Arduino verstehen und anwenden

Teil 11: Timer und Counter

Timer und Counter gehören zu den wichtigsten Funktionseinheiten eines Mikrocontrollers. Obwohl ihre Aufgabe vergleichsweise einfach ist, erschließen sie eine Fülle von Anwendungen.

Auch die in den verschiedenen Arduino-Versionen eingesetzten Controllertypen verfügen über diese nützlichen Funktionseinheiten. Im Rahmen dieses Beitrags werden die Counter- und Timereinheiten genauer unter die Lupe genommen und auch in praktischen Anwendungen eingesetzt.

Funktion und Verwendung

Prinzipiell bestehen Counter aus einem controller-internen Zählregister. Wird dieses Zählregister in regelmäßigen Zeitabständen um den Wert eins erhöht oder reduziert, dann entsteht ein sogenannter Timer. Die Aufgabe eines Timers könnte theoretisch auch von einer konventionell programmierten while-Schleife übernommen werden. Diese würde aber den gesamten Programmablauf blockieren und der Arduino könnte keine weiteren Aufgaben parallel dazu ausführen. Das ist natürlich in den meisten Fällen nicht akzeptabel.

Ein Counter dagegen verfügt über eine eigene Hardwareeinheit innerhalb des Controllers. Wenn er einmal gestartet ist, arbeitet er völlig unabhängig von anderen Controllereinheiten. Ein wichtiger Punkt ist dabei, dass der Counter bei einem bestimmten Zählstand einen Interrupt auslösen kann. Counter können sowohl von externen Ereignissen, wie etwa dem Pegelwechsel an einem bestimmten Prozessorpin als auch von internen Signalen gesteuert werden.

Wird der Counter über das interne Taktsignal des Controllers gesteuert, entsteht ein präziser Timer. Dieser erlaubt die sehr exakte Erzeugung von Zeittakten. Die Genauigkeit, mit der diese Takte zu Verfügung stehen, wird letztlich nur durch den Quarzoszillator des Mikrocontrollers bestimmt.

Neben dem Hauptoszillator des Controllers ist häufig auch noch ein sogenannter Uhren-Oszillator vorhanden. Dieser kann mit sehr präzise abgestimmten Uhrenquarzen angesteuert werden. Diese Quarze arbeiten meist mit einer Frequenz von 32.768 Hz (= 2^{15} Hz). Mithilfe dieser Quarze können Uhren realisiert werden, die innerhalb von vielen Monaten nur Zeitabweichungen von wenigen Sekundenbruchteilen aufweisen.

Neben ihrer eigentlichen Funktion als Timer kommen Counter auch bei der Erzeugung von Analogsignalen mithilfe der Pulsweitenmodulation zum Einsatz. Auch hier gestatten sie eine sehr präzise Einstellung der entsprechenden Signale, die mit anderen Mitteln nicht realisierbar wäre. Dieses Thema wird im nächsten Artikel eingehender behandelt.

Einfache Uhr mit programmgesteuerter Verzögerung

In diesem Anwendungsbeispiel wird eine einfache Uhr programmiert. Die Ausgabe erfolgt direkt auf die serielle Schnittstelle. Die Zeitanzeige kann also auf jedem RS232-Terminal ausgegeben werden. Natürlich ist auch der serielle Monitor aus der Processing-IDE dafür einsetzbar.

```
// Simple Clock

int cnt, hours = 17, mins = 30, secs = 00; // hours, minutes, seconds
int LED=13;

void setup()
{ Serial.begin(9600);
  pinMode(LED,OUTPUT);
}

void loop()
{ cnt++;
  if(cnt==50)digitalWrite(LED,LOW);
  if(cnt==100)
  { digitalWrite(LED,HIGH);
    Serial.print("Current Time: ");
    if (hours < 10) Serial.print("0"); Serial.print(hours); Serial.print(":");
    if (mins  < 10) Serial.print("0"); Serial.print(mins); Serial.print(":");
    if (secs  < 10) Serial.print("0"); Serial.print(secs); Serial.println();
    secs++;
    if(secs==60)
    { secs=0; mins++;
      if(mins==60)
      { mins=0; hours++;
        if(hours==24) hours=0;
      }
    }
    cnt=0;
  }
  delay(10);
}
```

