AVR Timers / counters.

Timer / Counter 0

General information:

Timer0 is a 8 bit timer/counter which can count from *0* to *0xFF*. In the timer mode this peripherie uses an internal clock signal and in the counter mode an external signal on *PORTB.0.* I take both modes of operation into consideration. Besides the timer can be operated either in the polling myode or in the interrupt mode.

Used registers:

Timer registers:

TCCR0 (Timer/Counter 0 Control Register)

TCNT0 (Timer/Counter 0 Value)

Interrupt registers:

TIFR (Timer Interrupt Flag Register)

TIMSK (Timer Interrupt Mask Register)

GIMSK (General Interrupt Mask Register)

Timer mode:

In this mode of operation the timer is provided by an internal signal. After each clock cycle the value of the *TCNT0* register is increased by one. The clock ratel is *x* times the oscillator frequency. The factor *x* can have the following values:

1, 8, 64, 256, 1024

(for example: 1024 - the timer is increased after 1024 cycles of the oscillator signal)

This prescaling is controlled by writing one of the following values into the register *TCCR0*:

initial value	**used** frequency
1	ck
2	ck/8
3	ck/64
4	ck/256
5	ck/1024

Polling mode:

In polling mode the timer is operated without using interrupts. Status information from the timer subsystem is retrieved by reading the appropriate registers in the foreground program loop.

Example:

```
/* Testprogramm for Timer/Counter 0 in the Polling Mode
   If the Timer has an overflow the overlow bit in the TIFR
   register will be set and the led variable increased.
   The led variable will be written to the PORTB.
   Leitner Harald 07/00
*/

#include <io.h>

uint8_t led;

int main( void )
{
      outp(0xFF, DDRB); /* use all pins on PORTB for output */
      outp(0, TCNT0); /* start value of T/C0 */
      outp(5, TCCR0); /* prescale ck/1024 */
      led = 0;

      for (;;)
      {
           // this while-loop checks the overflow bit
           // in the TIFR register
           while ((inp(TIFR) & 0x01) != 0x01) {}

           outp(~led,PORTB);
           led++;

           if (led == 255)
              led=0;

           // if a 1 is written to TOV0 bit,
           // the TOV0 bit will be cleared
           outp((1 << TOV0), TIFR);
      }
}
```

In that way the register *TCCR0* is loaded with *5 (ck/1024)* and the starting value of the timer in the register *TCNT0* is laid down with *0*. After each 1024th cycle of the oscillator the value of *TCNT0* is increased by one.

The *for(;;){}* defines an endless loop. In this loop a *do-while* loop is inserted, that constantly checks, if the bit at the place *0* of the *TIFR* register is set or not. This bit has the name *TOV0* (timer overflow 0) and is set when the 8 bit register *TCNT0* is of the value *0xFF* and tries to increase it -> overflow. In this case the *do-while* loop is left and the bontent of the variable led is written on *PORTB*.

Afterwards the variable *led* is increased by one and checks if *led* is of the value *0xFF*. In this case led is fixed to *0*. Otherwise you have to write a one into register *TIFR*, which has as a consequence that the *TOV0* is deleted and the timer starts counting from the beginning.

Interrupt mode:

This mode of operation is used more often than the polling mode. In this case the *TOV0* bit isn't constantly tested if it is set. In case of a timer overflow condition, the controller jumps from the actual position to the interrupt vector address in question. The interrupt routine itself is called from this vector address. After the interrupt routine is finished, the program returns to the instruction after the one, where it was interrupted.

Example:

```
/*
   Test program for Timer/Counter 0 in the Interrupt Mode
   Every time the Timer starts an interrupt routine the led
   variable is written on the PORTB and increased one time.
   Leitner Harald 07/00
*/

#include <io.h>
#include <interrupt.h>
#include <sig_avr.h>

uint8_t led;

SIGNAL (SIG_OVERFLOW0)
{
      outp(~led, PORTB); /* write value of led on PORTB */
      led++;
      if (led==255)
            led = 0;
      outp(0,TCNT0); /* reload timer with initial value */
}

int main( void )
{
      outp(0xFF, DDRB); /* use all pins on PORTB for output */
      /* enable the T/C0 overflow interrupt */
      outp((1 << TOIE0), TIMSK);
      outp(0, TCNT0); /* start value of T/C0 */
      outp(5, TCCR0); /* prescale ck/1024 */
      led = 0;
      sei(); /* set global interrupt enable */

      for (;;){}
}
```

