Microprocessor Systems Design

EEE 42101

Experiment 2: Timers and Interrupts.

Objectives:

- Study Timers
- Study interrupts and interrupts service routine.

Tools:

1- PC
2- Arduino Nano board
3- Testing board
4- MiniB-USB cable

Note: all material and sources of this course will be available on:

https://github.com/ashrafmalraheem/Mircoprocessor_Course

Feel free to download, study and modify for your own projects.

Part 1: Timer/Counter

Timer is a counter with 8, 16- or 32-bit register it depends on the architecture of the microcontroller unit. It increments its value by one every one clock cycle. The clock of the timer need not to be the same of the core clock. It can be its clock or its clock divided by a certain value. It also can have external clock source. All of this is set by bits on its control registers. You can get and set the value of the timer by accessing its data register. You can also generate interrupt if an overflow occurred or certain value is reached. Then you can define an interrupt service routine to perform certain function.

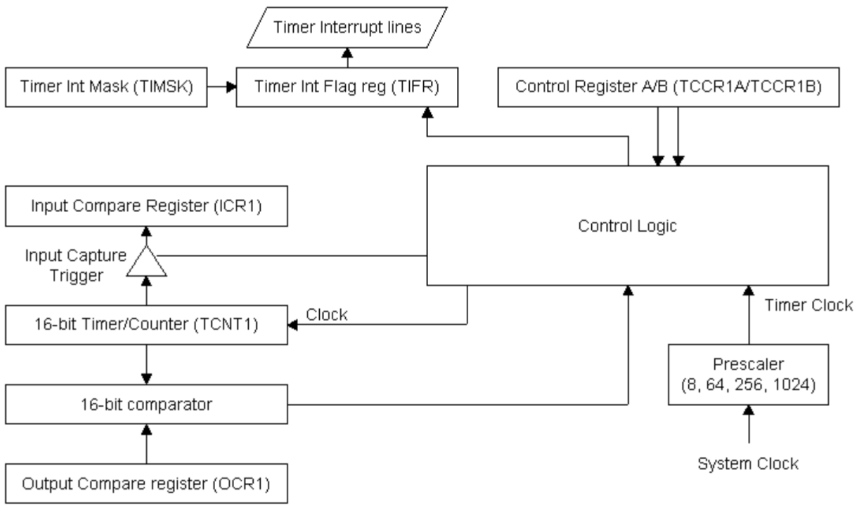

Timer Interrupt lines
Timer Int Mask (TIMSK)
Timer Int Flag reg (TIFR)
Control Register A/B (TCCR1A/TCCR1B)
Input Compare Register (ICR1)
Control Logic
Input Capture Trigger
Timer Clock
16-bit Timer/Counter (TCNT1)
Clock
Prescaler (8, 64, 256, 1024)
16-bit comparator
System Clock
Output Compare register (OCR1)

Figure 1 16-bit Timer/Counter Block Diagram

The advantage of using timers is that they are separate device run without the microcontroller core intervention. E.g. it can generate interrupts when it has an overflow. Therefore, they can be used as a delay without wasting any of the processor resources(time) Instead of delay function.

To set a timer to Blink LED every **1/2 seconds.** you should firstly decide which timer you should use: 8-bit or 16-bit timers?

$$interval\ time\ of\ interrupt = \frac{2 * prescaler * 2^{timer\ no.\ of\ bits}}{f_{clk}}\ , prescaler\ could\ be\ 1, 8, 64, 256\ or\ 1024$$

$$f_{clk} = 16MHz$$

Q: What is the selected value of prescaler? Solve the above equation using 16-bit timer