Das Prinzip dieser Uhr ist simpel: Es wird einfach kontinuierlich eine Endlosschleife durchlaufen. Die Durchlaufzeit dieser Schleife soll mit

```
delay(10);
```

auf 10 ms festgelegt werden. Bei jedem Schleifendurchlauf wird eine Zählvariable (cnt) um eins erhöht. Immer wenn diese Variable den Wert 50 erreicht hat, wird die LED ausgeschaltet. Wird der Wert 100 erreicht, d. h. nach

100 x 10 ms = 1000 ms = 1 s

wird die LED wieder aktiviert und die Uhr um eine Sekunde weitergeschaltet.

Die LED blinkt also im Sekundentakt und die Uhr sollte nach jeder abgelaufenen Sekunde um eine Sekunde weiter zählen. Nach jeweils 60 Sekunden wird der Minutenwert um eins erhöht, nach 60 Minuten der Stundenwert und nach 24 Stunden wird der Stundenwert auf 0 zurückgesetzt.

Da es üblich ist, Uhrzeiten im Format hh:mm:ss anzugeben, wurde im Programm noch für die „führenden Nullen" gesorgt. Ansonsten würde die Zeit 17:30:05 im eher ungewöhnlichen Format 17:3:5 angezeigt.

Wenn man die Uhr laufen lässt, kann man feststellen, dass soweit alles bestens funktioniert. Bild 1 zeigt die zugehörige Ausgabe der Uhrzeit auf dem seriellen Monitor.

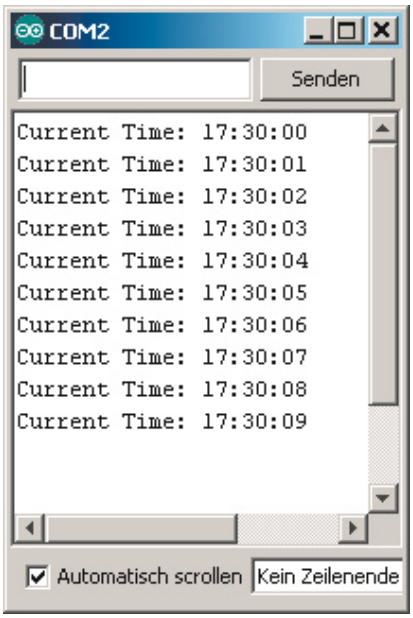

Bild 1: Zeitausgabe im seriellen Monitor

Die Time-Library

Wie im Arduino-Umfeld üblich existiert auch für die Arbeit mit Uhrzeiten eine Library. Der folgende Sketch zeigt, wie diese Library eingesetzt wird.

```
// clock using time lib

char str[2];

#include <Time.h>

void setup()
{ Serial.begin(9600);
  setTime(20, 0, 0, 1, 1, 15); // set time to 20:00:00 at 1. Jan. 2015
}

void loop()
{ sprintf(str,"%02d", hour());    Serial.print(str);  Serial.print(":");
  sprintf(str,"%02d", minute());  Serial.print(str);  Serial.print(":");
  sprintf(str,"%02d", second());  Serial.print(str);  Serial.println();
  delay(1000);
}
```

Die Time-Library kann unter
http://playground.arduino.cc/code/time
kostenlos aus dem Internet geladen werden.

Zusätzlich wurde hier auch noch die Funktion „sprintf" verwendet. Diese gestattet die Formatierung von Ausgaben an die serielle Schnittstelle. Hier sorgt die Angabe „%02" für das einfügen der führenden Nullen. Weitere Details dazu finden sich in [1].

Lässt man die Arduino-Uhren eine gewisse Zeit laufen, so wird man schnell bemerken, dass sie erheblich nachgehen. Warum? Das Problem liegt darin, dass auch die Anweisungen für die Ausgabe auf die Schnittstelle oder die Befehle für die Anzeige der Sekunden und Minuten eine gewisse Zeit erfordern.