The interrupt routine is introduced trough the keyword *SIGNAL*. As soon as an overflow occurs, this interrupt routine is carried out. In the main program you have to set the bits that enable the interrupt. In the register *TIMSK* you have to set the bit *TOIE0* and through the command *sei()* the *i*-bit (global interrupt enable) is enabled in the status register (*SREG*).

Counter mode:

In this mode of operation the status changes on the pin *T0* are counted. Instead of operating the signal on *T0* manually, using a frequency generator or any automated signal is equally possible. The following program counts the status changes on pin *T0* and increases the value of the counter register *TCNT0* by one.

One has to pay attention that the pin *T0* is situated on *PORTB*. Therefore pin 0 of *PORTB* has to be defined as an input and all others as an output.

The next step is the definition of the right mode of operation. The value *0x06* has to be written into the register *TCCR0*. Now the timer/counter is configured as a counter of falling edges on pin *T0*.

Polling mode:

Example:

```
/*
   Test program for Counter 0 in the Polling Mode
   If the Counter has an overflow the overlow bit
   in the TIFR register
   will be set and the led variable increased.
   The led variable will be written to the PORTB.
   Leitner Harald 07/00
*/

#include <io.h>

uint8_t led;

int main( void )
{
   outp(0xFE, DDRB); /* use pin 1-7 of PORTB as output; */
                     /* - and pin 0 (T0) as input */
   outp(0xFE, TCNT0);/* start value of counter */
   outp(6, TCCR0);   /* init the T/C as counter */
                     /* - triggered by falling edge on T0 */
   led = 2;

   for (;;) {
      /* this while-loop checks the overflow bit */
      while ((inp(TIFR) & 0x01) != 0x01) {}

      outp(0xFE, TCNT0); /* start value of counter */
      outp(~led,PORTB);
      led++;
      if (led==255)
         led=0;

      outp((1 << TOV0),TIFR);
   }
}
```

Interrupt mode:

```
/*
   Test program for Counter 0 in the Interrupt Mode
   Every time a falling edge is set on the T0 input
   the counter is increased for one time.
   Leitner Harald 07/00
*/

#include <io.h>
#include <interrupt.h>
#include <sig-avr.h>

uint8_t led;

SIGNAL (SIG_OVERFLOW0)
{
   outp(~led, PORTB); /* write value of led on PORTB */
   led++;
   if (led==255)
      led = 2;
   outp(0xFE,TCNT0); /* reload counter with initial value */
}

int main( void )
{
   outp(0xFE, DDRB); /* use all pins on PORTB for output */
   outp((1 << TOIE0), TIMSK); /* enables overflow interrupt*/
   outp(0xFE, TCNT0);      /* start value of counter */
   outp(6, TCCR0); /* count on falling edge on Pin T0 */
   led = 0;
   sei(); /* set global interrupt enable */

   for (;;){}
}
```

Timer / Counter 1

General information

In contrast to timer 0 or timer 2, timer 1 is a 16-bit timer/counter. Because of this, you can use it for longer counting sequences. The counting extent is between *0x0000* and *0xFFFF*. This area is being realised through two registers. Otherwise Timer 1 features compare/capture and a PWM.

Used registers:

Timer registers:

TCCR1A (Timer/Counter Control Register A)

TCCR1B (Timer/Counter Control Register B)

TCCR1L (Timer/Counter Value Low Byte)

TCCR1H (Timer/Counter Value High Byte)

OCR1AL (Output Compare Register A Low Byte)

OCR1AH (Output Compare Register A High Byte)

OCR1BL (Output Compare Register B Low Byte)

OCR1BH (Output Compare Register B High Byte)

ICR1L (Input Capture Register Low Byte)

ICR1H (Input Capture Register High Byte)

Interrupt registers:

TIFR (Timer Interrupt Flag Register)

TIMSK (Timer Interrupt Mask Register)

GIMSK (General Interrupt Mask Register)

Timer mode:

In this mode of operation the timer is supplied by an internal signal. After each takt cycle the meter reading is increased by 1. This signal is produced by a *n* times the amount of the oscillator signal. The factor *x* can have the following result:

1,8,64,256,1024

(for instance: 1024- only after 1024 cycles of the oscillators the timer is raised- the frequency is only fosc/1024) This results can be set with register *TCCR1B*.

