = 0,50 W; Kollektorstrom IC ≤ 100 mA; Sperrspannung UCE ≤ 45 V; Stromverstärkung B ≈ 290 (bei IC = 2 mA). Der größer der Buchstaben-Index () A, B oder C je größer die Stromverstärkung!

Eine **Diode** ist ein elektrisches Bauelement, das Strom nur in einer Richtung passieren lässt und in der anderen Richtung sperrt, also als Isolator wirkt.

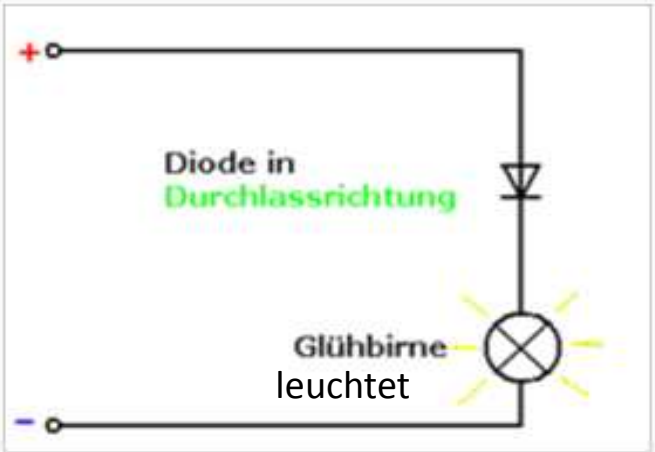
(Gleichrichter-)Dioden in verschiedenen Bauformen



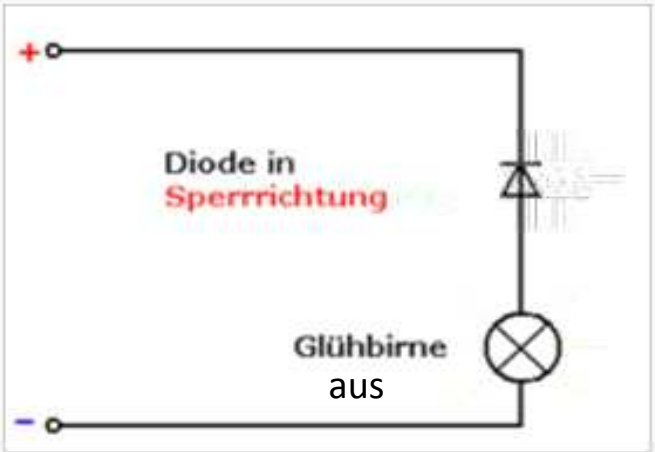
Schaltzeichen einer Diode und Abbildungen üblicher Gehäuse mit Markierung der Kathode. Kennzeichnung der positiven (+) und negativen (−) Seite bei Beschaltung in Durchlassrichtung.

Daher wird von Durchlassrichtung und Sperrrichtung gesprochen.

Bei Wechselstrom bewirkt die Diode aufgrund dieser Eigenschaft eine Gleichrichtung, also eine Umwandlung in Gleichstrom



Das bewirkt eine Diode



Das bewirkt eine Diode

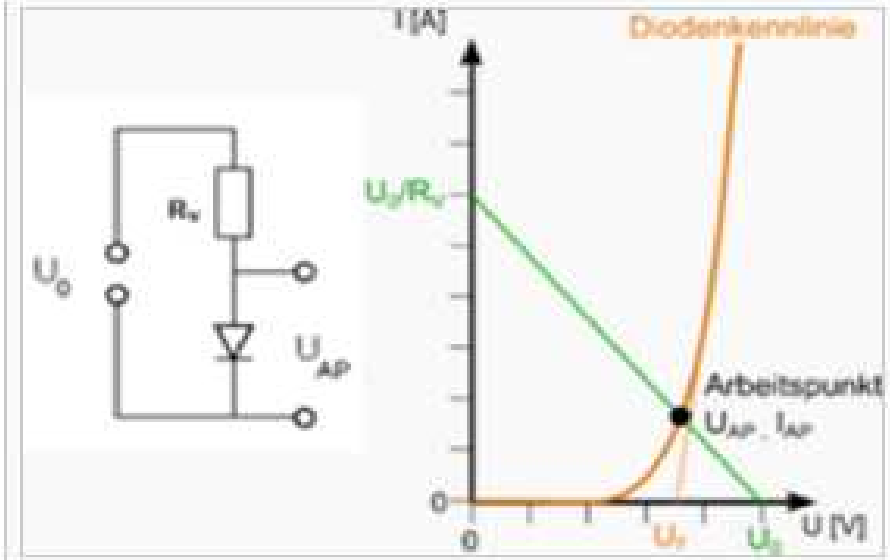
Die einfachste Möglichkeit der Versorgung einer LED an einer Spannungsquelle (Batterie) ist, in Reihe zu ihr einen Vorwiderstand zu schalten.