Der delay-Wert im ersten Sketch müsste also etwas kleiner als 10 ms sein. Entsprechend müsste der Wert im zweiten Sketch weniger als 1000 ms betragen. Natürlich könnte man jetzt versuchen, durch Vergleich mit einer präzisen Uhr die korrekten delay-Werte zu bestimmen. Dieses „experimentelle" Vorgehen ist allerdings recht aufwendig und wird auch nie zu einem präzisen Ergebnis führen, da die Durchlaufzeiten für die verschiedenen Programmteile stark und kaum vorhersehbar schwanken.

Zeitmessung mit Timern

Wirklich präzise Zeitmessungen sind am besten mithilfe eines Timers umsetzbar. Wie bereits erwähnt ist ein Timer nichts anderes als ein im Controller mit separater Hardware integrierter Zähler. Die Taktfrequenz des Zählers kann direkt aus dem Prozessortakt gewonnen werden. Der Timer läuft dann völlig unabhängig von der Programmausführung mit stabiler Quarzpräzision. Immer wenn der Timer abgelaufen ist, kann ein Interrupt ausgelöst werden. Dieser Interrupt erfolgt damit immer in präzise definierten Zeitabständen.

Die zugehörige Bibliothek kann unter
http://code.google.com/p/arduino-timerone/downloads/list
aus dem Internet geladen werden.

Ein Timer kann so mit zwei Befehlen festgelegt werden:

```
Timer1.initialize(i);
```

und

```
Timer1.attachInterrupt(procedure);
```

Mit Timer1.initialize(); wird der Zeitabstand definiert, in welchem ein Interrupt ausgelöst wird. Die Variable i muss dabei in µs angegeben werden. Für einen Sekundentakt muss i = 1.000.000 µs = 1000 ms = 1 s gewählt werden. Dann wir die in

```
Timer1.attachInterrupt(procedure);
```

festgelegte „procedure" exakt alle 1,000000 Sekunden aufgerufen, unabhängig von irgendwelchen anderen Aktivitäten des Mikrocontrollers.

Präzise Real Time Clock

In diesem Anwendungsbeispiel soll der Timer praktisch eingesetzt werden. In der ersten Programmzeile wird dazu die Funktions-Bibliothek für TimerOne eingebunden:

```
#include "TimerOne.h"
// include timer
```

Nach der Initialisierung des Timerinterrupts mit

```
Timer1.initialize(1000000);
Timer1.attachInterrupt(update_time);
```

wird die update_time()-Routine immer in einem Zeitabstand von exakt 1 Sekunde aufgerufen.

Die Hauptprogramm-Schleife kann in diesem Beispiel wieder völlig leer bleiben.
Wenn man nun die Ganggenauigkeit dieser Uhr mit den einfachen Uhrversionen mit Delay-Zeitverzögerung vergleicht, kann man schnell die deutlich verbesserte Ganggenauigkeit feststellen.

LED 0	LED 1	LED 2	LED 3	LED 4	Dezimalwert
1	1	1	1	1	31
0	1	1	1	1	30
1	0	1	1	1	29
0	0	1	1	1	28
1	1	0	1	1	27
0	1	0	1	1	26
1	0	0	1	1	25
0	0	0	1	1	24
1	1	1	0	1	23
0	1	1	0	1	22
1	0	1	0	1	21
0	0	1	0	1	20
1	1	0	0	1	19
0	1	0	0	1	18
1	0	0	0	1	17
0	0	0	0	1	16
1	1	1	1	0	15
0	1	1	1	0	14
1	0	1	1	0	13
0	0	1	1	0	12
1	1	0	1	0	11
0	1	0	1	0	10
1	0	0	1	0	9
0	0	0	1	0	8
1	1	1	0	0	7
0	1	1	0	0	6
1	0	1	0	0	5
0	0	1	0	0	
1	1	0	0	0	3
0	1	0	0	0	2
1	0	0	0	0	1
0	0	0	0	0	0

