

Programmieren ohne ‚delay‘

Eines der einfachsten Programme für einen Mikroprozessor ist der „Blink“-Sketch, der dementsprechend für die meisten Einsteiger Grundlage der ersten eigenen Programme ist:

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Eine LED wird im Sekunden-Takt ein- und ausgeschaltet, die Befehlskette lautet einfach

```
LED einschalten
1000 Millisekunden warten
LED ausschalten
1000 Millisekunden warten ...
```

Danach geht's wieder von vorne los.

Versucht der Programmier-Einsteiger nun, dieses Programm z.B. um weitere LEDs zu erweitern, so stößt er schnell auf die Problematik der Abhängigkeit und erhält auf seine Fragen die lapidare Antwort „schau' Dir ‚BlinkWithoutDelay an' – da ist das Prinzip erklärt“.

Zwar ist m.E. im zugehörigen Tutorial <https://www.arduino.cc/en/Tutorial/BlinkWithoutDelay> das Prinzip tatsächlich recht anschaulich erklärt – der Einsteiger hält den Aufwand aber zunächst nicht für gerechtfertigt, denn das Problem lässt sich ja vielleicht auch mit mehreren ‚delay‘-Befehlen lösen:

Anstelle der Befehlskette

```
Pizza in die Mikrowelle
10 Minuten warten
Pizza aus der Mikrowelle holen
eMail prüfen
```

prüft man z.B. alle 2 Minuten die eMails und kommt zu der Befehlskette

```
Pizza in die Mikrowelle
eMail prüfen
2 Minuten warten
```

```
eMail prüfen  
2 Minuten warten  
eMail prüfen  
2 Minuten warten  
eMail prüfen  
2 Minuten warten  
eMail prüfen  
2 Minuten warten  
Pizza aus der Mikrowelle holen
```

Wenn man sich das bildlich vorstellt, wird schnell klar, dass so niemand arbeitet und dass das spätestens dann nicht mehr funktioniert, wenn tatsächlich eine eMail kommt, oder man parallel dazu noch eine Tasse Tee (drei Minuten) zubereiten möchte.

Schon in dem einfachen ‚Blink‘-Programm steht als Kommentar „the loop function runs over and over again forever“ – und genau darum geht es: Die Ausführung des Programms nicht anzuhalten, sondern ganz gezielt Befehle auszuführen, wenn sie dran sind. Man stellt die Pizza in die Mikrowelle und merkt sich die Startzeit, gießt den Tee auf und merkt sich die Startzeit und prüft regelmäßig, ob die 10 Minuten für die Pizza oder die 3 Minuten für den Tee abgelaufen sind.

Genau das macht ‚BlinkWithoutDelay‘ nun für die LED – es prüft endlos, ob die Zeit bis zum nächsten Umschalten abgelaufen ist und wenn das Ergebnis „ja“ lautet, wird es ausgeführt:

```
void loop() {  
    unsigned long currentMillis = millis();           // aktuelle Zeit  
    bestimmen  
    if (currentMillis - previousMillis >= interval) { // Zeitintervall  
    abgelaufen ?  
        previousMillis = currentMillis;              // Schaltzeitpunkt  
    speichern  
        if (ledState == LOW) {                        // LED umschalten  
            ledState = HIGH;  
        }  
        else{  
            ledState = LOW;  
        }  
        digitalWrite(ledPin, ledState);  
    }  
}
```

Beim Blinken einer LED sieht man nun leider tatsächlich keinerlei Vorteil – das Programm erscheint dem Einsteiger nur viel unübersichtlicher. Ich habe es daher mal für 5 LEDs an einem ATtiny85 erweitert. Für den Einsatz auf einem Arduino sind natürlich auch mehr als 5 LEDs möglich und die Portnummern anzupassen. Das Programm ist nur unwesentlich größer, kann aber nun 5 LEDs unabhängig mit verschiedenen Frequenzen und unsymmetrisch (High-Low-Verhältnis) blinken lassen, weil pausenlos nacheinander für alle LEDs geprüft wird, ob sie geschaltet werden müssen – das ist mit Verwendung von ‚delay‘ so nicht mehr zu realisieren.

```
int ledZahl = 5;                                     // Anzahl der LEDs  
int ledPin[] = {0, 1, 2, 3, 4};                     // Pins der LEDs
```

```
unsigned long currentMillis = 0;
unsigned long previousMillis[] = {0, 0, 0, 0, 0};
int interval[] = {750, 800, 850, 950, 1000};           // Blink-Intervalle
int highlowq[] = {5, 5, 5, 2, 2};
void setup() {
  for (int i = 0; i < ledZahl; i++) {
    pinMode(ledPin[i], OUTPUT);
  }
}
void loop() {
  currentMillis = millis();
  for (int i = 0; i < ledZahl; i++) {      if(currentMillis -
previousMillis[i] > interval[i]) {
    previousMillis[i] = currentMillis;
    digitalWrite(ledPin[i], HIGH);
  }
  else if(currentMillis - previousMillis[i] > interval[i]/highlowq[i]) {
    digitalWrite(ledPin[i], LOW);
  }
}
}
```

Der prinzipielle Aufbau eines Programms ohne ‚delay‘ unterscheidet sich also grundlegend von Befehlsketten mit ‚delay‘ – und deshalb kann man ein Programm auch kaum „mal eben umschreiben“. Bei der Erstellung eines Programms ist es also sinnvoll, schon von Anfang an möglichst ohne ‚delay‘ zu arbeiten – sowohl um das Prinzip zu lernen, als auch um später bei Erweiterungswünschen nicht wieder komplett von vorne beginnen zu müssen.

From:

<https://wiki.modellbahn-anlage.de/> - **Wiki der Modellbahn-Anlage.de**

Permanent link:

<https://wiki.modellbahn-anlage.de/decoder/code/programmieren-ohne-delay>

Last update: **07.05.2025 15:05**