The timer is adjusted through writing the following results into the register initial value used frequency

1	ck
2	ck/8
3	ck/64
4	ck/256
5	ck/1024

Polling mode:

Example:

```
/*
      Test program for Timer/Counter 1 in the Polling Mode
      If the Timer has an overflow the overflow bit of the TIFR
      register is set and the led variable increased.
      The led variable is then written to PORTB.
      Leitner Harald 07/00  */

#include <io.h>

uint8_t led;

int main( void )
{
   outp(0xFF, DDRB); /* use all pins on PORTB for output */
   outp(0x00, TCNT1L); /* start value of T/C1 - low byte */
   outp(0x00, TCNT1H); /* start value of T/C1 – highbyte*/
   outp(0, TCCR1A); /* T/C1 in timer mode */
   outp(1, TCCR1B); /* prescale ck */
   led = 0;

   for (;;) {
      // this while-loop checks the overflow bit in TIFR reg.
      while ((inp(TIFR) & 0x04) != 0x04) {}

      outp(~led,PORTB);
      led++;
      if (led==255)
         led=0;

      outp(0x00, TCNT1L); // start value of T/C1 - low byte
      outp(0x00, TCNT1H);  // start value of T/C1 - high byte
      outp((1<<TOV1),TIFR); // if a 1 is written to a TOV1 bit
                           // - the TOV1 bit will be cleared
   }
}
```

The *TCCR1A* register is established with 0 the *TCCR1B* register with 1 (CK-one prescale) and the starting value of the registers *TCNT1L* and *TCNT1H* with 0.

After each cycle of the quartz oscillator the meter reading of the *TCNT1L* register is increased by 1. After reaching the value *0xFF* in register *TCNT1L, the next clock cycle will increment TCNT1H* by 1 and reset *TCNT1L* to 0.

There is a *while* loop that constantly monitors if the bit in position 4 of the *TIFR* register is set or not. This bit is called *TOV1* (time overflow1) and is set by the processor at the fist clock cycle after that both 8 bit registers (*TCNT1L, TCNT1H*) are of the same value *0xFF*. In this case the *while* loop is left and the content of the variable led is written on *PORTB*.

Afterwards the variable *led* is increased by one and tested if it has the value 255 (0xFF). If this is the case, *led* is put to *0*. If not, a one is written in position 4 of the register *TIFR*, which has the consequence that *TOV1* bit is deleted and the timer has to begin to count once again.

Interrupt mode:

This mode of operation is used more frequently than polling. The *TOV1* bit is not always controlled if it has been put. Because if there is an overflow the appropriate vector address is mentioned. The interrupt routine is called up of this vector address.

After the work of this routine the programm goes at the point where it was interrupted.

Example:

```
/*
   Test program for Timer/Counter 1 in the Interrupt Mode
   Every time the Timer starts an interrupt routine the led
   variable
   is written on the PORTB and increased one time.
   Leitner Harald 07/00
*/

#include <io.h>
#include <interrupt.h>
#include <sig-avr.h>

uint8_t led;

SIGNAL (SIG_OVERFLOW1)
{
   outp(~led, PORTB);      // write value of led on PORTB
   led++;
   if (led==255)
      led = 0;
   outp(0,TCNT1L);         // reload timer with initial value
   outp(0XFF, TCNT1H);
}

int main( void )
{
   outp(0xFF, DDRB);             // use port B for output
   outp((1 << TOIE1), TIMSK); // enables the T/C1 overflow
                                 // - interrupt in the T/C
                                 // - interrupt mask register
   outp(0xFF, TCNT1H);           // start value of T/C1
   outp(0, TCNT1L);
   outp(0, TCCR1A);              // no compare/capture/pwm mode
   outp(5, TCCR1B);              // prescale ck/1024
   led = 0;
   sei();                        // set global interrupt enable

   for (;;){}
}
```

The interrupt routine is introduced through the keyword *SIGNAL*. As soon as an overflow occurs the routine is carried out. In the main program necessary interrupts must be enabled.

