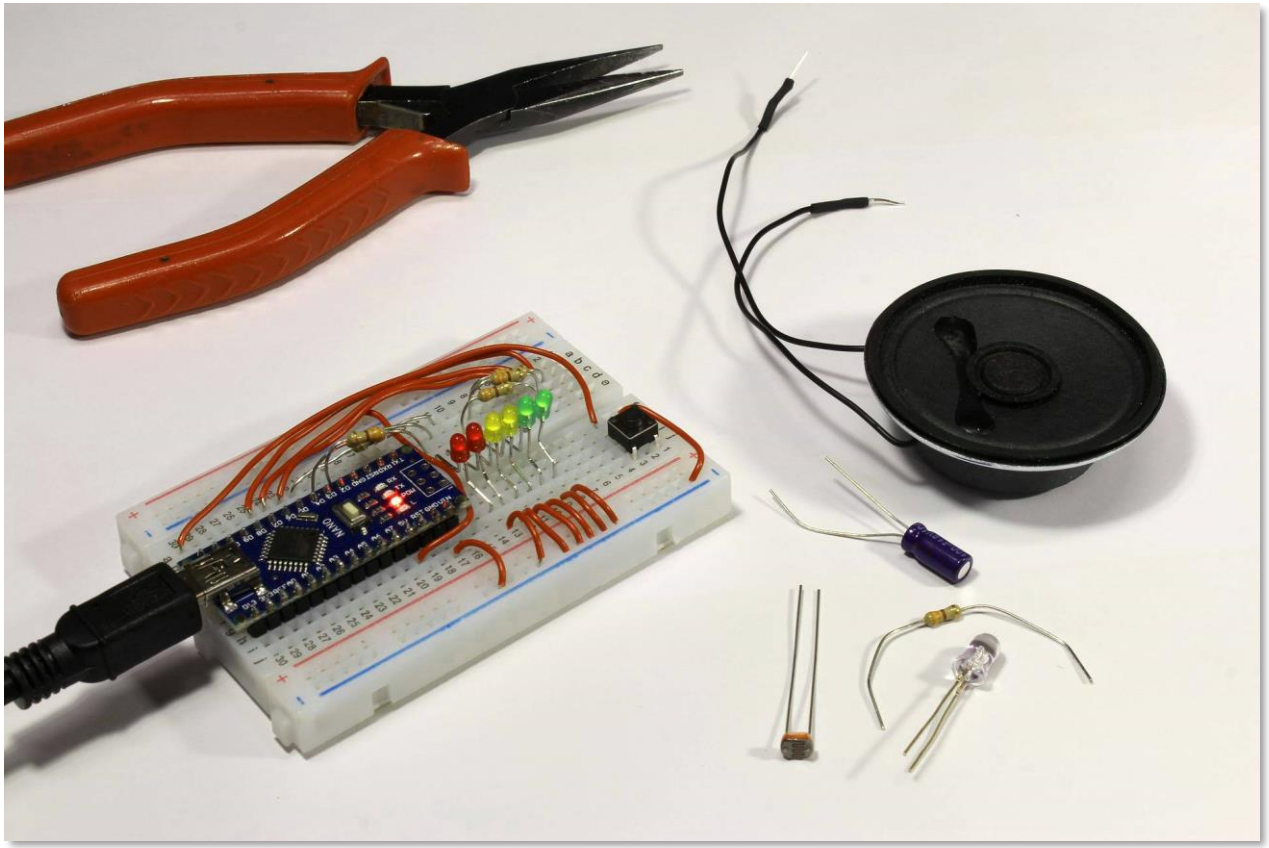
Arduino Kurs – Bits und Bytes

Stephan Laage-Witt
FES Lörrach - 2018



FREIE EVANGELISCHE SCHULE
LÖRRACH

Themen

Digitale Eingabe
Bit, Byte und Wort
Reaktionszeittester

Was ist ein Bit?

Ein Bit ist die kleinste Informationseinheit in der digitalen Informatik. Es kann zwei Zustände annehmen, 0 oder 1. Die Zustände können auch als **`false`** und **`true`**, oder **`LOW`** und **`HIGH`** bezeichnet werden.

Elektrisch werden dazu Spannungspegel verwendet. Eine Spannung < 0.6V wird als logische 0 und eine Spannung > 2.0V als logische 1 ausgewertet. Spannung zwischen 0.6 und 2.0V sollten in der digitalen Elektronik nicht (oder nur sehr kurz) vorkommen.

