Timers on the ATmega8

INTRODUCTION:

The timers are the heart of automation. We have seen in previous chapters how we could take in an input, perform mathematical functions on the data and, perform an action. Timers give us an extra level of control by giving us not only control over what happens but when it happens. We can put a time delay on a self destruct in our evil lair, we can control the flash rate of our seizure causing robot and, we could ensure that our death-ray isn't fired before its fully charged. We have control of time MUA HAHAHAHAHA !!!!

THEORY OF OPERATION:

A quick review of the previous tutorial (COMMON TIMER/COUNTER THEORY). The clock source (from the internal clock or an external source) sends pulses to the prescaler which divides the pulses by a determined amount. This input is sent to the control circuit which increments the TCNTn register. When the register hits its TOP value it resets to 0 and sends a TOVn (timer overflow) signal which could be used to trigger an interrupt.

Unfortunately, the AVR timer does process time in hours, minutes or seconds (what we are use to). However, being evil masterminds we have the solution, it just requires a bit of math and our old friend, the prescaler.

OCRn = [(clock_speed / Prescaler_value) * Desired_time_in_Seconds] - 1

Now for the bad news OCRn has to be a whole number. If you end up with a decimal number it means that your desired timer will not be exact. The other thing is that your number has to be able to fit into the register. So 255 for an 8bit timers and 65535 for the 16bit timer.

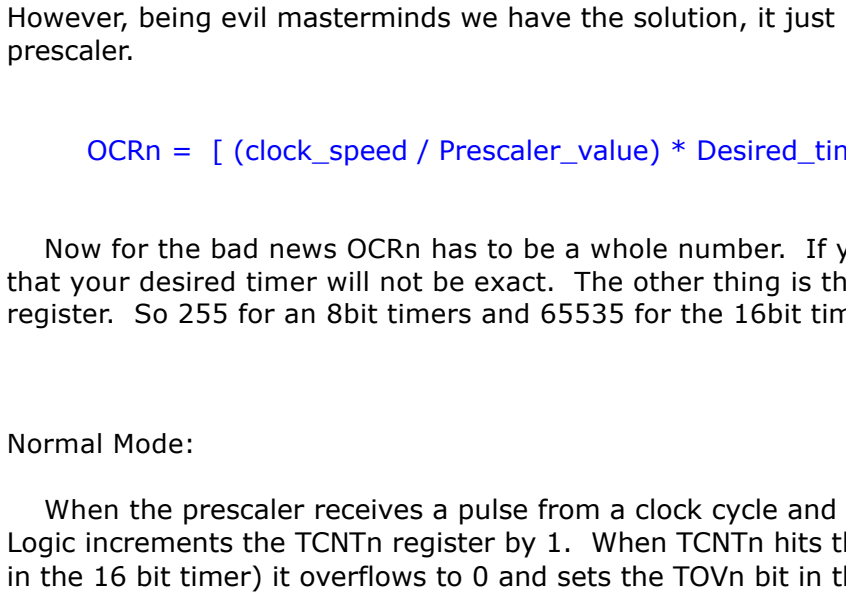
Normal Mode:

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 Normal Mode is that it is very hard to use for an exact interval, because of this Normal Mode is only useful if you need a none-specific time interval (say you don't care if it happens every 1 or 2 ms as long as it happens at the same time each time) its nice and easy option. Because, of this limitation many programmers choose to use CTC mode for their timers.

CTC Mode:

CTC stands for "Clear Timer on Compare" and it does the following. When the prescaler receives a pulse from a 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.



Flow 1: Activating The Timer

The flow diagram above describes how to activate the timer. More details below.

TIMER0 (8BIT PWM):