Es lässt sich der gewünschte Strom *I*, also die Helligkeit, über die Wahl des Widerstandes einstellen:

$$R = \frac{U_0 - U_{\mathrm{LED}}}{I}$$

Beispiel: $U_0 = 4{,}5\ \mathrm{V},\ \ I = 0{,}35\ \mathrm{A},\ \ U_{\mathrm{LED}} = 3{,}4\ \mathrm{V}$

$$R = \frac{4{,}5\ \mathrm{V} - 3{,}4\ \mathrm{V}}{0{,}35\ \mathrm{A}} = 3{,}1\ \Omega$$

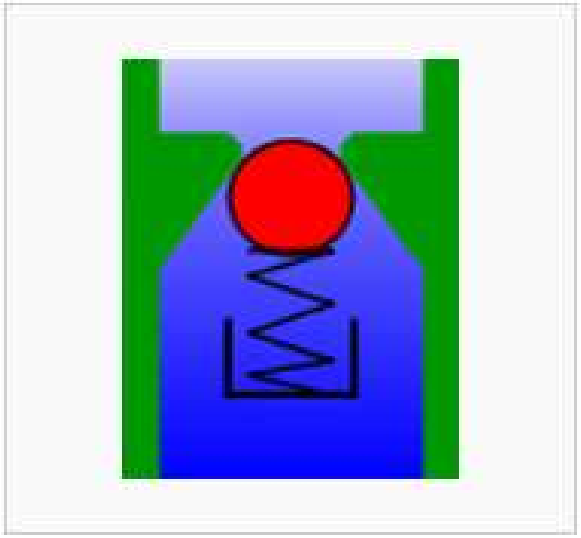


Bestimmung des Arbeitspunktes bei bekannter Diodenkennlinie

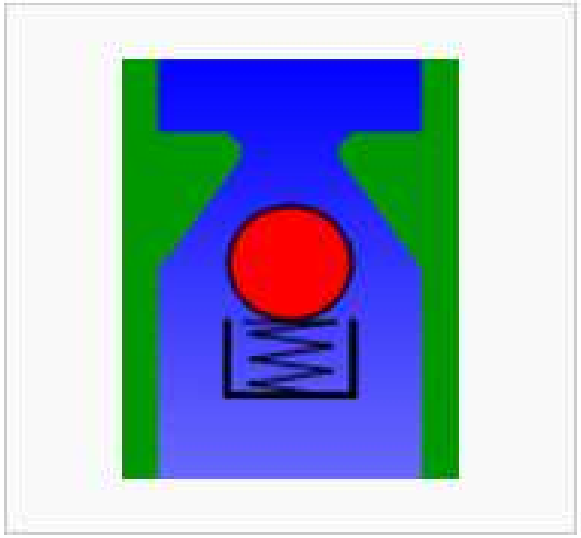
Die Funktion einer Gleichrichterdiode im Stromkreis kann man sich am einfachsten wie ein Rückschlagventil im Wasserkreislauf vorstellen:

Wenn Druck (Spannung) auf dieses Ventil (Diode) in Sperrrichtung erfolgt, so wird der Wasser-(Strom-)fluss blockiert. In der Durchlassrichtung muss der Druck (Spannung) groß genug werden, um die Federkraft des Ventils (Schleusen- oder Schwellenspannung der Diode) überwinden zu können. Dadurch öffnet das Ventil (Diode) und der Strom kann fließen. Diesem Druck, welcher im mechanischen Modell zum Überwinden der Federkraft notwendig ist, entspricht bei einer Diode die so genannte Schwellenspannung (U_S), die in Flussrichtung an der Diode anliegen muss, damit sie in den leitenden Zustand übergeht.

Bei gewöhnlichen Silizium-Dioden liegt diese bei ca. 0,7 V.



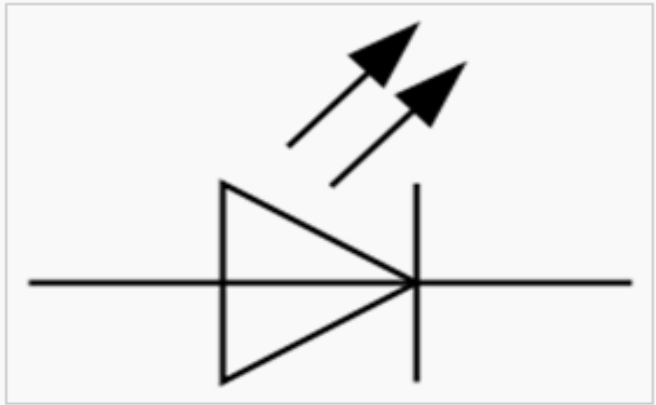
Ein geschlossenes Kugelrückschlagventil.



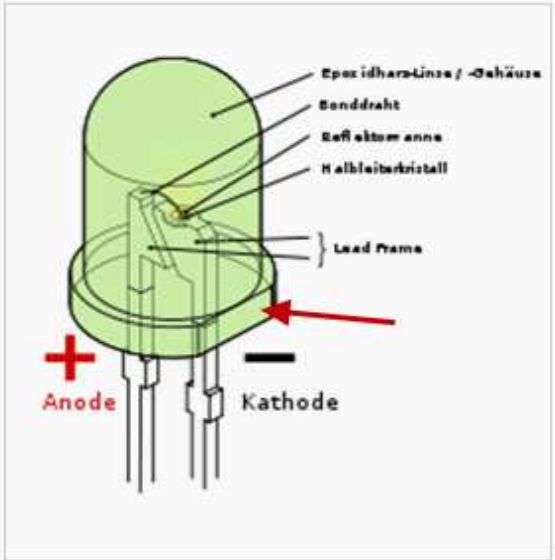
Ein geöffnetes Kugelrückschlagventil.