In register *TIMSK* the bit *TOIE1* has to be set and through the instruction the *I*-bit in the status register is enabled.

In all other cases the timer is initialised like in the polling mode.

Counter mode:

In this mode of operation the changes of the statue are counted on the external pin *T1*. In case of an overflow an interrupt routine is called up. Instead of the manual handling of entry *T1*, a supply through a frequency generator is equally possible.

The program is in the most parts equal to that of timer0. One should pay attention that pin *T1* is situated at *PORTB*. Because of that one has to define Pin1 of *PORTB* as an entry and all the others as an outlet. The next step is the determination of the suitable mode of operation. For that the value *0x6* is written into register *TCCR1B*. Now the timer/counter is configurated to pin T1 as a counter of falling edges.

Compare mode:

The Timer/Counter 1 supports two output compare functions using the registers *OCR1A* (low and high byte) and *OCR1B* (low and high byte) as the data sources to be compared to the content of the Timer/Counter register *TCNT1* (low and high byte). If there is a compare match it is possible to clear the content of the Timer/Counter register (only if compare with *OCR1A*) or take effects on the output pins. This pins are called *OC1A* (PORTD.5) and *OC1B* (PORTD.4).

The different functions are controlled by the register *TCCR1A* as follows:

Bit 0-3:	not used
Bit 4:	COM1B0
Bit 5:	COM1B1
Bit 6:	COM1A0
Bit 7:	COM1A1

Mode Select:

COM1X1	COM1X0	Description
0	0	T/C 1 disconnected from pin OC1X
0	1	Toggle the value of OC1X
1	0	Clear OC1X
1	1	Set OC1X

The *CS10*, *CS11* and the *CS12* bit (bit0-2) of register *TCCR1B* definesthe prescalling source of Timer/Counter 1 as follows:

CS12	CS11	CS10	Description
0	0	0	T/C 1 stopped
0	0	1	*CK*
0	1	0	*CK/8*
0	1	1	CK/64
1	0	0	CK/256
1	0	1	CK/1024
1	1	0	clocked by the pin T1, falling edge
1	1	1	clocked by the pin T1, rising edge

If you want to clear the content of Timer/Counter 1 on a compareA match, it is necessary to set bit3 in the register *TCCR1B*. In a compare match the suitable bit (*OCIE1A* -> bit4, *OCIE1B* -> bit3) is set in the *TIMSK* register or the interrupt routine (*SIG_OUTPUT_COMPARE1A, SIG_OUTPUT_COMPARE1B*) is carried out.

Example:

```
/*
   Flashes LED on STK200 Board with Compare - Mode of Timer 1
   Pulse width is regulated by switch PD2 and PD3 (Int0 Int1)
   07/00 Leitner Harald
*/

#include <io.h>
#include <interrupt.h>
#include <sig-avr.h>

uint8_t delay;

SIGNAL (SIG_OUTPUT_COMPARE1A)/* Compare interrupt routine */
{
   if (delay == 0)
      outp(0XFF, PORTB);
   else
      outp(0XFE, PORTB);
}

SIGNAL (SIG_OVERFLOW1)/* T/C1 overflow interrupt routine */
{
   if (delay == 0)
      outp(0XFF, PORTB);
   else
      outp(0XFF, PORTB);

   outp(0XFF, PORTB);
   outp(delay,TCNT1H);
   outp(0,TCNT1L);
}
```

```
SIGNAL (SIG_INTERRUPT0) /* PD2 */
{
   if (delay < 15)
      delay = 0;
   else
      delay = delay - 15;
}

SIGNAL (SIG_INTERRUPT1) /* PD3 */
{
   if (delay > 235)
      delay = 250;
   else
      delay = delay + 15;
}

int main( void )
{
   outp(0xFF, DDRB); /* define PORTB as Output (Leds)*/
   outp(0x00, DDRD); /* define PORTD as Input (Switches) */

   delay = 120; /* default of timer1 high byte */

   /* Switches PB3,PB4 for Interrupt 0 and 1 */
   outp((1 << TOIE1) | (1 << OCIE1A), TIMSK);
   outp((1 << INT0)  | (1 << INT1), GIMSK);
   outp((1 << ISC01) | (1 << ISC10) | (1 << ISC11), MCUCR);
   outp(delay, TCNT1H); /* Init T1 */
   outp(0, TCNT1L);
   outp(0XFF, OCR1AH); /* Compare value of T1 */
   outp(0X10, OCR1AL);
   outp(0, TCCR1A); /* Timer mode with no output */
   outp(1, TCCR1B); /* counting with ck */
   sei();

   for (;;){}
}
```

