Rechnerarchitektur, Foliensatz 6 Interrupts und Timer

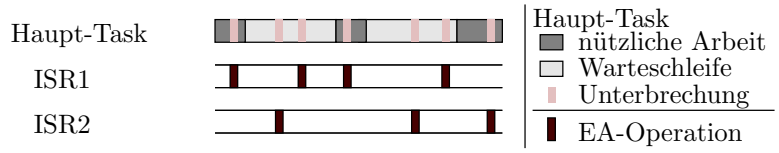
G. Kemnitz

13. November 2018

Contents

1 Interrupt

Interrupt-Service-Routinen (ISR)



Haupt-Task:

- Ohne zyklische Abfrage von Ereignisbits.
- Lokale (individuelle) und globale Interrupt-Freigabe.

Interrupt-Behandlung:

- Aufruf der ISR auf einer feste Hardware-Adresse.
- Interrupt sperren. Inhalte genutzter Register incl. Statusregister, Frame-Pointer, ... retten.
- EA-Operation ausführen
- Register wiederherstellen. Interrupt-Freigabe. Rücksprung.

Interrupt-Freigabe, Ereignisbits, ... ATmega 2560

Globale Interrupt-Freigabe: Bit »I« (SREG.7, EA-Adresse 0xFE):

Bitnummer:	7	6	5	4	3	2	1	0
Bitname:	I	T	H	S	V	N	Z	C

C-Anweisungen für des Setzen und löschen von »I«:

```
sei();        // Interrupts global ein
cli();        // Interrupts global aus
```

Adresse und Konfigurationsbits externer Interrupts an Port B:

Interrupt	Adresse*1	Ereignisbit	Freigabebit	weiter Einstellung*2
INT0 (PB0)	0x0002	EIFR.0	EIMSK.0	EICRA[1:0]
INT1 (PB1)	0x0004	EIFR.1	EIMSK.1	EICRA[3:2]
...	...	...	...	...
INT7 (PB7)	0x0010	EIFR.7	EIMSK.7	EICRB[7:6]

*1 Startadresse der Interrupt-Service-Routine (ISR)

*2 Interrupt bei null, steigender und/oder fallender Flanke.

Interrupt-Quellen des ATmega 2560

Ingesamt 57 (ATmega2560.pdf[1], Abschn. 14.1 Interrupts):

- 8× für Port B Anschlüsse. Bei Port B als Eingang auch als Software-Interrupts nutzbar.
- 3× Port-Change-Interrupts für programmierbare Bitänderungen auch an anderen Ports.
- 1× Watchdog (Timeout).
- 26× für Timer-Funktionen.
- 4×3 für die 4 universellen seriellen Schnittstellen (USARTs).
- 1× SPI (serielle Übertragung abgeschlossen).
- 1× Analogkomparator.
- 1× ADC (Digital-Analog-Wandlung abgeschlossen).
- 1× TWI (2-wire Serial Interface).
- 1× EEPROM (Schreiboperation abgeschlossen), ...

Vor- und Nachteile von Interrupts

Vorteile:

- Schnellere Reaktion auf externe Ereignisse.
- Der Haupt-Task muss nicht aller paar tausend Befehle zur Hauptschleife zurückkehren, darf Warteschleifen enthalten.

Nachteile:

- Haupt-Task wird an zufälligen Programmstellen unterbrochen,
- dadurch kein deterministischer Ablauf.
- Viele zusätzliche Fehlermöglichkeiten.
- Erschwerter Test, erschwerte Fehlersuche.

Einige Regeln für Interrupt-Routinen:

- kurze, vorhersagbare Abarbeitungszeit, keine Warteschleifen.
- keine Änderung von Daten und Registerinhalten, die das unterbrochene Programm möglicherweise gerade bearbeitet.

[1] http://techwww.in.tu-clausthal.de/site/Lehre/Rechnerarchitektur...

