Counter/Timer in Embedded Systems

Baris Aksanli
Department of Electrical and Computer Engineering
San Diego State University

Outline

- Embedded systems and the need for timing/counters
- Timer/counter in AVR
  - Modes
  - Pre-scaling
  - Time calculation

Embedded Systems

- Part of our every day life
- For example:
  - In a car, more than 60-70 microprocessors, e.g. lights, mirrors, airbags, etc.
  - In a house, more than 40-50 microprocessors, e.g. appliances, smart devices, PCs, etc.
- What do we see as an embedded system?
  - Hardware (chips) + Software (programs, OS)
  - CPU processor (ARM, PowerPC, Xscale/SA, 68K)
  - Memory (256MB or more)
  - Input/output interfaces (parallel and serial ports)

Timing Constraints and Properties

- Predicting and controlling timing and events
- Timing relationship: (can you guarantee it?)
  - Predictable actions in response to external stimuli
  - Deadline (absolute or relative) and jitter
- Difficult to control timing
  - Sequence, order, and race condition

Hard Real-time

- Responses to events for aperiodic systems or actions taken when a periodic task begins (time-driven systems) must complete by a specified deadline
- A timing constraint or deadline is “hard” if the failure to meet is considered to be a fatal fault
- The operation of the system is incorrect if the results are not produced according to the timing specifications

Soft Real-time

- Event responses shall be handled on average within a certain time
- Late completion of a scheduled event is undesirable. The operation is degraded if the results are not produced timely but the result is not fatal
- A certain number of event responses shall be handled within a specified timeframe
- A specified failure rate is permitted

Timers: Why we need them?

- Provide accurately timed delays or actions independent of code execution time
- How are Timers used?
  - Accurate delay
    - Read the timer, store value as K. Loop until timer reaches K+100.
  - Schedule important events
    - Setup an *Output Compare* to trigger an interrupt at a precise time
  - Measure time between events
    - When event#1 happens, store timer value as K
    - When event#2 happens, read timer value and subtract K
    - The difference is the time elapsed between the two events

Generic Timer/Counter

- Control register sets Modes
- Counter counts input pulses
- Timer counts clock cycles
- Status register
- Timer / Counter Modes
  - One shot
  - Auto reload
  - Enhanced modes:
    - PWM: Pulse Width Modulation
    - Capture/compare
- Generate Time/Count interrupts

Timer vs. Counter

- Timer: when the incoming clock frequency is known
  - Get clock from Oscillator frequency, then by setting the preload value. We can generate a fixed period of time known to the designers.

- Counter: when the incoming clock is ‘irregular’
  - Get clock from external pin, and we are only interested in the number of occurrence of this pulse. This is called counter (counting events)

Example in AVR

- First, how does clock work?
- Clock control module generates clocks for memory and IO devices
- Multiple internal clock sources
- Provisions for external crystal clock source (max 20 MHz)

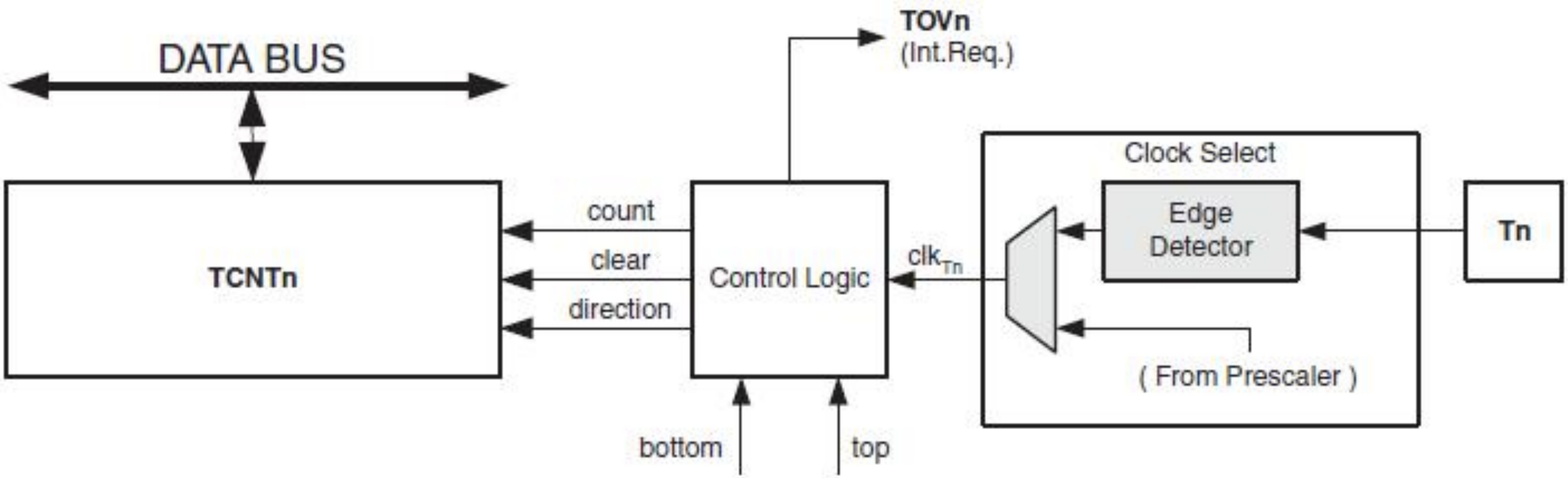
Timer / Counters

- 8/16 Bit register
  - Increments or decrements on every clock cycle
  - Can be read on data bus
  - Output feeds waveform generator
  - 2 8-bit timer/counters (0 and 2) and 1 16-bit timer/counter (1)
- Clock Sources
  - Internal from clock pre-scaler
  - External Tn Pin (Uses 1 port pin)

