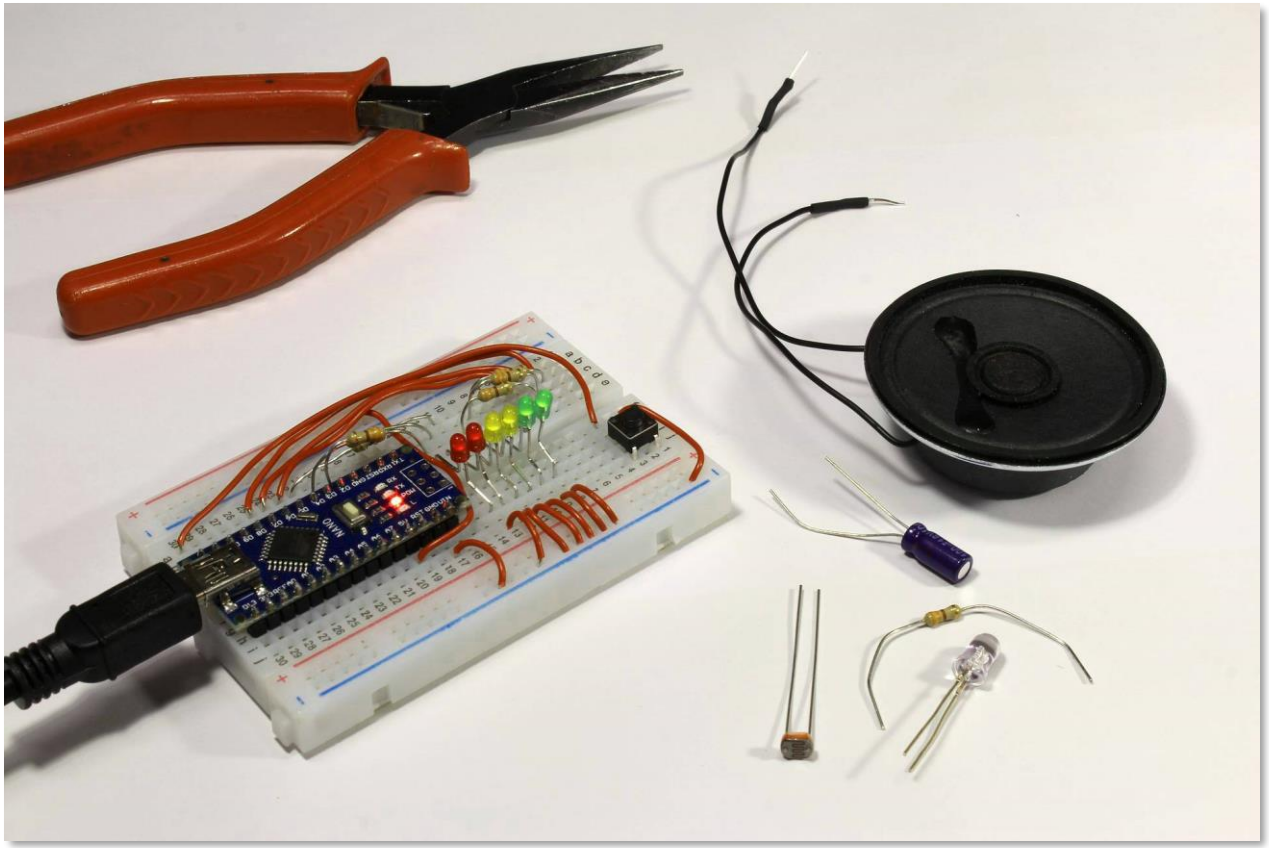
Arduino Kurs – Timer und Interrupts

Stephan Laage-Witt
FES Lörrach - 2018



FREIE EVANGELISCHE SCHULE
LÖRRACH

Themen

Timer
Interrupts
Regelmäßige Aufgaben ausführen
Exakte Zeitintervalle messen

Exakte Zeiten sind gar nicht so einfach!

