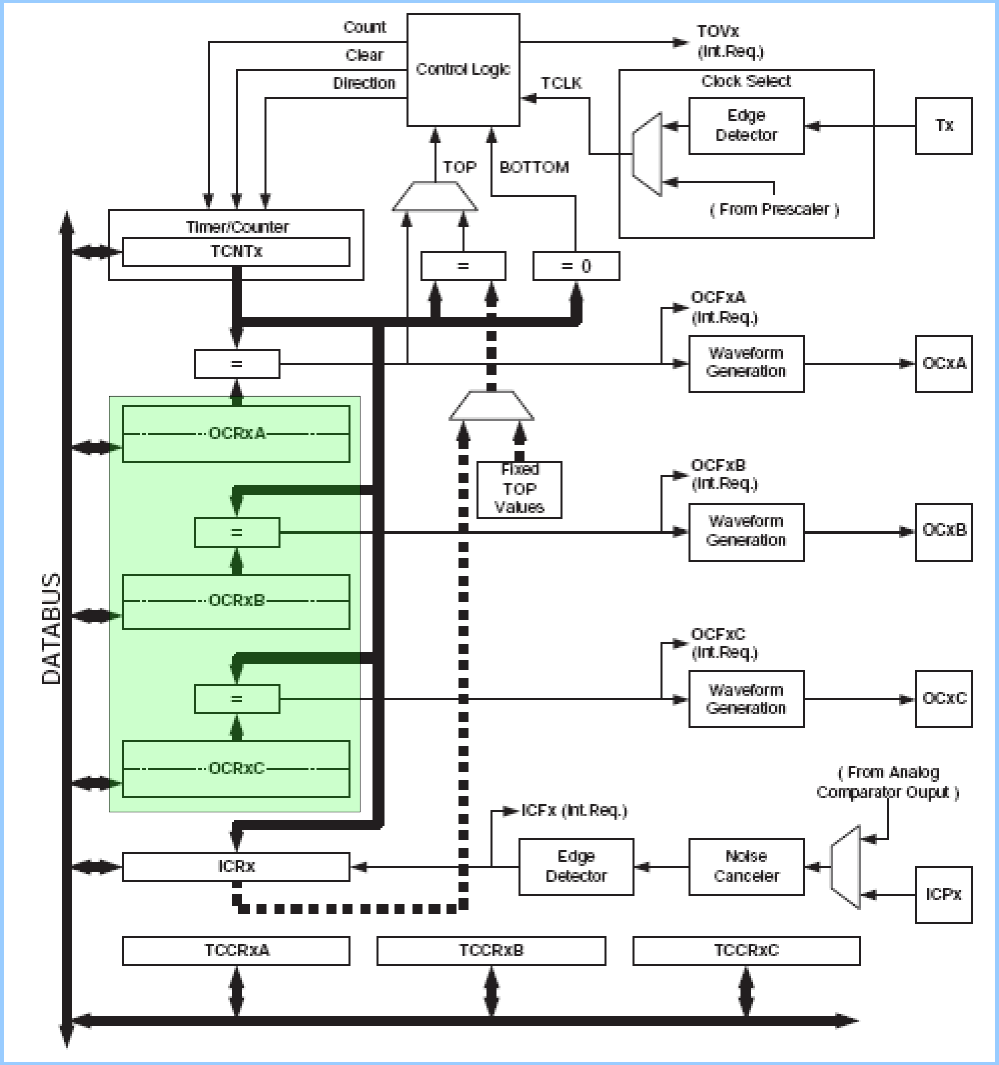
16-Bit Timer/Counter 1 and 3

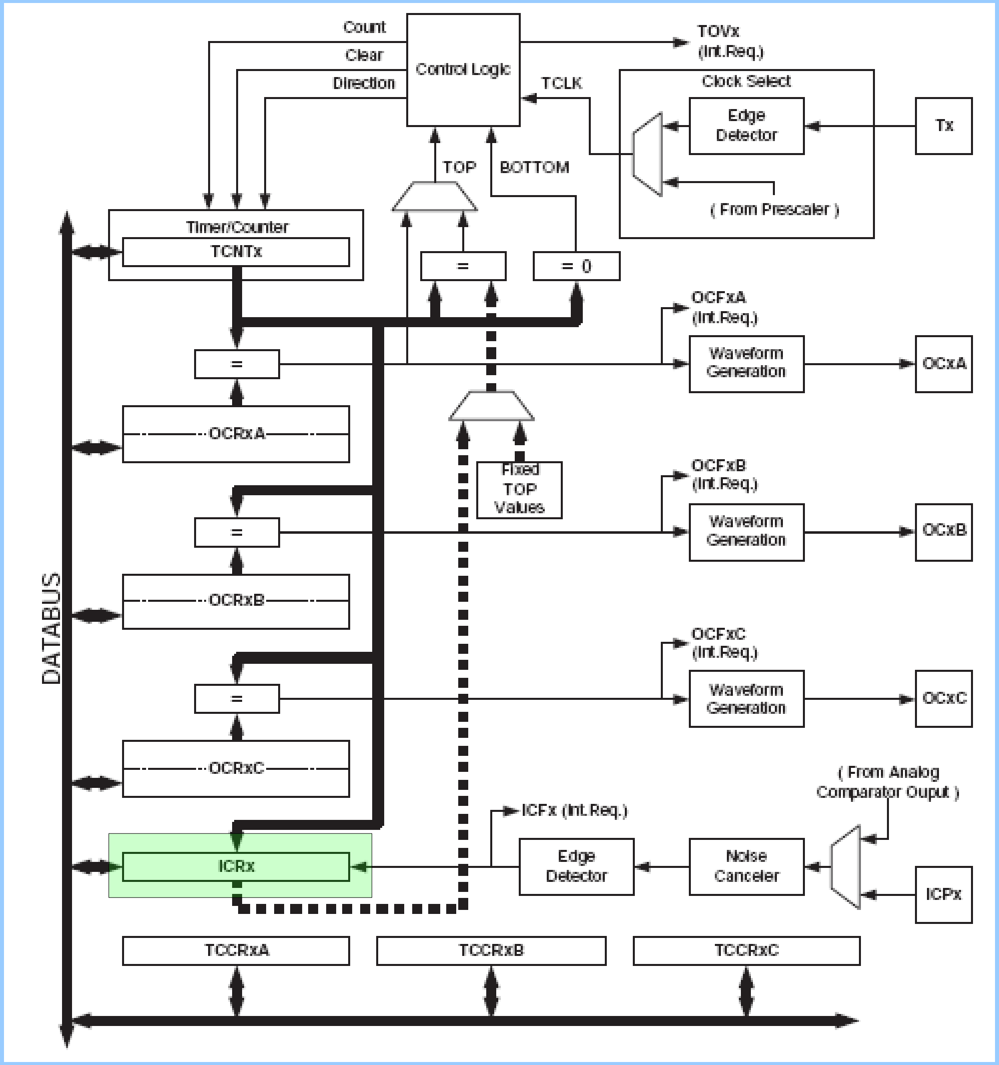
Counter/Timer 1,3 (TCNT1, TCNT3) are identical in function.