Neben dem Effekt der Gleichrichtung zeigt ein Halbleiterübergang weite nutzbare Eigenschaften, die z. B. in Zener-, Photo-, Leuchtdioden und Halbleiterdetektoren für Strahlung ausgenutzt werden

Eine **Leuchtdiode** (englisch *light-emitting diode*, **LED**) ist ein elektronisch Halbleiter-Bauelement. Fließt durch die Diode elektrischer Strom in Durchlassrichtung, so strahlt sie Licht, Infrarotstrahlung (als Infrarotleuchtdiode) oder auch Ultraviolettstrahlung mit einer vom Halbleitermaterial und der Dotierung abhängigen Wellenlänge ab



Schaltzeichen einer Leuchtdiode



Aufbau einer 5 mm großen LED. Markierung der Kathode: Gehäuse abgeflacht, Zuleitung im Auslieferungszustand kürzer.

Eine LED in transparentem Kunststoffgehäuse

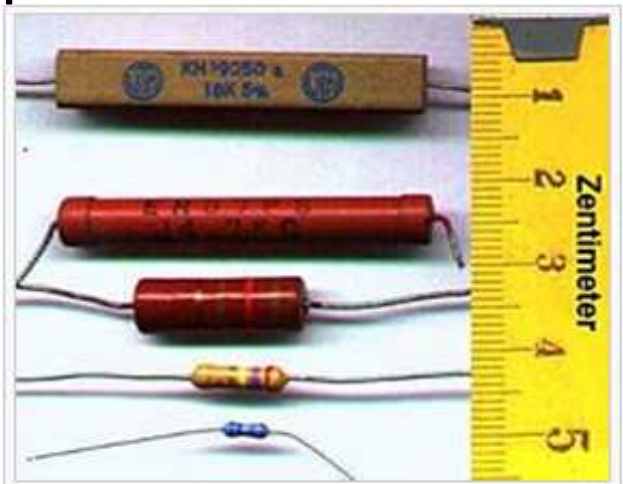
Flussspannungen der LEDs verschiedener Farben

Farbe	Wellenlänge λ in nm	Flussspannung ΔU in V	Werkstoff
Infrarot	λ > 760	ΔU < 1,9	Galliumarsenid (GaAs) Aluminiumgalliumarsenid (AlGaAs)
Rot	610 < λ < 760	1,63 < ΔU < 2,1	Aluminiumgalliumarsenid (AlGaAs) Galliumarsenidphosphid (GaAsP) Aluminiumgalliumindiumphosphid (AlGaInP) Galliumphosphid (GaP)
Orange	590 < λ < 610	2 < ΔU < 2,1	Galliumarsenidphosphid (GaAsP) Aluminiumgalliumindiumphosphid (AlGaInP) Galliumphosphid (GaP)
Gelb	570 < λ < 590	2,1 < ΔU < 2,2	Galliumarsenidphosphid (GaAsP) Aluminiumgalliumindiumphosphid (AlGaInP) Galliumphosphid (GaP)
Grün	500 < λ < 570	2 < ΔU < 3,4	Indiumgalliumnitrid (InGaN) / Galliumnitrid (GaN) Galliumphosphid (GaP) Aluminiumgalliumindiumphosphid (AlGaInP) Aluminiumgalliumphosphid (AlGaP) Zinkoxid (ZnO), in Entwicklung
Blau	450 < λ < 500	2,48 < ΔU < 3,7	Zinkselenid (ZnSe) Indiumgalliumnitrid (InGaN) Siliziumkarbid (SiC) Silizium (Si) als Träger, in Entwicklung Zinkoxid (ZnO), in Entwicklung
Violett	400 < λ < 450	2,76 < ΔU < 4,0	Indiumgalliumnitrid (InGaN)
Ultraviolett	230 < λ < 400	3,1 < ΔU < 4,4	Diamant (C) Aluminiumnitrid (AlN) Aluminiumgalliumnitrid (AlGaN) Aluminiumgalliumindiumnitrid (AlGaInN)[8]

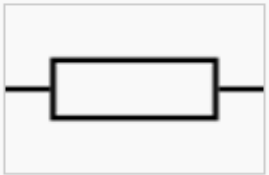
LED-Lampe in gleicher mechanischer Bauform wie eine Glühlampe

KFZ-Rückleuchte mit LED-Technik

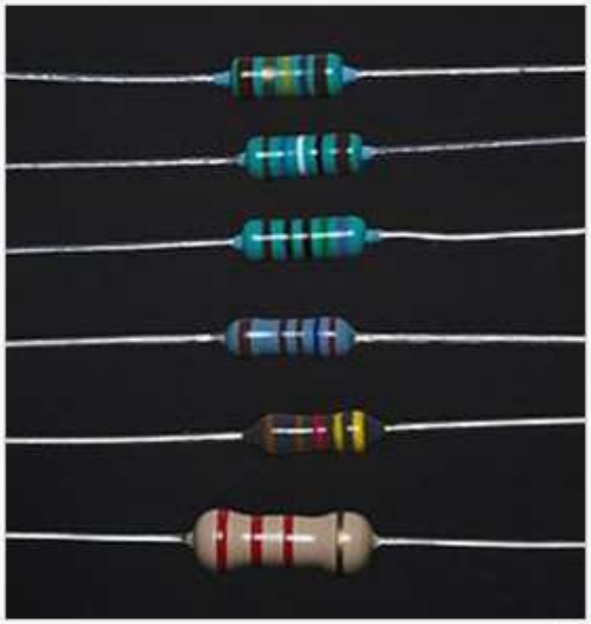
Der elektrische Widerstand: Er wird verwendet um **Strom und Spannung zu begrenzen** oder auf einen bestimmten gewünschten Wert einzustellen.