Das Einhalten von genauen Zeiten ist oft sehr wichtig, wenn man Steuerungen entwickelt und programmiert. Wenn Sekunden nicht genau eingehalten werden, kann man schon nach Minuten falsche Ergebnisse bekommen. Viele Maschinensteuerungen müssen nach Signale nach genauen Zeiten bekommen, um Motoren oder Ventile zu schalten. Andernfalls drohen Fehlfunktionen

Was ist das Problem? Wir haben beim Arduino die `delay()`-Funktion verwendet, um eine definierte Zeit abzuwarten. Aber was genau macht z.B. dieses Programm?

```
void loop() {
  if (digitalRead(led_pin) == HIGH)
    digitalWrite(led_pin, LOW);
  else
    digitalWrite(led_pin, LOW);
  delay(100);
}
```

Die Warte-Zeit von 100 msec wird ziemlich genau eingehalten. Aber die Programmteile davor und danach, also das Auslesen und das Setzen der Ports und der Sprung zurück zum Anfang der Loop-Schleife brauchen auch Zeit. Dieses Programm wird also nicht genau alle 100 msec die LED umschalten, sondern ist etwas langsamer.

Je komplexer der Programm-Code wird, desto schwieriger ist es, mit Programm-Statements alleine genaue Zeiten einzuhalten.

Timer 1

- Aus diesem Grund hat jeder Mikrocontroller Timer „on board“. Timer sind digitale Zähler, die mit einem Quarz-genauen Taktsignal versorgt werden. Sie arbeiten unabhängig von der CPU ganz ohne Software. Man kann sie aber per Software starten und ihren Wert per Software auslesen und verwenden.
- Die Programmierung von Timern ist nicht ganz einfach. Einige Register im Prozessor müssen gesetzt werden, bis ein Timer läuft. Aber zum Glück macht es das Arduino-Framework (wieder einmal) einfach für den Anwender.
- Die CPU des Arduino Nano hat 3 Timer. Für unsere Zwecke verwenden wir den Timer 1. Für den Timer gibt es eine kleine Bibliothek, die eingebunden werden muss:

  http://playground.arduino.cc/uploads/Code/TimerOne.zip

- Sobald die Bibliothek installiert ist, gibt es eine Reihe von neuen Funktionen, mit denen sich Intervalle exakt bearbeiten lassen.
- Im Arduino-Sketch muss mit der Anweisung

  ```
  #include "TimerOne.h"
  ```

  dem Compiler mitgeteilt werden, dass wir die Bibliothek verwenden möchten.

Timer und Interrupt-Funktionen

Zum Timer gehören 3 Komponenten:

1. **Setzen des Zeitintervalls:**
   Die Funktion `timer1.initialize(`**<Zeit in µsec>**`)` initialisiert den Timer. Der Timer läuft dann regelmäßig nach der angegebenen Zahl Mikro-Sekunden (Millionstel Sekunden) ab und wird sofort wieder neu gestartet.

2. **Definition einer Interrupt-Funktion**
   Was soll der Timer nach dem Ablauf der Zeit tun? Das wird mit einer Funktion festgelegt, in der die entsprechenden Programm-Anweisungen stehen. Diese Funktion wird als Interrupt – also als Unterbrechung der normalen Programm-Ausführung – aufgerufen. Nachdem die Interrupt-Funktion abgearbeitet ist, kehrt die Programm-Ausführung wieder dahin zurück, wo sie vor dem Interrupt stand und macht dort unbeirrt weiter. Wir nennen die Interrupt-Funktion `timer_isr()` (isr steht für „Interrupt Service Routine“).

3. **Interrupt-Funktion an den Timer binden**
   Im dritten Schritt muss dem Timer noch mitgeteilt werden, welche Funktion bei Ablauf aufgerufen werden soll. Das geschieht mit der Anweisung:

   `timer1.attachInterrupt(`**<function>**`)`.

LED mit Interrupt blinken lassen

- In diesem Beispiel wird eine LED durch die Interrupt-Routine ziemlich schnell hin und her geschaltet, während das Hautprogramm `loop()` „gemächlich“ Zahlen auf dem LCD anzeigt.
- Damit das Beispiel funktioniert, muss eine LED an den Ausgang D2 des Arduino angeschlossen werden.
- Der Arduino macht also zwei Sachen „quasi“ parallel und unabhängig voneinander.

