AVR 8/16 Bit Timers/Counters – Tutorial #11

T.K. HAREENDRAN

AVR tutorial

Share this:

A timer is a simple counter! The input clock of microcontroller and operation of the timer is independent of the program execution.All the Atmel microcontrollers have Timers as an inbuilt peripheral. In this tutorial our focus is on ATmega8 microcontroller. ATmega8 comes with two 8-bit and one 16-bit timer. This means that there are 3 sets of timers, each with the ability to count at different rates.

The two 8-bit counters can count to 255 whilst the 16- bit counter can count to 65,536. We learned that the simplest timer in Atmeag8 is TIMER0 with an 8-bit resolution (0-255). Timers can run asynchronous to the main AVR core hence timers are totally independent of CPU. A timer is usually specified by the maximum value to which it can count, beyond which it overflows and resets to zero.

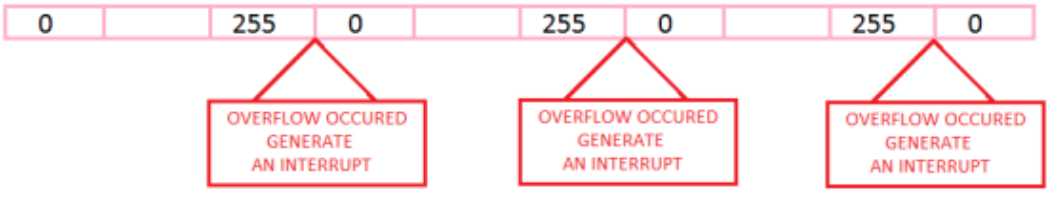
Speed of the counting can be controlled by varying the speed of clock input to it. As we know, all the componets of a microcontroller unit (MCU) are someway connected to the central processing unit (CPU) and some registers, to share information beween them. Next, I am discussing about few of the Registers in TIMER0 of ATmega8 in detail. This is infact a sequel of the previous part!

- **TCCR0:** In this register (used to configure the timer), there are 8 bits, but only last 3 bits CS02,CS01 and CS00 are used. These are CLOCK SELECT bits used to setup the prescaler
- **TCNT0:** This is the real counter in the TIMER0 counter. The timer clock counts this register as 1, ie the timer clock increases the value of this 8 bit register by 1 with every timer clock pulse. The timer clock can be defined by the TCCRO register
- **TIMSK0:** This register, used to activate/deactivate the INTERRUPTS related to timers,controls the interrupts of all three timers. BIT0 (first bit from the right) controls the the overflow interrupts of TIMER0. Note that TIMER0 has one interrupt, and the rest of the bits are for other counters

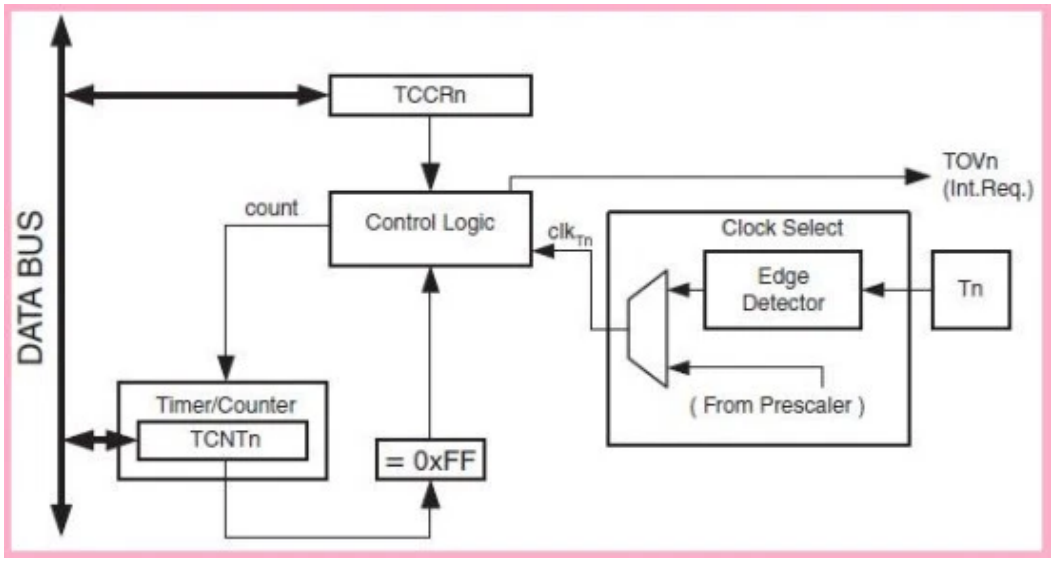
In short:
TCCR0 – Timer/Counter Control Register
TCNT0 – Timer/Counter Register
TIMSK0 – Timer/Counter Interrupt Mask Register