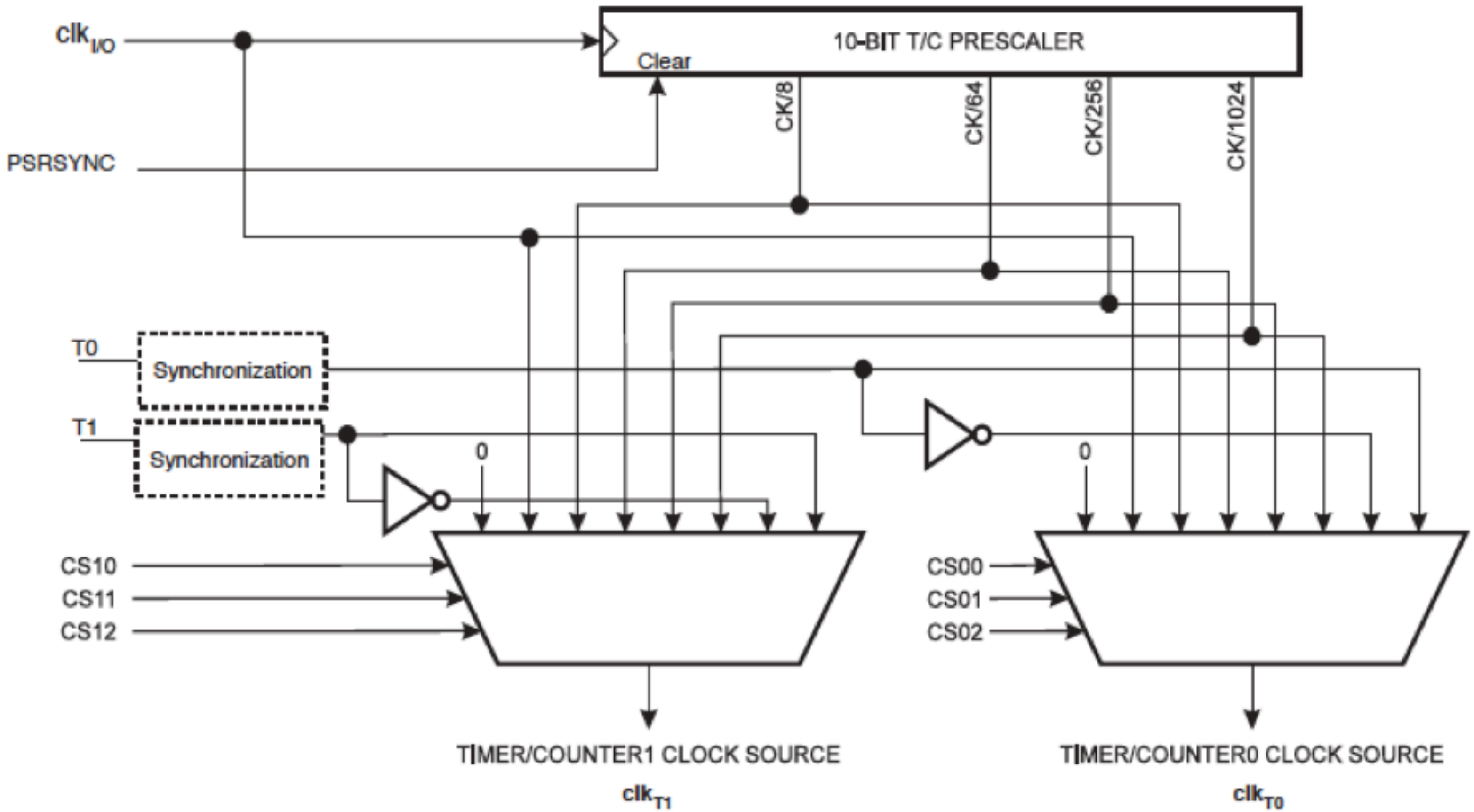
AVR 8-bit Timer 0

- 8 Bit Up Timer
  - Counts from 0 to 255 (0xFF), then loops to 0
  - Internal or external clock source
    - Pre-scaler
- Interrupt on Overflow
  - Transition from 255 to 0 can trigger interrupt if desired

