Jacobs University Bremen

Natural Science Laboratory

Embedded Systems Laboratory
CO26-300312

Fall Semester 2017

Timer/Counter Overflow Interrupt

Mailbox :
104

Instructor :
Ph.D. Fangning Hu

Author :
Biruk Amare

September 29, 2017

1 Introducion

The main objective of this lab is to use a separate counter called Timer in microcontroller in order to count clock cycles and blink the led by using the timer.

1.1 Introduction to Timer

A timer is a register which is not a normal one. The value of this register increases/decreases automatically. In AVR, timers are of two types : 8-bit and 16-bit timers. In an 8-bit timer, the register used is 8-bit wide where as in 16-bit timer, the register width is of 16 bits. This means that the 8-bit timer is capable of counting $2^8 = 256$ steps from 0 to 255.

Similarly, a 16-bit timer is capable of counting $2^{16} = 65536$ steps from 0 to 65535. Due to this feature, **timers are also known as counters**. Now what happens once they reach their MAX? If they reach a maximum value then it returns to initial value of zero. We say timer/ counter overflows.

The best part is that the timer is totally independent of the CPU. Thus, it runs parallel to the CPU and there is no CPU's intervention, which makes the timer quite accurate.

A Timer can either be an 8-bits counter(register) called **TCNT0** or 16-bits counter called **TCNT1**. The counter will increase 1 for each clock cycles independently from the CPU and trigger an internal interrupt signal when the counter overflows (becomes 0 again). The interrupt vector is called **TIMER0_OVF** (for TCNT0) or **TIMER1_OVF**(for TCNT1). In order to enable this interrupt, you need to set the corresponding bit to 1 in the Timer Interrupt Mask egister **TIMSK0**(for TCNT0) or **TIMSK1**(for TCNT1).

$$TimerCount = (RequiredDelay/Clocktimeperiod) - 1 \tag{1}$$

This technique of frequency division is called **Prescaling**. We do not reduce the actual *F_CPU*. The actual *F_CPU* remains the same (at $4MHZ$ in this case). So basically, we derive a frequency from it to run the timer. Thus, while doing so, we divide the frequency and use it. There is a provision to do so in AVR by setting some bits. It comes at a cost. There is a trade-off between resolution and duration.

Thus, we always choose prescaler which gives the counter value within the feasible limit (255 or 65535) and the counter value should always be an integer.

1.2 TCCR1B Register

The **Timer/counter1 Control Register B** - TCCR1B Register is as follows.

Bit	7	6	5	4	3	2	1	0	
(0x81)	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	

FIGURE 1 – Breadboard Diagram

The **bit 2 :0- CS12 :10** are the **Clock Select**

1.3 TCNT1 Register

The **Timer/Counter1** - TCNT1 Register is as follows. It is 16 bits wide since the TIMER1 is a 16-bit register. **TCNT1H** represents HIGH byte whereas **TCNT1L** represents the LOW byte. The timer/counter value is stored in these bytes.

1.4 TIMSK Register

The **Timer/Counter Interrupt Mask Register** - TIMSK Register is as follows : **Bits 5 :2** corresponding to TIMER1. Right now, we are interested in the yellow bit only. Other bits are related to CTC mode which we will discuss later. **Bit2 - TOIE1 - Timer/Counter1 Overflow Interrupt Enable** bit enables the overflow interrupt of TIMER1. We enable the overflow interrupt as we are making the timer overflow. Only **Bits 5 :2** are related to TIMER1. Of these, we are interested in **Bit2 - TOV1 - Timer/Counter1 Overflow Flag**. This bit is set to '1' whenever the timer overflows. It is cleared (to zero) automatically as soon as the corresponding Interrupt Service Routine (ISR) is executed. Alternatively, if there is no ISR to execute, we can clear it by writing '1' to it.