BIT	7	6	5	4	3	2	1	0
						CS02	CS01	CS00
READ/WRITE	R	R	R	R	R	R	R	R
INITIAL VALUE	0	0	0	0	0	0	0	0

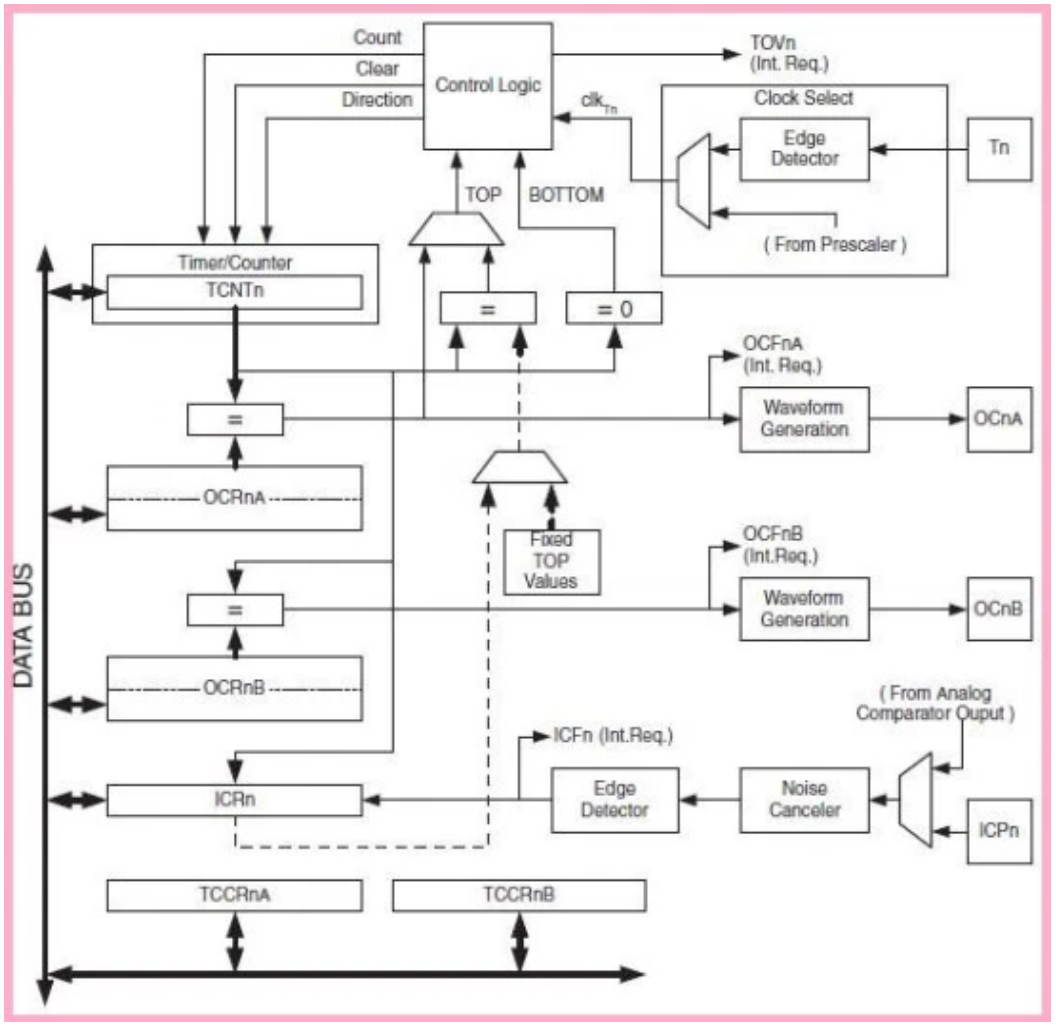
Hope you can recall the answer of this question; What is an ISR? ISR (Interrupt Service Routine) is a function that the CPU should execute whenever an interrupt occurs.