Die exakte Definition der Spannungspegel hängt vom jeweiligen System ab. Wichtig ist, dass die Grenzwerte eindeutig unter- bzw. überschritten werden

```
boolean flag;

flag = true;

if (flag) {
  // do this
} else {
  // do that
}
```

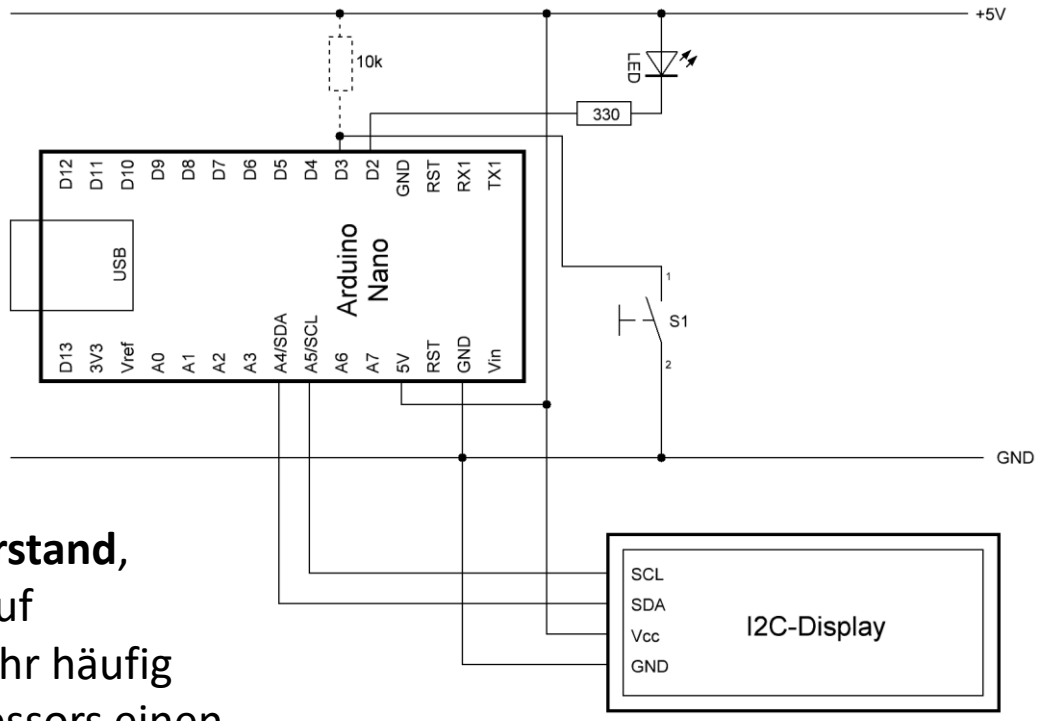
Im Arduino-Framework gibt es den Datentyp **`boolean`** für Konstanten und Variablen, die ein Bit repräsentieren.

gängige Logikpegel (alle Angaben in Volt)

	Eingang		Ausgang	
Technologie	Low (V_{IL})	High (V_{IH})	Low (V_{OL})	High (V_{OH})
TTL 5V	≤ 0,8	≥ 2,0	≤ 0,4	≥ 2,4
CMOS 5V	≤ 1,5	≥ 3,5	≤ 0,5	≥ 4,44
LVTTL 3,3V	≤ 0,8	≥ 2,0	≤ 0,4	≥ 2,4
CMOS 2,5V	≤ 0,7	≥ 1,7	≤ 0,2	≥ 2,3
CMOS 1,8V	≤ 0,7	≥ 1,17	≤ 0,45	≥ 1,2
ECL	≤ −1,4	≥ −1,2	?	?
RS-232(*)	−15 bis −3	+3 bis +15	−15 bis −5[1]	+5 bis +15[1]
HTL 10…30V	≤ 0,2 × U_B	≥ 0,6 × U_B		≈ U_B
(*) = negative Logik, d. h. low=1, high=0				

Digitaler Eingang mit Taster

In dieser Schaltung verwenden wir einen Taster, um ein digitales 0/1-Signal zu erzeugen. Der Taster liegt mit der einen Seite auf Masse. Die andere Seite kommt über einen Widerstand an die +5V-Schiene.



Mit dem Taster kann die Spannung am Widerstand zwischen +5V (Taste offen) und 0V (Taste geschlossen) hin- und her geschaltet werden. Der Port D3 des Arduino dient als digitale Eingabe.

Den Widerstand nennt man **Pull Up-Widerstand**, weil er die Spannung bei offenem Taster auf +5V zieht. Da diese Art der Beschaltung sehr häufig vorkommt, haben die Entwickler des Prozessors einen Pull Up-Widerstand bereits integriert. Er kann über Software ein- oder ausgeschaltet werden. Der Widerstand braucht auf dem Board also nicht gesteckt zu werden und ist deshalb gestrichelt gezeichnet.

