Timerinterrupts beim Arduino

Timer 1 (16bit)

Register:

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bits in den Registern und ihre Bedeutung:

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

Für den CTC-Mode kann demnach TCCR1A komplett auf 0 gesetzt werden, weil WGM11 und WGM10 in diesem Register liegen und beide laut Tabelle 0 sind:

```
TCCR1A = 0; //Register definiert zurücksetzen

TCCR1B = 0; //zuerst Register definiert zurücksetzen

TCCR1B |= (1 << WGM12); //CTC gesetzt durch WGM12
```

Im TCCR1B wird aber auch der Prescaler gesetzt.

CSX2	CSX1	CSX0	Beschreibung
0	0	0	Timer angehalten
0	0	1	1:1 (kein Prescaling)
0	1	0	1:8
0	1	1	1:64
1	0	0	1:256
1	0	1	1:1024
1	1	0	externe Taktquelle am Pin T0, Takt bei negativer Taktflanke
1	1	1	externe Taktquelle am Pin T0, Takt bei positiver Taktflanke

Das X in obiger Tabelle entspricht der Timernummer. Damit ist es beim Timer1: CS12, CS11 und CS10.

```
TCCR1B | = (0 << CS12) | (0 << CS11) | (1 >> CS10);  //kein Prescale

TCCR1B | = (0 << CS12) | (1 << CS11) | (0 >> CS10);  //Prescale auf 8

TCCR1B | = (0 << CS12) | (1 << CS11) | (1 >> CS10);  //Prescale auf 64

TCCR1B | = (1 << CS12) | (0 << CS11) | (0 >> CS10);  //Prescale auf 256

TCCR1B | = (1 << CS12) | (0 << CS11) | (1 >> CS10);  //Prescale auf 1024
```

Der Prescaler wird gebraucht, um den Systemtakt XTAL (16MHz) herunter zu teilen. Im damit erreichten Takt wird dann hochgezählt bis der Vergleichswert OCR1A (OCR1B gibt es auch) erreicht wurde. Im CTC-Mode gibt es dann bei Interruptsteuerung einen Reset und der Interrupt wird ausgelöst.

Berechnung: CPU-Frequenz (XTAL) / Prescale / Interruptfrequenz = OCR1A

Beispiel: OCR1A = 31250; //rechne 16MHz / 256 / 2Hz = 31250

Bei 16MHz und einem Prescale von 256 gibt es eine Frequenz von 62500Hz. Wenn man mit dieser Frequenz zählt ist man nach einer halben Sekunde (=2Hz) bei 31250. Der Interrupt löst also alle halbe Sekunde aus.

Ein weiteres Beispiel:

Bei 16MHz und Prescale 8 erhält man einen Zähltakt von 2MHz. Wird der Vergleichswert OCR1A auf 20 gesetzt, bekommt man alle 10µs einen Impuls.

2MHz / 20 = 100kHz; 100kHz = 100.000/1s = 100.000 / 1000ms =

100.000 / 1.000.000µs = 1 / 10µs also 1 Interrupt pro 10µs

Wichtig: Nur Timer 1 (16bit) zählt bis 65535 = 2^16-1. Alle anderen Timer (8bit)zählen nur bis 255 = 2^8-1. Prescaler entsprechend wählen.

Neben OCR1A gibt es noch OCR1B. Man kann beide verwenden. Im Register TIFR gibt es korrespondierende Flags, die gesetzt werden, wenn der betreffende Werte erreicht wurde:

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Sofern das im Interrupt passiert, braucht man die entsprechenden Bits nicht manuell zurücksetzen. Ohne Interrupt müssen die Bits jeweils mit ‚1‘ zurückgesetzt werden. Kein Scherz! 1 wird zum Löschen benutzt …

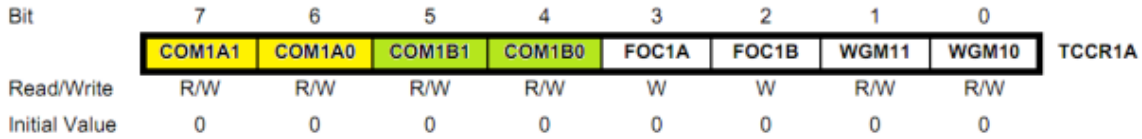
```
TIFR |= (1 << OCF1A);
```

Der Test könnte so aussehen:

```
if (TIFR & (1 << OCF1A)) { …
```

Der Wert des Zählers lässt sich aus TCNT1 auslesen.