Timer 0 on AVR 328

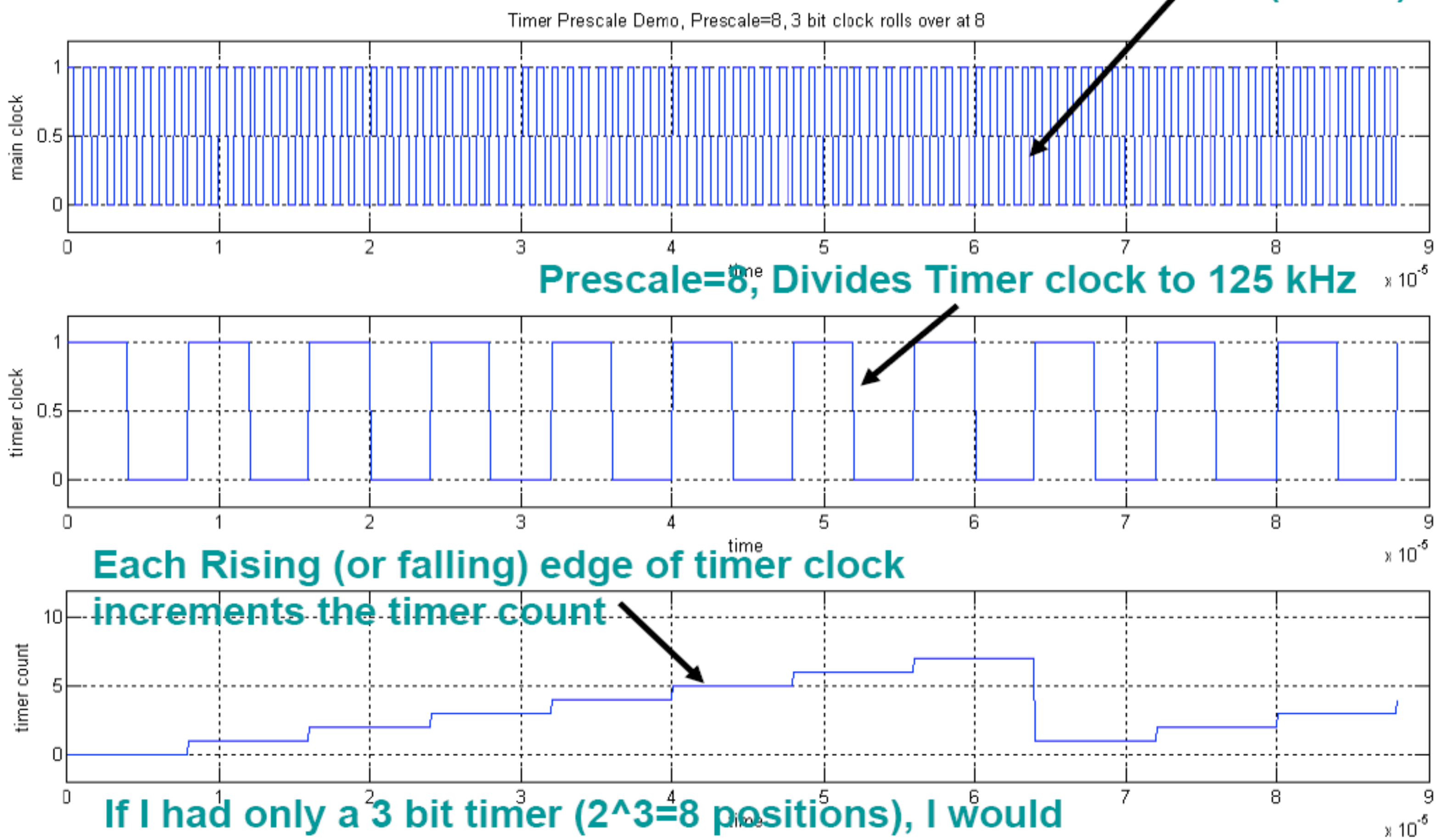


Pre-scaler for Timer 0 and 1



Pre-scale for a Timer

Main Clock (1 MHz)



Prescale=8, Divides Timer clock to 125 kHz

Each Rising (or falling) edge of timer clock increments the timer count

If I had only a 3 bit timer (2^3=8 positions), I would overflow at 7 (starts at 0) and wrap back to zero

Why Pre-scale?

- It allows setting how fast the timer counts and thus the resolution of time that can be measured
  - For the fastest count, set pre-scale to 1 (default). The timer will count at the same rate as main clock and have a resolution equal to the period of the main clock.
  - What if we were using an 8-bit timer (max count = 255). The counter would quickly fill up and overflow. We can:
    - Increase the pre-scale so that it counts slower
    - Switch over to a 16-bit timer (max = 65535)
- Note: Timers actually hold a count, but knowing the pre-scale and main clock, we know what the count means in time
- Actual time = (Change in Count)*(Timer Period)
  - Where Timer Period = Main Clock Period*Pre-scale value

Clock Source Select

- Using bits CS0[2:0] (in TCCR0B) and CS1[2:0] (in TCCR0B)
  - CPU frequency divided by 1,8,64,256,1024
    - At 16MHz that's: 1/16us, 1/2us, 4us, 16us, 64us

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(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 T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

- Where can you find T0/T1 pin:
  - T0 pin: Port D[4]
  - T1 pin: Port D[5]

