**Board 1 Teil 3**

**Board 1 – Teil 3 ( Der Timer )**

**Prozessor – Board 1 mit dem ATmega 1284 P,**

**3 x Ports, ISP, 2 x I2C – Bus, Taster und LED`s**

**Was ist ein Timer?**

Ein Timer ist im Grunde nichts anderes als ein bestimmtes Register im Mikrocontroller, das hardwaregesteuert fortlaufend um 1 erhöht (oder verringert) wird (statt *um 1 erhöhen* sagt man auch **inkrementieren**, und das Gegenstück, **dekrementieren**, bedeutet *um 1 verringern*). Anstatt also Befehle im Programm vorzusehen, die regelmäßig ausgeführt werden und ein Re-gister inkrementieren, erledigt dies der Mikrocontroller ganz von alleine. Dazu ist es möglich, den Timer mit dem Systemtakt zu verbinden und so die Genauigkeit des Quarzes auszunut-zen, um ein Register regelmäßig und vor allen Dingen unabhängig vom restlichen Programm-fluss (!) hochzählen zu lassen.

Davon alleine hätte man aber noch keinen großen Gewinn. Nützlich wird das Ganze erst dann, wenn man bei bestimmten Zählerständen eine Aktion ausführen lassen kann. Einer der 'bestimmten Zählerstände' ist zum Beispiel der **Overflow**.

Das Zählregister eines Timers kann natürlich nicht beliebig lange inkrementiert werden – z. B. ist der höchste Zählerstand, den ein 8-Bit-Timer erreichen kann, 28 – 1 = 255. Beim nächsten Inkrementierschritt tritt ein Überlauf (engl. **Overflow**) auf, der den Timerstand wieder zu 0 werden lässt. Und hier liegt der springende Punkt. Wir können uns nämlich an diesen **Overflow** "anhängen" und den Controller so konfigurieren, dass beim Auftreten des Timer-Overflows ein Interrupt ausgelöst wird.

Unter einem **Interrupt** versteht man eine vorübergehende Unterbrechung des laufenden Pro-gramms,um einen anderen in der Regel zeitkritischen und meist kurzen Vorgang abzuarbeiten.

Das auslösende Ereignis wird **IRQ** (Interrupt Request) genannt. Nach dieser Anforderung wird die **ISR** (Interrupt Service Routine) ausgeführt. Anschließend wird das unterbrochene Programm dort fortgeführt, wo es unterbrochen wurde.

Die Mikrocontroller der AVR-Familie besitzen je nach Typ eine unterschiedliche Anzahl an programmierbaren Timer. Bei dem AT1284p sind es 4 Timer.

* Timer 0 – 8 Bit ( 0 - 255 Schritte sind 256 )
* Timer 1 – 16 Bit ( 0 – 65536 Schritte sind 65537 )
* Timer 2 – 8 Bit
* Timer 3 – 16 Bit

Die Timer werden immer mit Timer x benannt, wobei x für die Timernummer steht (also 0,

1, 2, usw.). Die Konfigurationsmöglichkeiten sind von Timer zu Timer unterschiedlich.

**Allgemeine Funktionsweise**

Timer funktionieren nach dem allgemeinen Prinzip, dass sie eine Ganzzahl (im weiteren als Zähler bezeichnet) je nach Betriebsmodus auf- oder abwärtszählen, d.h. inkrementieren bzw. dekrementieren.

 **Board 1 Teil 3**

Angenommen, der Timer arbeitet im einfachsten Betriebsmodus, dem Normalen Modus. Die Zählrichtung des Timers ist aufsteigend gerichtet. Je nach Auflösung, also 8-Bit oder 16-Bit, folgt auf den maximalen Zäh-lerstand wieder die Null. Wenn z.B. bei einem 8-Bit Ti-mer der Wert 255 inkremen-tiert wird folgt die Null (siehe Grafik).