Three separate comparison registers exist. Thus, three separate outputs are available: OCxA, OCxB, OCxC



16-Bit Timer/Counter 1 and 3

An input capture register (ICRx) is available for capturing the counter value at the occurrence of external (edge) events such as an external pin change or comparator state change.



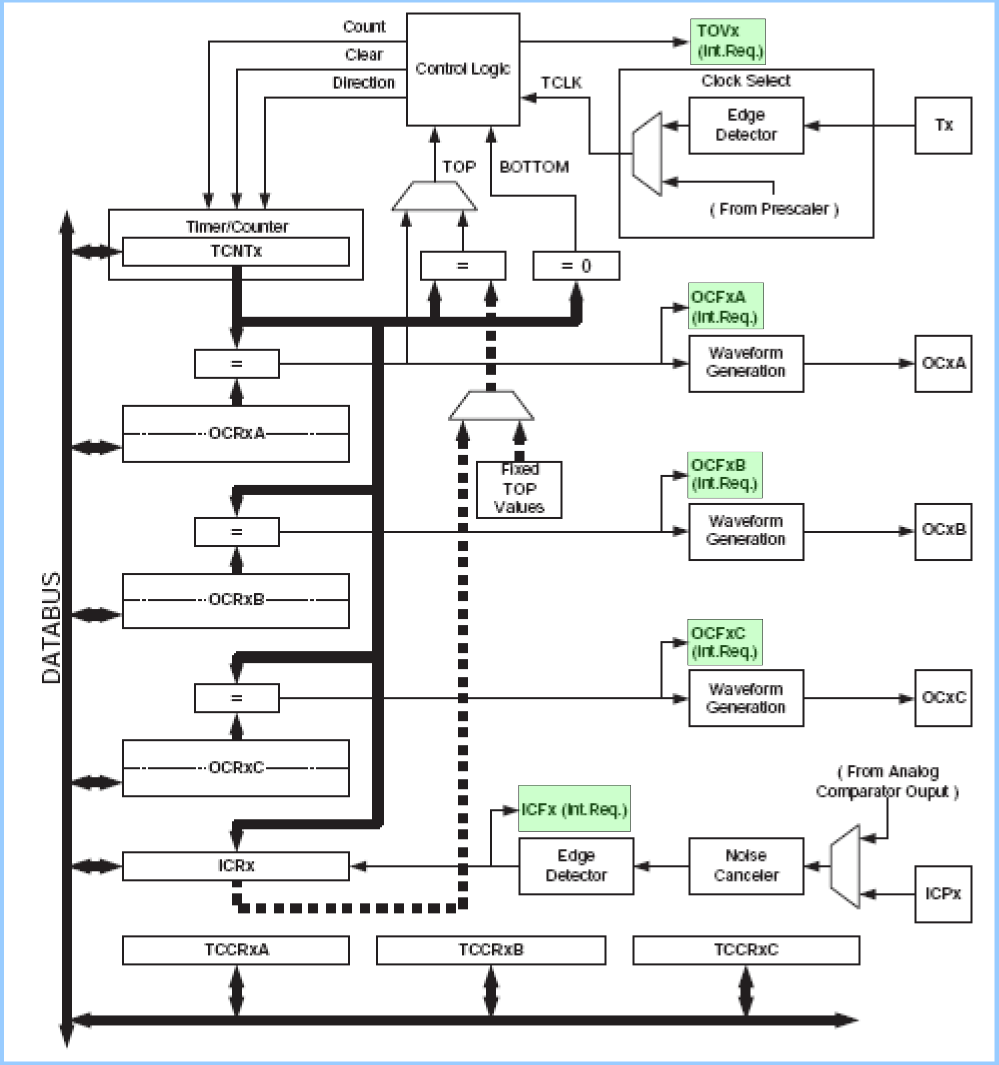
16-Bit Timer/Counter 1 and 3

Five possible interrupt sources exist for each counter/timer:

-overflow (counter register over or under flows)