This program shows a possible use of the compare function. The timer is initialized in the main program. To controll the delay of the compare mode there are two switches *PD2* and *PD3* on the STK 200 board. A variable delay is decreased by 15 when using *PD2* and when using *PD3* increased by 15. This value delay is loaded into the high-byte of the timer/counter register.

Out of this value the timer tries to reach the overflow. Before it can reach an overflow a compare match occurs and the suitableinterrupt routine is called up.

Because of this routine a led on the board flashes.Afterwards the timer runs to an overflow and carries out the overflowinterrupt routine. The led is switched off again and the timer new isinitialised with the value delay.

Capture mode:

This mode of operation enables to memorize the actual value of Timer/Counter 1 through an external signal.

When the rising or falling edge of the signal at the input capture pin *ICP* (*PORTD.6*) is detected, the current value of Timer/Counter 1 is transfered to the 16bit input capture register *ICR1* (*ICR1L, ICR1H*). This sets the input capture flag *ICF1* and is possible to call up a interrupt routine (*SIG_INPUT_CAPTURE*) if the bit *TICIE1* (bit5 of *TIMSK*) is set. The most important thing is that you have to read the low byte (*ICR1L*), for a full 16bit register read, first.

Example:

```
/*
   Flashes LED on STK200 Board with Capture - Mode
   of Timer 1. ---  With each capture signal on PORTD.6,
   the actual value of timer 1 (high byte) is written at PORTB.
   07/00 Leitner Harald
*/

#include <io.h>
#include <interrupt.h>
#include <sig_avr.h>

SIGNAL (SIG_OVERFLOW1)
{
   outp(0, TCNT1H); /* reset timer after overflow */
   outp(0, TCNT1L);
}

SIGNAL (SIG_INPUT_CAPTURE1)
{
   outp(~inp(ICR1L), PORTB);
   /* reading timer value of capture register */
   outp(~inp(ICR1H), PORTB); /* and write to PORTB */
}

int main( void )
{
   outp(0xFF, DDRB); /* define PORTB as Output (Leds) and */
   outp(0x00, DDRD); /* PORTD as Input (Switches) */
   outp(0xFF, PORTB);

   /* Enable interrupt for capture and overflow */
   outp((1 << TICIE1) | (1 << TOIE1), TIMSK);
   outp(0, TCNT1H); /* Init T1 */
   outp(0, TCNT1L);
   outp(0, TCCR1A); /* Timer mode with no output */
   outp(5, TCCR1B); /* counting with ck/1024 */
   sei();

   for (;;){}
}
```

This program reads the actual value of Timer/Counter 1 after a signal has occured on the pin *ICP*. Then the high-byte of this value is written to the Leds on *PORTB*.