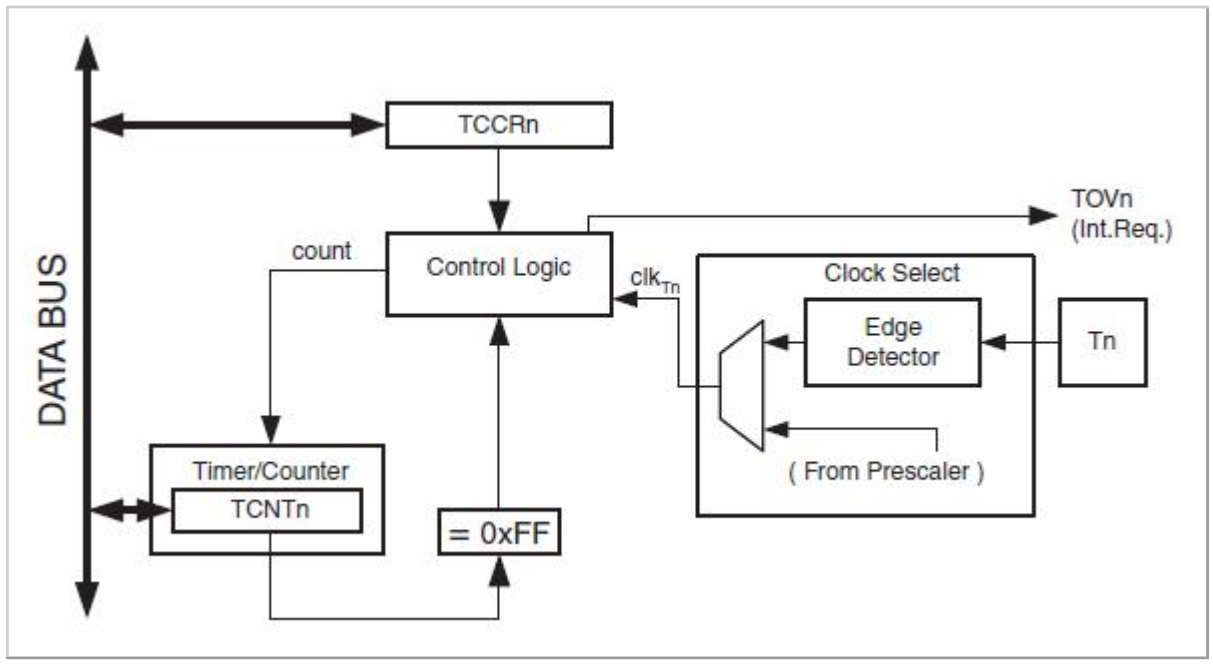


Figure 1: Timer0 on the ATmega8

Timer0 is fairly neutered counter, it does not have a OCR0 register therefore it is only capable of running in normal mode. Because of this limitations Timer/Counter0 is rarely used as a timer (it is also why you rarely see tutorials on Timer0 on the internet).

TOV0 can generate a Timer Overflow interrupt. In order to activate the timer0 interrupts you need to SET(1) the TOIE0 bit within the TIMSK register.

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCCR0	FOC0	-	-	-	-	CS02	CS01	CS00

Timer/Counter Control Register 0

CS02	CS01	CS00	DESCRIPTION
0	0	0	Timer/Counter0 Disabled
0	0	1	No Prescaling
0	1	0	Clock / 8
0	1	1	Clock / 64
1	0	0	Clock / 256
1	0	1	Clock / 1024
1	1	0	External clock source on T0 pin, Clock on Falling edge
1	1	1	External clock source on T0 pin, Clock on rising edge

CS bits

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0

Timer/Counter Interrupt Mask Register

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	-	TOV0

Timer/Counter Interrupt Flag Register

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCNT0								

Timer/Counter Register (stores the counter value)

Software:

ATmega8 Code:

```
// this code sets up a timer0 for 1ms @ 16Mhz clock cycle
// in order to function as a time delay at the begining of the main loop
// NOTE:    1ms = 0xF9 but we are going 0xFF, so we are actually creating a
//          1.024ms timer.


#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

    }
}
```

ATmega8 Code:

```
// this code sets up a timer0 for 4ms @ 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 << CS02);
    // set prescaler to 256 and start the timer


    while (1)
    {
        //main loop
    }
}


ISR (TIMER0_OVF_vect)  // timer0 overflow interrupt
{
    // event to be exicuted every 4ms here
}
```

Normal Mode to CTC Mode Work Around:

There is an interesting work around for timer/counter0 limitations. Say we want to counter to count to 1ms, in the first example we just lived with the fact that our 1ms timer was actually a 1.024ms timer. In order to get a true 1ms we need to count to 0xF9, which is only 6 lower then the max. So what we need to do is add 6 to the register when it overflows to 0. Of course this isn't needed if we have a timer with a CTC mode.