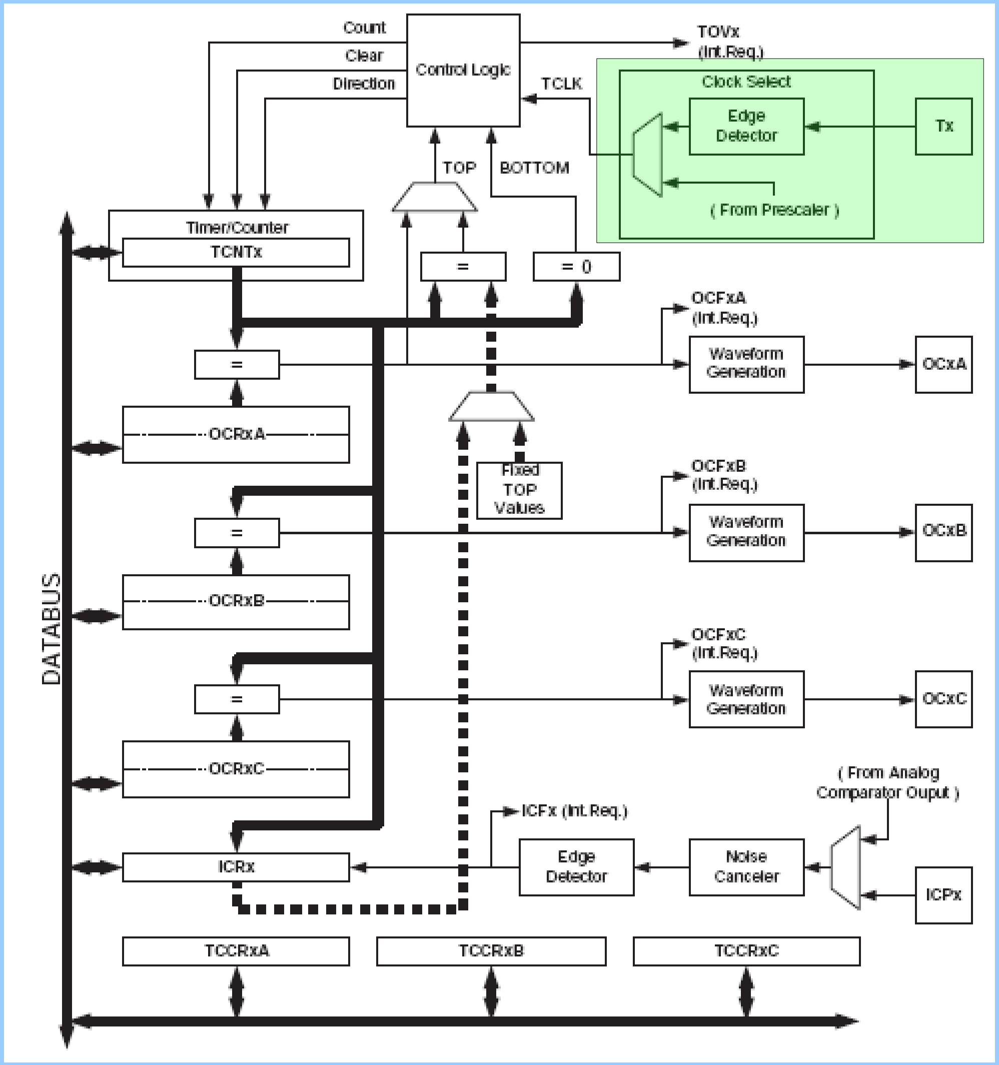
-output compare (counter register = a compare register

-input capture (something happened externally, capture the present count)



16-Bit Timer/Counter 1 and 3

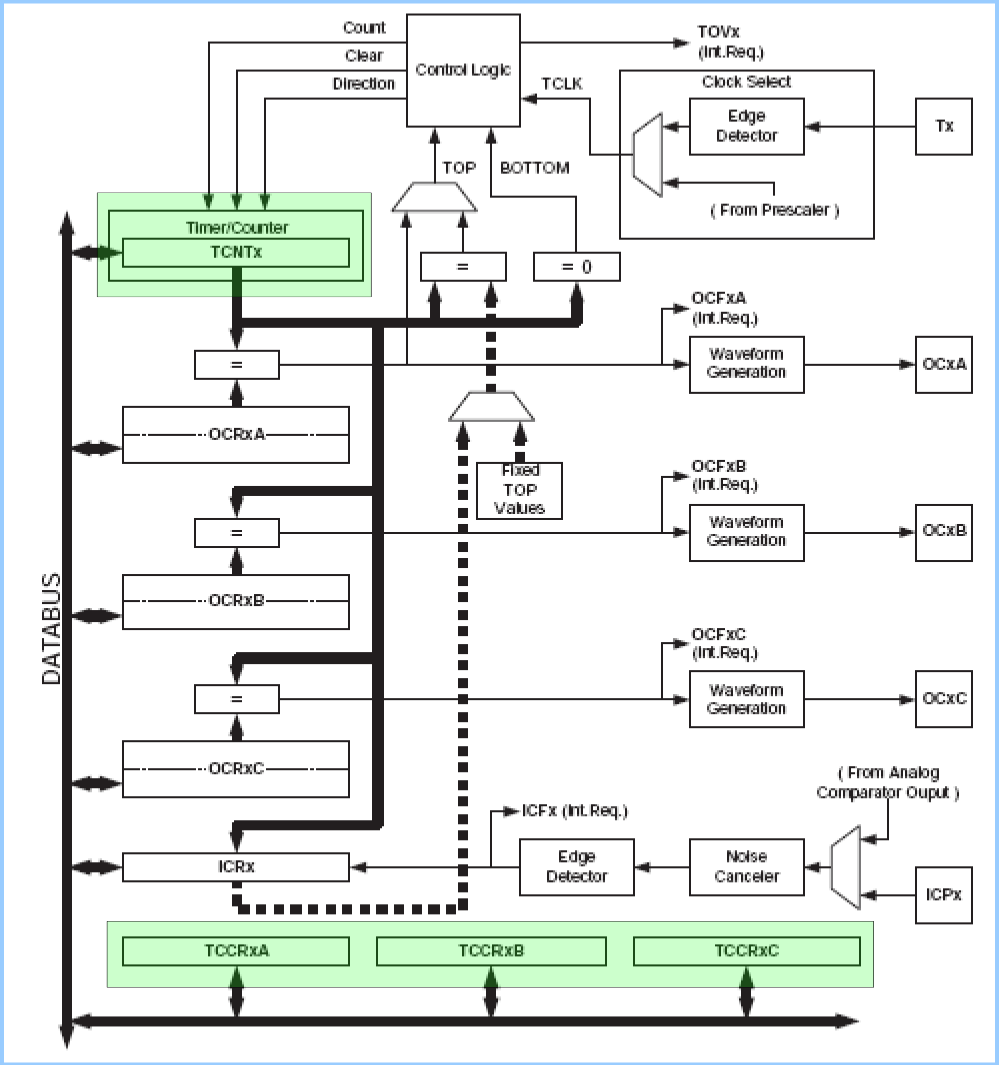
TCNT1/3 can be clocked internally, via a prescaler, or by an external clock.