**Der Prescaler**

Wenn also der Quarzoszillator mit 4 MHz schwingt, dann würde auch der Timer 4 Millionen mal in der Sekunde erhöht werden.

$$Overflow= \frac{Quarzfrequenz}{8 Bit}=\frac{4 000 000}{256}=15625 $$

Da der Timer (8 Bit) jedes Mal von 0 bis 255 zählt, bevor ein Overflow auftritt, heißt das auch, dass in einer Sekunde 15625 Overflows vorkommen. Leider ist das für uns zu schnell.

Um diese Raten zu verzögern, gibt es den Vorteiler, oder auf Englisch, Prescaler. Er kann z.B. auf die Werte 1, 8, 64, 256 oder 1024 eingestellt werden, je nach Timer. Dem Datenblatt des AT1284p habe ich die folgenden Angaben entnommen:

**Timer 0** (8 Bit) (CS00 bis CS02)

**Timer 1/3** (16 Bit) (CS10 bis CS12 und CS30 bis 32)

 **Board 1 Teil 3**

**Timer 2** (8 Bit) ( CS20 bis CS22)



Die Aufgabe eines Prescaler ist es, den Systemtakt um den angegebenen Faktor (8, 64, 256, oder 1024) zu teilen. Steht der Vorteiler also auf 1024, so wird nur bei jedem 1024-ten Impuls vom Systemtakt der Timer um 1 erhöht. Entsprechend weniger häufig kommen dann natürlich die Overflows.

$$Overflow=\frac{Systemtakt}{8 Bit x Prescaler}= \frac{4 000 000 (Hz)}{256 x 1024}=15,26 (Hz)$$

Der Systemtakt sei wieder 4 MHz (4 000 000 Hz). Dann wird der Timer in 1 Sekunde

4000000 / 1024 = 3906.25 mal erhöht. Da der Timer wieder jedes Mal bis 255 zählen muss bis ein Overflow auftritt, bedeutet dies, dass in 1 Sekunde 3906.25 / 256 = 15.26 Overflows (Timerfrequenz) auftreten.

Bei einem Systemtakt von 16 MHz (16 000 000 Hz) ergeben sich andere Werte.

$$Overflow= \frac{Systemtakt}{16 Bit x Prescaler}= \frac{16 000 000 (Hz)}{65537 x 32}=7,63 (Hz)$$

Berechnung der Timer Zeit:

$$Timerfrequenz= \frac{Systemtakt}{8 Bit x Prescaler}= \frac{16 000 000 (Hz)}{256 x 1024}=61 (Hz)$$

$$Timer Zeit=\frac{1}{Timerfrequenz}= \frac{1}{61 (Hz)}=0,01639 s=16,39 ms$$

**Die Betriebsmodi**

Die AVR-Timer können in unterschiedlichen Betriebsmodi betrieben werden. Diese sind:

* Normaler Modus
* CTC Modus
* PWM

 **Board 1 Teil 3**

**Normaler Modus (Normal Mode)**

Der einfachste Betriebsmodus ist der normale Modus. Er funktioniert wie im Abschnitt "Allgemeine Funktionsweise" beschrieben. Die Zählrichtung des Timers ist immer aufsteigend, bis zum Überlauf - da fängt der Zähler wieder bei 0 an. Der Überlauf kann einen Interrupt (Timer-Overflow) auslösen.

Im einfachsten Fall kann die-

ser Modus im folgenden Dia-gramm dargestellt werden:

Der Zähler des Timers ist in dem Register TCNTx gespeichert, wo-bei x für eine Zahl steht. Soll z.B. auf den Timer0 (siehe Daten-blatt des jeweiligen Controllers) des Controllers zuge-griffen werden, so ist an TCNT eine 0 anzuhängen, also TCNT0. Wie lange es braucht, bis der Zähler einen Overflow auslöst, ist von der Taktfrequenz des Controllers, dem eingestellten Prescaler-Wert und von der Timerauf-lösung abhängig.

