Interrupt-Programmierung

Am Beispiel des
ATMEGA16 Microcontrollers

Beispiel: Messung der Betriebszeit

- Die Betriebszeit zeigt an, wie lange der Rechner seit dem Booten läuft
- Hier: Aktualisierung der Betriebszeit in Millisekundenschritten
- Eingesetzte Bibliothek: WinAVR
- Methode:
  - Programmierung eines Timer-Interrupts
  - Dieser zählt 1000 mal in der Sekunde eine Variable für die Betriebszeit hoch

Inhalt der Folien

- Die Timer/Interruptregister des Microcontrollers (gemäß der Dokumentation von ATMEL)
- Einstellen des Timers in Software
- Programmierung des Interrupt-Handlers

Wichtige Register

- TCNT0 Timer/Counter Register
  Zählt von Null aufwärts
- OCR0 Output Compare Register
  Obergrenze für den Zähler TCNT0
- TCCR0 Timer/Counter Control Register
  Gibt die Funktionsweise des Zählers an
- TIMSK Timer/Counter Interrupt Mask
  Schaltet einzelne Interrupts ein/aus
- TIFR Timer/Counter Interrupt Flag Register
  Zeigt aufgetretene Interrupts an

Was tut das Register TCCR0?

TCCR0 – Timer/Counter Control Register								
Bit	7	6	5	4	3	2	1	0
Name	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initialer Wert	0	0	0	0	0	0	0	0

Durch das Setzen einzelner Bits kann das Verhalten der Timer/Counter-Register eingestellt werden.

Für uns sind folgende Bits interessant:

WGM00/01 (Waveform Generation Mode) →Aktivierung des Zählers
CS02/01/00 (Clock Select) → Zählgeschwindigkeit

Bit 3 und 6 im Register TCCR0

WGM01	WGM00	Waveform Generation Mode
0	0	Normal
0	1	Pulse Width Modulation, Phase Correct
1	0	Clear Timer on Compare Match
1	1	Fast Pulse Width Modulation

Unsere Betriebsart:

Das Zählregister TCNT0 wird mit dem in OCR0 eingestellten Wert verglichen. Wenn die Werte gleich sind, wird TCNT0 im nächsten Takt wieder auf Null gesetzt. Außerdem wird ein Interrupt ausgelöst, sofern dies im Register TIMSK eingestellt ist.

Bits 0 bis 2 im Register TCCR0

CS02	CS01	CS00	Clock Select
0	0	0	Timer/Counter stopped
0	0	1	$clk_{I/O}$
0	1	0	$clk_{I/O}$ / 8
0	1	1	$clk_{I/O}$ / 64
1	0	0	$clk_{I/O}$ / 256
1	0	1	$clk_{I/O}$ / 1024
1	1	0	External clock on T0 pin (falling edge)
1	1	1	External clock on T0 pin (rising edge)

Unsere Betriebsart: Gezählt wird mit dem Takt des Microcontrollers (16 Mhz) durch 64.

Änderungen am Register TCCR0

TCCR0 – Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0
Name	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initialer Wert	0	0	0	0	0	0	0	0
Unser Wert	0	0	0	0	1	0	1	1

Hier müssen wir Bits setzen, dann wird TCNT0 250.000 mal in der Sekunde erhöht (= 16Mhz / 64).

Das Register OCR0

- Der Zähler TCNT0 wird 250.000 mal in der Sekunde hochgezählt
- Der Maximalwert von TCNT0 steht in Register OCR0
- Falls der Maximalwert erreicht wird

  $\Rightarrow$ Wird TCNT0 im nächsten Takt auf Null zurückgesetzt

  $\Rightarrow$ Und ein Interrupt wird ausgelöst.
- Die Betriebszeit soll in Millisekunden gemessen werden

  $\Rightarrow$ 1000 Interrupts in der Sekunde
- OCR0 muß also auf 249 gesetzt werden.