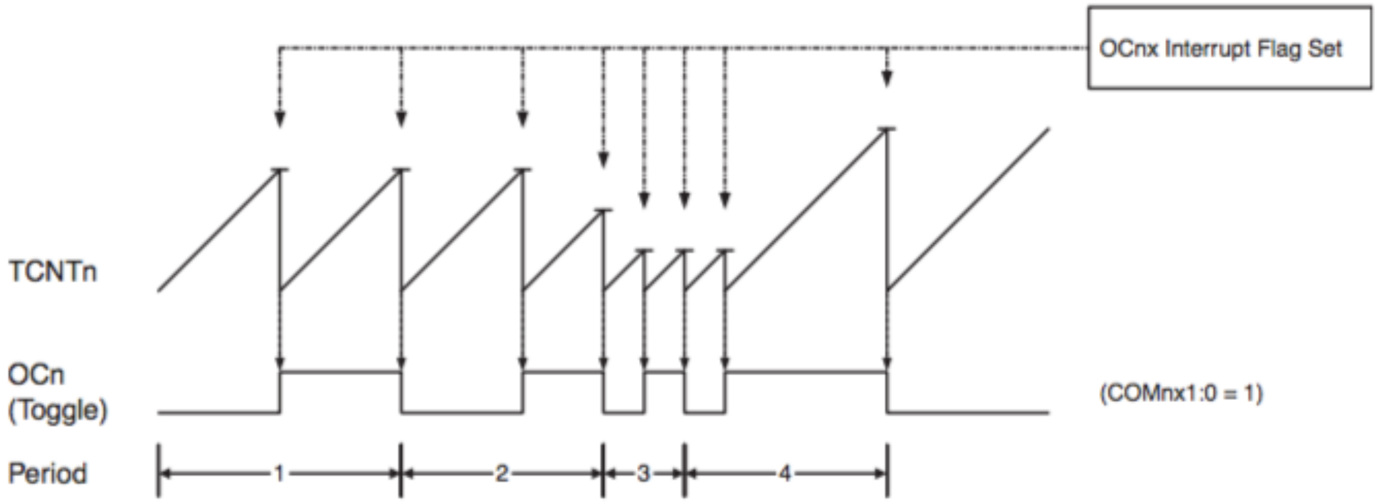
Normal Mode Operation

- Timer increments
- Wraps around at TOP = 0xFF (=255)
- Starts again at 0
- TOVn interrupt flag is set when TCNTn is reset to 0

- Useful for generating interrupts every N time units
- Useful for generating an interrupt in M time units
  - Setting the initial value of TCNTn to (255-M) (for 8-bit)

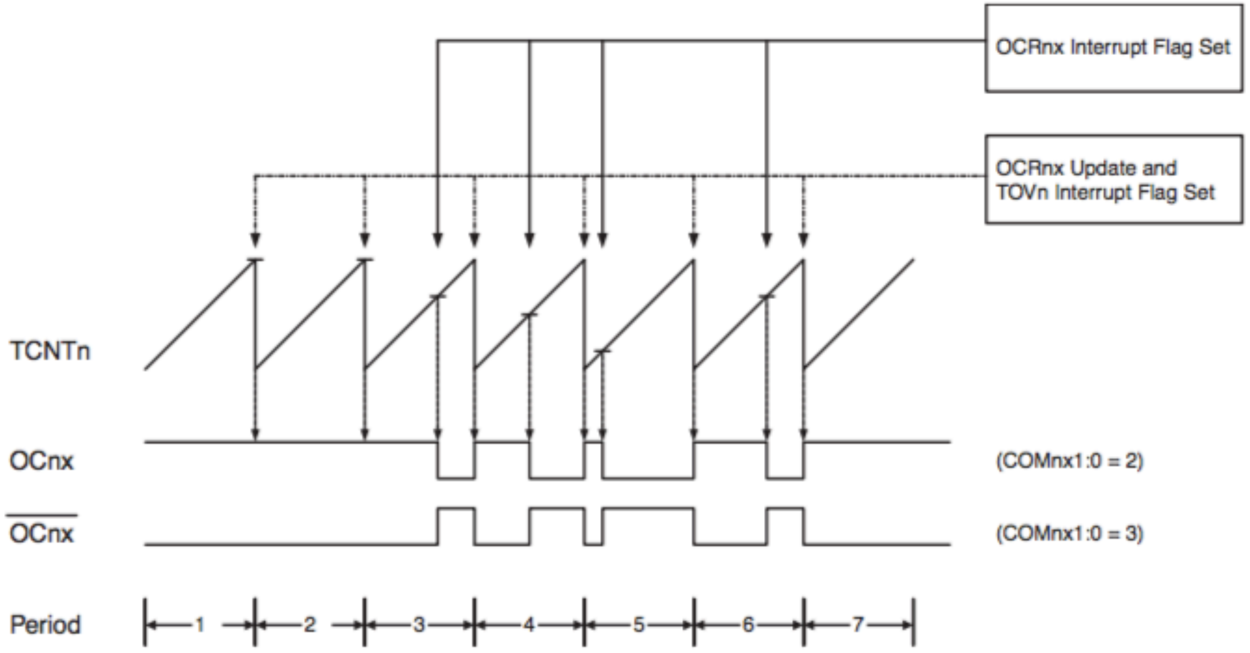
Clear Timer on Compare (CTC Mode)

- Timer increments
- Wraps around at OCRnA (n=0, 1,2 for Timer0, Timer1, Timer2)
  - OCRnA defines top value of counter
- Starts again at 0
- OCFnA interrupt flag set when TCNTn reset to 0
- Pin OCnA can be made to toggle when counter resets
  - Generate output waveform