**CTC Modus (Clear Timer on Compare Match)**

Im CTC Modus des Timers ist es möglich, anstelle der durch die Hardware bedingten Ober-grenze des Timers, einen anderen Wert zu benutzen, an dem der Timer einen Interrupt aus-löst und wieder bei 0 zu zählen anfängt. Neben dem Aktivieren des CTC Modus genügt es da-zu, einfach den gewünschten Endwert in ein spezielles Regis-ter, das OCR0A, zu laden. Und natürlich hat auch die ISR dann einen anderen Namen.

Dazu wird der Zähler vorgela-den, bevor dieser wieder vom eigentlichen Timer hochgezählt wird.

**PWM**

PWM – Puls-Weiten-Modulation (Puls-Breiten-Modulation)

Viele sprechen über das Thema, doch wissen scheinbar nicht alle wie das funktioniert.

Ein Mikrocontroller ist ein rein digitales Bauteil. Er schaltet einen Pin entweder ein- oder aus. Definieren wir einen Pin als Ausgang, dann können wir diesen Ausgang entweder auf HIGH setzen worauf am Ausgang die Versorgungsspannung Vcc anliegt, oder aber wir setzen den Ausgang auf LOW wonach dann 0V am Ausgang liegt.

Was passiert aber nun, wenn wir per-iodisch mit einer festen Frequenz zwi-schen HIGH und LOW umschalten? Wir erhalten eine Rechteckspannung. Diese Rechteckspannung hat nun einen geometrischen Mittelwert, der je nach Pulsbreite kleiner oder grösser ist.

 **Board 1 Teil 3**

Die blaue Linie bezeichnet den geometrischen Mittelwert

Bei einer kleinen Pulsbreite haben wir einen kleinen geometrischen Mittelwert

Bei einer grösseren Pulsbreite haben wir einen grösseren geometrischen Mittelwert

Wiederholen wir den Vorgang fortlaufend, haben wir ein PWM an einem Pin anliegen.

Mit den AVR's können wir direkt PWM-Signale erzeugen. Dazu dient der 16-Bit Zähler, wel-cher im sogenannten PWM-Modus betrieben werden kann.

Dabei gibt es noch verschiedene Arten der PWM:

* Fast PWM
* Phasen-korrekte PWM
* Phasen- und frequenzkorrekte PWM

Die genaue Funktionsweise und Werte bitte den entsprechenden Datenblättern entnehmen.

Das alles zum PWM ist aber ein ganz anderes Thema.

**Welchen Prescaler brauchen wir?**

Der AT1284p verfügt über 4 Timer. Im Datenblatt bzw. oben sind die Werte angegeben.

Beispiel:

Wir wollen mit einem 8-Bit Timer Impulse von 1 ms erzeugen. Unser Quarz hat eine Frequenz von 16 MHz. Erste Überlegung:

* 8 - Bit Timer 0 (zählt von 0 bis 255)
* 4 Prescaler (Teiler) (8, 64, 256, 1024)
* 16 MHz Quarz (16 000 000 Schwingungen pro Sekunde)

Treibe ich den Timer mit 16Mhz an, dann brauch ich gar nicht rechnen um zu wissen, dass der wesentlich schneller bei 256 angelangt ist, als in 1ms.

Um wieviel muss ich die 16Mhz (16 Millionen Pulse pro Sekunde) 'untersetzen', damit 256 Pulse 1 Millisekunde dauern?

Wenn ich dem Timer zugestehe, in 1 Millisekunde bis 255 zählen zu dürfen, wie weit kommt er dann in 1 Sekunde?

$$256 x 1000 \left(1 Sekunde\right)=256 000$$

In einer Sekunde zählt unser Timer bis 256 000.

So schnell darf der Timer also zählen, damit sich meine Zeitvorgabe erfüllt. Tatsächlich zählt er aber 16 000 000 Pulse pro Sekunde.