PWM mode:

When the PWM (PulseWidthModulation) mode is selected the Timer/Counter 1 can be used as an 8,9 or 10bit, free running PWM. Timer/Counter 1 acts as an up/down counter that is counting up from *0x0000* to the selected top (8bit -> *0x00FF*, 9bit -> *0x01FF*, 10bit -> *0x03FF*), where it turns and counts down to *0x0000* and repeats this cycle endlessly. When the counter value matches the content of the compare register (*OCR1A, OCR1B*) it has an effect on the output pins *OCA1* and *OCB1* as follows:

COM1X1	COM1X0	Effect on OCX1
0	0	no
0	1	no
1	0	cleared on compare match, up-counting, set on compare match, down-counting
1	1	cleared on compare match, down-counting, set on compare match, up-counting

These bits are set in the register *TCCR1A:*

(*COM1A1* -> bit7, *COM1A0* -> bit6, *COM1B1* -> bit5, *COM1B0* -> bit4)

The right PWM mode is selected bits *PWM10* (bit0 of *TCCR1A*) and *PWM11* (bit1 of *TCCR1A*) as follows:

PWM11	PWM10	Description
0	0	PWM mode disabled
0	1	8bit PWM
1	0	9bit PWM
1	1	10bit PWM

Example:

```
/*
   Testprogramm for Timer/Counter 1 PWM Mode 10bit
   The T/C is used as a free running 10bit-PWM.
   The T/C is counting from 0x00 up to 0x3FF and after
   reaching 0x3FF the T/C is counting down to 0x00.
   This cycle repeats endless.
   When the counter value matches the content of the output
   compare register during counting up, PD5(OC1A) pin
   is cleared
   and if matches while counting down the PD5(OC1A) is set.
   When the counter value matches the content of the output
   compare register during counting up, PD4(OC1B) pin is set
   and if matches while counting down the PD4(OC1B) is cleared.
   Leitner Harald 07/00
*/

#include <io.h>
```

```
int main( void )
{
   outp(0xFF, DDRD); /* use all pins on port D for output */
   outp(0xB3, TCCR1A); /* init the counter */
   outp(0x5, TCCR1B); /* init the counter */
   outp(0x00, TCNT1L); /* value of T/C1L */
   outp(0x00, TCNT1H); /* value of T/C1L */
   outp(0xFF, OCR1AL); /* value of Compare reg. A Low-Byte */
   outp(0x00, OCR1AH); /* value of Compare reg. A High-Byte */
   outp(0xFF, OCR1BL); /* value of Compare reg. B Low-Byte */
   outp(0x00, OCR1BH); /* value of Compare reg. B High-Byte */

for (;;){}
}
```

This program creates a free running PWM signal on the two outputs *OCA1* and *OCB1*. The two compare registers *OCR1A* and *OCR1B* define the pulswidth.

Selected Mode:

TCCR1A:
10bit PWM

Com1A1 -> 1, *COM1A0* -> 0

Cleared on compare match, up-counting,

set on compare match, down-counting

Com1B1 -> 1, *COM1B0* -> 1

Cleared on compare match, down-counting,

set on compare match, up-counting

TCCR1B:
prescale: ck/10

Timer 2

General information:

Timer2 is a 8 bit timer that counts from *0* to *0xFF*. In the timer mode this peripherie uses an internal clock signal. Besides the timer can be operated either in the polling mode or in the interrupt mode.

Used registers:

Timer registers:

TCCR2 (Timer0 Control Register)

TCNT2 (Timer0 Value)

OCR2 (Output Compare Register of Timer2)

ASSR (Asynchronous Status Register)

Interrupt registers:

TIFR (Timer Interrupt Flag Register)

TIMSK (Timer Interrupt Mask Register)

GIMSK (General Interrupt Mask Register)

Timer mode:

In this mode of operation the timer is provided by an internal signal. The value of the *TCNT2* register is increased by one after each clock cycle. This clock signal is produced out of *x* times the amount of the oscillator signal. The factor *x* can have the following values:

1, 8, 32, 64, 128, 256, 1024

(for example: 1024 - the timer is increased after 1024 cycles of the oscillator signal)

This prescaling is controlled by writing one of the following values into the register *TCCR2*:

initial value	used frequency
1	ck
2	ck/8
3	ck/32
4	ck/64
5	ck/128
6	ck/256
7	ck/1024

Timer2 is no Timer/Counter but only a Timer for internal signals.