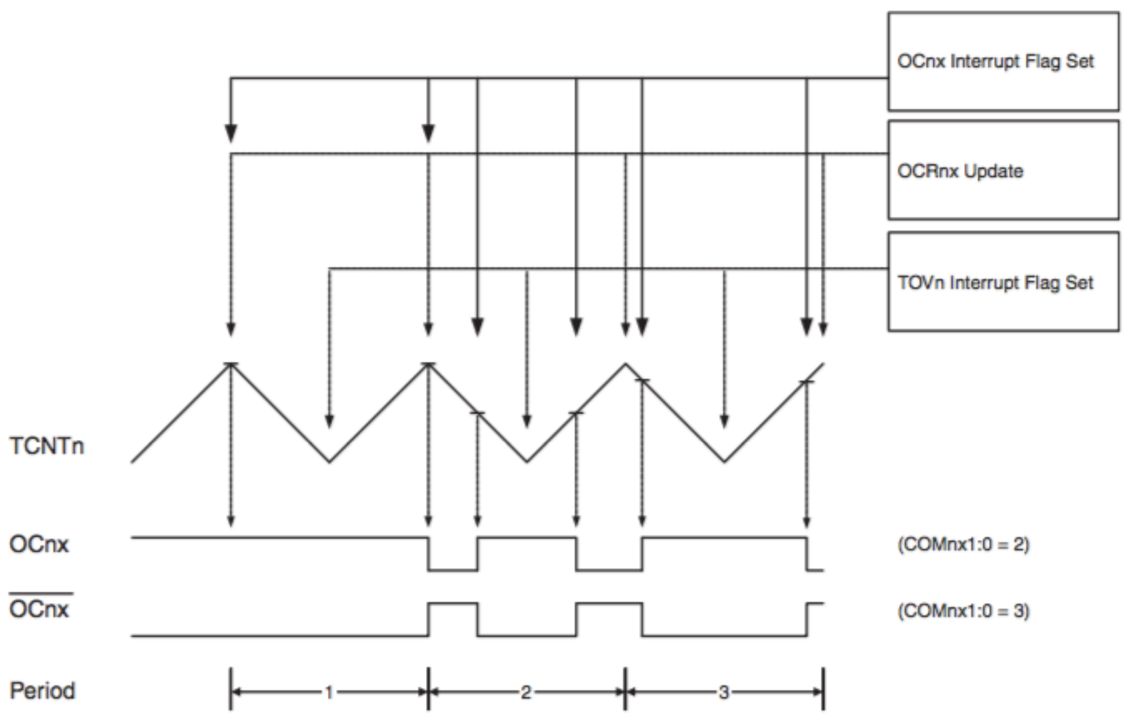
Fast PWM (Phase Width Modulation) Mode

- Timer increments
- Wraps around at 0xFF (mode 3 – Slide 25) or OCR0A (Mode 7 – Slide 25)
- Start again at 0
- Pin OCnx generates waveform (n =
  - Set(reset) when timer=0
  - Reset(set) when timer=OCRnx (n = 0,1,2- x = A,B)



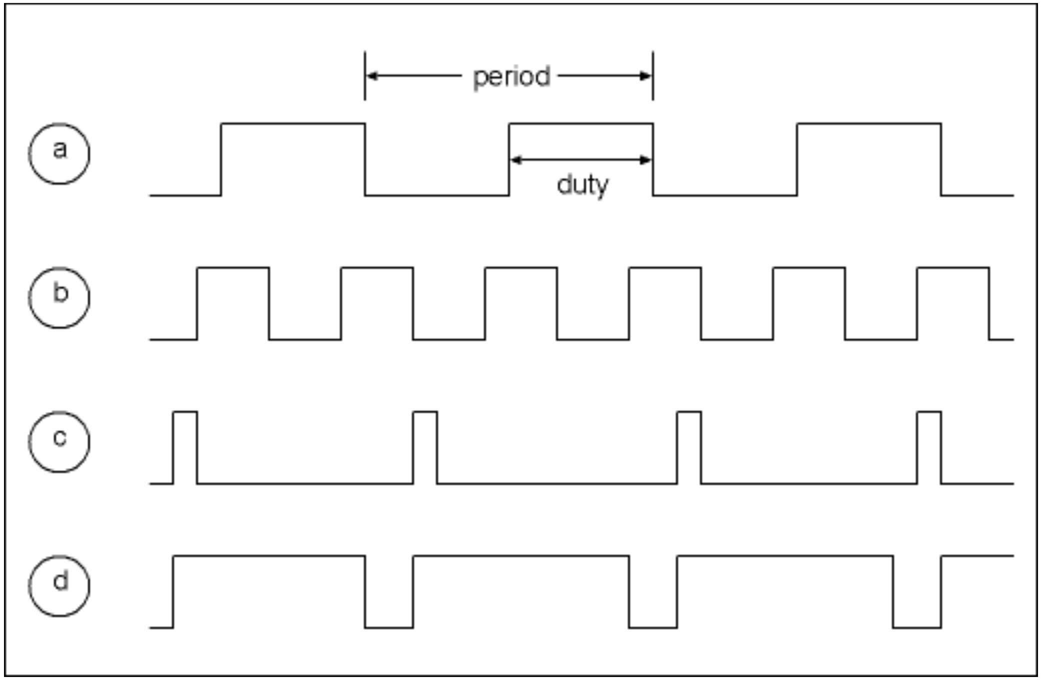
Phase Correct Pulse Width Modulation (PWM)

- Phase Correct Pulse-Width Modulation (mode 1 and 5)
  - Useful for using digital circuits to achieve analog-like control of motors, LEDs, etc.