Um wieviel muss ich ihn daher untersetzen?

$$\frac{16 000 000}{256 000}= \frac{16 000}{256}=62,5$$

Ausgerechnet habe ich 62,5

 **Board 1 Teil 3**

Das wäre also mein idealer Teiler Faktor (Prescaler). Wenn es den gäbe, dann würde dieser Timer in genau 1 Millisekunde bis 255 zählen. Nur blöderweise gibt es den nicht.

Ein kleinerer Teiler Faktor ist sinnlos, dann zählt der Timer ja schneller. Also nehme ich den nächst größeren. Das ist 64.

 **Prescaler = 64**

Jetzt rechnen wir in die umgekehrte Richtung. Bei einem Prescaler von 1 würde der Timer

16 000 000 Pulse pro Sekunde zählen.

$$\frac{16 000 000}{64}=250 000$$

Mit einem Prescaler von 64 schafft er nur noch 250 000

Wenn der Timer also in 1 Sekunde bis 250000 zählen würde, wie weit kommt der dann in 1 Millisekunde? Bis 250.

Ich muss den Timer so einstellen, dass er jeweils nur bis 250 zählt. Dann hab ich bei 16 MHz Taktfrequenz und einem Vorteiler von 64 es so eingerichtet, dass diese 250 Pulse abzählen genau 1 Millisekunde dauert.

**Was muss ich genau einstellen?**

Als erstes muss ich angeben, in welcher Betriebsart / Mode ich den Timer betreiben will



Angabe

WGM 1

Angabe Mode

Angabe CTC

Mit dieser Einstellung (**1<<WGM01**) setze ich den Timer auf **CTC**



Angabe CS02, CS01, CS00

Angabe Prescaler 64

Mit dieser Einstellung (**1<<CS01, 1<<CS00**) setze ich den Prescaler auf **64**

 **Board 1 Teil 3**

Ein kleines Programm als Beispiel: (mit Erklärung)

/\* ATB\_B1\_Timer\_1.c Created: 25.11.2014 19:23:07 Author: AS \*/

#define F\_CPU 16000000UL // Angabe der Quarzfrequenz, wichtig für die Zeit

#include <avr/io.h> // Einbindung Datei Ausgänge

#include <avr/interrupt.h>

volatile int8\_t flag\_1ms; // Globale Variable flag\_1ms

volatile int16\_t warten; // Globale Variable warten

void timer\_init() // Timer 0 konfigurieren

 {

 TCCR0A = 0; // Es werden keine Bits gesetzt

 TCCR0B = (1<<WGM01)|(1<<CS01)|(1<<CS00); // Einstellung CTC Modus, Prescaler 64

 TCNT0=1; // Initialisiert Timer

 OCR0A=249; // Laden des Vergleichswertes

 TIMSK0|=(1<<OCIE0A); // Interrupt erlauben

 }

ISR (TIMER0\_COMPA\_vect) // ISR

 {

 flag\_1ms=1; // setzt flag\_1ms auf 1

 }

int main(void)

 {

 timer\_init(); // Initiiert Timer (erster Aufruf)

 DDRC=0b01000000; // setzt Port C, Pin PC6 auf Ausgang

 sei(); // gibt Interrupts frei

 while(1) // Beginn Programmschleife while

 {

 if(flag\_1ms) // Abfrage ob flag\_1ms wahr (1) ist

 { // Wenn Abfrage wahr ist, dann …

 flag\_1ms=0; // setzt flag\_1ms auf 0

 warten++;

 if(warten==500) // Angabe Zeit 500ms

 {

 PORTC &= ~(1<<PC6); // Schaltet PC6 ein

 }

 else

 {

 if(warten==1000) // Angabe Zeit 500 ms

 {

 PORTC |= (1<<PC6); // Schaltet PC6 aus

 warten=0; // setzt Zled7 auf 0

 }

 } // ende if warten

 } // ende if flag

 } // ende while

 } // ende main

 **Board 1 Teil 3**

Sehen wir uns das Programm genauer an:

#define F\_CPU 16000000UL // Angabe der Quarzfrequenz, wichtig für die Zeit

* Angabe der Quarzfrequenz

#include <avr/io.h> // Einbindung Datei Ausgänge

#include <avr/interrupt.h>

* Angabe der Dateien

volatile int8\_t flag\_1ms; // Globale Variable flag\_1ms

volatile int16\_t warten; // Globale Variable warten

* Definition der Variablen

void timer\_init() // Timer 0 konfigurieren

 {

 TCCR0A = 0; // Es werden keine Bits gesetzt

 TCCR0B = (1<<WGM01)|(1<<CS01)|(1<<CS00); // Einstellung CTC Modus, Prescaler 64

 TCNT0=1; // Initialisiert Timer

 OCR0A=249; // Laden des Vergleichswertes

 TIMSK0|=(1<<OCIE0A); // Interrupt erlauben

 }

* Unterprogramm timer\_init(), Angabe der notwendigen Einstellungen für 1 ms (siehe Text)

ISR (TIMER0\_COMPA\_vect) // ISR

 {

 flag\_1ms=1; // setzt flag\_1ms auf 1

 }

* Aufruf ISR und Übergabe an flag\_1ms

int main(void)

* Beginn Hauptprogramm

 {

 timer\_init(); // Initiiert Timer (erster Aufruf)

* Erster Aufruf Unterprogramm timer\_init()

 DDRC=0b01000000; // setzt Port C, Pin PC6 auf Ausgang

* Ausgang Port C, Pin 6 als Ausgang freigeben

 sei(); // gibt Interrupts frei

* Interrups freigeben

 while(1) // Beginn Programmschleife while

 {

 if(flag\_1ms) // Abfrage ob flag\_1ms wahr (1) ist

* Wenn flag\_1ms ( wahr / 1 ) gesetzt ist, dann mache …

 **Board 1 Teil 3**

 { // Wenn Abfrage wahr ist, dann …

 flag\_1ms=0; // setzt flag\_1ms auf 0

* Setze flag\_1ms auf 0

 warten++;

* Erhöhe warten um 1

 if(warten==500) // Angabe Zeit 500ms

* Hat warten den Wert von 500 dann schalte …

 {

 PORTC &= ~(1<<PC6); // Schaltet PC6 ein

 }

* Port C / Pin 6

else

* Wenn nicht, dann mache …

 {

 if(warten==1000) // Angabe Zeit 500 ms

* Hat warten den Wert von 1000 dann schalte …

 {

 PORTC |= (1<<PC6); // Schaltet PC6 aus

* Port C / Pin 6

 warten=0; // setzt warten auf 0

* Setze warten auf 0

 }

 } // ende if warten

 } // ende if flag

 } // ende while

 } // ende main

**Funktionsbeschreibung:**

Der **Timer 0** zählt bis **249** und schaltet zurück auf **0**. Das entspricht 250 Schritten. Bei je-dem Durchlauf wird **flag\_1ms** gesetzt. Ist **flag\_1ms** gesetzt wird **warten** um jeweils **1** hoch-gezählt. Erreicht **warten** den Wert von **500** wird **PC6** geschaltet. Erreicht **warten** den Wert von **1000** wird **PC6** wiederum geschaltet.

Da der Timer mit 1ms schaltet und der Zähler bei 500 und 1000 schaltet blinkt die LED (PC6) mit 0,5s ein und 0,5s aus. Das entspricht einer Blinkfrequenz von 1 Sekunde.