Polling mode:

Example:

```
/*
   Test program for Timer/Counter 2 in the Polling Mode
      If the Timer has an overflow the overlow bit in the TIFR
   register
   is set and the led variable increased.
   Then the variable led is written to PORTB.
   Leitner Harald 07/00
*/

#include <io.h>

uint8_t led;
uint8_t state;

int main( void )
{
   outp(0xFF, DDRB); /* use all pins on PORTB for output */
   outp(0, TCNT2); /* start value of T/C2 */
   outp(7, TCCR2); /* prescale ck/1024 */
   led = 0;
   for (;;) {
      do { // this while-loop checks the overflow bit in TIFR
         state = inp(TIFR) & 0x40;
      } while (state != 0x40);

      outp(~led,PORTB);
      led++;
      if (led==255)
         led=0;

      outp((1 << TOV2), TIFR);  // if a 1 is written to TOV2,
                                // -- the TOV2 will be cleared
   }
}
```

In that way the register *TCCR2* is loaded with *7* (*ck/1024*) and the starting value of the timer in register *TCNT2* is laied down with *0*. After each 1024th cycle of the oscillator the value of TCNT2 is increased by one. The *for(;;){}* defines an endless loop. In this loop a *do-while* loop is inserted, that constantly checks, if the bit on the place 6 of the *TIFR* register is set or not. This bit has the name *TOV2* (timer overflow 2) and is set when the 8 bit register *TCNT2* is of the value *0xFF* and tries to increase it -> overflow. In this case the *do-while* loop is left and the bontent of the variable *led* is written on *PORTB*.

Afterwards the variable *led* is increased by one and checks if *led* is of the value *0xFF*. In this case *led* is fixed to *0*. Otherwise you have to write a one into register *TIFR on position 6* , what has as a consequence that the *TOV2* is deleted and the timer starts counting from the beginning.

Interrupt mode:

This mode of operation is used more often than the polling mode. In this case the *TOV2* bit isn't constantly proved if it was set. Because in case of an overflow the controller jumps from the actual position to the suitable interrupt vector address. The interrupt is called from this vector address.

After this execussion the program goes on the place, where it was interrupted.

Example:

```
/*
   Test program for Timer/Counter 2 in the Interrupt mode
   Every time the Timer starts an interrupt routine the
   led variable
   is written on the PORTB and increased one time.
   Leitner Harald 07/00
*/

#include <io.h>
#include <interrupt.h>
#include <sig_avr.h>

uint8_t led;

SIGNAL (SIG_OVERFLOW2)
{
   outp(~led, PORTB); /* write value of led on PORTB */
   led++;
   if (led==255)
      led = 0;
   outp(0,TCNT2); /* reload timer with initial value */
}

int main( void )
{
   outp(0xFF, DDRB); /* use all pins on PORTB for output */
   /* enable the T/C2 overflow interrupt in the T/C interrupt
      mask register for */
   outp((1 << TOIE2), TIMSK);
   outp(0, TCNT2); /* start value of T/C2 */
   outp(7, TCCR2); /* prescale ck/1024 */
   led = 0;
   sei(); /* set global interrupt enable */

   for (;;){}
}
```

The interrupt routine is introduced trough the keyword *SIGNAL*. As soon as an overflow occurs, this interrupt routine is carried out. In the main program you have to fix the bits that enable the interrupt. In the register *TIMSK* you have to set the bit *TOIE2* and through the command *sei()* the *i*-bit (global interrupt enable) is enabled in the status register (*SREG*).

Compare mode:

The Timer2 supports one output compare function using the register *OCR2* as the data source to be compared to the content of the Timer2 data register *TCNT2.* If there is a compare match it is possible to clear the content of Timer2 data register *TCNT2* or to take effect on the output compare pin *OC2 (PORTD.7).*

The different functions are controlled by the register *TCCR2* in the following way:

Bit 0-2	used for prescale (CS20 -> bit0, CS21 -> bit1, CS22 -> bit2)
Bit 3	CTC2 (Clear Timer on Compare Match)
Bit 4	COM20
Bit 5	COM21
Bit 6	PWM2
Bit 7	not used

Mode select:

Com21	Com20	Description
0	0	Timer2 disconnected from pin OC2
0	1	Toggle the value of OC2
1	0	Clear OC2
1	1	Set OC2

Bits 0-2 of register *TCCR2* defines the prescaling source of timer2 in the following way:

CS22	CS21	CS22	Description
0	0	0	Timer2 stopped
0	0	1	ck
0	1	0	ck/8
0	1	1	ck /32
1	0	0	ck /64
1	0	1	ck /128
1	1	0	ck /256
1	1	1	ck /1024

If you want to clear the content of timer2 on a compare match it is necessary to set the CTC2 bit (bit 3) of the TCCR2 register. On a compare match the bit OCIE2 (bit 7) in the TIMSK register is set or a interrupt routine is called up (SIG_OUTPUT_COMPARE2).

