AVR-Mikrocontroller in BASCOM programmieren, Teil 3

Alle Beispiele in diesem Kapitel beziehen sich auf den Mega8.
Andere Controller können unterschiedliche Timer haben.

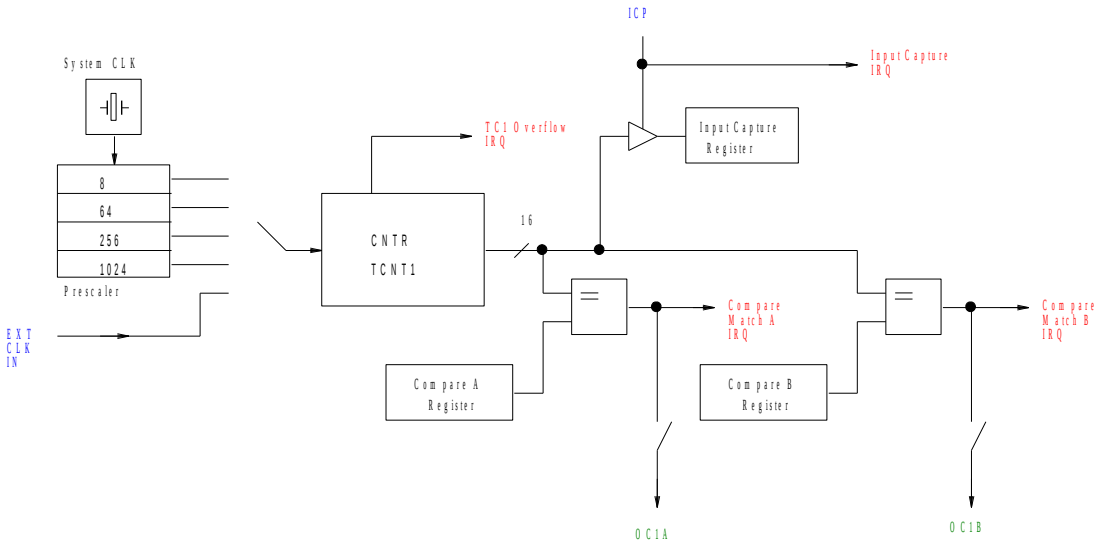
14. Timer

14.1 Allgemeines

- Beim Mega 8 gibt es drei Timer:
  Timer0: 8 bit (0...255)
  Timer1: 16 bit (0...65535)
  Timer2: 8 bit (0...255)

- Timer1 bietet die meisten Möglichkeiten:
  - Erzeugung von präzisen Impulsen, ohne dass dafür Rechenzeit benötigt wird
  - Regelmäßiges Ausführen von Unterprogrammen
  - Uhr
  - Pulsweitenmodulation (PWM)
  - Erfassen von Impulslängen und Zeiten

Grob gesehen hat der Timer1 folgendes Blockschaltbild:



Rot: Interrupt Requests (Interrupt-Anforderungen durch Timer1)

Blau: Eingangspins

Grün: Ausgangspins

- Der Takt des Timers kann gleich dem Takt des Controllers sein, oder um einen Faktor 8,

64, 128 oder 1024 heruntergeteilt werden.

Dies erledigt der **Prescaler**.

Für einen 4MHz-Quarz ergibt sich z.B. folgendes:

Prescaler	**1**	**8**	**64**	**256**	**1024**
Auflösung in µs (T_{CLK})	0.25	2	16	64	256
Tmax Timer0 (8 bit)	64µs	512µs	4096µs	16384µs	65 536µs
Tmax Timer1 (16 bit)	16.384ms	131.072ms	1.048s	4.194s	16.777s

14.2 Timer1 mit Overflow-Interrupt

Der Timer zählt vor sich hin, bei jedem Overflow wird ein Interrupt ausgelöst.

Diese Betriebsart ist nur selten von Nutzen, da sie als Zeitintervall nur wenige Werte zuläßt, nämlich $T = 65536 \cdot T_{CLK}$, wobei T_{CLK} die durch den Prescaler heruntergeteilte Taktfrequenz des Controllers ist.

Eine Anwendung wäre das regelmäßige Ausführen einer Aktion, wenn die Zeitspanne frei gewählt werden kann.