Digitale Eingabe mit Taster

DigitaleEingabe

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

#define PIN_LED 2
#define PIN_BUTTON 3

// Set the LCD address to 0x3f for a 20 chars / 4 line display
LiquidCrystal_I2C lcd(0x3f, 20, 4);

/* setup() ------------------------------------------------------------------*/
void setup()
{
  pinMode(PIN_LED, OUTPUT);                    // set Ports
  pinMode(PIN_BUTTON, INPUT_PULLUP);

  lcd.init();                                  // initialize the LCD

  lcd.backlight();                             // activate backlight
  lcd.setCursor(0, 0);                         // set cursor and show title
  lcd.print("Digitale Eingabe");
}

/* loop() -------------------------------------------------------------------*/
void loop()
{
  lcd.setCursor(0, 2);                         // position the text cursor
  if (digitalRead(PIN_BUTTON) == LOW) {        // read input bit from button
    lcd.print("Off");
    digitalWrite(PIN_LED, LOW);                // switch LED off
  } else {
    lcd.print("On");
    digitalWrite(PIN_LED, HIGH);               // switch LED on
  };
  delay(100);                                  // wait a moment
}
```

Dieses Programm testet die digitale Eingabe. Wenn der Taster gedrückt ist, leuchtet die LED und das Display zeigt „Ein“.

`PinMode(PIN_BUTTON, INPUT_PULLUP);`

definiert den Port D3 als Eingabe mit aktiviertem Pull Up-Widerstand.

`digitalRead(PIN_BUTTON);`

fragt den Status des Ports ab und liefert **`LOW`** oder **`HIGH`** als Rückgabewert.

Wichtig: Der Operator **`==`** vergleicht zwei Werte und liefert **`true`**, wenn sie gleich sind. Der Vergleichsoperator darf nicht mit dem Zuweisungsoperator **`=`** verwechselt werden!

8 Bits: Was ist ein `byte`?

Wenn man komplexere Daten als 0/1 verarbeiten möchte, z.B. Zahlen, dann kann man mehrere Bits zusammen nehmen. Die nächst größere Einheit ist die Zusammenfassung von 8 Bits in ein Byte. Jedes Bit hat dabei eine zugewiesene Stelle im dualen Zahlensystem:

MSB = most signifcant bit — LSB = least signifcant bit

Bit-Nummer	7	6	5	4	3	2	1	0
Wert	128	64	32	16	8	4	2	1

Der Wertebereich eines Bytes liegt damit zwischen 0 (alle Bits gelöscht) und 255 (alle Bits gesetzt.

Im Arduino-Framework gibt es den Datentyp `byte` um einen Speicherplatz mit der Größe von genau einem Byte zu deklarieren. Für die Schreibweise der Bits hat sich das Präfix `0b` etabliert.

Hinweise: Oft wird für eine Variable mit der Größe von einem Byte auch der Datentyp `char` verwendet.

```
byte b;

b = 0b00000000;          // 1
b = 0b00000111;          // 7
b = 0b00001000;          // 8
b = 0b10000000;          // 128
b = 0b10101010;          // 170
```

16 Bits: Und was ist ein `word`?

Ein Byte mit seinem beschränkten Wertebereich ist oft nicht ausreichend. Die nächst größere Einheit ist ein `word`, das 16 Bit zusammenfasst.

	MSB															*LSB*
Bit-Nummer	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Wert (ohne Vorzeichen)	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Wert (mit Vorzeichen)	+/-	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

Es gibt zwei Möglichkeiten, den Wert einer 16-Bit-Variablen zu interpretieren:

- Ohne Vorzeichen: Der Wertebereich reicht von 0 (alle Bits gelöscht) bis 65535 (alle Bits gesetzt).
- Mit Vorzeichen: Man kann vereinbaren, dass das höchste Bit als Vorzeichen-Bit interpretiert werden soll. 0 bedeutet positiv, und 1 bedeutet negativ. Dann überstreicht der Wertebereich -32768 bis +32767.

Im Arduino-Framework ist der Datentyp `int` (abhängig vom Prozessor-Typ) eine 16-Bit Variable mit Vorzeichen. Der Datentyp `uint16_t` definiert eine 16-Bit-Variable ohne Vorzeichen.

Achtung: In der Programmiersprache C gibt es keine Warnung vor einem Datenüberlauf! Für eine `int`-Variable ergibt `32767 + 1` das Ergebnis `-1`.

Reaktionszeit-Tester

Mit dem Taster, LED und Display können wir einen Reaktionszeit-Tester programmieren. Hier ist der Programmablauf:

`setup()`

Ports setzen

Titel ausgeben

`loop()`

Message „Fertig?“

Auf Taster warten

Message „Achtung!“

Zwischen 3 und 20 Sekunden warten

    Abbruch, falls Taster gedrückt wurde

Message „jetzt …“, LED einschalten

Millisekunden zählen bis 10000

    Wenn Taster gedrückt, stoppen

LED ausschalten

Message „Ergebnis“ und Zeit ausgeben

Auf Taster warten

Zur Messung der Reaktionszeit zählt das Programm einen Zähler hoch und wartet nach jedem Zählschritt 1 msec, solange, bis die Taste gedrückt ist. Die Anzahl der msec wird als Ergebnis ausgegeben.

Das hier gezeigte Programm ist nicht wirklich exakt. Im nächsten Modul zeigen wir, wie mit der Verwendung eines Timers Zeiten präzise gemessen werden können.

Reaktionszeit-Tester, Teil 1

Reaktionszeit-Tester §

