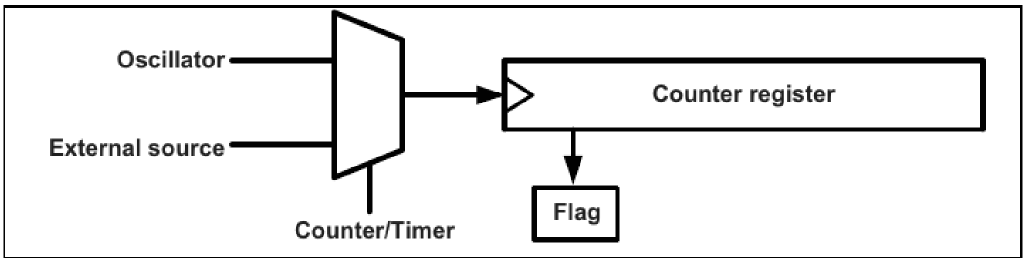
TIMER AND COUNTER

Many applications need to generate time delays or count events. ATmega32 has three Timers – called Timer0, Timer1 and Timer2. Timer0 and Timer2 are 8 bit timer and Timer1 is 16 bit timer. To work with timer, we use the internal clock source (oscillator) and to work with counter we use external clock source. Following figure illustrates this concept.



Basic registers of timers

- TCNTn (Timer/Counter)
  - ATmega32 has TCNT0, TCNT1 and TCNT2
  - It is a counter and counts up with each pulse
  - When a timer overflows, its TOVn flag will be set.

- TCCRn (Timer/Counter Control Register)
  - Used for setting different operation modes of Timers
  - For example, we can specify Timer0 to work as a timer or a counter by loading proper values into the TCCR0.

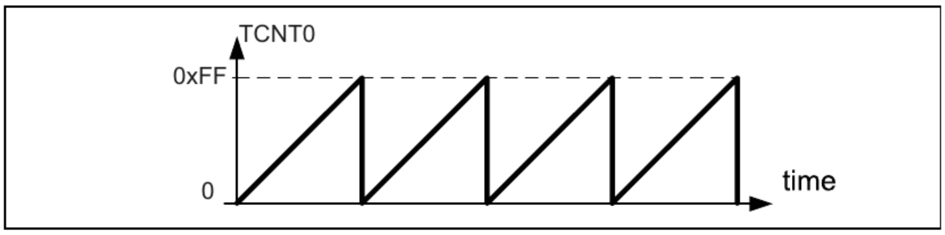
- OCRn (Output Compare Register)
  - The content of the OCRn is compared with the content of the TCNTn. When they are equal the OCFn (Output Compare Flag) flag will be set.

These timer registers are located in the I/O register memory.