16-Bit Timer/Counter 1 and 3

TCNT1/3 counter register and control registers:

16-bit counter



control registers

16-Bit Timer/Counter 1 and 3

Each register is 16 bits wide. Access is taken care of by our library (iom128.h):

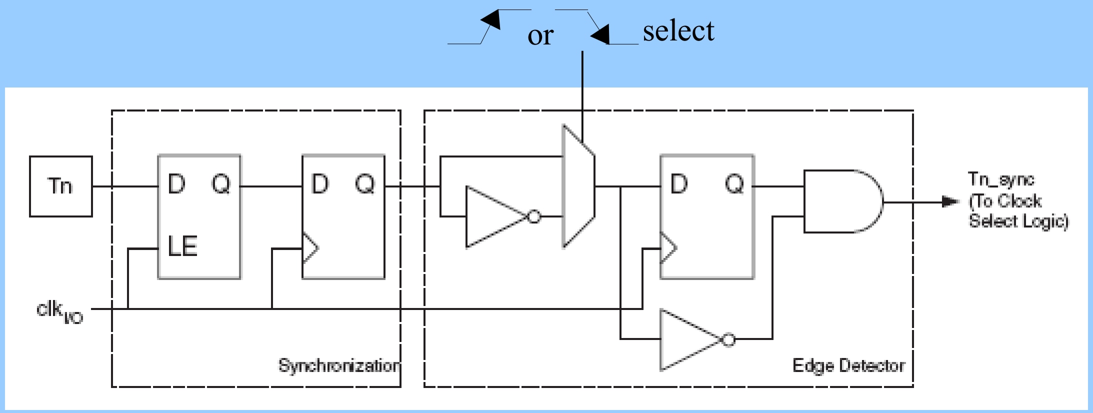
```
....
....
/* Timer Counter 1 */
#define TCNT1  _SFR_IO16(0x2C)
#define TCNT1L _SFR_IO8(0x2C)
#define TCNT1H _SFR_IO8(0x2D)
....
....

thus can say:  TCNT1 = 0x0000; //this works!
```

16-Bit Timer/Counter 1 and 3

Timer/Counter Sources:

- -control register B determines the clock source
  - -directly off of internal clock f_{clk} (16Mhz)
  - -internal clock prescaled by $f_{clk}/8$, $f_{clk}/64$, $f_{clk}/256$, or $f_{clk}/1024$
  - -external clock sources (*T1* or *T3* pin)
    - -no prescaler available for external clocks
    - -they first get synchronized, then edge detected
    - -one pulse for either rising or falling edge
    - -external clocks incur 2.5-3.5 system clock delay

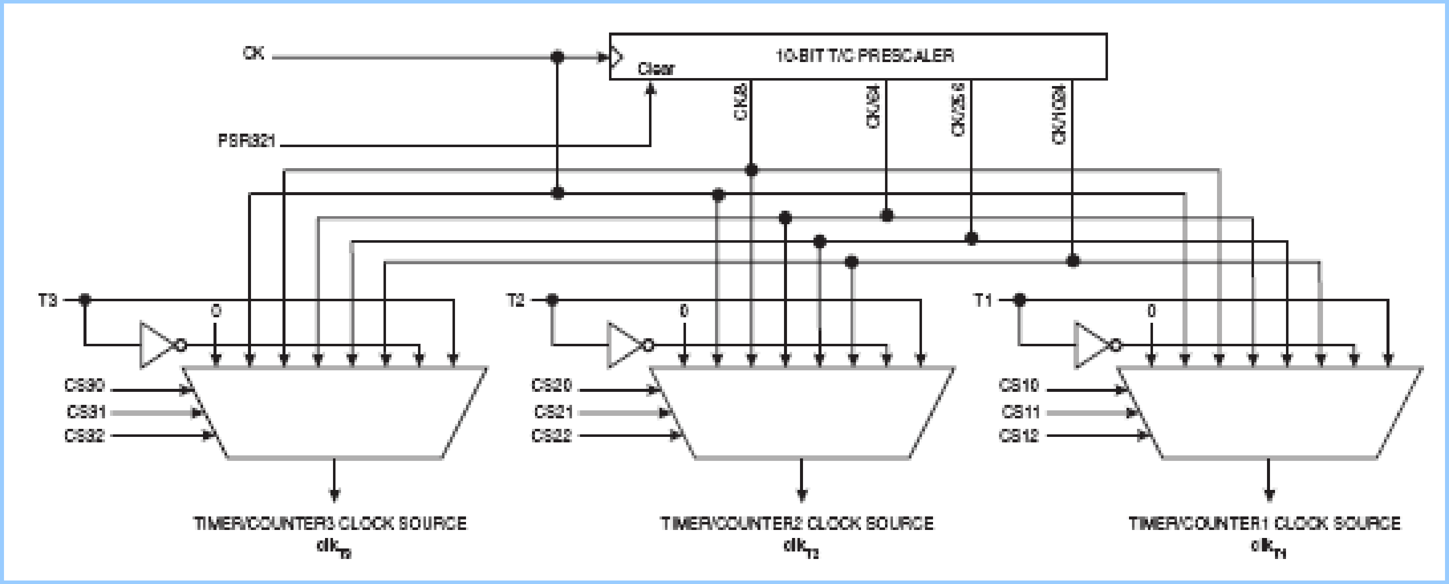


-To sample an external clock it must be longer than 2 f_{clk} periods

16-Bit Timer/Counter 1 and 3

Timer/Counter Sources:

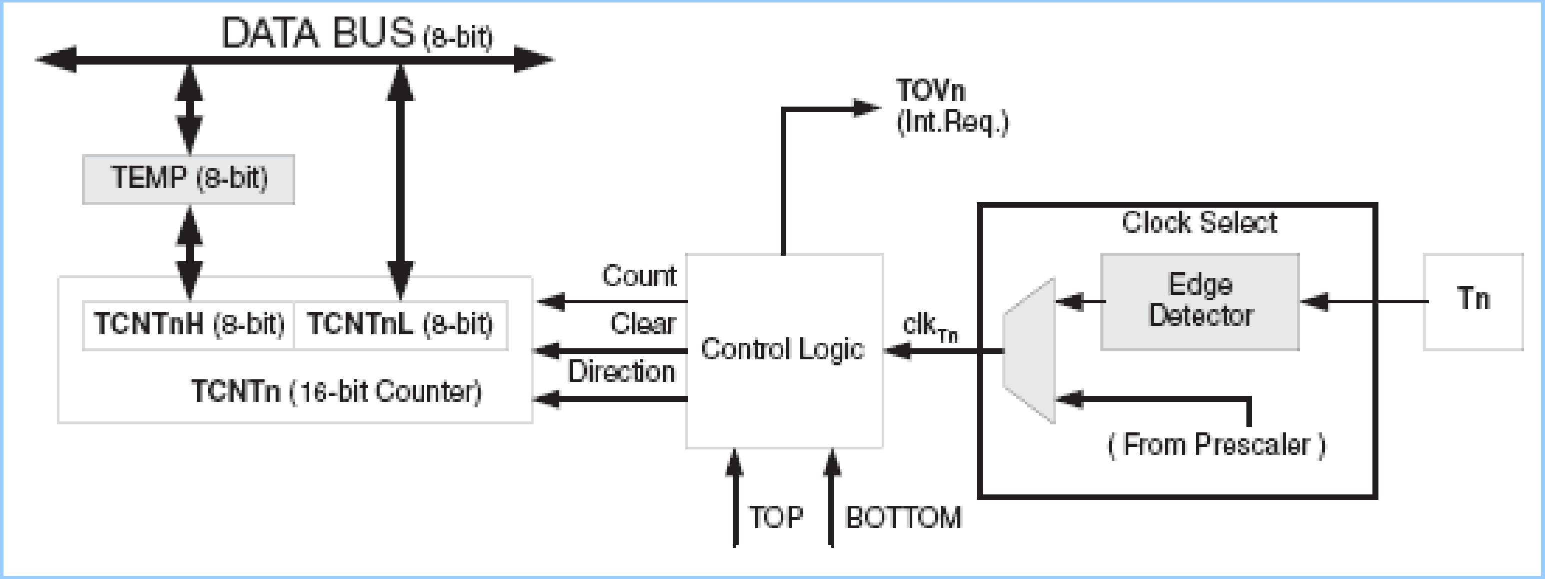
-internal clock divided by prescaler at $f_{clk}/8$, $f_{clk}/64$, $f_{clk}/256$, or $f_{clk}/1024$



CK
10-BIT T/C PRESCALER
Clear
PSR321
CK/8
CK/64
CK/256
CK/1024
T3
T2
T1
0
CS30
CS31
CS32
CS20
CS21
CS22
CS10
CS11
CS12
TIMER/COUNTER3 CLOCK SOURCE clkT3
TIMER/COUNTER2 CLOCK SOURCE clkT2
TIMER/COUNTER1 CLOCK SOURCE clkT1

16-Bit Timer/Counter 1 and 3

Counter Unit:



The counter is incremented, decremented or cleared at each clk$_{Tn}$

Counter signals:

count: increment or decrement enable for counter

direction: count up or down

clear: clear counter

clk$_{Tn}$: the selected clock source

TOP: indicates counter has reached largest value allowed ***(not 0xFFFF)***

BOTTOM: indicates counter has reached lowest value allowed ***(not 0x0000)***

16-Bit Timer/Counter 1 and 3

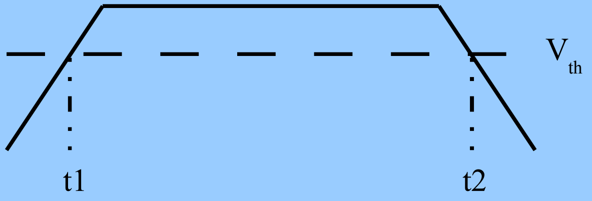
Input Capture Unit:

The input capture unit can detect events and give them a time stamp.

-can calculate frequency

-duty cycle

-duty cycle at selected voltages



Trigger sources for the ICU:

External events come in via the ICP1 or ICP3 pins

-sampled like *Tn* pins (latch, synchronization, edge detection)

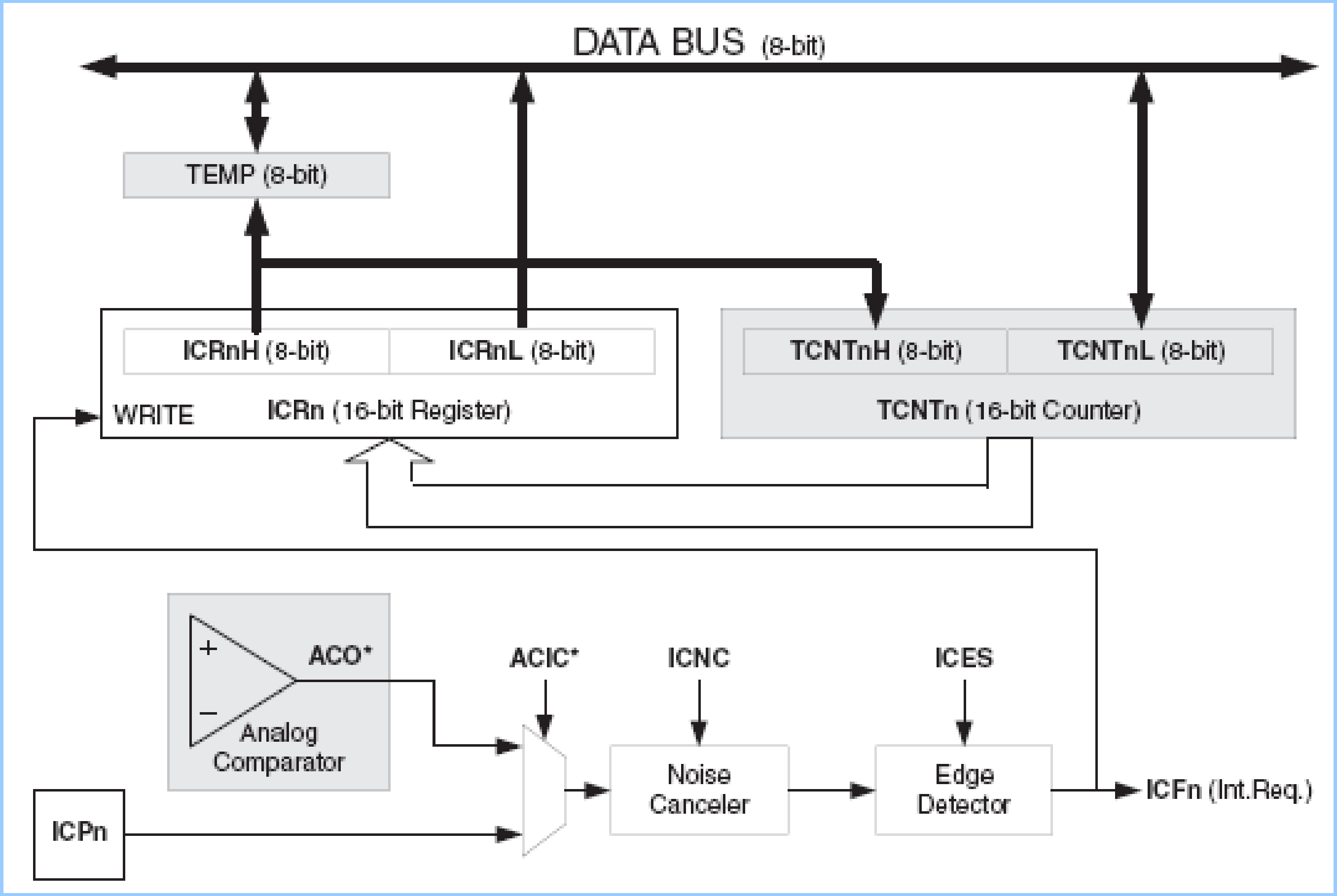
Internal events come from the analog comparator.