verschiedene Bauformen


Schaltzeichen für elektrischen Widerstand nach DIN EN 60617

Schaltzeichen für elektrischen Widerstand nach ANSI


Bedrahtete Widerstände mit verschiedenen Werten

OHM: der elektrische Widerstand wird in der **Einheit Ohm** gemessen

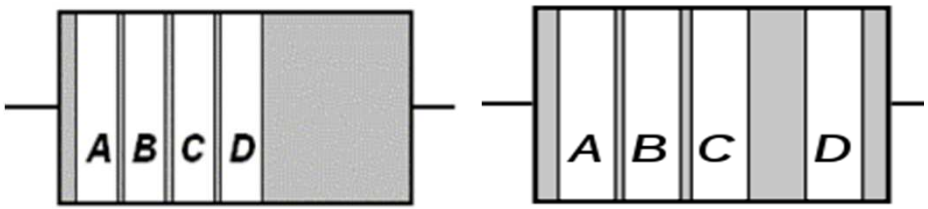
- milli Ohm
- Ohm
- Kilo Ohm
- Mega Ohm.

Ω

Ein 2-MΩ-SMD-Widerstand in der Baugröße 1206 (Raster in mm)

Das ohm'sche Gesetz: **U = R x I**

Spannung = Widerstand mal Strom



Die **Farbkodierung für Widerstände** ist eine Kennzeichnung für die elektrischen Werte von kleinen zylindrischen Widerständen. Um die daraus resultierende Schwierigkeit zu umgehen, lesbare Zahlen unabhängig von ihrer Einbaulage auf der Oberfläche dieser Bauteile abzubilden, wurde der umlaufende Aufdruck von Farbkennringen erdacht.
Es gibt Farbcodes mit vier, fünf oder sechs Ringen.
Bei vier Ringen geben die ersten beiden Ringe die Zahlenwerte an (siehe Tabelle), der dritte Ring gibt den Multiplikator (0 ≙ $\times 10^0 = \times 1$, 1 ≙ $\times 10^1 = \times 10$, 2 ≙ $\times 10^2 = \times 100$ usw.) und der vierte Ring gibt die Toleranzklasse an.

Farbkodierung von Widerständen mit 4 Ringen

Farbe		Widerstandswert in Ω			Toleranz
		1. Ring (1. Ziffer)	2. Ring (2. Ziffer)	3. Ring (Multiplikator)	4. Ring
„keine“	✕	—	—	—	±20 %
silber		—	—	$10^{-2} = 0{,}01$	±10 %
gold		—	—	$10^{-1} = 0{,}1$	±5 %
schwarz		—	0	$10^0 = 1$	—
braun		1	1	$10^1 = 10$	±1 %
rot		2	2	$10^2 = 100$	±2 %
orange		3	3	$10^3 = 1.000$	—
gelb		4	4	$10^4 = 10.000$	—
grün		5	5	$10^5 = 100.000$	±0,5 %
blau		6	6	$10^6 = 1.000.000$	±0,25 %
violett		7	7	$10^7 = 10.000.000$	±0,1 %
grau		8	8	$10^8 = 100.000.000$	±0,05 %
weiß		9	9	$10^9 = 1.000.000.000$	—

Zur kompakten Beschriftung von Widerstandswerten kann der Buchstabe „R“ als Dezimaltrennzeichen verwendet werden:

- 10R = 10 Ω
- 1R5 = 1,5 Ω
- R005 = 0,005 Ω = 5 mΩ

Auch werden SI-Präfixe als Dezimaltrennzeichen verwendet. Der Wert des Präfix stellt dabei einen zusätzlichen Multiplikator dar.
Diese Darstellungsform wird vor allem in Schaltplänen eingesetzt:

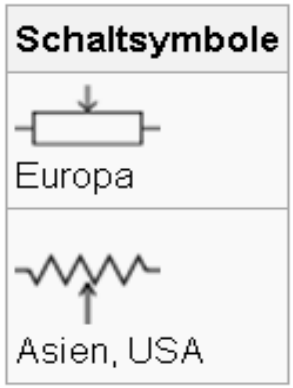
- 10k = 10 kΩ
- 1M5 = 1,5 MΩ
- 0k5 = 0,5 kΩ = 500 Ω

Die **Widerstands-Nennwerte** werden nach geometrischen Folgen abgestuft, die E-Reihen.

Je nach Toleranz können Widerstände mit Werten aus der E12- (10 %), E24- (5 %), E48- (2 %) oder E96-Reihe (1 %) hergestellt werden.
Die Prozent-Zahlen geben Mindestgenauigkeiten für die jeweilige Reihe an.
Bsp. sind die Werte der Reihe E12 = {10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, 82}.
Die Werte sind so gewählt, dass sich überlappende Toleranzbereiche ergeben.