Im Grunde besteht das Programm aus den folgenden Teilstücken:

* Angabe Frequenz
* Unterprogramm Timer
* ISR – Schleife
* Hauptprogramm

 **Board 1 Teil 3**

**Ein weiteres Beispiel:**

Wir wollen mit einem Timer Impulse von **10** ms erzeugen. Unser Quarz hat eine Frequenz von 16 MHz. Erste Überlegung:

* Welcher Timer? 8-Bit oder 16-Bit? Welchen Timer brauche für andere Aufgaben?
* Welcher Prescaler (Teiler) ? (8, 64, 256, 1024)
* 16 MHz Quarz (16 000 000 Schwingungen pro Sekunde)

Mit meinem Quarz 16MHz treibe ich den Timer mit 16 000 000 Pulse je Sekunde.

Mein **8-Bit Timer** hat **256** Schritte und mein **16-Bit Timer** hat **65537** Schritte.

Um wieviel muss ich die 16Mhz (16 Millionen Pulse pro Sekunde) 'untersetzen', damit **256** oder **65537** Pulse **10** Millisekunden dauern?

$$10 ms x 100=1 Sekunde (1000 ms)$$

In einer 1 Sekunde werden **100** Pulse ausgelöst.

Wenn ich dem **8-Bit Timer** zugestehe, in **10** Millisekunde bis **255** zählen zu dürfen, wie weit kommt er dann in 1 Sekunde?

$$256 x 100=25 600$$

In einer Sekunde zählt unser **8-Bit Timer** bis **25 600**.

Bei einem **16-Bit** Timer erfolgt eine Zählung in **10** Millisekunden bis **65536**, wie weit kommt er in 1 Sekunde?

$$65 537 x 100=6 553 700$$

In einer Sekunde zählt der **16-Bit Ti**mer bis **6 553 700**.

So schnell darf der Timer also zählen, damit sich meine Zeitvorgabe erfüllt. Tatsächlich zählt er aber 16 000 000 Pulse pro Sekunde. Um wieviel muss ich ihn daher untersetzen?

Bei einem 8-Bit Timer:

$$\frac{16 000 000}{25 600}= \frac{160 000}{256}=625$$

Bei einem 16-Bit Timer:

$$\frac{16 000 000}{6 553 700}=\frac{160 000}{65 537}=2,44 $$

Bei einem 8-Bit Timer benötige ich einen Prescaler von mindestens 625 und bei einem 16-Bit Timer einen Prescaler von mindestens 2,44.

Das wären also meine idealen Teiler Faktoren (Prescaler). Nur blöderweise gibt es die nicht.

Ein kleinerer Teiler Faktor ist sinnlos, dann zählt der Timer ja noch schneller. Also nehme ich die nächst größeren. Das ist beim 8-Bit Timer 1024 und beim 16-Bit Timer 8.

Jetzt rechnen wir in die umgekehrte Richtung. Bei einem Prescaler von 1 würde der Timer

16 000 000 Pulse pro Sekunde zählen.

Bei einem 8-Bit Timer:

$$\frac{16 000 000}{1024}=15625$$

Mit einem Prescaler von 1024 schafft er nur noch 15 625.

 **Board 1 Teil 3**

Bei einem 16-Bit Timer:

$$\frac{16 000 000}{8}=2 000 000$$

Mit einem Prescaler von 8 schafft er noch 2 000 000.

Wenn der 8-Bit Timer also in 1 Sekunde bis 15625 zählen würde, wie weit kommt der dann in 10 Millisekunden? Bis 156.

Ich muss den 8-Bit Timer so einstellen, dass er jeweils nur bis 156 zählt. Dann hab ich bei 16 MHz Taktfrequenz und einem Vorteiler von 156 es so eingerichtet, dass diese 256 Pulse abzählen genau 10 Millisekunden dauern.

Wenn der 16-Bit Timer also in 1 Sekunde bis 2 000 000 zählen würde, wie weit kommt der dann in 10 Millisekunden? Bis 20 000.

Ich muss den 16-Bit Timer so einstellen, dass er jeweils nur bis 20 000 zählt. Dann hab ich bei 16 MHz Taktfrequenz und einem Vorteiler von 8 es so eingerichtet, dass diese 20 000 Pulse abzählen genau 10 Millisekunden dauert.