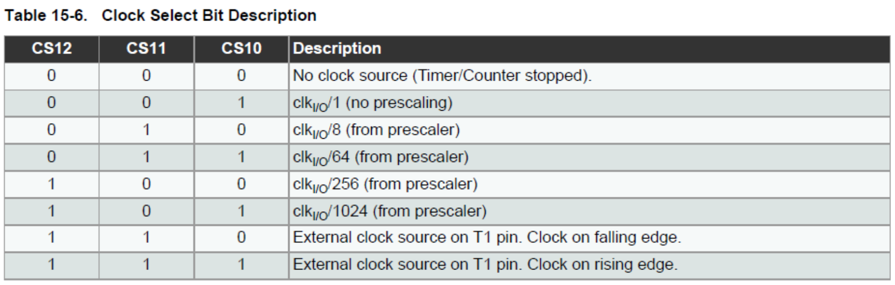


Table 15-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Figure 2 Clock Select Bit description

From the prescaler value you obtained above, now you should what bits to set in the timer control register **TCCR1B**.

You can use bit manipulation using bit shift operators with bit mask like this:

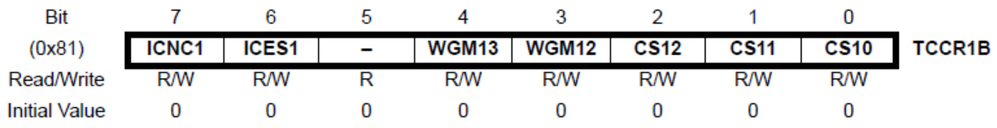
```
TCCR1B |= 1<<CS10; // this will set the CS10 bit to 1

TCCR1B |= 1<<CS11; // this will set the CS11 bit to 1

TCCR1B |= 1<<CS11; // this will set the CS12 bit to 1
```

Or:

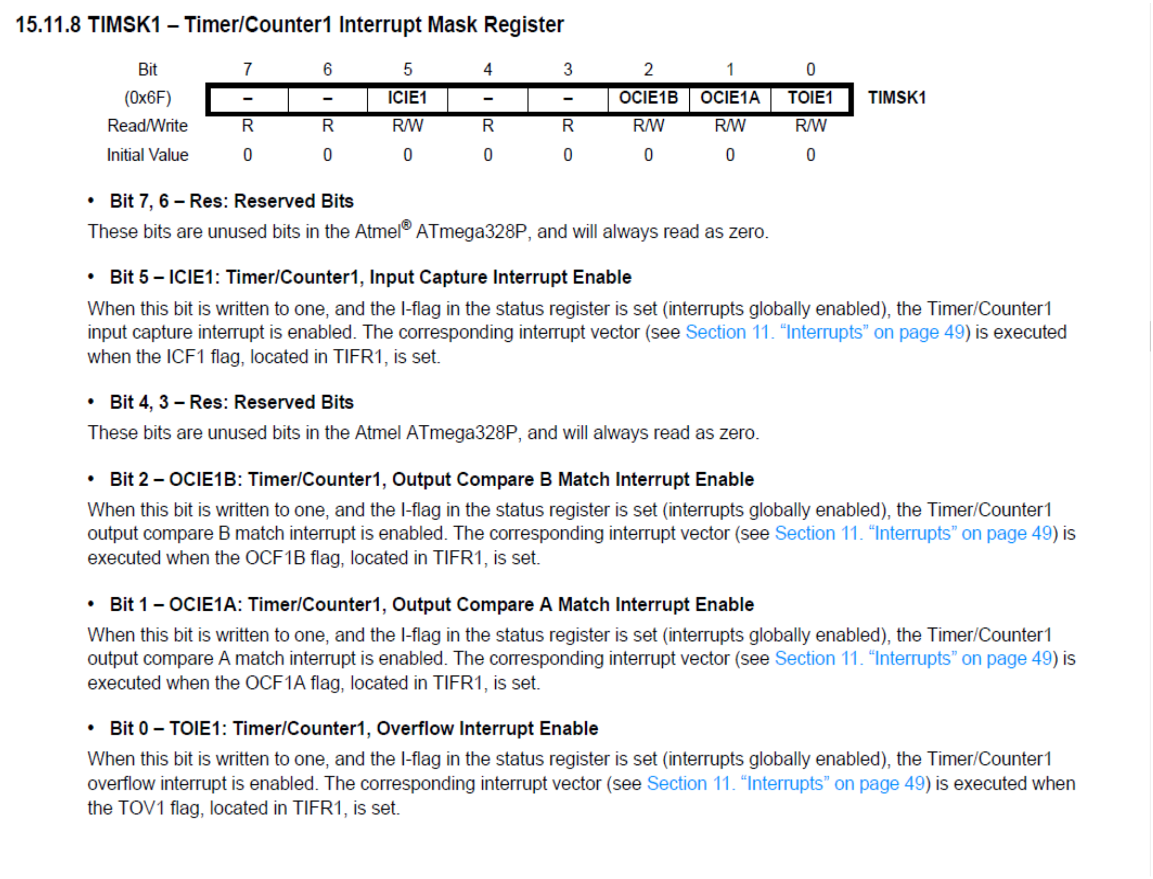
```
TCCR1B |= BIT0; // set bit 0
```



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 3 Timer1 B Control Register

You should enable the overflow interrupt in the timer interrupt mask register **TIMSK1**.



15.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

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	

- **Bit 7, 6 – Res: Reserved Bits**

These bits are unused bits in the Atmel® ATmega328P, and will always read as zero.

- **Bit 5 – ICIE1: Timer/Counter1, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 input capture interrupt is enabled. The corresponding interrupt vector (see Section 11. "Interrupts" on page 49) is executed when the ICF1 flag, located in TIFR1, is set.

- **Bit 4, 3 – Res: Reserved Bits**

These bits are unused bits in the Atmel ATmega328P, and will always read as zero.

- **Bit 2 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 output compare B match interrupt is enabled. The corresponding interrupt vector (see Section 11. "Interrupts" on page 49) is executed when the OCF1B flag, located in TIFR1, is set.

- **Bit 1 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 output compare A match interrupt is enabled. The corresponding interrupt vector (see Section 11. "Interrupts" on page 49) is executed when the OCF1A flag, located in TIFR1, is set.

- **Bit 0 – TOIE1: Timer/Counter1, Overflow Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 overflow interrupt is enabled. The corresponding interrupt vector (see Section 11. "Interrupts" on page 49) is executed when the TOV1 flag, located in TIFR1, is set.

Figure 4 Timer1 Overflow interrupt

Code:

```
TIMSK1 |= 1<<TOIE1; // enable overflow interrupt
```

Now you should define what is in your interrupt service routine. What the interrupt should do after it occur. You can either set the LED on or off inside this routine. Or you can set a flag to 1 or 0 and then use this flag to turn the LED.

```
ISR(TIMER1_OVF_vect){ // interrupt service routine for timer1 overflow
     /* Replace with your interrupt service routine */
}
```

Part 2: Pulse width Modulation PWM

The PWM is one of application of the timers. It generates a square pulse with programmable pulse width (duty cycle) and frequency. It is widely used in controlling speed of motors, dimming of light, clock generation and many other different applications. The PWM use the compare functionality of the timer. If the timer reaches certain value, the compare register generates an interrupt.

We will use the PWM to dim the LED.

Now you should select a high frequency for your timer (prescaler = 1). The interrupt now is generated from the compare registers not the overflow flags. See Figure 4 above.

Use Timer 1: 16-bit timer

```
Set OCIE1B & OCIE1A on the TIMSK1 register
```

Set the value of compare registers:

```
OCR1A & OCR1B
```

Their values will determine the duty cycle.

Define the ISR when the compared value is reached for each compare register. One should switch on the LED and the other switch it off.

```
ISR(TIMER1_COMPA_vect)
```

```
ISR(TIMER1_COMPA_vect)
```

Additional work:

Try to generate a variable dimming LED while the code is running.