ATmega8 Code:

```
// this code sets up a timer1 for a TRUE 1ms @ 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)
}
```

TIMER1 (16BIT PWM):

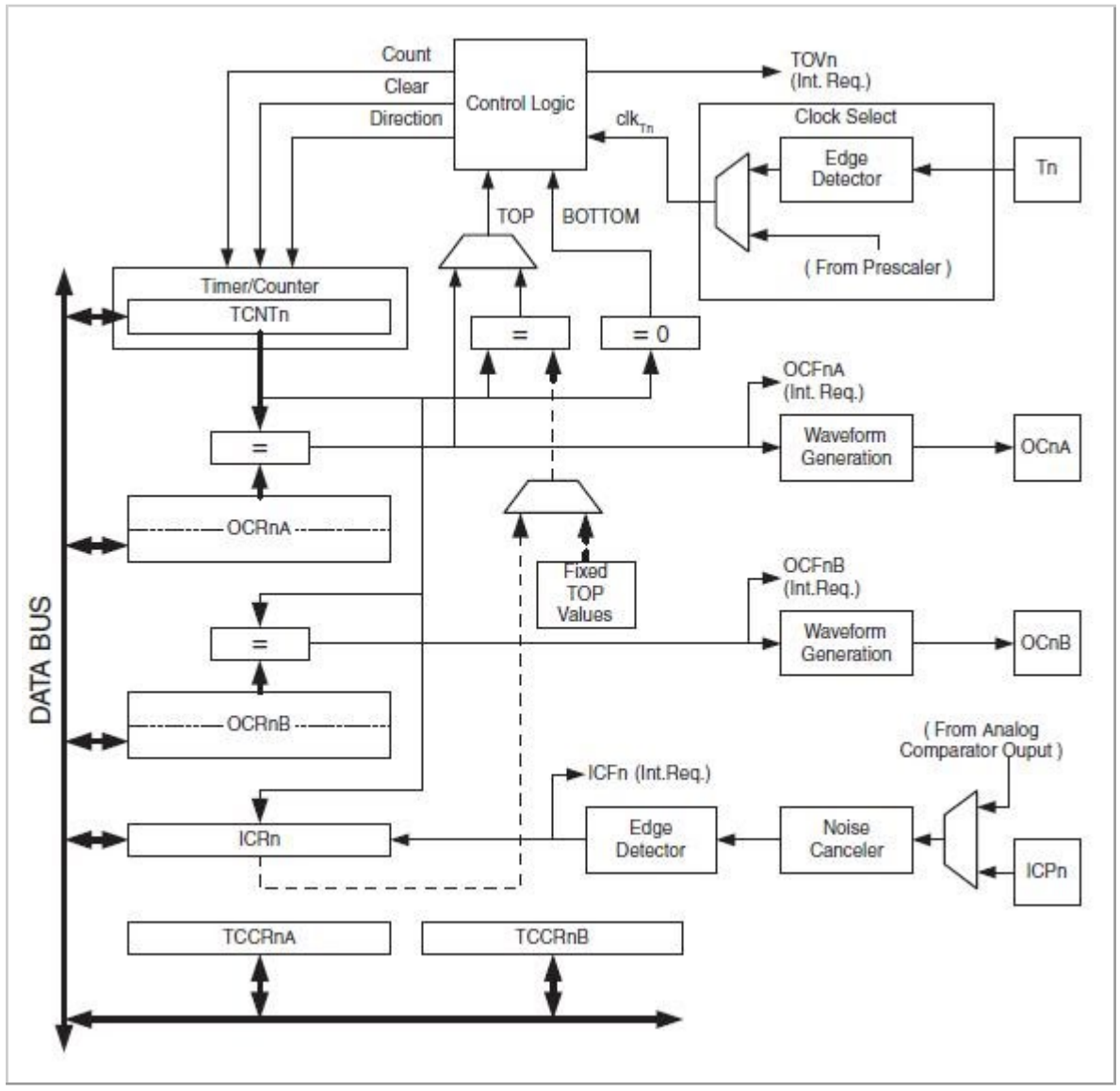


Figure 2: Timer1 on the ATmega8

Timer/Counter1 is the big daddy of timers. It can run in Normal mode (0xFFFF) and 2 CTC modes. The difference between the 2 CTC modes 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 order to activate the timer1 overflow interrupts you need to SET(1) the TOIE1 bit within the TIMSK register.

In CTC (mode 4) mode OCIF1A can generate an interrupt when it detects a compare match. In order to activate the timer1 CTC interrupt SET(1) the OCF1A bit within the TIMSK register.

In CTC (mode 12) mode TICIE1 can generate an interrupt when it detects a compare match. In order to activate the timer1 CTC interrupt SET(1) the TICIE1 bit within the TIMSK register.