Pulse Width Modulation (PWM)

- Dynamically change duty cycle of a waveform
- Used to control motor speed



Registers (n=0, 1, 2)

- TCNTn – Timer/Counter Register (8-bit)
- OCRnA – Output Compare Register A
- OCRnB – Output Compare Register B
- TCCRnA/B – Timer/Counter Control Registers
- TIMSKn – Timer/Counter Interrupt Mask Register
  - TOV interrupt
  - Compare A&B interrupts
- TIFRn – Timer/Counter Interrupt Flag Register
  - TOV interrupt
  - Compare A&B interrupts

Timer/Counter 0 Control Registers

Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer/Counter Mode Summary (Timer 0)

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on[1][2]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes: 1. MAX = 0xFF
2. BOTTOM = 0x00

Modes (cont'd for Timer 0)

Table 14-2. Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match
1	1	Set OC0A on Compare Match

Table 14-3. Compare Output Mode, Fast PWM Mode[1]

0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal Port Operation, OC0A Disconnected. WGM02 = 1: Toggle OC0A on Compare Match.
1	0	Clear OC0A on Compare Match, set OC0A at BOTTOM, (non-inverting mode).
1	1	Set OC0A on Compare Match, clear OC0A at BOTTOM, (inverting mode).

Table 14-4. Compare Output Mode, Phase Correct PWM Mode[1]

0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal Port Operation, OC0A Disconnected. WGM02 = 1: Toggle OC0A on Compare Match.
1	0	Clear OC0A on Compare Match when up-counting. Set OC0A on Compare Match when down-counting.
1	1	Set OC0A on Compare Match when up-counting. Clear OC0A on Compare Match when down-counting.

Pre-scaler values

Table 15-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(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 T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

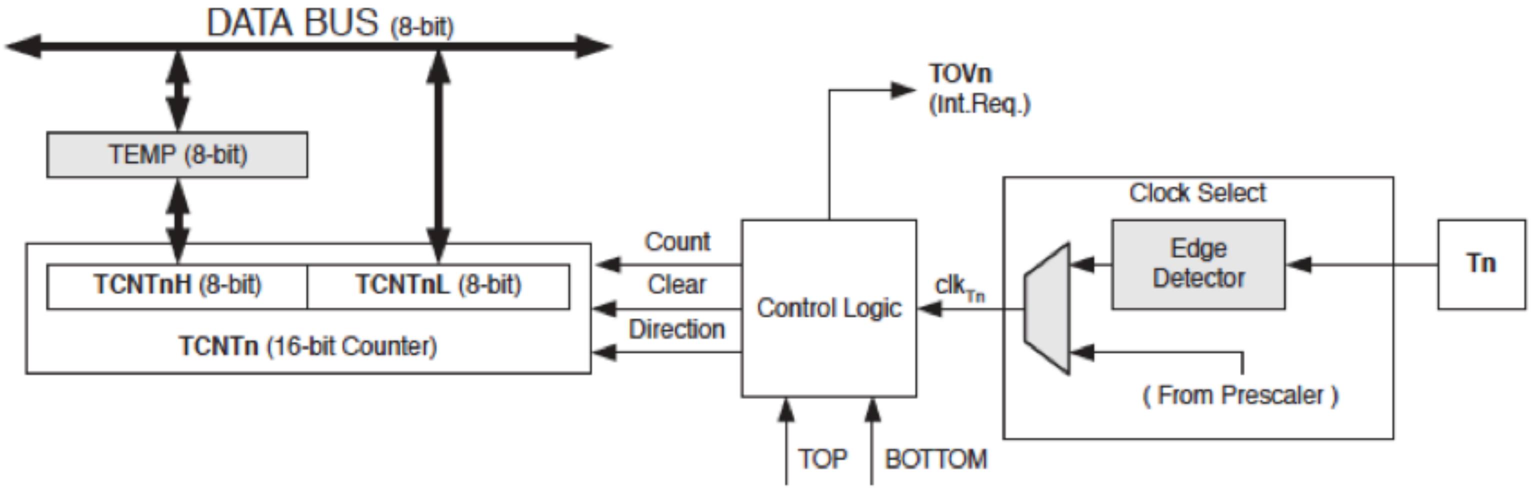
TIFRo (Timer Interrupt Flag Register)