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

FIGURE 2 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x70)	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	TIMSK2
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

FIGURE 3 – Timer/Counter2 Interrupt Mask Register

2 Prelab

2.1 Read the ATmega328 datasheet (Chapter Timer/Counter0 or Timer/Counter1) and study the functions of necessary registers in order to have a Timer Overflow Interrupt. Calculate whether Timer 0 is big enough to have a 1 second overflow. If not, you should change to the 16-bits Timer/Counter1. Please calculate a proper combination of the initial value and a couting frequency to produce 1 second delay.

To check if Timer 0 is big enough to have a 1 second overflow, we first calculate the least number of cycles needed to generate a 1 second delay. Considering the biggest prescaler ,which is 1024, we can get the number of cycles by $\frac{clk_{IO}}{Prescaler} = \frac{8 \cdot 10^8}{1024} = \underline{\underline{7812.5}}$. This means that 7812 cycles are needed to generate a 1 second delay. The 8-bit register has a maximum count of 255 and therefore cannot create a 1 second delay, so we need to use the 16-bit register. Thus using the prescaler 1024, we can start count at 57722 or using the prescaler 256, we can start counter at 34285.

2.2 Read the ATmega328 datasheet (Chapter Interrupts) and find out the name of the Timer Overflow Interrupt Vector in the Interrupt Vector Table. Be careful to the different devices in the datasheet, they have different Tables!!

Table 1 – Interrupt Vector Table

Vector No.	Program Address	Source	Interrupt Definition
10	0x0012	TIMER2_OVF	Timer/Counter2 Overflow
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow

The Timer Overflow Interrupt Vectors in the Interrupt Vector Table are `TIMER0_OVF_vect` and `TIMER1_OVF_vect` for Timer 0 and Timer 1 respectively.

3 Lab Assignments :

3.1 Make your LED blink by using Timer both in C and Assembly language

```
1  #include <avr/io.h>
2  #define F_CPU 8000000UL
3  #include <util/delay.h>
4  #include <avr/interrupt.h>
5
6  int main(void)
7  {
8    cli(); //Clear interrupt
```

```
9     DDRD = 0xFF; //set the portd as output, turn on
10
11    TCNT1 = 57724;
12    TCCR1B = 0b00000100; //prescale value = 256
13    TIMSK1 = 0x01; //TOIE1 enabling overflow
14    sei(); //set global interrupt
15    while(1){}
16  }
17  ISR(TIMER1_OVF_vect)
18  {
19    TCNT1=57724; //16 bit counter value
20      PORTD ^= 0xFF;
21  }
```