Ergebnis:

Takte / Impulse mit 10ms kann ich sowohl mit einem 8-Bit Timer oder einem 16-Bit Timer erzeugen. Beim 8-Bit Timer kommt es allerdings zu Abweichungen.

Einschränkung:

Der 8-Bit Timer ist so ziemlich am Ende, da ich den Prescaler 1024 verwende. Beim 16-Bit Timer kann ich noch andere Prescaler einsetzen und somit noch grössere Zeiten erreichen.

Natürlich gibt es auch entsprechende Computerprogramme dazu z.B den

**AVR Timer Calculator**

Im ersten Bild wird mit 16Mhz, 8-Bit Timer, einem Prescaler von 1024 ein Compare von 155 berechnet. Es entsteht ein Fehler dabei. Leider ist das die einzige Möglichkeit



Im zweiten Bild wird mit 16MHz, 16-Bit Timer und einem Prescaler von 8, 64, 256 ein Compare von 19999, 2499, 624 gerechnet. Ohne Fehler

Welcher Timer genommen wird, bleibt jedem selbstüberlassen.

 **Board 1 Teil 3**

Als Beweis habe ich die Berechnung noch mal

mit 16Mhz, 8-Bit Timer bei 20ms durchgeführt.

Damit ist es nicht möglich

Bei 16MHz, 16-Bit Timer, 20ms gibt es wieder verschiedene Möglichkeiten der Einstellung



Bei 16MHz, 8-Bit Timer und 20 ms habe ich keine Möglichkeiten der Einstellung



Bei 8MHz, 16-Bit Timer habe ich wieder verschiedene Möglichkeiten der Einstellung

 **Board 1 Teil 3**



Als letztes noch 8MHz, 16-Bit Timer und

20ms mit den verschiedenen Einstellungen



Zum Schluss der Bilder noch meine Einstellungen für 16MHz, 8-Bit Timer und 1 ms



**Welche Einstellungen muss ich zur Berechnung vornehmen?**

Angabe der Quarzfrequenz in Hz

Angabe der Zeit in µs

Welcher Modus Compare = CTC

Welcher Timer

8-Bit oder 16-Bit

 **Welche Ergebnisse be-**

 **komme ich?**

Welchen Compare Wert muss ich einstellen

Welche Zeit wird ermittelt bei welchem Fehler

Welche Prescaler kann ich verwenden bzw. sind möglich

 **Board 1 Teil 3**

**Was muss ich genau einstellen?**

Als erstes muss ich angeben, in welcher Betriebsart / Mode ich den Timer betreiben will



Angabe

WGM 1

Angabe CTC

CTC-Mode

Angabe Mode

Mit dieser Einstellung (**1<<WGM01**) setze ich den Timer auf **CTC**

Diese Einstellung ist für den AT1284p. Andere Prozessoren können andere Einstellungen haben. Bitte die Daten dem jeweiligen Datenblatt entnehmen.

TCCR0A = 0; // Es werden keine Bits gesetzt

**TCCR0B** = (**1<<WGM01**)|( . . . )|( . . . ); // Einstellung CTC Modus

Mit dieser Einstellung setze ich in meinem Programm das Register TCCR0B des Timers 0 auf CTC Modus.

Als nächstes muss ich den Prescaler einstellen. Je nach verwendeten Quarz und gewählter Zeit können es unterschiedliche Einstellungen sein. Die Werte muss ich wieder einer Tabelle aus dem Datenblatt entnehmen.