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

Timer/Counter Control Register 1 A

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCCR1B	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

Timer/Counter Control Register 1 B

CS12	CS11	CS10	DESCRIPTION
0	0	0	Timer/Counter1 Disabled
0	0	1	No Prescaling
0	1	0	Clock / 8
0	1	1	Clock / 64
1	0	0	Clock / 256
1	0	1	Clock / 1024
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

CS bits

MODE	WGM13	WGM12	WGM11	WGM10	DESCRIPTION	TOP
0	0	0	0	0	Normal	0xFFFF
1	0	0	0	1	PWM, Phase Corrected, 8bit	0x00FF
2	0	0	1	0	PWM, Phase Corrected, 9bit	0x01FF
3	0	0	1	1	PWM, Phase Corrected, 10bit	0x03FF
4	0	1	0	0	CTC	OCR1A

5	0	1	0	1	Fast PWM, 8bit	0x00FF
6	0	1	1	0	Fast PWM, 9bit	0x01FF
7	0	1	1	1	Fast PWM, 10bit	0x03FF
8	1	0	0	0	PWM, Phase and Frequency Corrected	ICR1
9	1	0	0	1	PWM, Phase and Frequency Corrected	OCR1A
10	1	0	1	0	PWM, Phase Correct	ICR1
11	1	0	1	1	PWM, Phase Correct	OCR1A
12	1	1	0	0	CTC	ICR1
13	1	1	0	1	RESERVED	
14	1	1	1	0	Fast PWM	ICR1
15	1	1	1	1	Fast PWM	OCR1A

Waveform Generator Mode bits

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0

Timer/Counter Interrupt Mask Register

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	-	TOV0

Timer/Counter Interrupt Flag Register

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCNT1H								
TCNT1L								

Timer/Counter Register (stores the counter value, 16 bit)

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
OCR1AH								
OCR1AL								

Output Compare Register A (stores the compare value, 16 bit)

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
ICR1								
ICR1								

Input Compare Register (can be used to stores the compare value, 16 bit)

Software:

ATmega8 Code:

```
// this code sets up timer1 for a 1s  @ 16Mhz Clock (mode 4)


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


int main(void)
{
    OCR1A = 15624;

    TCCR1B |= (1 << WGM12);
    // Mode 4, CTC on OCR1A

    TIMSK |= (1 << OCIE1A);
    //Set interrupt on compare match

    TCCR1B |= (1 << CS12) | (1 << CS10);
    // set prescaler to 1024 and start the timer


    sei();
    // enable interrupts


    while (1);
    {
        // we have a working Timer
    }
}

ISR (TIMER1_COMPA_vect)
{
    // action to be done every 1 sec
}
```

ATmega8 Code:

```
// this code sets up timer1 for a 200ms  @ 16Mhz Clock (Mode 12)


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

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


int main(void)
{
    ICR1 = 12499;

    TCCR1B |= (1 << WGM13)| (1 << WGM12);
    // Mode 12, CTC on ICR1

    TIMSK |= (1 << TICIE1);
    //Set interrupt on compare match

    TCCR1B |= (1 << CS12);
    // set prescaler to 256 and starts the timer


    sei();
    // enable interrupts


    while (1)
    {
        // we have a working Timer
    }
}

ISR (TIMER1_COMPA_vect)
{
    // action to be done every 200ms
}
```

TIMER2 (8BIT PWM):