4 Evaluation

4.1 Give your circuit diagram. Explain you circuit design

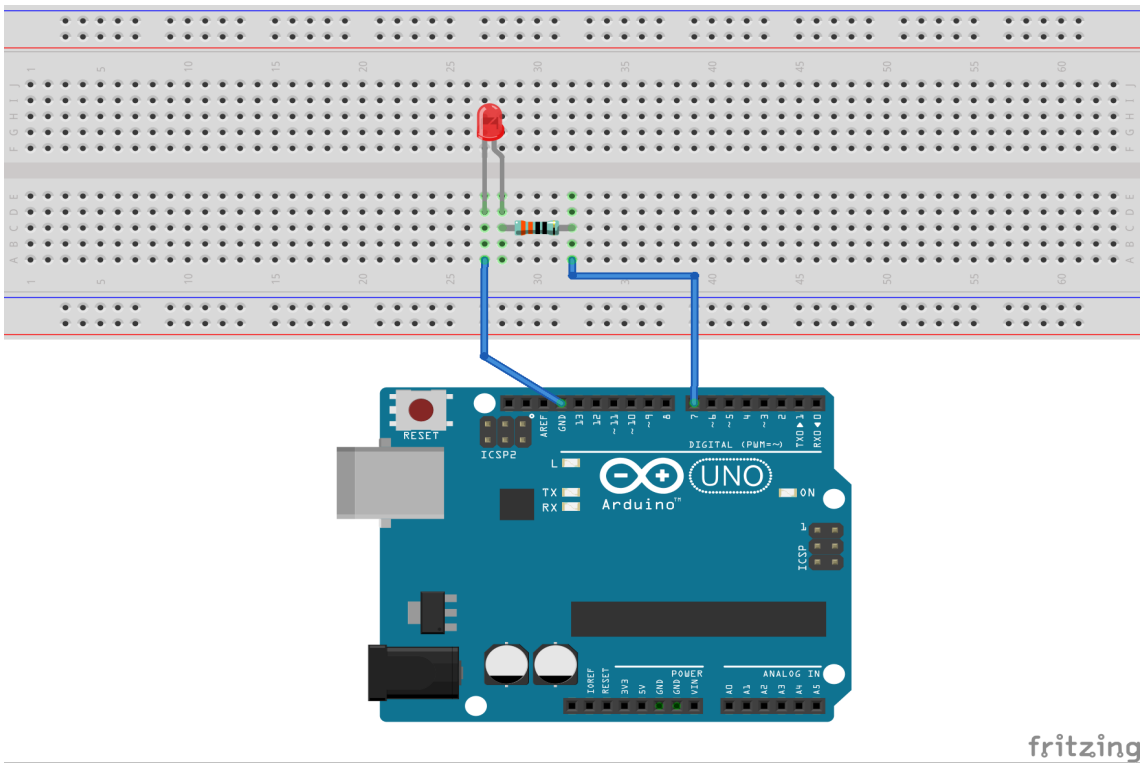

FIGURE 4 – Breadboard Diagram

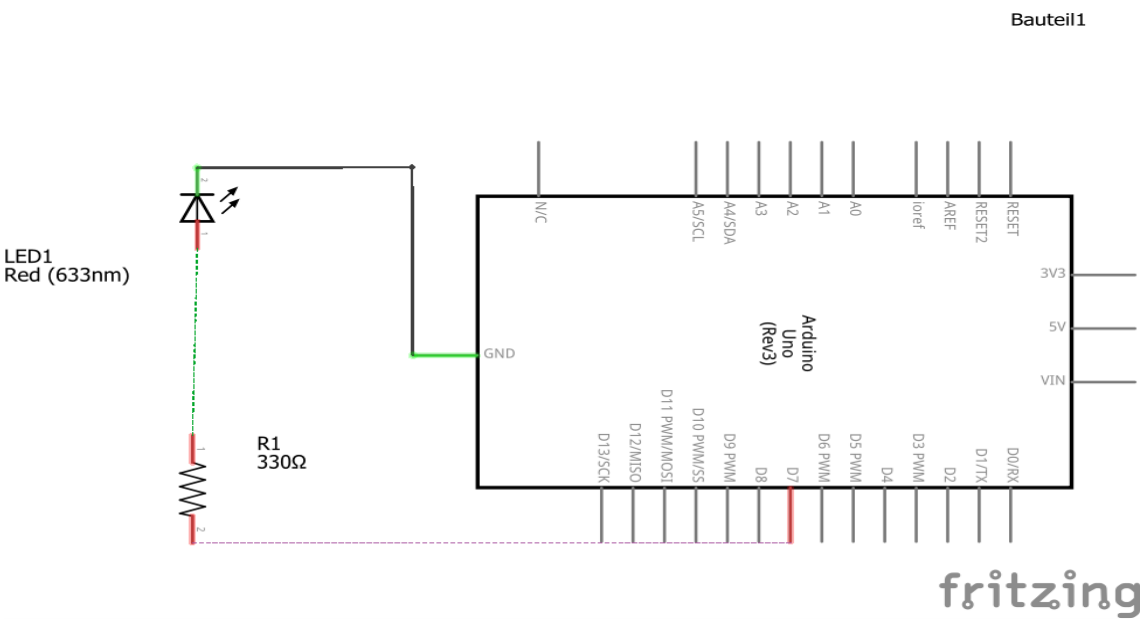

FIGURE 5 – Schematic Diagram

5 References

[1] http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-
datasheet_Complete.pdf1

[2] http://embsys-fhu.user.jacobs-university.de/?page_id=49