Interrupt-Sperren

Ein unterbrechbares Programm muss die ISR immer sperren, während es Übergabedaten bearbeitet:

```
...
uint8_t tmp = <Interrupt-Freigaberegister>;
<Interrupt-Freigabebit löschen>
<Bearbeitung der Übergabedaten>
<Interrupt-Freigaberegister> = tmp;
```

Beispiel einer ISR:

```
#include <interrupts.h>//Header für Int.-Nutzung
#include <avr/io.h>
ISR(INT0_vect){    //Int. bei Tastendruck an PB0
 PORTJ^=1;         //PJ0 (LED 1) invertieren
}
```

INT0_vect – Startadresse der Interrupt-Service-Routine (ISR).

Beispiel für ein Hauptprogramm hierzu:

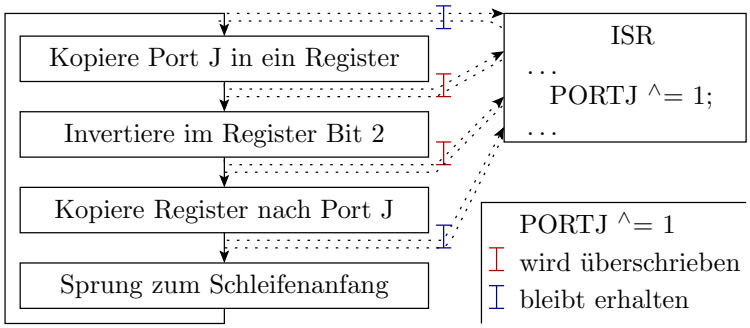
```
int main(){
  DDRB  = 0;      // Port B Eingänge
  DDRJ  = 0xFF;   // Port J Ausgänge
  EIMSK =1<<INT0;// Freigabe Interrupt an PB0
  EICRA = 0b11;   // Interrupt bei 01-Flanke
  sei();          // globale Interrupt-Freigabe
  while (1){      // Endlosschleife
   uint8_t tmp = EIMSK; // Int.-Freigabe speichern
   EIMSK &= ~1;   // INT0 an PB0 sperren
   PORTJ^=2;      // PJ0 (LED 2) invertieren
   EIMSK=tmp;     // Int-Freigabe wiederherstellen
  }
}
```

Wenn INT0 nicht während »PORTJ^=2« gesperrt wird, wird die Hälfte der Anweisungen »PORTJ^=1« in der ISR bei Tastendruck nicht wirksam. Warum?

$$\Rightarrow \text{Bearbeitung derselben Daten (PORTJ).}$$

Ohne die drei Anweisungen zur Interrupt-Sperre wird der Schleifenkörper in die Schrittfolge übersetzt:

- Port J lesen,
- Wert bearbeiten,
- Wert schreiben und
- Sprung zum Schleifenbeginn:



An 50% der Unterbrechungsmöglichkeiten wird die Invertierung von PJ0 in der ISR vom Rückschreibwert für die Invertierung von PJ1 überschrieben, d.h. Reaktion nur auf 50% Tastendrucke.

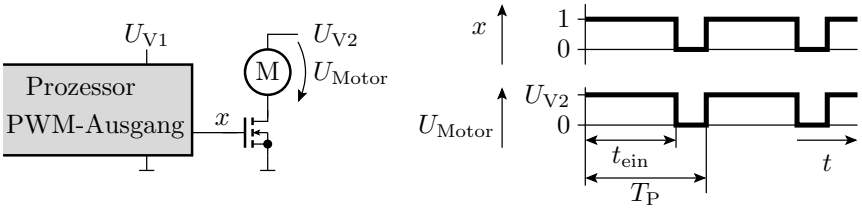
2 Timer

Timer

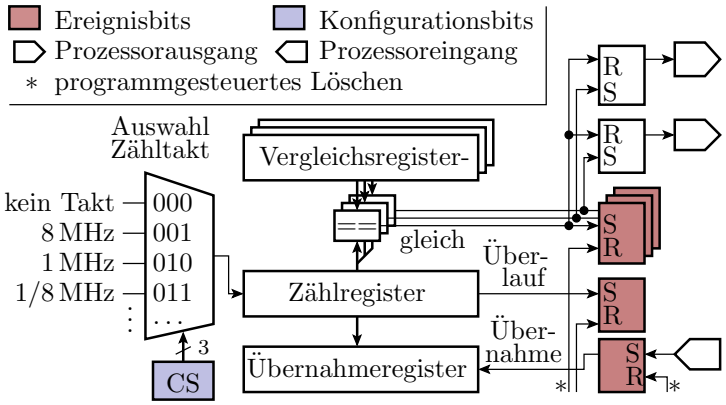
Ein Timer ist eine Hardware-Einheit aus Zähl-, Vergleichs-, Konfigurationsregistern, ... zur

- Erzeugung von Wartezeiten,
- zeitgesteuerten Ereignisabarbeitung,
- Erzeugung pulsweitenmodulierter (PWM-) Signale und
- Zeitmessung.