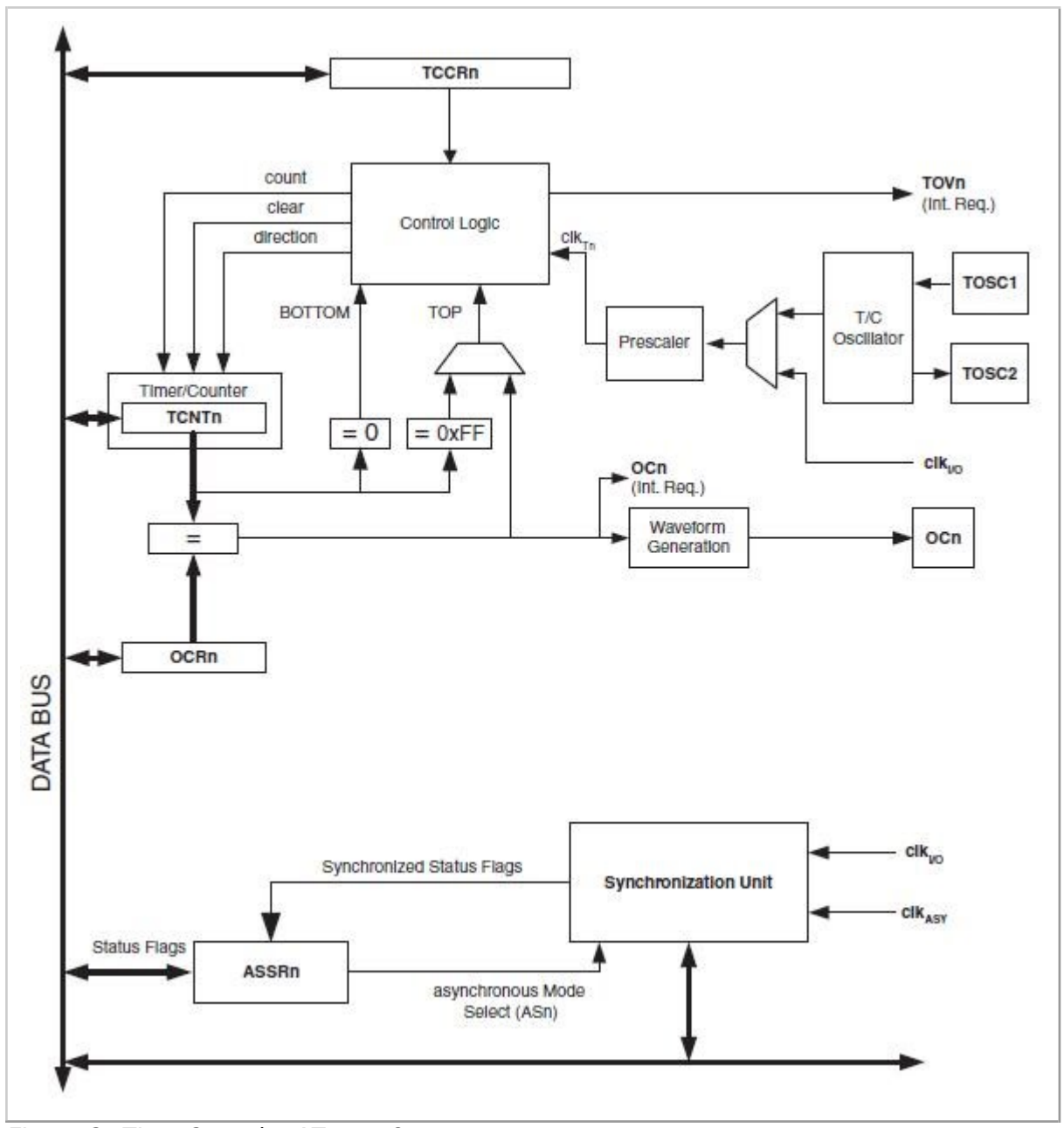


Figure 3: Timer2 on the ATmega8

Timer/Counter2 is the preferred timer among programmers for short time delays. It can run in Normal mode (0xFF) or CTC modes.

In normal mode TOV2 can generate a Overflow interrupt. In order to activate the timer1 overflow interrupts you need to SET(1) the TOIE2 bit within the TIMSK register.

In CTC mode OCIF2 can generate an interrupt when it detects a compare match. In order to activate the timer1 CTC interrupt SET(1) the OCF2 bit within the TIMSK register.

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit

TCCR2 FOC2 WGM21 COM21 COM20 WGM20 CS22 CS21 CS20
Timer/Counter Control Register 2

CS22	CS21	CS20	DESCRIPTION
0	0	0	Timer/Counter2 Disabled
0	0	1	No Prescaling
0	1	0	Clock / 8
0	1	1	Clock / 32
1	0	0	Clock / 64
1	0	1	Clock / 128
1	1	0	Clock / 256
1	1	1	Clock / 1024

CS bits

MODE	WGM21	WGM20	DESCRIPTION	TOP
0	0	0	Normal	0xFF
1	0	1	PWM Phase Corrected	
2	1	0	CTC	OCR2
3	1	1	Fast PWM	

Waveform Generator Mode bits

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0

Timer/Counter Interrupt Mask Register

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	-	TOV0

Timer/Counter Interrupt Flag Register

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCNT2								

Timer/Counter Register (stores the counter value)

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
OCR2								

Output Compare Register (stores the compare value)

Software:

```
// this code sets up timer2 for a 250us  @ 16Mhz Clock


#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
}
```

Can't type playing Psyconauts.

Cheers.
Q

Comments? Questions? Problems with content? q.qeewiki@gmail.com

Comments