Im TCCR1A Register liegen noch die Flags für den Compare Output Mode = COM:



Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on compare match
1	0	Clear OC1A/OC1B on compare match (Set output to low level)
1	1	Set OC1A/OC1B on compare match (Set output to high level)

Das bezieht sich auf die beiden Ausgänge OC1A(PD5) und OC1B(PD4) am Prozessor. Nimmt man die gelbe Zeile, so schaltet der PD5 Ausgang jeweils um (z.Bsp.) LED-Blinken, wenn der Zählerstand erreicht wird:

```
TCCR1A |= (1 << COM1A0); // PD5 toggelt

TCCR1A |= (0 << COM1A0) | (0 << COM1A1); // PD4 und PD5 unbenutzt
```

Nutzung der Timer mit Interrupts

Das TIMSK Register ist für die Nutzung der Timer im Rahmen von Interrupts verantwortlich:

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Die gelb markierten OCIE1A und OCIE1B sind hier von Interesse.

```
TIMSK |= (1 << OCIE1A); // aktiviert den Interrupt
```

Im obigen Beispiel wird der Interrupt immer ausgelöst, wenn der Zähler TCNT1 gleich dem Vergleichswert OCR1A wird. Das geht auch mit OCIE1B für den Vergleich von TCNT1 und OCR1B.

Die Interrupt Service Routine (ISR) wird dann wie eine Funktion beschrieben:

```
ISR (TIMER1_COMPA_vect) {…}
```

Ein kompletter Programmcode könnte dann so aussehen, wie auf der folgenden Seite:

```
long i = 1;

void setup() {
  cli(); // disable interrupts

  // reset
  TCCR1A = 0; // set TCCR1A register to 0
  TCCR1B = 0; // set TCCR1B register to 0
  TCNT1  = 0; // reset counter value

  OCR1A = 31249; // compare match register

  // set prescaler
  TCCR1B |= (1 << CS12) | (1 << CS10);  //Prescale 1024

// 16MHz/1024=15625Hz mit OCR1A=31249 Interrupt alle 2s

  TCCR1B |= (1 << WGM12); // turn on CTC mode
  TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt

  sei(); // allow interrupts


  Serial.begin(9600); // start serial connection
}

ISR(TIMER1_COMPA_vect) { // function which will be called when an interrupt
occurs at timer 1

  Serial.println(i); // send current value of i to the pc
}

void loop() {
  i++; // increment i
}
```

Noch ein Beispiel: Timer 0 und 2 unterscheiden sich deutlich davon, da sie nur 8bit sind…

```
// timer0 will interrupt at 2kHz
// timer1 will interrupt at 1Hz
// timer2 will interrupt at 8kHz
void setup() {
  cli(); // disable interrupts
   // timer0
  TCCR0A = 0; // set TCCR0A register to 0
  TCCR0B = 0; // set TCCR0B register to 0
  TCNT0  = 0; // set counter value to 0
  OCR0A = 124; // set compare match register
   TCCR0B |= (1 << CS01) | (1 << CS00); // Set CS01 and CS00 bits for 1:64 prescaler
  TCCR0A |= (1 << WGM01); // turn on CTC mode
  TIMSK0 |= (1 << OCIE0A); // enable timer compare interrupt
  // timer1
  TCCR1A = 0; // set TCCR1A register to 0
  TCCR1B = 0; // set TCCR1B register to 0
  TCNT1  = 0; // set counter value to 0
  OCR1A = 15624; // set compare match register
  TCCR1B |= (1 << CS12) | (1 << CS10); // Set CS12 and CS10 bits for 1:1024 prescaler
  TCCR1B |= (1 << WGM12); // turn on CTC mode
  TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
 // timer2
  TCCR2A = 0; // set TCCR2A register to 0
  TCCR2B = 0; // set TCCR2B register to 0
  TCNT2  = 0; // set counter value to 0
  OCR2A = 249; // set compare match register
```

```
  TCCR2B |= (1 << CS21); // Set CS21 bits for 1:8 prescaler
  TCCR2A |= (1 << WGM21); // turn on CTC mode
  TIMSK2 |= (1 << OCIE2A); // enable timer compare interrupt
  sei(); // allow interrupts
}
ISR(TIMER0_COMPA_vect) {
// timer0 interrupt to-do code here
}
ISR(TIMER1_COMPA_vect) {
 // timer1 interrupt to-do code here
}
ISR(TIMER2_COMPA_vect) {
 // timer2 interrupt to-do code here
}
void loop() {
  // other code
}
```