TimerOneTest§

```
#include <LiquidCrystal_I2C.h>
#include <TimerOne.h>
#define PIN_LED 2

LiquidCrystal_I2C lcd(0x27, 20, 4);
int cnt = 0;

// Arduino setup()
void setup() {
  lcd.init();                       // lcd initialisieren
  lcd.backlight();                  // backlight einschalten
  pinMode(PIN_LED, OUTPUT);         // LED an Pin 2
  Timer1.initialize(100000);        // Timer starten, 10 mal pro Sekunde
  Timer1.attachInterrupt(timer_isr); // Inetrrupt-Routine an Timer 1 anhängen
}

// Arduino loop()
void loop() {
  lcd.setCursor(0, 2);              // Cursor setzen
  lcd.print(cnt);                   // Zahl ausgeben
  ++cnt;                            // Zahl heraufzählen
  delay(1000);                      // ungefähr 1 Sekunde warten
}

// Interupt-Routine, wird genau 10 mal pro Sekunde ausgefhührt
void timer_isr(void) {
  if (digitalRead(PIN_LED) == LOW) {  // falls der Output Low ist
    digitalWrite(PIN_LED, HIGH);      // ... auf High schalten
  } else {
    digitalWrite(PIN_LED, LOW);       // sonts auf Low schalten
  };
}
```

Interrupt und Haupt-programm im Gespräch

- Oft möchte man die Interrupt-Funktion dazu nutzen, dem Hautprogramm mitzuteilen, das es Zeit ist, irgendetwas zu tun. Dafür eignet sich ein „Job-Flag“, also ein Flagge, die von der Interrupt-Routine gehisst wird, wenn es Zeit ist.
- In diesem Beispiel setzt die Interrupt-Routine genau einmal pro Sekunde ein Job-Flag. Diese Information wird im Hauptprogramm verwendet, um eine Ampelsteuerung weiter zu schalten.
- Das Programm geht davon aus, dass rote, gelbe und grüne Leuchtdioden an den Ausgängen D2, D3 und D4 geschaltet sind.
- Die grüne und die rote Phase sind jeweils 4 Sekunden lang. Gelb und gelb/rot sind jeweils 1 Sekunde lang.

AmpelMitTimer §

```
#include <TimerOne.h>
#define PIN_RED 2
#define PIN_YELLOW 3
#define PIN_GREEN 4

volatile boolean job_flag = false;
int ampel_status = 0;

void setup() {
  pinMode(PIN_RED, OUTPUT);           // definiere die LEDs als Output
  pinMode(PIN_YELLOW, OUTPUT);
  pinMode(PIN_GREEN, OUTPUT);
  Timer1.initialize(1000000);         // Timer starten, 1 mal pro Sekunde
  Timer1.attachInterrupt(timer_isr);  // Inetrrupt-Routine an Timer 1 anhängen
}

void loop() {
  if (job_flag) {
    job_flag = false;
    if (ampel_status <= 3) {          // Grün
      digitalWrite(PIN_GREEN, HIGH);
      digitalWrite(PIN_YELLOW, LOW);
      digitalWrite(PIN_RED, LOW);
    } else if (ampel_status == 4) {   // Gelb
      digitalWrite(PIN_GREEN, LOW);
      digitalWrite(PIN_YELLOW, HIGH);
      digitalWrite(PIN_RED, LOW);
    } else if (ampel_status <= 8) {   // Rot
      digitalWrite(PIN_GREEN, LOW);
      digitalWrite(PIN_YELLOW, LOW);
      digitalWrite(PIN_RED, HIGH);
    } else {                          // Rot + Gelb
      digitalWrite(PIN_GREEN, LOW);
      digitalWrite(PIN_YELLOW, HIGH);
      digitalWrite(PIN_RED, HIGH);
    };
    ++ampel_status;                   // nächster Ampel-Status
    if (ampel_status == 10) ampel_status = 0;
  };
};

// Interupt-Routine, wird genau 1 mal pro Sekunde ausgefhührt
void timer_isr(void) {
  job_flag = true;                  // Hallo Hauptprogramm, es gibt Arbeit!
}
```

Zeiten messen mit dem Timer

Timer können auch verwendet werden, um Zeiten zu messen. Dazu kann die Interrupt-Funktion einen Zähler hochzählen, der dann vom Hauptprogramm ausgelesen wird.

Hier ein Beispiel, um die Dauer eines Tastendrucks zu messen. Die Zeit wird in Millisekunden (ms) gemessen und auf dem LCD angezeigt.