-sampled like Tn pins but with the option of a noise canceler

-noise canceler is probably a 4 stage shift register with a 4-input AND gate

16-Bit Timer/Counter 1 and 3

Input Capture Unit:

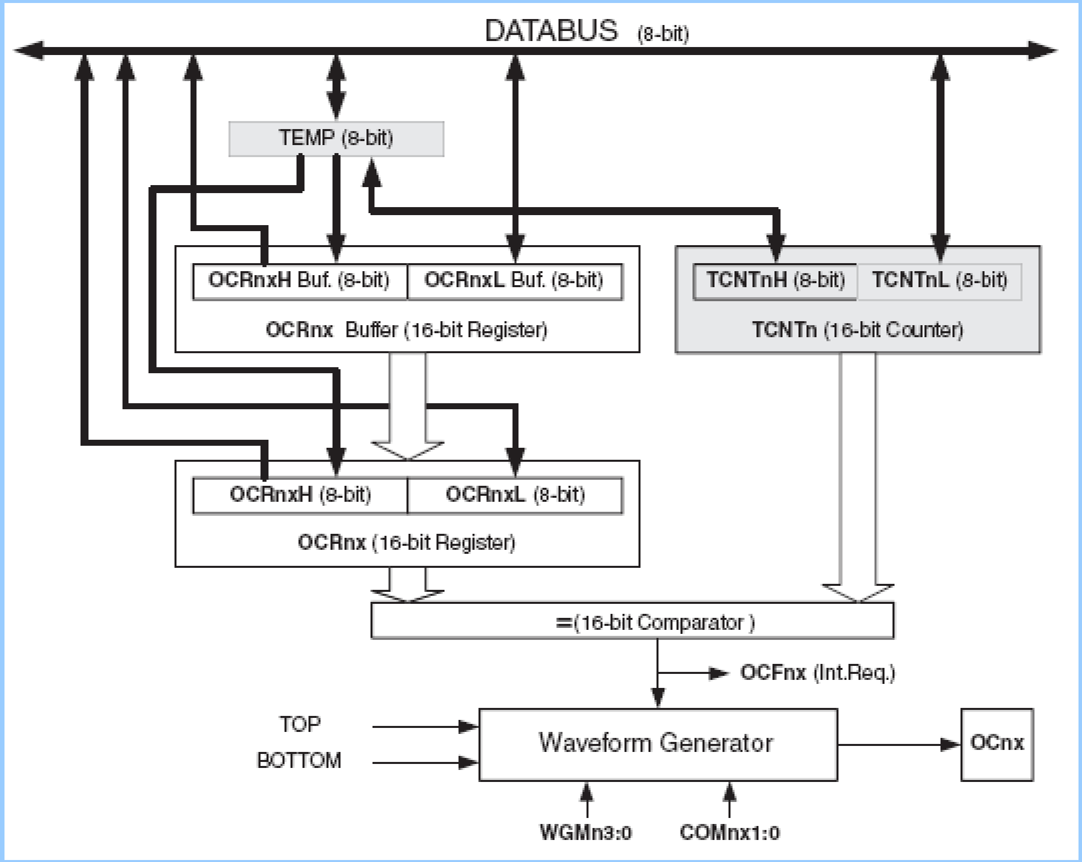


When an event occurs, the value of TCNTn is captured in ICRn

Note: analog comparator can trigger TCNT1 but not TCNT3

16-Bit Timer/Counter 1 and 3

Output Compare Unit:



16-bit comparator continuously compares TCNTn and OCRnx.

If equal, the output compare flag is set (OCFnx) and an interrupt can be issued.

The waveform generator uses this signal to generate an output to a pin.

16-Bit Timer/Counter 1 and 3

Modes of Operation:

Mode is determined by:

-Waveform Generation Mode (WGMn3:0)

-Compare Output Mode (COMnx1:0)

Normal Mode

-simplest mode

-count up to 0xFFFF and wrap around to 0x0000

-no clear is ever performed

-TOV flag is set when the wrap around occurs (*overflow*)

-to reset TOV, must execute ISR or clear flag manually

-no output pins are used

16-Bit Timer/Counter 1 and 3

Modes of Operation:

Clear Timer on Compare Match (CTC) Mode

-resolution of counter is manipulated by output compare register A (OCRnA) or input capture register (ICRn)