PWM-Signale dienen z.B. zur stufenlosen Leistungssteuerung von Elektromotoren (Ersatz für Digital-Analog-Wandler).



Aufbau und Funktionsweise eines Timers



- Kern eines Timers ist ein Zählregister mit einem vom Programm zuschaltbaren programmierbaren Takt.
- Die Ereignisbits (Überlauf, Gleichheit, externe Flanke) für Polling und Interrupt.

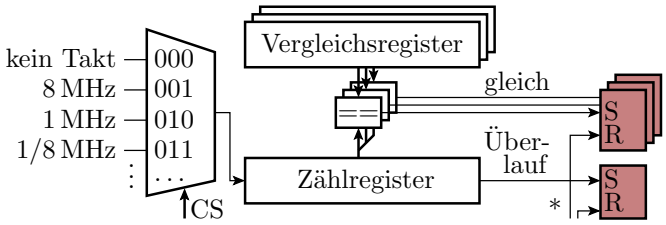
Timer des ATMega 2560

- Zwei 8-Bit Timer (Tmr0 und Tmr2).
- Vier 16-Bit-Timer (Tmr1, Tmr3, Tmr4 und Tmr5).

Die Bit-Anzahl beschreibt die Größe der Zähl- und Vergleichsregister.

2.1 Normalmodus, Programmzeitmessung

Normalmodus



- Zählregister zählt zyklisch bis zum Überlauf.
- Beim Überlauf wird eine Überlaufbit und bei Gleichheit mit einem Vergleichsregister ein Gleichheitsbit gesetzt.
- Beispiel Wartefunktion:

```
void wait(uint32_t tw){
 <berechne+setze Taktauswahl+Vergleichswert>
 <Lösche Zähler und Gleichheitsbit>
 <warte bis Gleichheitsbit wieder gesetzt ist>
 <schalte Zähltakt aus> }
```

Zeitmessung für Float-Operationen

Gleitkommaoperationen (+, *, ...) werden auf unserem Prozessor mit Unterprogrammen realisiert und dauern lange. Wie lange?

```
#include <avr/io.h>
float a=26.34516, b=1045.6734;
uint16_t t;
int main(void){
   TCNT1 = 0;   // Normalmodus, Zähltakt 8MHz
   TCCR1B = 1;  // WGM:=0b0000, CS:=0b001
   a = (a + b) * 22.9856;
   TCCR1B = 0;  // Zähltakt aus
   t = TCNT1;
}
```

Zählwert in t am Programmende: 317 Takte (Befehlszyklen)

- davon ca. 20 für den Unterprogammaufruf und
- je 150 für die Abarbeitung einer Gleitkommaoperation.