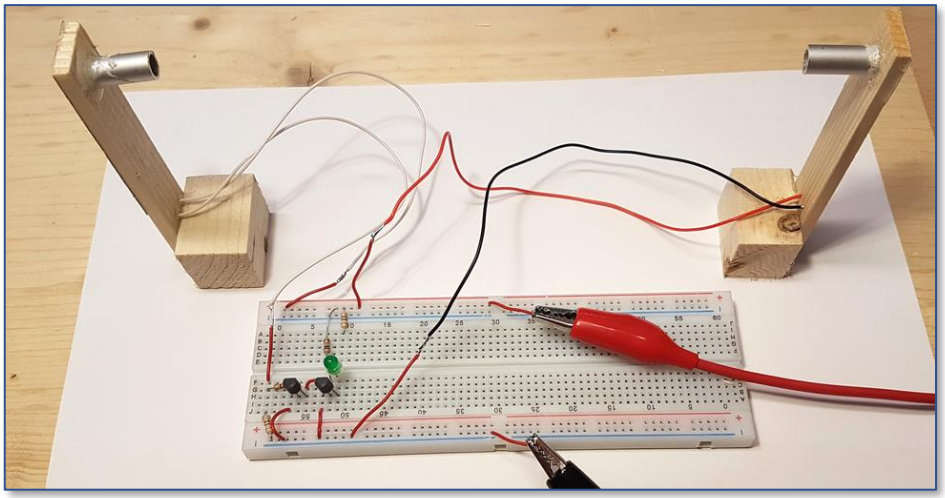
Es werden zwei Taster nach Masse an D4 und D5 benötigt.

ZeitMessungMitTimer

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

#define PIN_KEY_START 4
#define PIN_KEY_RESET 5

LiquidCrystal_I2C lcd(0x27, 20, 4);
volatile int counter = 0;

void setup() {
  pinMode(PIN_KEY_START, INPUT_PULLUP);   // setze die beiden Tasten als Eingang
  pinMode(PIN_KEY_RESET, INPUT_PULLUP);
  lcd.init();                             // LCD initialisieren
  Timer1.initialize(1000);                // Timer mit 1000us = 1ms Interval-Zeit starten
  Timer1.attachInterrupt(timer_isr);      // Interrupt-Routine an den Timer binden
}

void loop() {
  lcd.setCursor(5, 1);                    // Cursor setzen
  lcd.print(counter);                     // Zählerstand ausgeben
  lcd.print("ms    ");                    // Einheit ausgeben
  if (digitalRead(PIN_KEY_RESET) == LOW)
    counter = 0;                          // wenn Reset gedrückt wurde, Zähler zurück setzen
  delay(250);                             // und Pause machen ... kein Stress!
}

// Interrupt-Routine, wird 1000 mal pro Sekunde aufgerufen
void timer_isr(void) {
  if (digitalRead(PIN_KEY_START) == LOW)
    ++counter;                            // wenn der Start-Taster gedrückt wurde, weiter zählen
}
```

Aufgaben und Übungen

- Entwickle ein Programm für eine Ampelkreuzung, die mit realistischen und exakten Zeiten für die einzelnen Phasen arbeitet.
- Baue den Reaktionszeit-Tester aus dem vorherigen Modul so um, dass die Zeit mit einem Timer gemessen wird. Die Messung soll in Millisekunden erfolgen. Kann man auch genauer messen, z.B. 10tel Millisekunden noch erfassen?
- Eine schöne Anwendung ist die Messung von Geschwindigkeiten.
  - Mit etwas zusätzlicher Elektronik (aber überschaubarem Aufwand) lassen sich zwei Lichtschranken bauen, die an die digitalen Eingänge angeschlossen werden können.
  - Die Lichtschranken werden in einem definierten Abstand montiert. Die erste Schranke setzt einen Zähler auf 0 und startet den Timer. Die zweite Lichtschranke stoppt den Zähler.
  - Aus der Zeit und dem Abstand kann man die Geschwindigkeit errechnen und auf dem LCD-Display anzeigen.
  - Wie schnell sind z.B. Tischtennis-Bälle?



Lichtschranke mit Fotodiode und 2 Transistoren zum Anschluss an den Arduino