Einfacher ausgedrückt: Die E-Reihen kennzeichnen, wie viele Widerstände pro Dekade, (z. B. von 100 bis 1 kΩ) vorhanden sind. E12 hat z.B. zwölf Widerstände.

Der verstellbare Widerstand:



Ein Bauteil, mit dessen Hilfe sich ein beliebiger Widerstandswert zwischen zwei Grenzwerten einstellen lässt, heißt Potentiometer, auch „Poti“ genannt. Es besitzt drei Anschlüsse: die zwei des einfachen Widerstand und einen dritten Schleifer-Anschluss für das Abgreifen des eingestellten Widerstandswertes.
Potentiometer sind für häufiges Verstellen geeignet.
Hauptanwendung sind Pegeleinstellung oder Erfassung einer lateralen oder Winkelposition.
Trimmwiderstände/Trimmpotentiometer (geringe Leistung) und Stellwiderstände (große Leistung) sind nur für gelegentliches Verstellen geeignet.

Spezielle Widerstände:

Parameterabhängige Widerstände werden auch als nichtlineare Widerstände bezeichnet. Wesentliches Merkmal ist, dass der Widerstandswert von physikalischen Parametern wie Spannung, Temperatur, Druck, Licht etc., abhängt. Wichtig ist, dass bei nichtlinearen Widerständen der Zusammenhang zwischen „Spannung am“ und „Strom durch“ den Widerstand ***nicht*** durch die ohmsche Beziehung mit einem konstanten Widerstandswert *R* beschrieben werden kann! (sie haben also keinen festen Widerstandswert)

Beispiele:

PTC-Widerstände (Kaltleiter, positiver Temperaturkoeffizient): der Widerstandswert steigt mit steigender Temperatur.

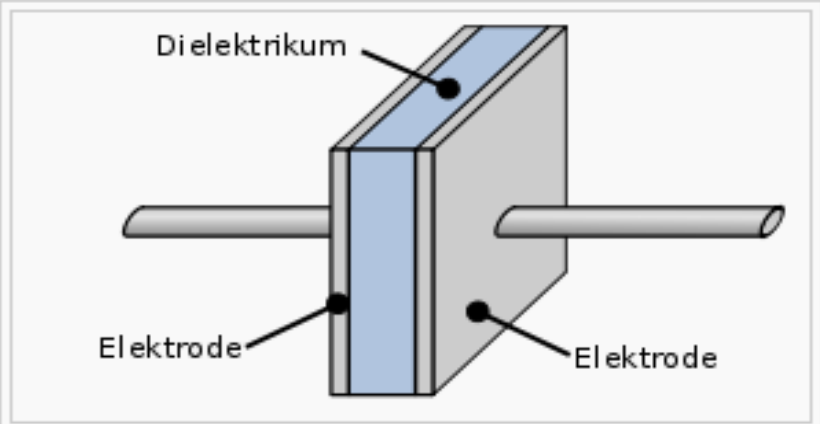
NTC-Widerstände (Heißleiter, negativer Temperaturkoeffizient): der Widerstandswert sinkt mit steigender Temperatur.

Ein Fotowiderstand ändert seinen Widerstand unter Lichteinwirkung. Trifft Licht auf die fotoempfindliche Fläche des Fotowiderstands, verringert sich der Widerstand durch den inneren fotoelektrischen Effekt.

Spannungsabhängige Widerstände auch Varistoren genannt bestehen aus Metalloxiden. Sie verringern ihren Widerstandswert bei steigender Spannung plötzlich, ähnlich einer Zener-Diode (jedoch für beide Polaritäten). Sie werden zur Begrenzung von Überspannungsimpulsen eingesetzt.

Dehnmessstreifen sind Folienwiderstände, die aufgeklebt werden. Sie ändern ihren Widerstandswert in Abhängigkeit ihrer Dehnung beziehungsweise Zugspannung.

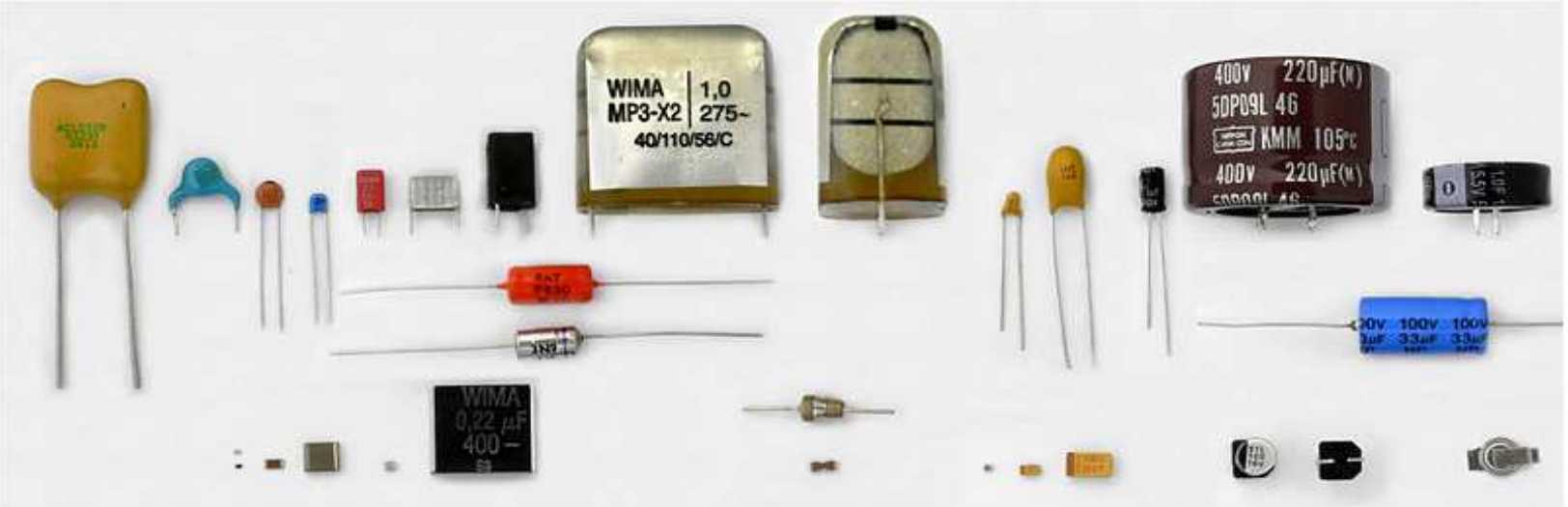
Ein **Kondensator** (von lateinisch *condensare* ‚verdichten') ist ein passives elektrisches Bauelement mit der Fähigkeit, elektrische Ladung und damit Energie zu speichern.
Die Fähigkeit, Ladung zu speichern, wird als elektrische Kapazität bezeichnet und mit Einheit Farad (F) gemessen.