Disassemblierter Programmausschnitt

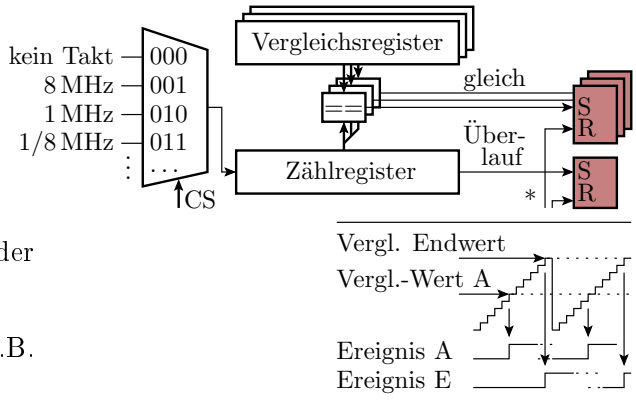
```
int main(void){
  TCNT1 = 0;
  TCCR1B = 1;  // WGM=0b0000, CS=0b001
  a = (a + b) * 22.9856;
// 0x009F  LDS R18,0x0200 ; r21:r18 := a
// 0x00A1  LDS R19,0x0201 ; (4 Byte)
// 0x00A3  LDS R20,0x0202 ;
// 0x00A5  LDS R21,0x0203 ;
// 0x00A7  LDS R22,0x0204 ; r25:r22 := b
// 0x00A9  LDS R23,0x0205 ; (4 Byte)
// 0x00AB  LDS R24,0x0206 ;
// 0x00AD  LDS R25,0x0207 ;
// 0x00AF  RCALL PC+0x001E; Gleitkommaaddition
// ... Konstante 22.9856 in r21:r18 laden
// ... Gleitkommamultiplikation
// ... a := r25:r22




// in r25:r24:r23:r22 steht a + b
// 0x00B0  LDI R18,0x82   ; r21:r18 := 22.9856
// 0x00B1  LDI R19,0xE2   ; (4 Byte)
// 0x00B2  LDI R20,0xB7   ;
// 0x00B3  LDI R21,0x41   ;
// 0x00B4  RCALL PC+0x00CE; Gleitkommamultipl.
// 0x00B5  STS 0x0204,R22 ; a := r25:r22
// 0x00B7  STS 0x0205,R23 ; (4 Byte)
// 0x00B9  STS 0x0206,R24 ;
// 0x00BB  STS 0x0207,R25 ;
   TCCR1B = 0;  // Zähler aus
   t = TCNT1;   // Zählwert speichern
}
```

2.2 CTC-Modus

CTC- (Clear on Compare) Modus

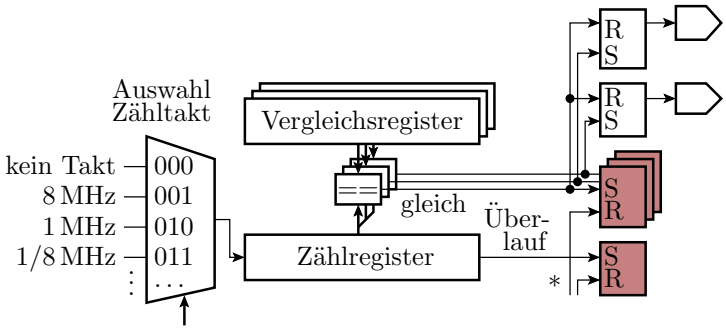


- Zähler wird bei Gleichheit mit einem der Vergleichsregister rückgesetzt.
- Auslösung zyklischer Ereignisse, z.B. Uhrenprozess:

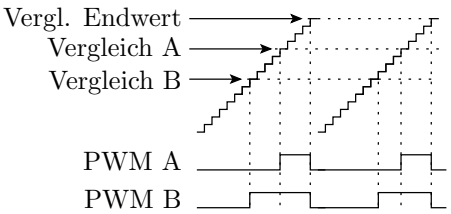
```
void Schrittfunktion Uhr(){
  if (<Vergleichs-Rücksetz-Ereignis>)
   <lösche Ereignisbit(s) und schalte Uhr weiter>
}
```

2.3 PWM-Erzeugung

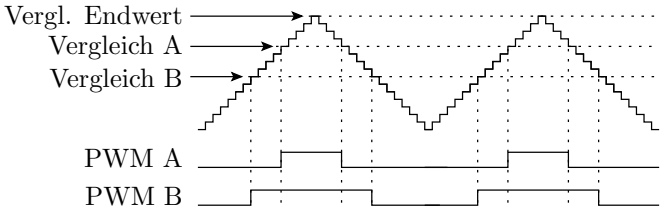
PWM-Erzeugung



- Zählerüberlauf setzt Ausgabebit.
- Vergleichsereignis löscht Ausgabebit.
- Pulsgenerierung z.B. zur Motoransteuerung ohne Polling und ISR.