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

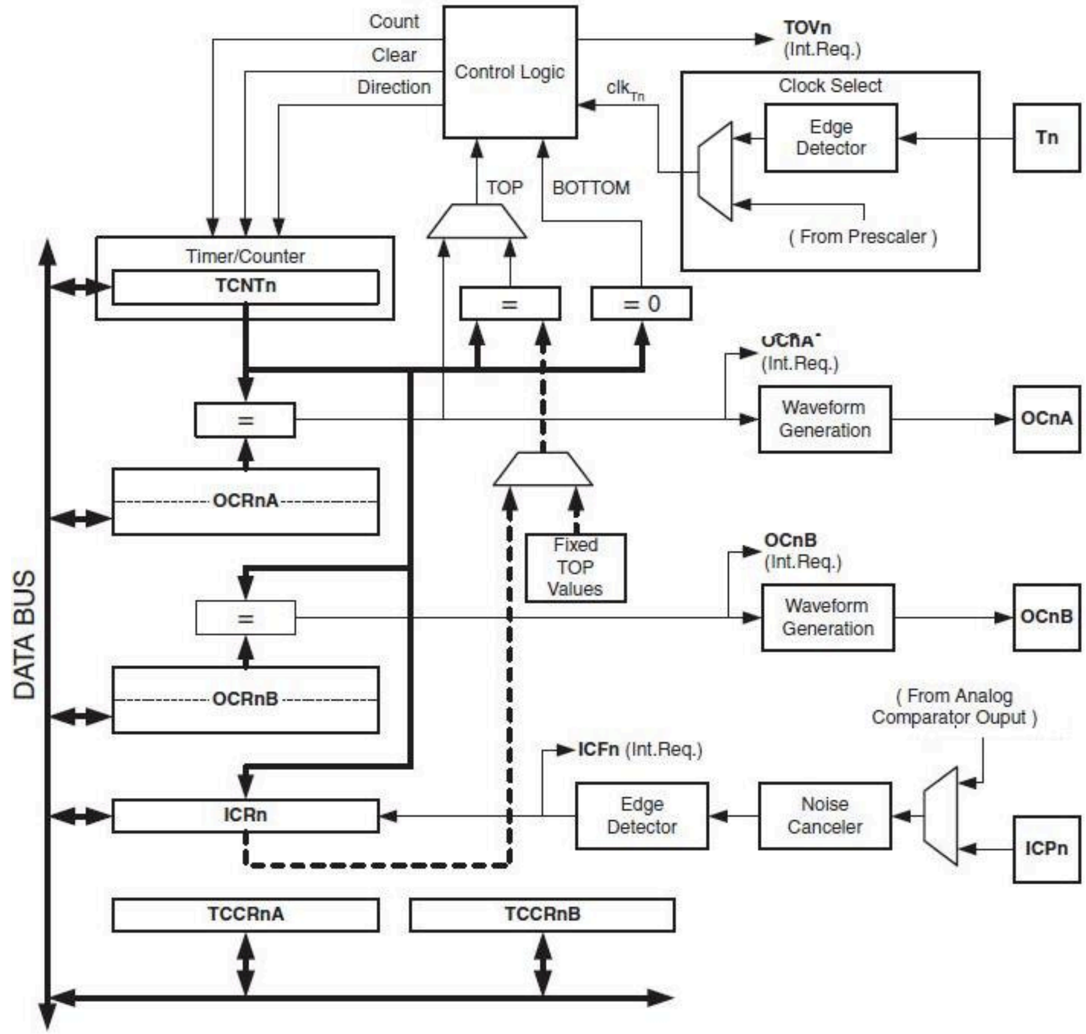
- The bit TOVo is set when an overflow occurs in Timer/Countero
  - TOVo is cleared by hardware when executing the corresponding interrupt handling vector
  - Alternatively, TOVo is cleared by writing a logic one to the flag
- The OCFoA/OCFoB bit is set when a Compare Match occurs between the Timer/Countero and the data in OCRoA/OCRoB
  - OCFoA/OCFoB is cleared by hardware when executing the corresponding interrupt handling vector
  - Alternatively, OCFoA/OCFoB is cleared by writing a logic one to the flag

AVR 16-bit Timer/Counter

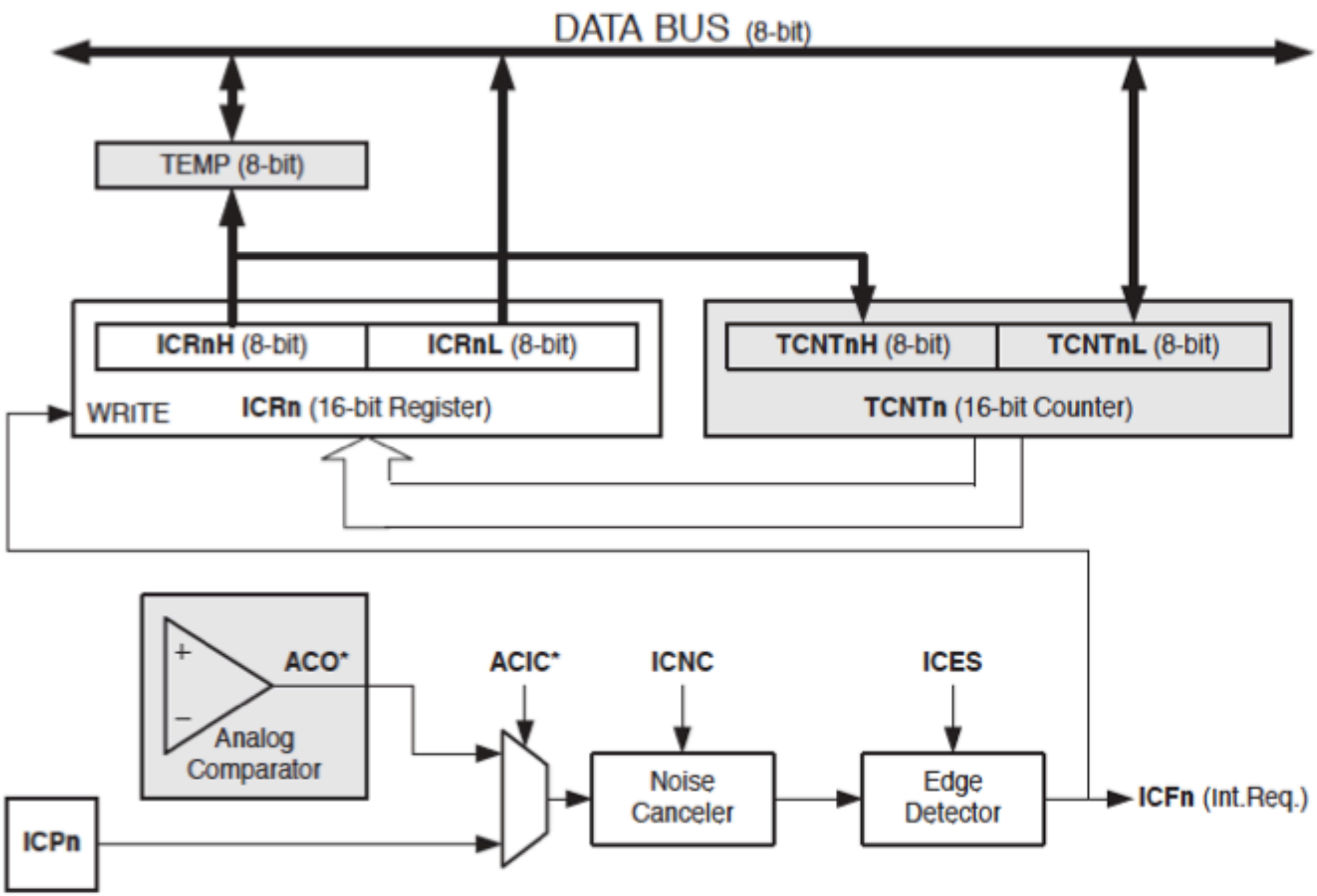
- 16 Bit Up Counter (Timer/Counter 1)
  - Counts from 0 to 65535 (0xFFFF), then loops
  - Internal clock source with pre-scaler or External Clock
- 16-bit comparators
- Interrupts possible on:
  - Overflow
  - Compare A/B
  - Input Capture of external event on ICP pin
- Input capture register