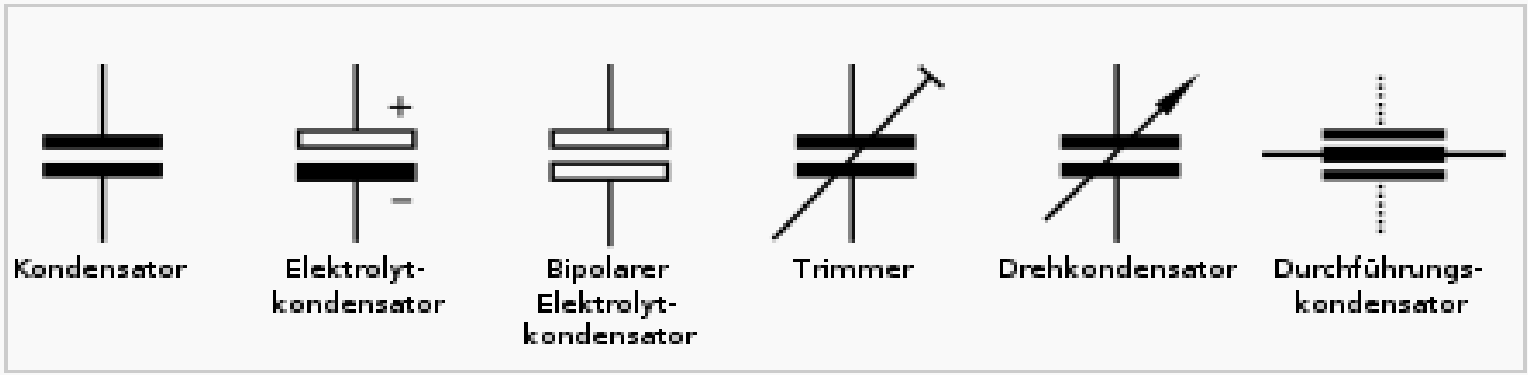


Prinzipdarstellung eines Kondensators mit Dielektrikum

Die technische Ausführung von Kondensatoren besteht aus zwei elektrisch leitenden Flächen in geringem Abstand, den Elektrode auch Kondensatorbeläge genannt.

Dazwischen befindet sich immer ein isolierender Bereich, ein Dielektrikum. Bei den meisten Bauformen werden die Elektroden und das Dielektrikum aufgerollt oder als Stapel angeordnet.



