Gruppe: Leander Stark, Jonas Dollinger Datum: 10.11.2020


Aufgabe 2

- Parametrierung und Verwendung eines Hardware Timers
- Im frei laufenden Betrieb (Free-Running-Mode)
- Abfrage des Timers mittels Polling
- Verbesserung der blockierenden delay Funktion, durch genaue hardware basierte Zeiten

Gruppe: Leander Stark, Jonas Dollinger Datum: 10.11.2020

## Parametrierung und Verwendung eines Hardware Timers:

Der dsPic33 Mikroprozessor besitzt einen Hardwarebasierten 16 Bit Timer, welcher auf Basis des externen (oder wahlweise auch internen) Oszillators funktioniert. Dieser Timer mit 16-Bit (65536 Stellen) inkrementiert bei jeder Periode des Oszillatortaktes um +1. Der Takt des Quarzes liegt bei 32,768 kHz. Damit ergibt sich eine Periodendauer von $T = \frac{1}{f} = 30{,}5\mu s$.

Um nun eine Delay-funktion zu schreiben, welche in Millisekunden zählt, muss zuerst sichergestellt werden dass der 16-Bit Zähler ausreichend Stellen zu Verfügung hat. Um dies zu gewährleisten wurde der Prescaler auf ein Verhältnis von 1:64 eingestellt.

Da die Systemfrequenz des Mikroprozessors sich ändern kann, sollte der Zähler auch unabhängig von dieser sein. Aufgrund dessen wurde der Wert PR1 auf 1ms normiert. Deshalb ergibt sich:

$$PR1 = \left(\frac{FCY}{64000}\right) - 1$$

Am Ende wird von diesem Wert noch die Eins abgezogen, da der Zähler beim Wert von 0 startet und damit jeweils eine Stelle zu weit ist.

## Delay_ms_t1:

```
1.  void delay_ms_t1(uint16_t milliseconds){
2.      int Count = 0;
3.      while(Count < milliseconds){
4.          while (! _T1IF){
5.              Nop();
6.              }
7.
8.          _T1IF = 0;
9.          Count++;
10.         }
11.     }
```

Sobald der Timer bis zum gesetzten PR1-Wert gezählt hat, wird ein Interuppt-Flag gesetzt (T1IF). In dieser Funktion wird solange Nop() ausgeführt (hier 1ms), bis dieses Flag gesetzt wurde. Der Parameter milliseconds bestimmt, wie oft dieser Wartevorgang wiederholt wird. Das Flag muss jedes Mal manuell zurückgesetzt werden.

## Software_PWM

```
1.  void PWM(float freq, float tast){
2.      if (tast==100) {
3.          LED0=1;
4.          PR1 = ((tast/(freq*100))*FCY/64)-1;
5.          delay_ms_t1(1);
6.      }
7.      else if (tast == 0) {
8.          LED0=0;
9.          PR1 = ((100/(freq*100))*FCY/64)-1;
10.         delay_ms_t1(1);
11.     }
12.     else {
13.     LED0=1;
14.     PR1 = ((tast/(freq*100))*FCY/64)-1;
15.     delay_ms_t1(1);
16.     LED0=0;
17.     PR1 = (((1/freq)*(1-tast/100))*FCY/64)-1;
18.     delay_ms_t1(1);
19.     }
20. }
```

Der Funktion werden 2 Parameter übergeben: die gewünschte PWM-Frequenz und der Tastgrad. Um die gewünschte Verzögerung zwischen den Pulsen zu erreichen wurde die Periodendauer PR1 angepasst auf:

$$PR1 = \frac{tast}{freq * 100} * \frac{FCY}{64} - 1$$

Danach wird die delay_ms_t1() Funktion mit Parameter 1ms aufgerufen, um die mit PR1 festgelegte Zeit abzuwarten.

Für die Grenzwerte tast=0 und tast=100 werden extra Fälle aufgerufen, in denen die LED zwar durchgehend aus- oder eingeschaltet bleibt, trotzdem aber eine Verzögerung durch die gewünschte Frequenz auftritt.

## Dreicksförmiger Verlauf der LED PWM

```
1.  while(1)
2.      {
3.          int n = 0;
4.          int c = 0;
5.          while (n<= 100){
6.              while (c < 3){
7.                  PWM(121.212,n);
8.                  c++;
9.
10.             }
11.             c = 0;
12.             n++;
13.
14.         }
15.         LED1 = !LED1;
16.         n = 100;
17.         while (n>=0){
18.             while (c < 3){
19.                 PWM(121.212, n);
20.                 c++;
21.             }
22.             c = 0;
23.             n--;
24.
25.         }
26.     }
```

Mit dieser Funktion wird der Tastgrad n des PWM-Signals von 0 auf 100 bzw. von 100 auf 0 variiert. Pro Tastgrad wird die PWM-Funktion 3 mal wiederholt aufgerufen. Für eine Systemfrequenz von 100MHz führt eine PWM-Frequenz von 121,212Hz zu einer Periodendauer von 5s. Dieser Wert konnte ermittelt werden, in dem die Periodendauer mit Hilfe der 2. LED gemessen wurde (Zeile 15).

Problem: Diese Konfiguration funktioniert nur mit einer Systemfrequenz von 100Mhz. Für andere Frequenzen müssen die Parameter angepasst werden.

Generell scheint die Software-PWM-Funktion für Frequenzen jenseits der 1kHz zu unerwünschtem Verhalten zu führen.