```
// Set the LCD address to 0x3f or 0x27 for a 20 chars / 4 line display
LiquidCrystal_I2C lcd(0x27, 20, 4);

// global variables

/* setup() ------------------------------------------------------------*/
void setup() {
  pinMode(PIN_LED, OUTPUT);                 // set Ports
  pinMode(PIN_BUTTON, INPUT_PULLUP);
  digitalWrite(PIN_LED, LOW);               // switch LED off

  lcd.init();                               // initialize the LCD
  lcd.backlight();                          // activate backlight
  lcd.setCursor(0, 0);                      // set cursor and show title
  lcd.print("Reaktionszeit-Tester");
}

/* loop() -------------------------------------------------------------*/
void loop()
{
  int reac_time = 0;                        // reaction time in msec
  int wait_time = random(17000);            // wait time in msec, initialize with random number
  boolean too_early_flag = false;           // flag for error (key pressed too early)
  int i;

  // preparations
  while (digitalRead(PIN_BUTTON) == LOW);   // wait for button to be released
  lcd.setCursor(0, 2);                      // set cursor and show title
  lcd.print("Fertig?          ");
  while (digitalRead(PIN_BUTTON) == HIGH);  // wait for button

  // run wait time between 3 and 20 secs
  lcd.setCursor(0, 2);                      // set cursor and show title
  lcd.print("Achtung!");
  delay(3000);                              // wait 3 seconds
  for (i = 0; i < wait_time; ++i) {         // wait for a random time (<= 17 secs)
    if (digitalRead(PIN_BUTTON) == LOW) {   // check whether button was pressed
      too_early_flag = true;
      break;
    };
    delay(1);
  };
```

Die Variable **`reac_time`** misst die Reaktionszeit.

Die Variable **`wait_time`** steuert die Wartezeit und wird mit dem Zufallszahlengenerator **`random()`** mit einem Wert zwischen 0 und 17000 initialisiert.

Mit der Variable **`too_early_flag`** wird festgehalten, ob die Taste schon vorher gedrückt wurde.

Das Programm gibt „Fertig?“ aus und wartet auf einen Tastendruck.

Danach gibt das Programm „Achtung!“ aus und wartet zwischen 3 und 20 Sekunden.

Reaktionszeit-Tester, Teil 2

```
  lcd.setCursor(0, 2);                          // set cursor
  if (too_early_flag == false) {

    // run test
    lcd.print("Jetzt ...");
    digitalWrite(PIN_LED, HIGH);                // switch LED on
    for (reac_time = 0; reac_time < 10000; ++reac_time) {
      if (digitalRead(PIN_BUTTON) == LOW) break;
      delay(1);
    };
    digitalWrite(PIN_LED, LOW);                 // switch LED on

    // show result
    lcd.setCursor(0, 2);                        // set cursor
    if (reac_time == 10000) {
      lcd.print("Zu spaet ...");
    } else {
      lcd.print("Ergebnis: ");
      lcd.print(reac_time);
    };

  } else {
    // show error message
    lcd.print("Fehler, zu frueh!");             // print error message
  };
  while (digitalRead(PIN_BUTTON) == LOW);       // wait for button to be released
  delay(1000);

  // done, wait for key pressed to start the next round
  while (digitalRead(PIN_BUTTON) == HIGH);      // wait for button
}
```

Falls die Taste jetzt bereits gedrückt wird, wird das `too_early_flag` gesetzt.

Für die Messung wird die LED eingeschaltet und gewartet, bis die Taste gedrückt ist oder 10 Sekunden verstrichen sind.

Dann wird das Ergebnis ausgegeben.

Schließlich wartet das Programm auf einen Tastendruck und startet wieder von vorne.