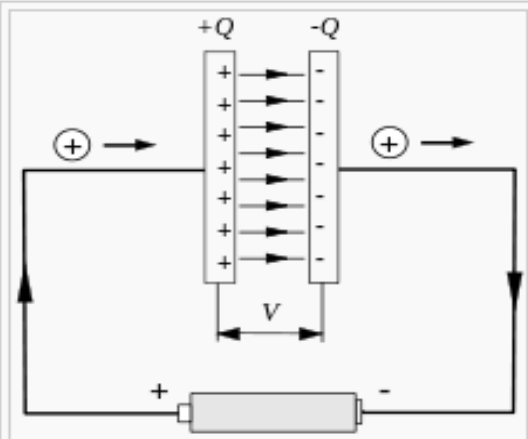


Auswahl verschiedener Schaltzeichen von Kondensatoren nach Typen

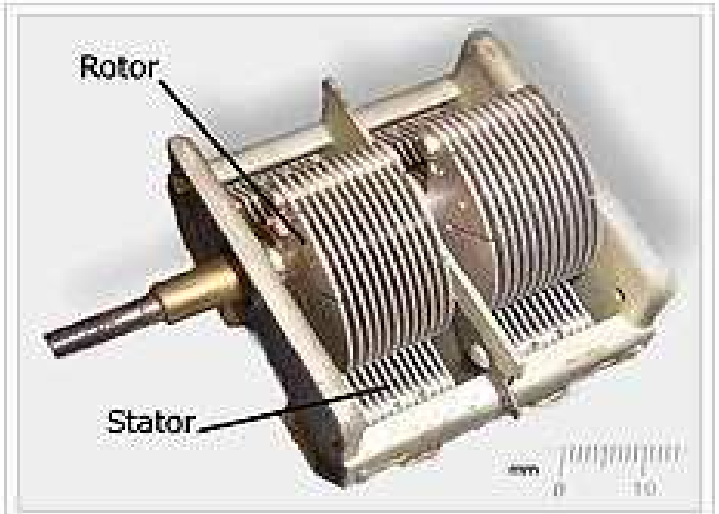
Ein geladener Kondensator speichert elektrische Energie im elektrischen Feld, das *zwischen* den geladenen Platten besteht.
Ist ein Kondensator der Kapazität C
Auf die Spannung *U* geladen, so
enthält sein Feld die Energie *E* gemäß:

$$E = \frac{1}{2} C \cdot U^2$$

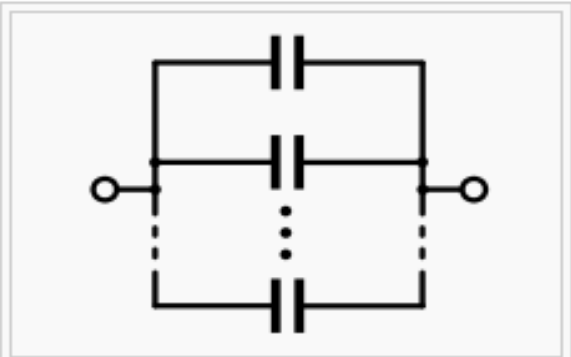
$$Q = C \cdot U$$



Prinzipielle Darstellung des Stromflusses durch einen Kondensator mit der Bildung eines elektrischen Feldes und der Aufladung der Elektroden, wobei eine Spannung V entsteht



Beim Drehkondensator wird das Kondensatorprinzip der sich gegenüberliegenden Elektrodenplatten besonders deutlich. Hier abgebildet ist ein Zweifachdrehkondensator.

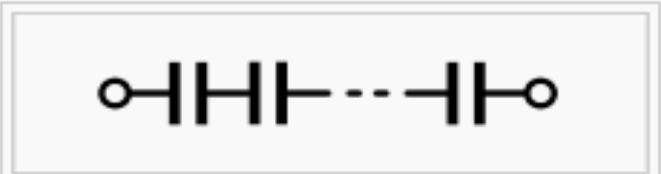


Parallelschaltung von Kondensatoren

Parallelschaltung:

Kondensatoren sind als Parallelschaltung miteinander verbunden, wenn dies Spannung an allen Bauteilen anliegt. Dann addieren sich die Kapazitäten der einzelnen Bauteile zur Gesamtkapazität:

$$C_{\text{ges}} = C_1 + C_2 + \cdots + C_n$$

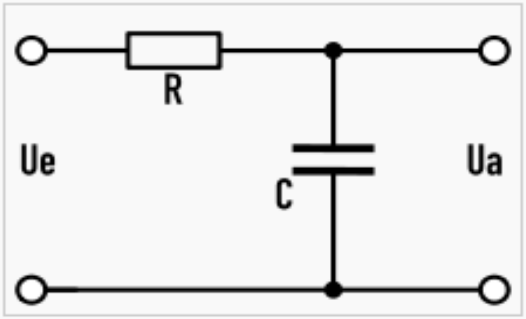


Reihenschaltung von Kondensatoren

Eine Reihenschaltung liegt vor, wenn durch zwei oder mehr Kondensatoren derselbe elektrische Strom fließt. Dann addiert sich der Kehrwert der Kapazität der einzelnen Bauteile zum Kehrwert der Gesamtkapazität (sie wird also kleiner):

$$\frac{1}{C_{\text{ges}}} = \frac{1}{C_1} + \frac{1}{C_2} + \cdots + \frac{1}{C_n}$$

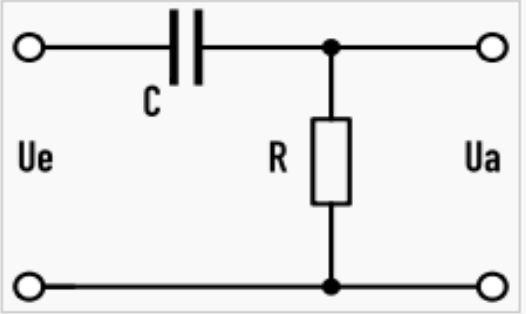
Filteranwendungen:
Die Frequenzabhängigkeit des Widerstands wird benutzt, um Wechselspannungen frequenzabhängig anzuheben oder abzusenken, zu „filtern“. Eine einfache Schaltung ist das RC-Glied, das je nach Schaltung als Hoch- oder Tiefpass wirkt.



Einfacher RC-Tiefpass
Ue: Eingangsspannung
Ua: Ausgangsspannung

Einen Grenzfall des Hochpasses stellt der Koppelkondensator dar, der dazu dient, Gleichströme von überlagerten, Wechselstromanteilen zu trennen. Das ist unter anderem nötig, um den Arbeitspunkt bei Analog-Verstärkern einstellbar zu halten.