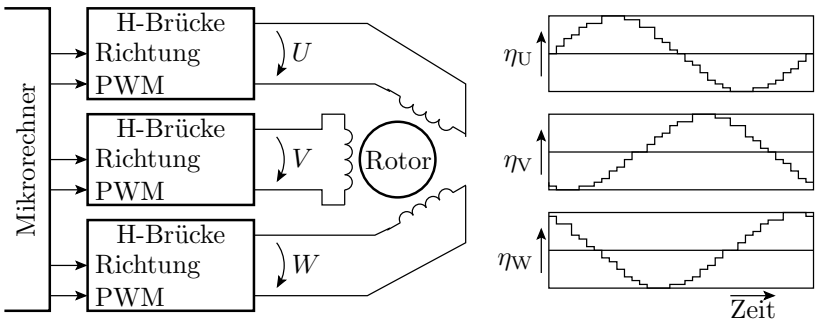


Symmetrische PWM[2]



- An den Endwerten schaltet die Zählrichtung um.
- Bei Gleichheit und Hochzählen wird die Ausgabe ein- und bei Gleichheit und Abwärtszählen ausgeschaltet.
- Bei dieser und der vorherigen PWM kann auch eine invertiert Ausgabe programmiert werden, so dass der Vergleichswert statt der Ausschalt-, die Einschaltzeit festlegt.

Typische Motoransteuerung

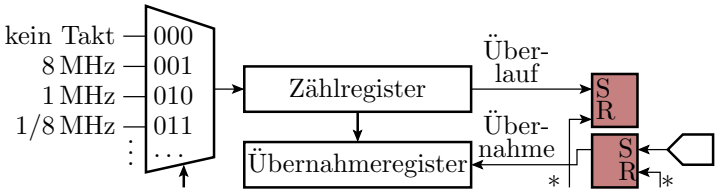


- Die Erzeugung von 3 sinusförmigen Mittelwertverläufen erfordert eine PWM-Einheit mit drei Vergleichsregistern.
- Ansteuerung über H-Brücken.
- Stufenlose Positions-, Geschwindigkeits- und Drehmomentsteuerung für viele Typen von Elektromotoren.

[2] Im Datenblatt unseres Prozessors ist das die phasenrichtige und die vorhergehende normale PWM die schnelle (Fast-) PWM.

2.4 Pulsweitenmessung

Pulsweitenmessung



- Externes Ereignis (Schaltflanke) bewirkt Übernahme des Zählwerts in das Übernahmeregister und setzt das Ereignisbit.
- Polling auf oder ISR-Aufruf bei Ereignisbitaktivierung.
- Programmgesteuerte Differenzbildung der Übernahmewerte zwischen den Übernahmeereignissen.

2.5 Watchdog

Watchdog-Timer (WDT)

Jedes größere Programm enthält statisch gesehen Fehler, die unter anderem auch dazu führen, dass das Programm abstürzt. Der WDT begrenzt die Dauer der Nichtverfügbarkeit durch Abstürze auf wenige ms bis s.

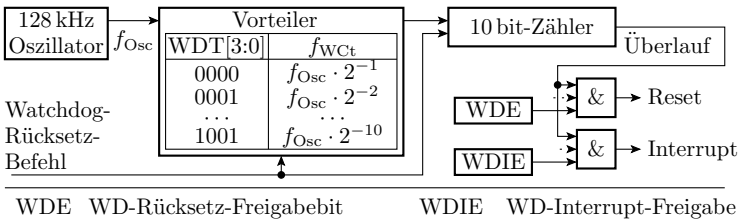
Funktionsprinzip:

- Zeitzähler, der Zeitimpulse zählt und bei Überlauf das System neuinitialisiert (und/oder Interrupt auslöst).
- Um Überläufe (Neuinitialisierungen) zu verhindern, muss das Programm in einer vorprogrammierten Mindestzeit Rücksetzbefehle für den Watchdog ausführen.

Programmierbar sind:

- die Zeit bis zum Überlauf und
- die Reaktion bei Überlauf (Interrupt, Neustart).

Watchdog-Timer (WDT) des ATmega 2560



- Zeit bis zum Überlauf: programmierbar von 16 ms bis 8 s.
- Nur-Interrupt: »Wiederbelebung« per Software.
- Interrupt + Rücksetzen: Datenretten + Neustart.
- WDT-Reset mit Fuse-Bit »WDTON« auch dauerhaft aktivierbar. Dann keine Deaktivierung durch Software (-Fehler) möglich.