Timer

> Löst ein „internes Ereignis“ aus, welches zu einem Interrupt führt

- Bestimmten Zeitpunkt das Hauptprogramm zu unterbrechen --> Unterprogramm, die Interrupt Service Routine, auszuführen.
- Zählerüberlauf wird ein Flag gesetzt (TF0 bzw. TF1)
- Beim 16-Bit Mode muss der Zählerstand aus 2 Bytes TH0:TL0 gebildet werden.
- Im 8-Bit Mode erfolgt ein automatischer Reload aus dem TH-Register in das TL-Register.
- Timer müssen gestartet werden --> Dazu muss das TR0 (bzw. TR1) Bit gesetzt werden.

1. Timer
    - Bezeichnung: T0
    - Port: 3.4
    - Einsprungadresse: 000Bh
2. Timer
    - Bezeichnung: T1
    - Port: 3.5
    - Einsprungadresse: 001B

1 Timerstartwert berechnen

- Frequenz des Mikrocontrollers --> 1 MHz (1*10^6 Hz)
- gewünschte Zeit, nach der ein Interrupt ausgelöst werden soll.(Bsp 50 ms)

1.1 Zeit errechnen, die ein Takt benötigt

- T = 1/f = 1/ 1* 10^6 = 1*10^-6 = 1µs!
    - → 1 Takt benötigt 1µs!

> Meistens ist 1µs in der Aufgabenstellung gegeben!

1.2 Wie viele Takte notwendig?

- 0,05s/1*10^-6s = 50000

1.3 Startwert errechnen

- 16-Bit-Timer sind 2^16-1 (= 65.535) Takte notwendig bis Überlauf --> Mit Überlauf sind es somit 2^16 (= 65.536) Takte

- Da wir nur 50.000 Takte brauchen, ist kein Multiplikator notwendig.

  → 65.536 Takte – 50.000 Takte = 15.536 Takte

2 Stackpointer setzen

- Da das Programm beim Auslösen eines Interrupts in ein Unterprogramm springt, muss eine Rücksprungadresse im Speicher (SFR) abgelegt werden. Damit diese nicht dort abgelegt wird, wo sich möglicherweise wichtige Daten befinden (werden), weist man dem Stackpointer (SP) eine bestimmte (freie) Adresse im Speicher zu.

In diesem Fall weise ich die Adresse 0x80 zu:

```
mov SP,#0x80
```

3 Timerstartwert setzen

3.1 Startwert zuerst einmal in eine Hexadezimal-Zahl umwandeln!

- 15.53610 = 3CB016

3.2 Wert teilen & „Timer High-Byte“ und ein „Timer Low-Byte“ zuweisen

- TH0 --> „Timer High-Byte“(Timer 0)

```
mov TH0,#0x3C
```

- TL0 --> „Timer Low-Byte“(Timer 0)

```
mov TL0,#0xB0
```

4. Timermodus setzen

M1	M0		
0	0	Modus 0	
0	1	Modus 1	16-Bit Timer ohne nachladen
1	0	Modus 2	8-Bit-Timer mit Auto-Reload
1	1	Modus 3	2 Stück 8-Bit-Timer

- Modus 0 **--> Wird automatische gestartet, aber veraltet!**
- Modus 1 = 16-Bit Timer ohne nachladen **--> 16-Bit Timer/Counter, der Zählerstand ist auf 2 Register aufgeteilt.**
- Modus 2 = 8-Bit-Timer mit Auto-Reload **--> 8-Bit Timer/Counter mit Auto-Reload, gezählt wird im Register TL0 bzw. TL1(der Reload erfolgt bei Überlauf aus TH0 bzwTH1.)**
    - Timer mode "2" is an 8-bit auto-reload mode. What is that, you may ask? Simple. When a timer is in mode 2, THx holds the "reload value" and TLx is the timer itself. Thus, TLx starts counting up. When TLx reaches 255 and is subsequently incremented, instead of resetting to 0 (as in the case of modes 0 and 1), it will be reset to the value stored in THx.
- Modus 3 = 2 Stück 8-Bit-Timer **--> Veraltet**
- Einstellen der Betriebsart erfolgt in einem Register (TMOD) für beide Timer!
    - unteren 4 Bit gelten für Timer 0, die oberen für Timer 1

Für Beispiel Modus 1, damit wir 50.000 Takte ohne nachladen realisieren können.

```
mov TMOD, #0x01
```

5 Timer & Interrupts freigeben

4.1 Timer 0 frei:

```
setb ET0
```

4.2 „generell“ alle Interrupts frei:

```
setb EA
```

6 Timer starten

- Zum starten des Timer 0, setzt man ein Bit bei **TR0**. Um ihn zu stoppen entfernt man es.

```
setb TR0   //Timer starten
clr TR0    //Timer stoppen
```

7 ISR erstellen

```
cseg at 0x000B        //cseg auf die Einsprungsadresse setzen
isr:                  //Label setzen
                      //Irgendwelche Befehle
   reti               //ISR beendet
```

8 Zusammenfassung

1. Timerstartwert errechnen
2. Stackpointer setzen
3. Timerstartwert setzen
4. Timermodus setzen
5. Timer freigeben
6. Timer starten

9 Beispiel Programm

```
$NOMOD51                //Vordefinitionen vergessen
#include <at89c5131.h>  //Neue Definitionen einbinden

cseg at 0x0000
jmp main                //zu Sprungmarke main springen
cseg at 0x000B          //vordefinierte Einsprungadresse (siehe Einfuehrung)
jmp isr                 //zu Sprungmarke isr springen

main:                   //Sprungmarke main
   mov SP,#0x80         //Stackpointer auf 0x80 setzen
   mov TH0,#0x3C        //TH0-Startwert setzen
   mov TL0,#0xB0        //TL0-Startwert setzen
   mov TMOD,#0x01       //Modus setzen
   setb ET0             //T0 freischalten
```

```
    setb EA              //Alle Interrupts generell freischalten
    setb TR0             //T0 starten

loop:                    //Endlosschleife
    nop
    jmp loop

isr:                     //Sprungmarke isr
    clr TR0              //Timer stoppen
    mov TH0,#0x3C        //TH0 zuruecksetzen
    mov TL0,#0xB0        //TL0 zuruecksetzen
    setb TR0             //Timer starten
    cpl P2.0             //Bit bei P2.0 komplementieren
    reti                 //ISR beendet - zurueckspringen

end
```