Bild 2: Zeitausgabe im Binärformat

```
// precise interrupt controlled real time clock

#include "TimerOne.h"
#include <Time.h>
char str[2];

void setup()
{ Serial.begin(9600);
  setTime(15, 20, 0, 1, 1, 15);
  Timer1.initialize(1000000);     // timeCounter = 1000000 µs = 1.000000 s
  Timer1.attachInterrupt(update_time);
}

void update_time()
{ sprintf(str,"%02d", hour());    Serial.print(str);  Serial.print(":");
  sprintf(str,"%02d", minute());  Serial.print(str);  Serial.print(":");
  sprintf(str,"%02d", second());  Serial.print(str);  Serial.println();
}

void loop() { /* main loop empty!*/ }
```

Countdown-Timer mit LEDs

Natürlich möchte man die Zeit nicht nur am PC anzeigen, hier steht ja ohnehin meist eine integrierte Uhr zur Verfügung. In diesem Beispiel werden deshalb sechs LEDs als Anzeigeelemente verwendet. Um einen möglichst großen Bereich abzudecken, werden zur Anzeige nicht die einzelnen LEDs direkt verwendet, sondern es kommt die binäre Zahlendarstellung zum Einsatz. Damit kann man mit nur sechs LEDs immerhin $2^6 = 64$ verschiedene Zustände anzeigen. Einem Sekunden-Countdown für Zeiten bis über eine Minute steht damit nichts mehr im Weg.

Die LEDs werden der Reihe nach an Port B angeschlossen, d. h. an den Digitalpins 8 bis 13. Die einzelnen Pins werden hier direkt als Port angesprochen:

```
// PORT B as output
DDRB = 0b1111111;
```

definiert alle Portpins des Ports B als Ausgänge. DDR steht dabei für Data DiRection. Weitere Details zur Verwendung der PORT-Anweisung finden sich in [1].

Im Sekundentakt wird dann der Wert seconds per Timerinterrupt um jeweils eins reduziert:

```
void update_time()
{ if (seconds >0) seconds--;
}
```

Der aktuelle Wert von seconds wird im Hauptprogramm direkt als Binärzahl entsprechend dem Muster in Bild 2 ausgegeben.

Aus Platzgründen wurden hier nur die Werte von 0 bis 31 angegeben.

Wenn die Zeit abgelaufen ist, blinken alle LEDs simultan mit einer Frequenz von 10 Hz:

```
if (seconds == 0)
  { PORTB = 0b111111; delay (100); PORTB = 0; delay (100);}
```

Durch die Verwendung der PORT-Befehle fällt das Programm sehr kompakt aus:

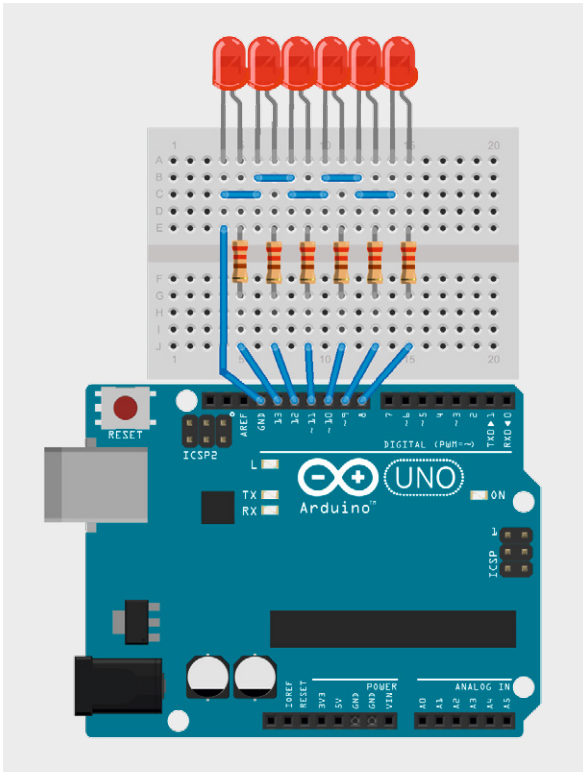

Bild 3: Binärer Countdowntimer

Bild 4: Individuell gestaltete Digitaluhr

Weitere Infos:

[1] Mikrocontroller-Onlinekurs, Franzis-Verlag, exklusiv für ELV, 2011, Best.-Nr. VA-10 20 44, € 99,–

[2] G. Spanner: Arduino – Schaltungsprojekte für Profis, Elektor-Verlag 2012, Best.-Nr. VA-10 94 45, € 39,80

[3] Grundlagen zur elektronischen Schaltungstechnik finden sich in der E-Book-Reihe „Elektronik!" (www.amazon.de/dp/B000XNCB02)

[4] Lernpaket „AVR-Mikrocontroller in C programmieren", Franzis-Verlag 2012, Best.-Nr. VA-09 73 52, € 39,95

Preisstellung Juni 2015 – aktuelle Preise im Web-Shop

```
// binary LED countdown timer

#include "TimerOne.h"          // include timer

double tc = 1000000;           // timeCounter = 1000000 µs = 1.000000 s
int seconds = 60;               // start value

void setup()
{ DDRB = 0b1111111;            // PORT B as output
   Timer1.initialize(tc);
   Timer1.attachInterrupt(update_time);
}

void update_time()
{ if (seconds >0) seconds--;
}

void loop()
{ PORTB = seconds;
  if (seconds == 0)
  { PORTB = 0b111111; delay (100); PORTB = 0; delay (100);}
}
```

Den Aufbau zum Binärtimer zeigt Bild 3.

Digitaluhren mit individueller Gestaltung

Nachdem nun klar ist, wie man mithilfe von Timern präzise Zeiträume messtechnisch erfassen kann, kommt vielleicht der Wunsch auf, eine „richtige" Digitaluhr mit echter Ziffernanzeige anstelle einer Binäranzeige aufzubauen. Diese Aufgabe ist mit dem Arduino durchaus lösbar. Allerdings muss man dafür noch das Problem der Ansteuerung von Siebensegment-Displays lösen. In einem späteren Beitrag zu dieser Reihe werden verschiedene Möglichkeiten aufgezeigt werden, wie eine solche Ansteuerung aussehen kann.

In Kombination mit den hier vorgestellten Timerfunktionen stellt dann aber der Bau von individuellen Digitaluhren kein Problem mehr dar.

Natürlich kann man solche Uhren auch käuflich erwerben. Der Eigenbau hat aber den Vorteil, dass man eigene Ideen realisieren kann. So stehen inzwischen Anzeigelemente in den verschiedensten Formen, Größen und Farben zur Verfügung. Bild 4 zeigt ein Beispiel für eine Eigenbau-Uhr mit blauer Anzeige.

Ausblick

Eine weitere Anwendung von Timern und Countern ist die sogenannte Pulsweitenmodulation (PWM). Sie gestattet es, quasi-analoge Spannungen zu erzeugen und an einen Arduino-Pin auszugeben. Diese Technik wird im Mittelpunkt des nächsten Beitrags zu dieser Artikelserie stehen.

Darüber hinaus werden auch die Möglichkeiten des Einsatzes von Digital-Analog-Konvertern (DACs, für engl. Digital/Analog-Converter) diskutiert werden. Diese bilden das Gegenstück zu den bereits im Artikel VI, „Sensortechnik und Messwerterfassung" besprochenen ADCs. Neben der präzisen Steuerung von Aktoren gestatten DACs auch die Ausgabe von Tönen und Musik.

Empfohlene Produkte/

Bauteile:	**Best.-Nr.**	**Preis**
Arduino UNO	VA-10 29 70	€ 27,95
Mikrocontroller-Onlinekurs	VA-10 20 44	€ 99,–

Alle Arduino-Produkte wie Mikrocontroller-Platinen, Shields, Fachbücher und Zubehör finden Sie unter: www.arduino.elv.de