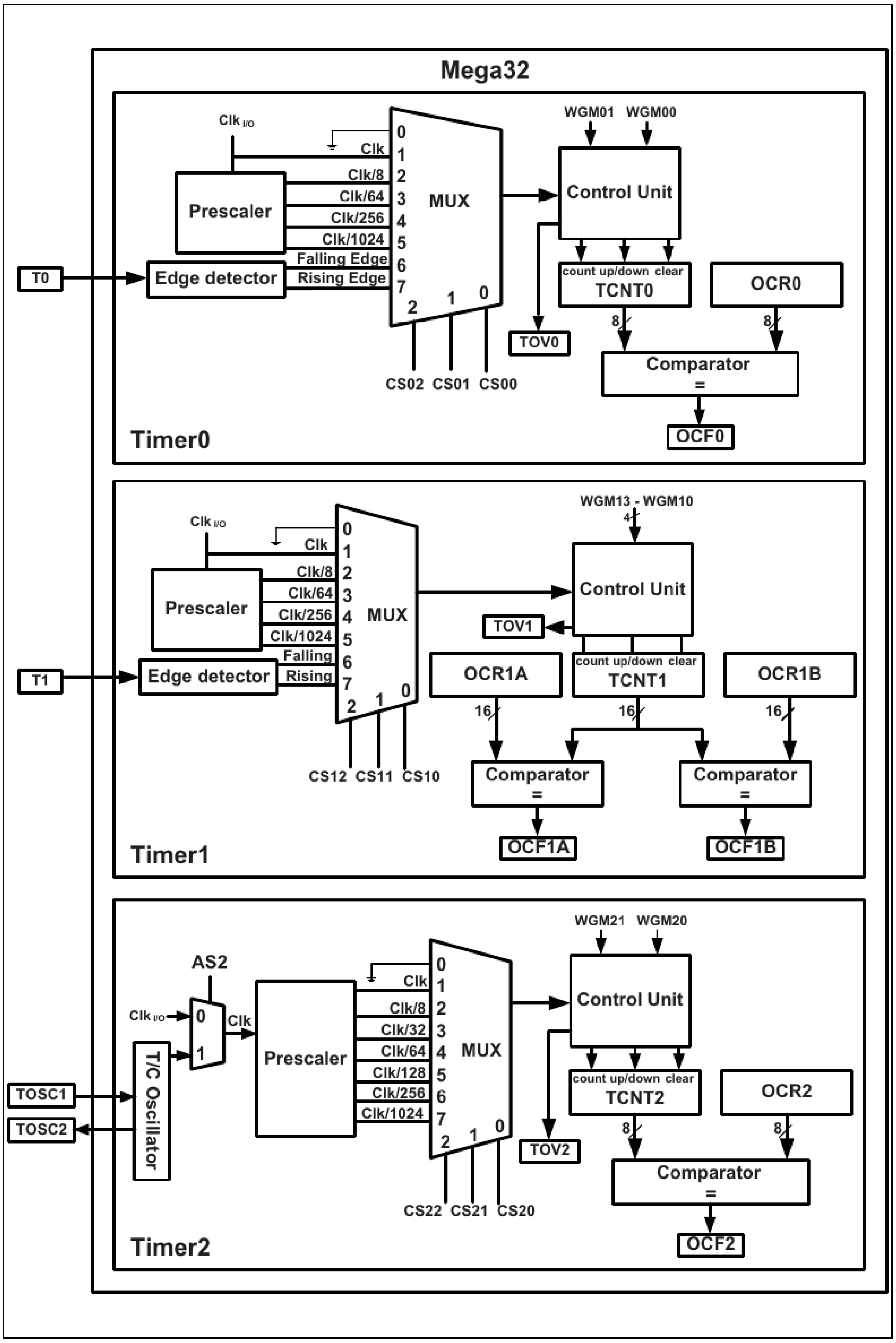
Operating Modes:
Mainly there are two operating modes of timers:
1. Normal Mode
2. CTC mode

Normal Mode
In the normal mode, the Timer/Counter register (TCNTn) increments on each clock pulse up to 0xFF (in Timer0 and Timer2) and up to 0xFFFF (in Timer1). After reaching this value, it rolls over to zero. When it rolls over, a flag, called TOV flag (Timer Overflow flag) is set. This flag can be monitored using the program to check the time delay.

Following diagram illustrates the working of Normal mode of Timer0.



Block Diagrams of Timer0, Timer1 and Timer2



To generate time delay using Timer0 in normal mode, the following steps are taken:

1. Load the TCNT0 register with the initial count value.
2. Load the value into the TCCR0 register, to indicate the operating mode
3. Keep monitoring the timer overflow flag (TOV0). Get out of the loop when TOV0 becomes high.
4. Stop the timer by loading 0x00 to TCCR0
5. Go back to Step 1 to load TCNT0 again.

<u>Finding the value to be loaded in to the timer</u>

- If we know the amount of time delay needed, then we have to load a counter value to TCNTn register. The counter value can be calculated using the following steps:
  1. Calculate the period of timer clock
  2. Calculate number of clocks needed by dividing time delay by clock period
  3. Subtract result of step2 from 256
  4. Find its hex value which is the counter value to be loaded.

Example: Assume crystal frequency is 8MHz, find the counter value to get 6.25 microsecond time delay

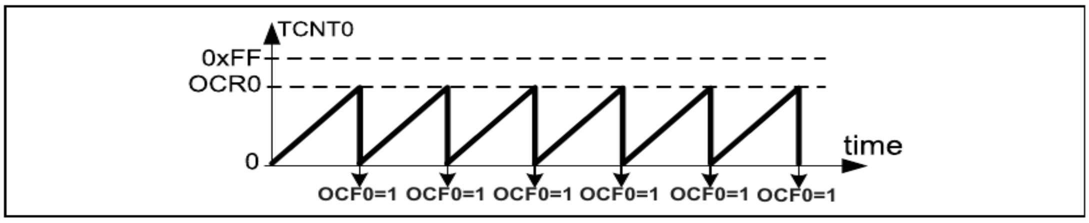
Step 1: Clock period = 1/8MHz = 0.125 microseconds
Step 2: 6.25/0.125 = 50 clock cycles needed
Step 3: 256-50 = 206
Step 4: 206 = 0xCE
So counter value is 0xCE

<u>Using Prescaler to generate large time delay</u>

The largest time delay is achieved by setting TCNTn to zero. If want large delay, we use the prescaler option in the TCCRn register to increase the delay by reducing the frequency. In Timer0, we can reduce the frequency by dividing 8, 64, 256 and 1024. For example, if we use prescaler 8, then a 8Mhz clock frequency will be reduced to 1Mhz.