Zusammen mit Spulen, die als Kenngröße eine bestimmte Induktivität aufweisen, ohmschen Widerständen und eventuell aktiven Bauelementen werden Kondensatoren in elektronischen Schaltungen auch für Schwingkreise, Bandfilter und Frequenzweichen z.B. in Lautsprechern, verwendet.



Einfacher RC-Hochpass

Abblockkondensatoren:
Keramik-Kondensatoren sind in allen Elektronikschaltungen notwendig, um den Innenwiderstand der Stromversorgung so weit zu vermindern, dass auch hochfrequente Stromimpulse keine unzulässigen Spannungsschwankungen am induktiven Widerstand des Zuleitungsdrahtes hervorrufen. Deshalb müssen Abblock-Kondensatoren direkt mit dem IC verbunden werden

Informationsspeicher

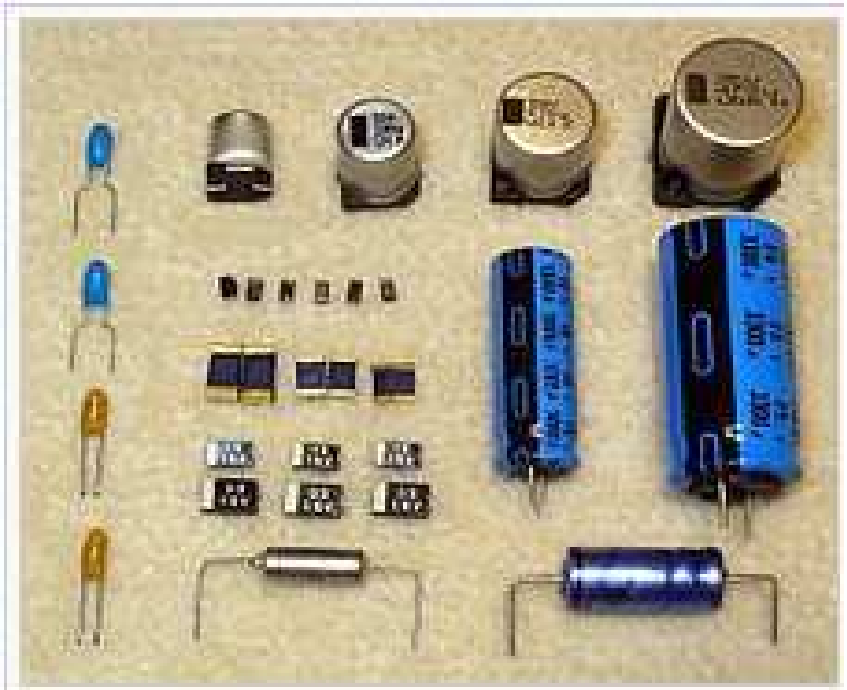
Der Ladungszustand eines Kondensators kann Information in digitaler oder analoger Form repräsentieren.

Analoge Informationsspeicherung mittels eines Kondensators findet beispielsweise in der Abtast-Halte-Schaltung statt: während der Abtastphase wird ein Kondensator mit einer Eingangs-Signalspannung verbunden, von welcher er während der Haltephase getrennt wird. Der Spannungswert steht dann zur Weiterverarbeitung, typischerweise einer Analog-Digital-Wandlung (ADC), konstant zur Verfügung, z.B. CD-Player.
Zur Speicherung von großen **digitalen** Informationsmengen können einige Milliarden Kondensatoren in einer integrierten Schaltung zusammengefasst werden, z.B. dynamische RAM (DRAM) in jedem PC.

Wandler

Die Kondensator-Spannung ist proportional dem Integral des Ladestromes über die Zeit. Auf diese Weise werden Kondensatoren zur Festlegung von Schaltzeiten genutzt, zum Beispiel die Schaltzeiten einer astabilen Kippstufe. Dazu wird ein Kondensator über eine Stromquelle geladen; seine Spannung nimmt proportional zur verstrichenen Zeit zu. Sobald die Spannung einen bestimmten Wert überschreitet, erfolgt ein Zustandswechsel der Schaltung.

Elektrolytkondensatoren (auch Elko genannt) sind gepolte Kondensatoren, dessen Anoden-Elektrode aus einem Metall (Aluminium, Tantal und Niob) besteht, auf dem durch Elektrolyse (anodische Oxidation) eine äußerst dünne, elektrisch isolierende Schicht aus
dem Oxid des Anodenmetalls erzeugt wird, die das Dielektrikum des Kondensators bildet. Der Elektrolyt kann aus einem flüssigen Elektrolyten (Ionenleiter) oder einem festen Elektrolyten (Elektronenleiter) bestehen und bildet die Kathode des Elektrolytkondensators. Die Stromzuführung zum Elektrolyten erfolgt über Folien gleichen Metalls wie das der Anode oder über eine andere geeignete Kontaktierung. Elektrolytkondensatoren sind, mit Ausnahme bipolarer Elektrolytkondensatoren, immer gepolte Bauelemente, die Anode ist der positive Anschluss.



Achtung:
Sie dürfen niemals mit falsch gepolter Spannung betrieben werden (Explosionsgefahr).
Durch gegenpolige Serienschaltung zweier Anodenfolien in einem Kondensatorgehäuse werden für spezielle Anwendungen (zum Beispiel Tonfrequenzweichen) auch Bipolar-Elektrolytkondensatoren für Wechselspannungsbetrieb hergestellt.