Im folgenden Beispiel blinkt eine LED, ohne dass das Hauptprogramm etwas dafür tun muß:

```
$crystal = 8000000
$regfile = "m8def.dat"

Config Pind.5 = Output                          'LED

Config Timer1 = Timer , Prescale = 64
On Timer1 Timer1serv
Enable Timer1
Enable Interrupts

'-----------------------------------------------
'Hauptprogramm:
Do
   'Dreh Däumchen
   'oder tu sonstwas....
Loop
'-----------------------------------------------
Timer1serv:
  Toggle Portd.5
Return
```

Mit dem Prescaler von 64 wird der 8MHz-Takt heruntergeteilt, dies entspricht einer Takt-Periodendauer von 0.125µs · 64 = 8µs.
Ein Overflow tritt also alle 65536 · 8µs = 524ms auf.
Die Blinkfrequenz ist also ca. 1Hz.

Zuerst müssen Timer-Interrupt und Interrupts allgemein freigeschaltet werden, damit ein Timer-Interrupt möglich ist! Dies geschieht mit:

```
On Timer1 Timer1serv
Enable Timer1
Enable Interrupts
```

14.3 Timer1 mit Output Compare

Dies ist die geeignete Betriebsart, um zu definierten Zeitpunkten eine Aktion auszuführen oder ein Signal mit definierter Frequenz zu erzeugen.

Im Prinzip wird der Zustand des Zählregisters mit dem Zustand des Compare-Registers verglichen und bei Übereinstimmung ein Interrupt ausgelöst oder der Zustand des OC1A bzw. OC1B-Pins geändert.

a) Regelmäßig auszuführender Interrupt

Beispiel: alle 100ms soll im Programm etwas geschen, hier der Einfachheit halber eine LED blinken.

Wenn wir einen 8MHz-Quarz benutzen haben wir eine Taktperiodendauer T_{CLK} = 0.125µs.

Wir müssen den Prescaler so bemessen, dass sich einerseits eine möglichst hohe Auflösung ergibt (kleiner Wert), andererseits aber auch der Zähler nicht vor dem Erreichen des Endwertes überläuft großer Wert).

Wir probieren ein wenig herum:

Prescaler	Auflösung	Endwert
1 8 64 ...	 8·0.125µs = 1µs 64·0.125µs = 8µs	 65536 · 1µs = 65.536ms (zu klein!) 65536 · 8µs = 524.288ms (reicht)

Also wählen wir Prescaler = 64.

Dabei wird alle 8µs um eins weitergezählt.

Der Interrupt soll alle 100ms ausgeführt werden, dies entspricht $\frac{100000\,\mu s}{8\,\mu s} = 12500$ Zählschritten.

Diese Zahl muß also ins Compare Register geschrieben werden.

```
$crystal = 8000000
$regfile = "m8def.dat"

Config Pind.5 = Output                    'LED

Ocr1a = 12500
Config Timer1 = Timer , Prescale = 64 , Clear Timer = 1 , Compare A = Disconnect
On Oc1a Timer1serv
Enable Oc1a
Enable Interrupts

'-----------------------------------------------
'Hauptprogramm:
Do
    'Dreh Däumchen
    'oder tu sonstwas....
Loop
'-----------------------------------------------
Timer1serv:
   'statt eine LED zum Blinken zu bringen kann hier
   'regelmässig etwas Wichtiges getan werden
   Toggle Portd.5
```

```
Return
```

In der Konfiguration des Timers

```
Config Timer1 = Timer , Prescale = 64 , Clear Timer = 1 , Compare A = Disconnect
```

wird **Clear Timer = 1** gesetzt. Dies bewirkt, dass bei jedem Compare Match (Übereinstimmung zwischen Zähl- und Vergleichsregister) der Zähler zurückgesetzt wird und somit wieder von null mit zählen beginnt.
Das **Disconnect** bewirkt, dass am OC1A-Pin kein Signal erscheint.

Vorsicht!
Bei hohen Frequenzen ist die Einsprungzeit für den Interrupt nicht mehr zu vernachlässigen.
(Siehe Kapitel Interrupts)

b) Generierung eines Signals am OC1A-Pin

Wenn ein Signal mit fester Frequenz an einem OC1-Pin erzeugt werden soll, ist noch weniger Software-Aufwand nötig, da sich die Hardware des Controllers um alles kümmert.

Nur das Compare-Register muß initialisiert werden.

Beispiel: es soll ein **1kHz-Signal** erzeugt werden, bei einer Quarzfrequenz von 4MHz.

Die Taktperiodendauer beträgt $T_{CLK} = 0.25\mu s$.

Bei Prescale = 1 ist dies auch die Auflösung des Zählers.