Interrupt Generation



Timer0 on the ATmega8



Timer1 on the ATmega8

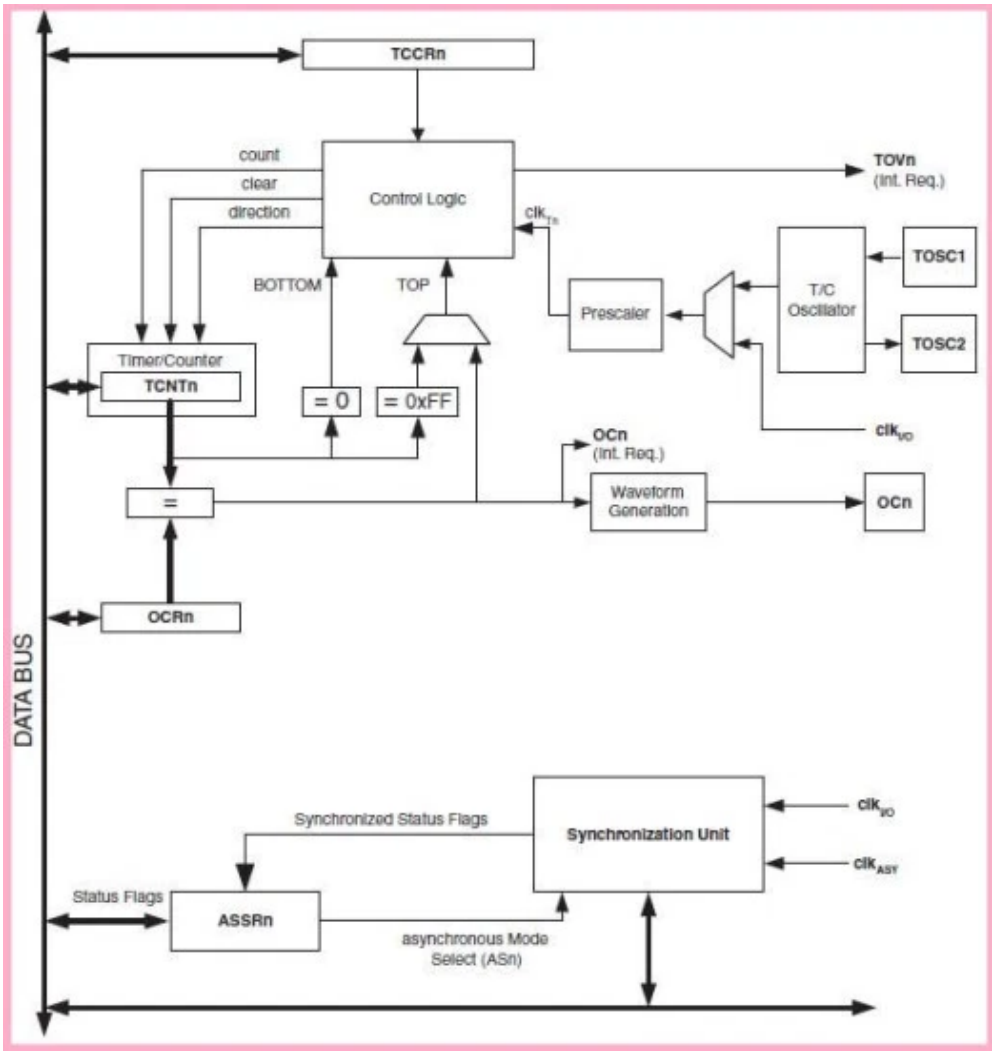
TCNT1	16-bit counter register
TCCR1A	Mode of operation and other settings
TCCR1B	Mode of operation, prescaler and other settings
OCR1A	16-bit Compare Register A
OCR1B	16 bit Compare Register B
TIMSK	Interrupt Mask Register
TIFR0	Timer/Counter Interrupt Flag Register

Timer1 Registers

Timer Modes

Timers are usually used in Normal Mode and CTC mode. In Normal Mode,when the prescaler receives a pulse from a clock cycle and passes it onto the Control Logic, the Control Logic increments the TCNTn register by 1. When TCNTn hits the TOP (0xFF in the 8 bit timers and 0xFFFF in the 16 bit timer) it overflows to 0 and sets the TOVn bit in the TIFR register. The problem with this Normal Mode is that it is very hard to use for an exact interval. CTC stands for “Clear Timer on Compare”. In CTC Mode, when the prescaler receives a pulse from the clock cycle and passes it onto the Control Logic, the Control Logic increments the TCTn register by 1.