Timer/Counter1 (cont'd)



Input Capture Unit (ICU)



ICU Properties

- Event on input causes:
  - Counter value (TCNT1) to be written to ICR1
    - Time-stamp
  - Interrupt flag ICF1 to be set
    - Causing an interrupt, if enabled
- Pin ICP1 – Port B [0]
- Useful for measuring frequency and duty cycle
  - PWM inputs

Timer/Counter Mode Summary (Timer 1)

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Coding Examples

Set Timer 0 for 1ms

```
// this code sets up a timer0 for 1ms @ 16Mhz clock cycle
// using no interrupts

#include <avr/io.h>

int main(void){
    while (1){
        TCCR0A |= (1 << WGM01); // Set the Timer Mode to CTC

        OCR0A = 0xF9; // Set the value that you want to count to

        // set prescaler to 64 and start the timer
        TCCR0B |= (1 << CS01) | (1 << CS00);

        // wait for the overflow event
        while ( (TIFR0 & (1 << OCF0A) ) == 0){}

        TIFR0 |= (1 << OCF0A); // reset the overflow flag

    }
}
```

Set Timer 0 for 1ms and 50% Duty Cycle

```
// this code sets up a timer0 for 1ms @ 16Mhz clock cycle
// in order to function as a time delay at the beginning of the main loop
// using no interrupts

#include <avr/io.h>

int main(void){
    while (1){
        TCCR0A |= (1 << WGM01); // Set the Timer Mode to CTC

        OCR0A = 0xF9; //represents 1ms timer
        OCR0B = 0x7D; //represents 50% duty cycle

        TCCR0B |= (1 << CS01) | (1 << CS00); // set prescaler to 64 and start the timer

        //CREATE A LOGIC 1 SIGNAL (E.G. TURN ON LED)

        while ( (TIFR0 & (1 << OCF0B) ) == 0){} // wait for OCR0B overflow event

        TIFR0 |= (1 << OCF0B); // reset OCR0B overflow flag

        //CREATE A LOGIC 0 SIGNAL (E.G. TURN OFF LED)

        while ( (TIFR0 & (1 << OCF0A) ) == 0){} // wait for OCR0A overflow event

        TIFR0 |= (1 << OCF0A); // reset OCR0A overflow flag

    }
}
```

Set Timer 0 for 4ms

```
// this code sets up a timer0 for 4ms @ 16Mhz clock cycle
// using no interrupts

#include <avr/io.h>

int main(void){
       while (1){
        TCCR0A |= (1 << WGM01); // Set the Timer Mode to CTC

        OCR0A = 0xF9; // Set the value that you want to count to

        TCCR0B |= (1 << CS02); // set prescaler to 256 and start the timer

        while ( (TIFR0 & (1 << OCF0A) ) == 0){}
       // wait for the overflow event

        TIFR0 |= (1 << OCF0A); // reset the overflow flag
    }
}
```

Set Timer 0 for 4ms (uses interrupts)

```
// 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){

    // Set the Timer Mode to CTC
    TCCR0A |= (1 << WGM01);

    // Set the value that you want to count to
    OCR0A = 0xF9;

    TIMSK0 |= (1 << OCIE0A);    //Set the ISR COMPA vect
    sei();         //enable interrupts

    TCCR0B |= (1 << CS02);
    // set prescaler to 256 and start the timer

    while (1){ //main loop }
}

ISR (TIMER0_COMPA_vect)  // timer0 overflow interrupt
{
    //event to be exicuted every 4ms here
}
```