<u>CTC Mode (Clear Timer on Compare match)</u>

In the CTC mode, the timer is incremented with a clock pulse. But it counts up until the content of the TCNTn register becomes equal to the content of OCRn (compare match occurs); then the timer will be cleared and the OCFn flag will be set in the next clock. The OCRn register and OCFn flag are used in the CTC mode. Following diagram shows the working of Timer0 in CTC mode.



<u>Comparison among timers:</u>

- Timer0 and Timer2 are 8 bit timers and Timer1 is 16 bit timer
- For Timer1, certain 8 bit registers are combined to form 16 bit register (Eg: TCNT1H:TCNT1L is the 16 bit Timer/Counter register)
- Timer0 and Timer1 has prescaler options: 8,64,256,1024
- Timer2 has prescaler options: 8,32,64,128,256,1024
- Timer2 can be used as a real time counter. To do so, we have to connect a crystal of 32.768 kHz to the TOSC1 and TOSC2 pins of AVR

TIMER/COUNTER PROGRAMMING

To program Timer/Counter, we have to know the bit format of the TCCRn register. This register is used to set the operation modes. Also we should know the format of TIFR register, to find the position TOVn flag.

For the Timer0, the format of TCCR0 register is:

7	6	5	4	3	2	1	0
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

- To set timer modes, we use WGM00 and WGM01 (Bits D6 and D3)
  - For normal mode, D6=0, D3=0
  - For CTC mode, D6=0, D3=1
- To select timer source, we use CS00, CS01, CS02 (Bits D0,D1 and D2)

CS02:00	D2	D1	D0	Timer0 clock selector
	0	0	0	No clock source (Timer/Counter stopped)
	0	0	1	clk (No Prescaling)
	0	1	0	clk / 8
	0	1	1	clk / 64
	1	0	0	clk / 256
	1	0	1	clk / 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.

Format of TIFR register ((Timer/counter Interrupt Flag Register):
This register contains the Timer overflow flags (TOVn) and OCF flags.

7	6	5	4	3	2	1	0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

<u>Programming Examples:</u>
Qn: Write a C program to toggle all the bits of PORTB continuously with some delay. Use Timer0, Normal mode, and no prescaler options to generate the delay.

Program:

```
#include <avr/io.h>
void T0Delay ();
int main ()
{
  DDRB = 0xFF;      //PORTB output port
  while (1)
  {

    PORTB = 0x55;
    T0Delay ();       //delay
    PORTB = 0xAA;
    T0Delay ();       //delay
  }
}
```