Das Umschalten des Pins muß alle 500µs ausgeführt werden, dies entspricht $\frac{500\,\mu\,s}{0.25\,\mu\,s} = 2000$ Zählschritten.

Dies entspricht einem Comparewert von 1999, da der Zähler dann von 0...1999 zählt und dabei 2000 Schritte macht.

```
$crystal = 4000000
$regfile = "m8def.dat"

Ocr1a = 1999
Config Timer1 = Timer , Prescale = 1 , Clear Timer = 1 , Compare A = Toggle

Do
Loop
```

Mit **Compare A = Toggle** wird dafür gesorgt, dass der OC1A-Pin bei jedem Erreichen des Vergleichswerts umgeschaltet wird.

Wichtig ist auch **Clear Timer = 1** , sodass der Timer bei jedem Compare Match rückgesetzt wird.

Der Vorteil dieser Methode im Gegensatz zum Arbeiten mit Interrupts ist, daß die Frequenz völlig unabhängig davon ist, was der Controller sonst noch alles tun muß.

Soll die Frequenz geändert werden, genügt es, der Wert im Register OCR1A zu ändern.

c) Generierung eines Signals am OC1B-Pin

Man könnte meinen hierzu würde es genügen, überall OC1B statt OC1A zu schreiben.
Dies funktioniert jedoch nicht, obwohl das Blockschaltbild so aussieht, als ob es funktionieren müsste.

Mit dieser Methode geht es:

```
$crystal = 4000000
$regfile = "m8def.dat"

Ocr1a = 1999
Config Timer1 = Timer , Prescale = 1 , Clear Timer = 1 , Compare B = Toggle

Do
Loop
```

d) Generierung von phasenverschobenen Signalen an OC1A und OC1B

Obschon der Controller zwei Compare-Register hat, kann man keine unterschiedlichen Frequenzen oder Signale mit einem Tastverhältnis ungleich 0.5 mit einem Timer erzeugen.

Mit dem Compare B – Register kann aber ein phasenverschobenes Signal erzeugt werden.

Leider gibt es dazu keine vernünftige Dokumentation, oder ich habe sie noch nicht gefunden.

14.4 Timer2 mit Output Compare für Sekundentakt

Timer 2 hat auch eine Output Compare – Möglichkeit, seine Breite ist aber nur 8 Bit.

Wenn man einen Sekundentakt benötigt, Timer1 aber für PWM oder sonstiges benötigt wird, bietet sich Timer2 an.

Man benötigt dann aber noch einen Software-Zähler um auf die fehlende Breite auszugleichen.

Um die richtigen Einstellungen zu finden, muß man ein wenig herumprobieren.

Beispiel: Atmega8 mit 8MHz-Quarz.

Taktperiode: 0.125µs
Zählerperiode mit Prescale = 256: Tcounter = 256 * 0.125µs = 32µs
Maximale Zählerzeit: 256 * 32µs = 8.192ms
Mit Compare-Wert 249 ergeben sich 250 Zählschritte bis zum Interrupt, also 250 * 32µs = 8ms.

Um auf 1s zu kommen, muß der Software-Zähler bis 1000ms/8ms = 125 zählen.

```
'Sekundentakt mit Timer2

$crystal = 8000000
$regfile = "m8def.dat"
$hwstack = 100

$baud = 9600

Config Portd.4 = Output                 'LED


'Timer21 für  8ms - Interrupt
'Tcounter = 256*0.125us = 32us
'Compare = 249 -> 250*32us = 8ms
Config Timer2 = Timer , Prescale = 256 , Clear Timer = 1 , Compare = Disconnect
Ocr2 = 249
On Oc2 Timer2_isr
Enable Oc2
Dim T2ticks As Byte

Enable Interrupts


Dim Secs As Long
Dim New_flag As Bit


Do
      If New_flag = 1 Then
          Print Secs
          New_flag = 0
      End If
Loop
'-------------------------------------------------------------------------------

Timer2_isr:
'Alle 8ms

Incr T2ticks
If T2ticks > 125 Then
      T2ticks = 0
      Gosub Everysecond
End If
Return
'-------------------------------------------------------------------------------
Everysecond:
   Secs = Secs + 1
   New_flag = 1
   Toggle Portd.4
   Return
```

14.5 Pulsweitenmodulation (PWM)

Interessante Links:

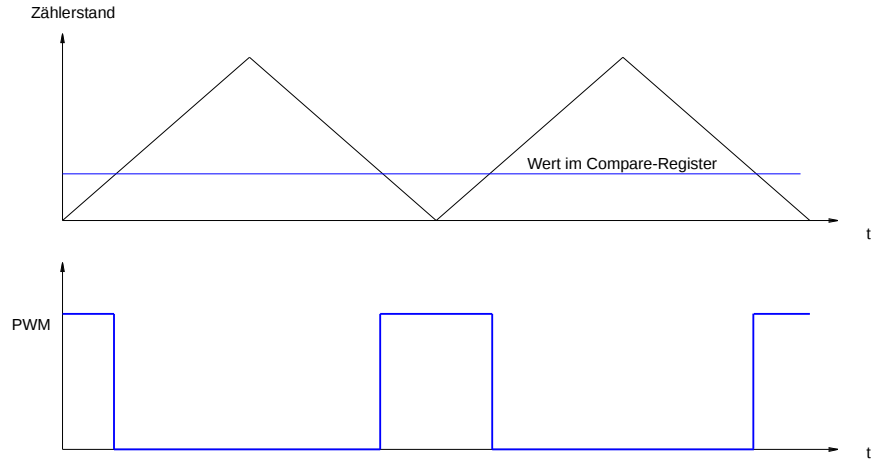
http://bascom-forum.de/mediawiki/index.php/Timer#PWM_-_PulsWeiten_Modulation
http://rn-wissen.de/wiki/index.php/Bascom_und_PWM

Der Timer1 bietet die Möglichkeit, zwei unabhängige PWM-Signale mit einer Auflösung von 8, 9 oder 10 Bit zu erzeugen. Dies geschieht rein hardwaremäßig, d.h. der Prozessor bleibt frei für andere Aufgaben.
Achtung: für Hardware-PWM **muss** man diese Ausgänge, OC1A und OC1B benutzen, da diese das Signal des Timers führen, andere Pins kommen nicht in Frage.

Je nach Controllertyp sind dies andere Portpins, z.B.:

	Mega8	Mega16 / Mega32
OC1A	PortB.1	PortD.5
OC1B	PortB.2	PortD.4

Es gibt verschiedene Modi für die PWM-Erzeugung, BASCOM benutzt standardmässig den „phasenkorrekten“ Modus. Dabei zählt der Timer hoch bis zu seinem maximalen Wert und dann wieder herunter. Dies entspricht analog gesehen einem Dreiecksignal. Erreicht der Zähler den Wert des Komparatorregisters, wird der Ausgang je nach der Konfiguration entweder gesetzt oder rückgesetzt.



Die Programmierung ist einfacher als die Beschreibung im Datenblatt.

Beispiel mit Mega8 und festem PWM-Wert 64/255 = ca. 25%:

```
$crystal = 16000000                    'geändert
$regfile = "m8def.dat"


'Timer1 als PWM mit
'  Prescale=1 -> fCTR = FXTAL = 16MHz  -> fPWM = 16MHz/510 = 31.4kHz
' 510 = 2*2^8-2   (rauf und runter)
'  PWM = 8  -> 8bit PWM    (möglich: 8/9/10 bit)
Config Timer1 = Pwm , Prescale = 1 , Pwm = 8 , Compare A Pwm = Clear Up
Config Portb.1 = Output                '=OC1A


Pwm1a = 64     '64/255 = ca. 25%
```

```
'Hauptschleife
Do
   '...   tu irgendwas
Loop
```

Der Wert von Pwm1a kann während des Programms verändert werden. Im Beispiel ist er der Einfachheit halber fest vorgegeben.

Die Bezeichnung „Compare A Pwm = Clear Up“ in BASCOM ist unklar. Diese Einstellung ergibt jedenfalls ein nichtinvertiertes PWM-Signal wie im obigen Diagramm, während die Einstellung „Compare A Pwm = Clear Down“ ein invertiertes Signal ergibt.

Die Frequenz des PWM-Signals hängt ab vom Prescaler-Wert und der Auflösung des PWM-Signals.

$$f_{PWM} = \frac{f_{Crystal}}{p \cdot (2^B - 2)}$$

f_{PWM} = Frequenz des PWM-Signals
$f_{Crystal}$ = Quarzfrequenz
p = Wert des Prescalers
B = Bit-Auflösung des PWM (8, 9, 10)

Wenn auch OC1B benutzt wird, können 2 unabhängige PWM-Signale erzeugt werden.

Mit hoher Frequenz und einem RC-Tiefpassfilter kann ein PWM-Ausgang praktisch einen DA-Wandler ersetzen.