The TCNTn register is compared to the OCRn register, when a compare match occurs the TOVn bit is set in the TIFR register. It’s interesting that Timer/Counter1 can run in Normal mode (0xFFFF) and in two CTC modes! The difference between the two CTC modes is that mode 4 uses the OCR1A register for its compare value, and mode 12 uses the ICR1 register. In normal mode TOV1 can generate a Overflow interrupt, in CTC (mode 4) mode OCIF1A can generate an interrupt when it detects a compare match, and in CTC (mode 12) mode TICIE1 can generate an interrupt when it detects a compare match. The 3rd, Timer/Counter2 (8-Bit) is the preferred one for short time delays. It can run in Normal mode or CTC mode!



Timer2 on the ATmega8

TCCR2A	Timer/Counter Control Register A
TCCR2B	Timer/Counter Control Register B
TCNT2	Timer/Counter Register
OCR2A	Output Compare Register A
OCR2B	Output Compare Register B
TIMSK2	Timer/Counter Interrupt Mask Register
TIFR2	Timer/Counter Interrupt Flag Register

Timer2 Registers

TIMER 1 in Output Compare Mode

Timer0 is fairly neutered counter, and because of some limitations Timer/Counter0 is rarely used as a timer. In TIMER0 Overflow mode CPU executes ISR when TCNT0 register overflows,but in some applications we need a very specific time period, or occurance of an event before the register overflows. This time we want the CPU to execute ISR when TCNT0 register reaches a particulr value. TIMER1 in Output Compare Mode is the practical solution for this task. TIMER1 is a 16-bit counter with seperate Prescaler, Compare Mode, and Capture Mode.Here are few of the registers in Output Compare Mode:

OCR1A – Output Compare Register A
OCR1B – Output Compare Register B
TIMSK – Timer/Counter Interrupt Mask Register

TIFR – Timer/Counter Interrupt Flag Register
TCCR1A – Timer/Counter1 control Register A
TCCR1B – Timer/Counter1 control Register B

Yes, we have seen various features (and modes) of 8/16 Bit Timers/Counters. Handling all these require plentiful knowledge of datasheet, so always remember to keep handy and refer the Atmega8 datasheet.

Code Examples

Here is a simple code which sets up timer0 for 1ms at 16Mhz clock cycle

```
#include <avr/io.h>
int main(void)
{
    while (1)
    {
        // start the timer
        TCCR0 |= (1 << CS01) | (1 << CS00);
        // set prescaler to 64 and start the timer
        while ( (TIFR & (1 << TOV0) ) > 0) // wait for the overflow event
        {
        }
        TIFR &= ~(1 << TOV0);
        // reset the overflow flag
    }
}
```

Here is the simple code to set up timer1 for 1ms at 16Mhz clock cycle. An interrupt is triggered each time the interval occurs.

```
#include <avr/io.h>
#include <avr/interrupt.h>
int main(void)
{
    TIMSK |= (1 << TOIE0);
    sei();
    //enable interrupts
    TCCR0 |= (1 << CS01) | (1 << CS00);
    // set prescaler to 64 and start the timer
    while (1)
    {
        //main loop
    }
}
ISR (TIMER0_OVF_vect) // timer0 overflow interrupt
{
    TCNT0 += 6;
    // add 6 to the register (our work around)
}
```

Timer2 Code for 250uS at 16Mhz clock cycle

```
#include <avr/io.h>
#include <avr/interrupt.h>
int main(void)
{
    OCR2 = 62;
    TCCR2 |= (1 << WGM21);
    // Set to CTC Mode
    TIMSK |= (1 << OCIE2);
    //Set interrupt on compare match
    TCCR2 |= (1 << CS21);
    // set prescaler to 64 and starts PWM
    sei();
    // enable interrupts
    while (1);
    {
    // we have a working timer
    }
}

ISR (TIMER2_COMP_vect)
{
    // action to be done every 250us
}
```

→ Part 12: **AVR Pulse Width Modulation**
← Part 10: **ATmega8 Advanced Guide: 8-bit Timers**