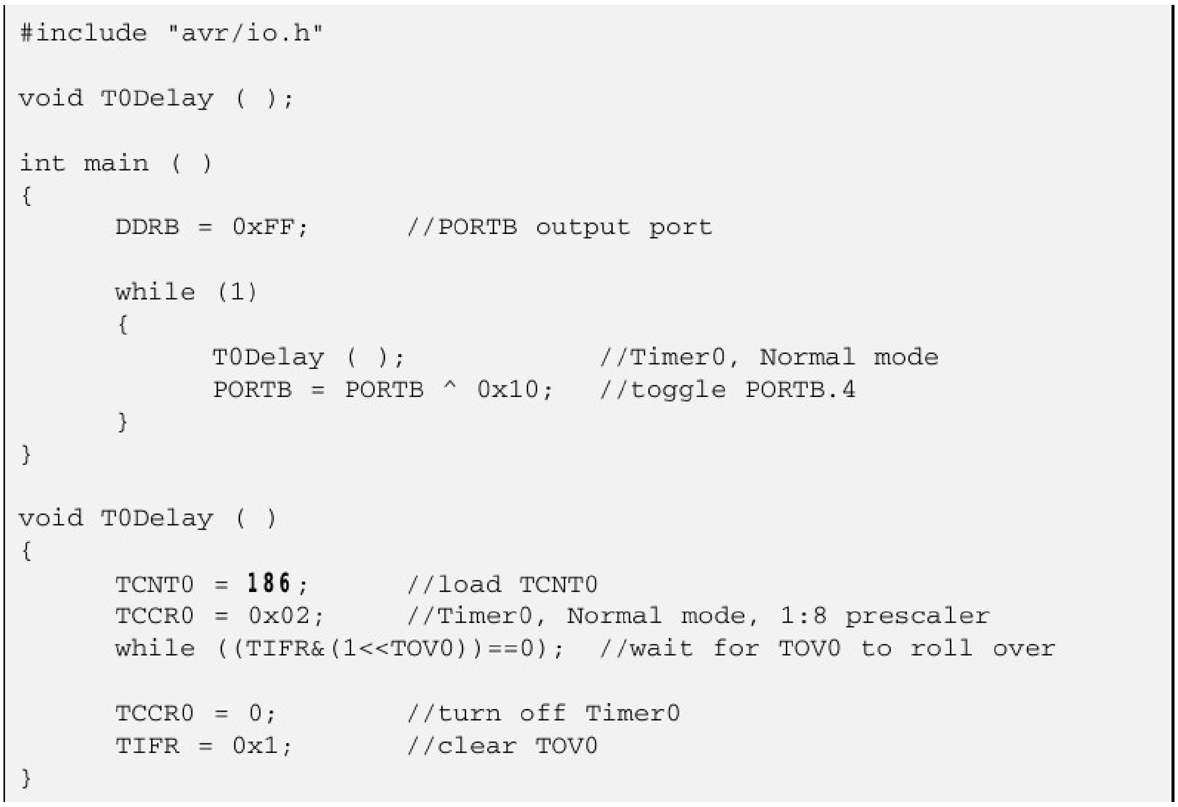
```
void T0Delay ()
{
  TCNT0 = 0x20;           //load TCNT0
  TCCR0 = 0x01;           //Timer0, Normal mode, no prescaler
  while ((TIFR&0x1)==0); //wait for TOV0 to roll over
  TCCR0 = 0;              //stop timer
  TIFR = 0x1;             //clear TOV0
}
```

Qn: Write a C program to toggle only the PORTB.4 bit continuously every 70 µs. Use Timer0, Normal mode, and 1:8 prescaler to create the delay. Assume XTAL = 8 MHz.

Solution:

Given clock frequency = 8 MHz
reduced frequency (1:8 prescaler) = 8 MHz/8 = 1 MHz
Duration of on clock period = 1 microsecond
So for 70 microsecond, 70 clock periods are required.
To calculate TCNT0 value:

256-70 = 186 = 0xBA



```
#include "avr/io.h"

void T0Delay ( );

int main ( )
{
      DDRB = 0xFF;      //PORTB output port

      while (1)
      {
            T0Delay ( );            //Timer0, Normal mode
            PORTB = PORTB ^ 0x10;   //toggle PORTB.4
      }
}

void T0Delay ( )
{
      TCNT0 = 186;      //load TCNT0
      TCCR0 = 0x02;     //Timer0, Normal mode, 1:8 prescaler
      while ((TIFR&(1<<TOV0))==0);  //wait for TOV0 to roll over

      TCCR0 = 0;        //turn off Timer0
      TIFR = 0x1;       //clear TOV0
}
```

<u>Counter Programming:</u>

Timer0 and Timer1 can be used as event counters. In counter programming, instead of using the frequency of the crystal oscillator as the clock source, we provide external pulses from outside the chip. By providing pulses to the T0 (PB0) and T1 (PB1) pins, we use Timer0 and Timer1 as Counter0 and Counter1, respectively.

Assuming that a 1 Hz clock pulse is fed into pin T0, use the TOV0 flag to extend Timer0 to a 16-bit counter and display the counter on PORTC and PORTD.

Solution:

```
#include "avr/io.h"

int main ( )
{
      PORTB = 0x01;            //activate pull-up of PB0
      DDRC = 0xFF;             //PORTC as output
      DDRD = 0xFF;             //PORTD as output

      TCCR0 = 0x06;            //output clock source
      TCNT0 = 0x00;

      while (1)
      {
            do
            {
                  PORTC = TCNT0;
            }while((TIFR&(0x1<<TOV0))==0);//wait for TOV0 to roll over

            TIFR = 0x1<<TOV0;        //clear TOV0
            PORTD ++;                //increment PORTD
      }
}
```