Das Register TIMSK

- Für die Zähler/Timerregister wurden passende Werte gewählt
- Damit auch Interrupts ausgelöst werden, müssen noch die Register TIMSK und TIFR behandelt werden
- Damit überhaupt irgendwelche Interrupts ausgelöst werden, muß außerdem das Interrupt Bit im Statusregister (SREG) des Microcontrollers gesetzt sein.

Das Register TIMSK

TIMSK – Timer/Counter Interrupt Mask Register								
Bit	7	6	5	4	3	2	1	0
Name	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initialer Wert	0	0	0	0	0	0	0	0
Wir brauchen	0	0	0	0	0	0	1	0

Das Register TIMSK gibt an, bei welchen Ereignissen Interrupts ausgelöst werden. Für uns ist das Bit

OCIE0
(Timer/Counter0 Output Compare Match Interrupt Enable)

relevant, das Interrupts bei Gleichheit des Zählers TCNT0 und des Maximalwerts OCR0 auslöst.

Das Register TIFR

- Gesetzte Bits im Interrupt Flag Register zeigen an, daß gerade ein Interrupt aufgetreten ist.
- Sie werden nach Aufruf des Interrupt Handlers automatisch gelöscht.

TIFR – Timer/Counter Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
Name	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initialer Wert	0	0	0	0	0	0	0	0

- Der Interrupt in diesem Beispiel wird durch das Bit OCF0 (Output Compare Flag 0) angezeigt.

Programmierung

- Die Register sind in einer Datei io.h definiert:

```
#include <avr/io.h>
```

- Dort finden sich solche Dinge:

```
...
#define TCCR0   _SFR_IO8(0x33)
#define CS00    0
#define CS01    1
#define CS02    2
#define WGM01   3
#define COM00   4
#define COM01   5
#define WGM00   6
...
```

Initialisierung des Timerinterrupts

```
int main( void )
{
        // Timer 0 kann auch Interrupts bei einen Zählerüberlauf
        // auslösen => Flag löschen, da hier nicht gewollt.
        // (Verwirrend aber wahr: Das TOV0-Bit wird durch
        // Schreiben einer 1 gelöscht)
        TIFR = 1 << TOV0;

        // Interrupts für das Zählerregister TCNT0 freischalten
        TIMSK = 1 << OCIE0;

        OCR0 = 249; // Zahlenobergrenze für das Zählerregister

        // Timer 0: Takt = Systemtakt / 64, CTC-Modus
        TCCR0 = ( 1 << WGM01 ) | ( 1 << CS01 ) | ( 1 << CS00 );

        sei(); // Interrupt-Bit im Statusregister setzen
               // => Alle Interrupts werden aktiviert
        while( 1 );
}
```

Behandlung von Interrupts

- Die Interruptvektortabelle speichert für jeden Interrupt die Startadresse einer Behandlungsroutine
- Mit dem SIGNAL-Makro aus der WinAVR-Bibliothek kann eine eigene Routine in die Interruptvektortabelle eingetragen werden:

```
SIGNAL( SIG_OUTPUT_COMPARE0 )
{
        // Betriebszeit aktualisieren
        uptime++;    // (ein 32-Bit Integer-Wert)
};
```

Auslesen der Betriebszeit

- Das Auslesen der Betriebszeit (32 Bit) dauert auf einem 8-Bit-Microcontroller mehrere Takte
- Während des Lesens sollte die Betriebszeit nicht durch eventuelle Interrupts geändert werden

```
uint32_t get_uptime()
{
        uint8_t  tmp_sreg;  // für Statusregister mit IRQ-Flag
        uint32_t tmp_time;  // Gelesene Betriebszeit

        tmp_sreg = SREG;    // Statusregister retten
        cli();              // Alle Interrupts abschalten
        tmp_time = uptime;  // Betriebszeit lesen
        SREG = tmp_sreg;    // ggf. Interrupts wieder einschalten
        return( tmp_time );
};
```