Angabe CS02, CS01, CS00



Angabe z.B. Prescaler 64

 **Board 1 Teil 3**

Folgende Einstellungen sind möglich

**1<<CS01** setze den Prescaler auf **8**

**1<<CS01, 1<<CS00** setze den Prescaler auf **64**

**1<<CS02** setze den Prescaler auf **256**

**1<<CS02, 1<<CS00** setze den Prescaler auf **1024**

Diese Einstellung ist für den AT1284p. Andere Prozessoren können andere Einstellungen haben. Bitte die Daten dem jeweiligen Datenblatt entnehmen.

TCCR0A = 0; // Es werden keine Bits gesetzt

**TCCR0B** = ( . . . )|(**1<<CS01**)|(**1<<CS00**); // Einstellung CTC Modus, Prescaler 64

Mit dieser Einstellung setze ich in meinem Programm das Register TCCR0B des Timers 0 auf einen Prescaler von 64.

Sehen wir uns das Unterprogramm noch mal an:

void timer\_init() // Timer 0 konfigurieren

 {

 TCCR0A = 0; // Es werden keine Bits gesetzt

 TCCR0B = (1<<WGM01)|(1<<CS01)|(1<<CS00);// Einstellung CTC Modus, Prescaler 64

 TCNT0=1; // Initialisiert Timer

 OCR0A=249; // Laden des Vergleichswertes

 TIMSK0|=(1<<OCIE0A); // Interrupt erlauben

 }

* jede Zeile habe ich mit entsprechenden Kommentaren versehen
* habe alle Werte kommentiert
* Angabe Prescaler
* Angabe Timer Nr.
* Angabe Mode

Damit kann ich die Einstellungen auch zu einem späteren Zeitpunkt nachvollziehen.

**Wie kann ich meinen Timer anders starten?**

Meinen Timer kann ich ohne Unterprogramm starten. Bisher habe ich so gemacht:

int main(void)

 {

 **timer\_init();** // Initiiert Timer (erster Aufruf)

In das Unterprogramm springen, Timer starten und zurück springen. Es geht auch anders:

int main(void)

 {

 TCCR0A = 0; // Es werden keine Bits gesetzt

 TCCR0B = (1<<WGM01)|(1<<CS01)|(1<<CS00); // Einstellung CTC Modus, Prescaler 64

 TCNT0=1; // Initialisiert Timer

 OCR0A=249; // Laden des Vergleichswertes

 TIMSK0|=(1<<OCIE0A); // Interrupt erlauben

Beim Start meines Programmes muss der Timer einmalig aufgerufen werden.

 **Board 1 Teil 3**

**Wie bekomme ich andere Zeiten?**

Mein Timer produziert jede ms einen Puls. Diesen kann ich einfach weiter zählen und entsprechend auswerten. Bisher habe ich so gemacht:

ISR (TIMER0\_COMPA\_vect) // ISR

 {

 **flag\_1ms=1;** // setzt flag\_1ms auf 1

 }

Man könnte es auch so machen und damit einen zusätzlichen Puls alle 10ms bekommen.

ISR (TIMER0\_COMPA\_vect) // ISR Timer 0

 {

 **flag\_1ms=1;** // setzt flag\_1ms auf 1

 if(wait<=9) // bei 9 sind es 10ms

 {

 wait++;

 } // erhöht

 else // wenn dann ...

 {

 wait=0; // setzt wait auf 0

 **flag\_10ms=1;** // setzt flag \_10ms auf 1

 }

Nicht vergessen, innerhalb des Programmes die Werte wieder auf 0 setzen.

Damit kann ich mir jede Zeit, die ich benötige selbst einstellen.

Mit diesem Tut kann jeder seine Timer einstellen, wie er es gern möchte. Denkt bitte daran, jeder Timer hat spezielle Fähigkeiten und Möglichkeiten der Anwendung. Seht euch doch mal die anderen Timer an. Da gibt es noch viele andere Möglichkeiten.

Einige Teile des Textes wurden zur besseren Übersicht farblich gestaltet.

Die Nutzung erfolgt auf eigenes Risiko.

Ich wünsche viel Spaß beim Bauen und programmieren

Achim

myroboter@web.de