Example:

```
/*
   Flashes LED on STK200 Board with Compare - Mode of Timer 2
   Timer starts with value 0x10, if the counter register
   has the same
   value as the ocr register the Led will be switched on.
   If the timer overflow flag is set the leds will be
   switched off.
   Leitner Harald 07/00
*/
```

```
#include <io.h>
#include <interrupt.h>
#include <sig_avr.h>

uint8_t delay;

SIGNAL (SIG_OUTPUT_COMPARE2)
{
   outp(0X00, PORTB); /* turn on leds on PORTB */
}

SIGNAL (SIG_OVERFLOW2)
{
   outp(0XFF, PORTB); /* turn off leds on PORTB */
   outp(delay,TCNT2);
}

int main( void )
{
   outp(0xFF, DDRB); /* define PORTB as output (leds) */
   delay = 0x10;
   /* enables interrupt of timer: */
   outp((1 << TOIE2)|(1 << OCIE2), TIMSK);
   outp(delay, TCNT2); /* default value of timer */
   outp(0x80, OCR2); /* value of compar register */
   outp(7, TCCR2); /* selected prescale : ck/1024 */
   sei();

   for (;;){}
}
```

This program shows a possibility of using timer2 in the compare-mode. The timer2 is initialized in the *main()* function of the program. Value *delay* (0x10) is the starting value for the register *TCNT2* and the value of the compare register *OCR2* is set to *0x80.*

Timer2 counts from *0x10* to *0x80*. At this point the program calls the interrupt routine for the compare match up and turns on the leds on *PORTB*. Then Timer2 counts until an overflow has occured. Then the interrupt routine for the overflow is reached and the leds are turned off. Then the cycle starts counting endlessly from *delay (0x10)*.

PWM mode:

When the PWM (Pulse Width Modulation) mode is selected, timer2 is used as a 8bit free running PWM. Timer2 acts as an up/down counter that is counting up from *0x00* to *0xFF*, where it turns and counts down to zero and repeats this cycle endlessly.

To choose this mode you have to set the bit *PWM2* (bit 6) of the register *TCCR2*. When the counter value matches the content of the compare register *(OCR2*) there will be effects on the output pin *OC2 (PORTD.7)* in the following way:

COM21	COM20	Description
0	0	no effects
0	1	no effects
1	0	cleared on compare match, up-counting, set on compare match, down-counting
1	1	cleared on compare match, down-counting,set on compare match, up-counting

(COM21 -> bit5 and COM20 -> bit4 of register TCCR2)

This program creates a free running PWM signal on the output pin OC2. The compare register OCR2 defines the signals form.

Example:

```
/*
   Test program for Timer/Counter 2 PWM Mode
   The T/C is used as a free running PWM.
   The T/C is counting from 0x00 up to 0xFF and after
   reaching 0xFF the T/C is counting down to 0x00.
   Then the cycle repeats.
   When the counter value matches the content of the output
   compare register during counting up, PD7(OC2) pin is cleared
   and if matches while counting down the PD7(OC2) is set.
   Leitner Harald 07/00
*/

#include <io.h>
#include <interrupt.h>
#include <sig-avr.h>

int main( void )
{
   outp(0xFF, DDRD); /* use all pins on PORTD for output */
   outp(0, TCNT2); /* start value of T/C2 */
   outp(0x67, TCCR2); /* init the counter */
   outp(0x19, OCR2); /* value of OCR2 */

   for (;;){}
}
```

Selected mode:

Prescale: *ck/1024*

PWM*: COM20 -> 1, COM21 -> 1*