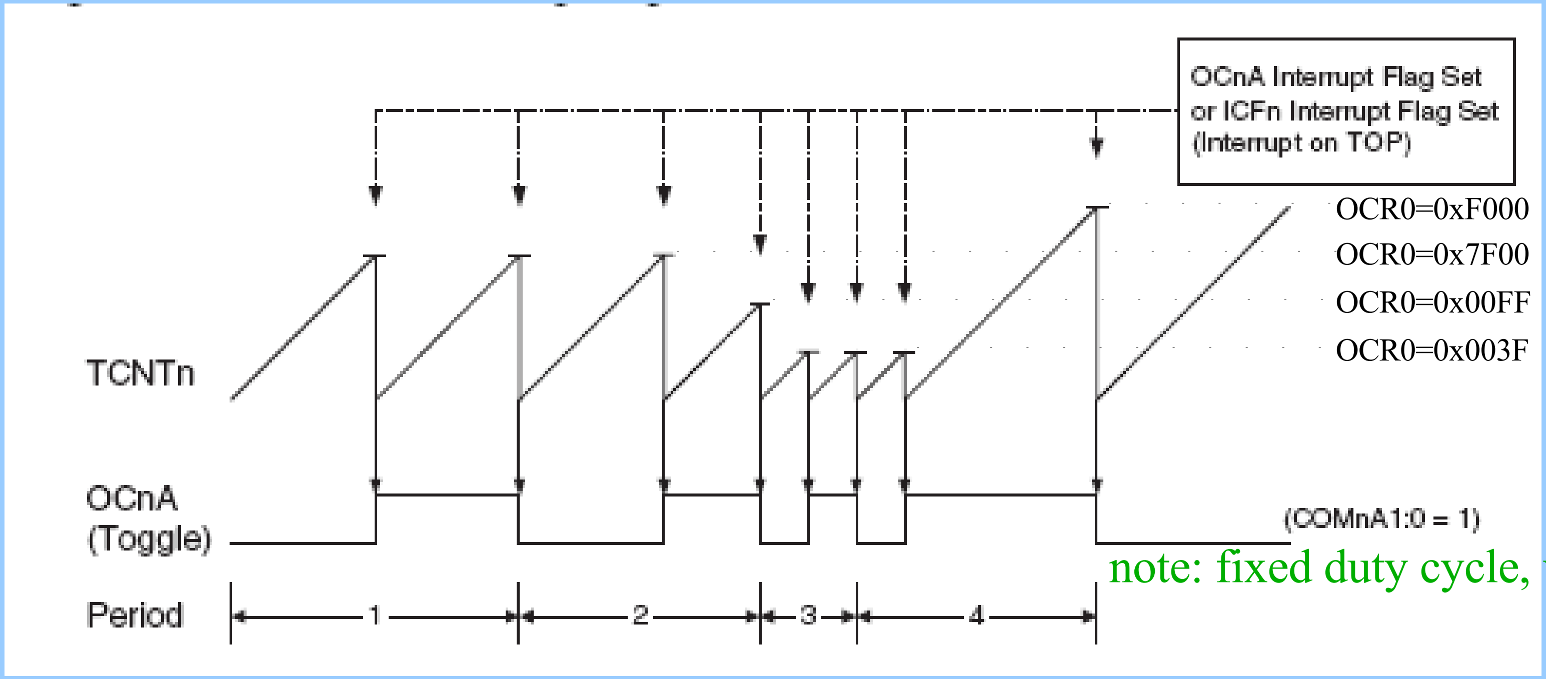
-counter is cleared to zero when its value equals either ICRn or OCRnA

-TOP is defined by ICRn or OCRnA

-interrupt can be generated at compare point

-output pins (OCnx) can be utilized

-toggle, set, or clear on match



note: fixed duty cycle, variable frequency

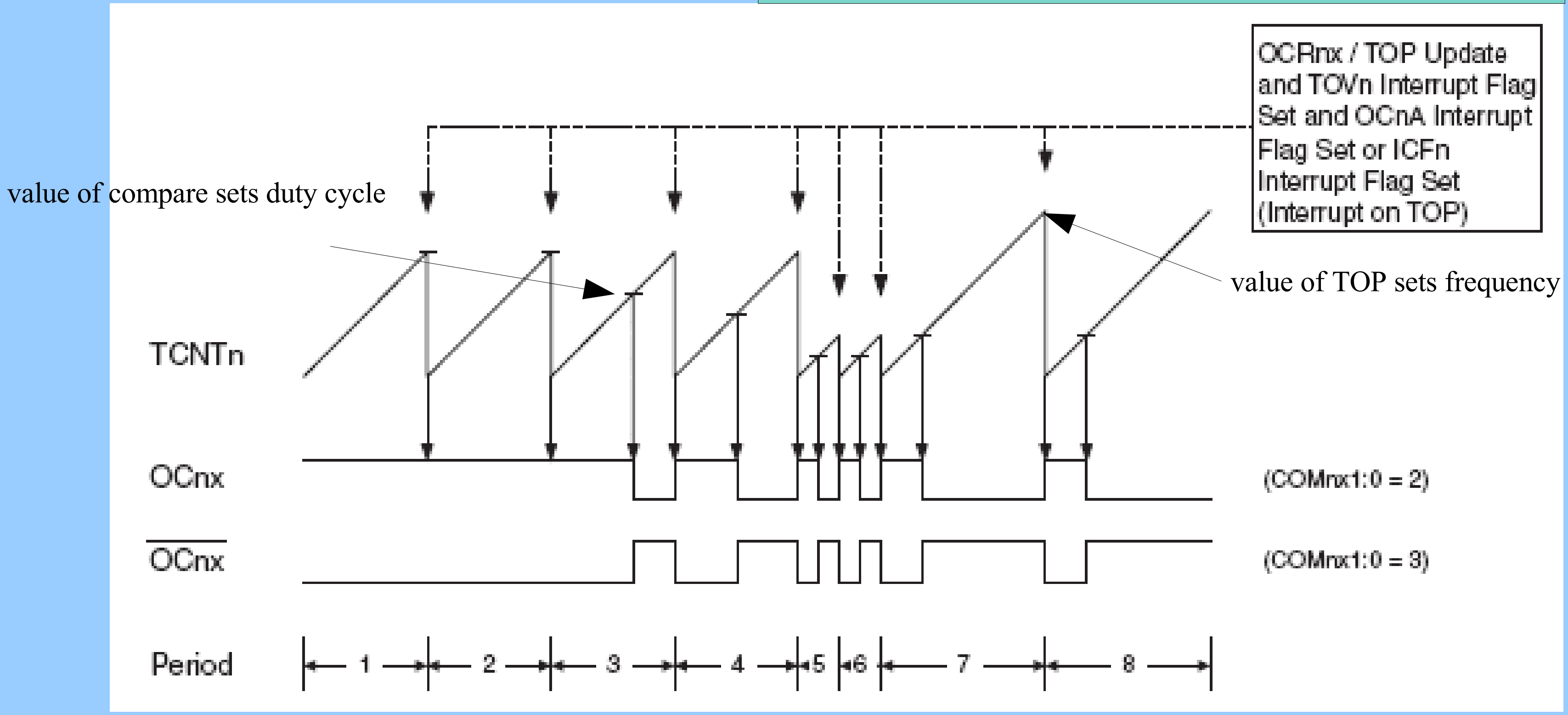
16-Bit Timer/Counter 1 and 3

Modes of Operation:

Fast PWM Mode

- -used to create high resolution PWM waveforms
- -same frequency, different duty cycle
- -count from bottom to top, then reset to bottom
- -output compare behavior:
  - -set on compare match
  - -reset at TOP*

*Top defined as:
0x00FF, 0x01FF, 0x03FF, ICRn or OCRnA


OCRnx / TOP Update and TOVn Interrupt Flag Set and OCnA Interrupt Flag Set or ICFn Interrupt Flag Set (Interrupt on TOP)
value of compare sets duty cycle
value of TOP sets frequency
TCNTn
OCnx
(COMnx1:0 = 2)
$\overline{\text{OCnx}}$
(COMnx1:0 = 3)
Period
1
2
3
4
5
6
7
8

16-Bit Timer/Counter 1 and 3

Code examples

```
// tcnt1_normal.c
// setup TCNT1 in normal mode and blink PB0 LED
// blink frequency = (16,000,000)/(2^16 * 64 * 2) = 1.91 cycles/sec
//
#include <avr/io.h>
int main()
{
  DDRB   = 0x01;                       //set port B bit zero to output
  TCCR1A = 0x00;                       //normal mode
  TCCR1B = (1<<CS11) | (1<<CS10);      //use clk/64
  TCCR1C = 0x00;                       //no forced compare

  while(1) {
    if (TIFR & (1<<TOV1)) {    //if overflow bit TOV1 is set
      TIFR  |= (1<<TOV1);      //clear it by writing a one to TOV1
      PORTB ^= (1<<PB0);       //toggle PB0 each time this happens
    }  //if
  }  //while
}  // main
```

16-Bit Timer/Counter 1 and 3

Code examples

```
// tcnt1_ctc.c
// setup TCNT1 in ctc mode
// set OC1A (PB5) to toggle on compare
// blink frequency ~= (16,000,000)/(2^15 * 64 * 2) = 3.8 cycles/sec
//
#include <avr/io.h>
int main()
{
  DDRB   = 0x20; //set port B bit five to output

  //ctc mode, toggle on compare match
  TCCR1A |= (1<<COM1A0);

 //use OCR1A as source for TOP, use clk/64
  TCCR1B = (1<< WGM12) | (1<<CS11) | (1<<CS10);

  TCCR1C = 0x00;    //no forced compare

  OCR1A = 0x7FFF;  //compare at half of 2^16

  while(1) {
    if (TIFR & (1<<OCF1A))   //if output compare flag is set
      TIFR |= (1<<OCF1A);     //clear it by writing a one to OCF1A
  }  //while
}  // main
```

16-Bit Timer/Counter 1 and 3

Code examples

```
// tcnt1_pwm.c
// setup TCNT1 in pwm mode
// set OC1A (PB5) as pwm output
// pwm frequency:  (16,000,000)/(1 * (61440 + 1)) = 260hz
//
#include <avr/io.h>
int main()
{
  DDRB   = 0x20;          //set port B bit five to output

  //fast pwm, set on match, clear at top, ICR1 holds TOP
  TCCR1A |= (1<<COM1A1) | (1<<COM1A0) | (1<<WGM11);

  //use ICR1 as source for TOP, use clk/1
  TCCR1B |= (1<<WGM13) | (1<< WGM12) | (1<<CS10);

  //no forced compare
  TCCR1C = 0x00;

  //20% duty cycle, LED is a bit dimmer
  OCR1A = 0xC000; //set   at 0xC000
  ICR1  = 0xF000; //clear at 0xF000

}  // main
```

16-Bit Timer/Counter 1 and 3

Code examples

Counter/Timers are one of the most valuable resources at your disposal.

Creation of multiple timed events from one timer:

```
ISR(xxx_vect){
  static uint8_t timer_tick;

  timer_tick++;  //increment the clock tick
  if(timer_tick % 8 == 0){do_this();}   //do every 8 ticks
  if(timer_tick % 16 == 0){do_that();}  //do every 16 ticks
}
```

When `timer_tick` reaches `0xFF`, it will advance to `0x00`. As long as the MOD operation is by a power of two, we are OK. However, if the MOD operation is not a power of two, the function calls will not occur at regular intervals once every roll-over of `timer_tick`.

This technique is convenient for multiple, repetitive events that occur at regular times.

16-Bit Timer/Counter 1 and 3

Code examples

From the previous example:

Each call is separated by either 8 or 16 timer_ticks.

timer_tick	function calls
0	do_this();do_that();
8	do_this();
16	do_this(); do_that();
24	do_this();
32	do_this(); do_that();
40	do_this();
48	do_this(); do_that();
....	
240	do_this(); do_that();
248	do_this();
255	
0	do_this(); do_that();
8	do_this();

16-Bit Timer/Counter 1 and 3

Code examples

What if the timer ticks were on different intervals?

```
ISR(xxx_vect){
  static uint8_t timer_tick;

  timer_tick++;  //increment the clock tick
  if(timer_tick == 39){timer_tick = 0);}
  if(timer_tick % 10 == 0){do_this();}   //do every 10 ticks
  if(timer_tick % 20 == 0){do_that();}   //do every 20 ticks
}//ISR
```

As a general rule the maximum count for timer_tick in this case would need to be multiple of (2n-1) where n is the value of the biggest MOD operator divisor.

16-Bit Timer/Counter 1 and 3

Code examples

Now lets look at another way to generating multiple timed function calls off one timer.

```
ISR(xxx_vect){
  static uint8_t timer_tick;

  timer_tick++;  //increment the clock tick, rollover at 255
  switch(timer_tick){
    Case 0:   do_this();                break;
    Case 10:  do_that();                break;
    Case 17:  do_something();           break;
    Case 21:  do_anything();            break
    Case 50:  do_nothing(); do_this();  break;
    Case 150: do_most();                break;
    Case 250: do_last_stuff();          break;
    Default: break;
  }//switch
}//ISR
```

Here we place functions at specific, non-reoccurring times. There may be many entries in the case statement, making for many code lines in the ISR but the execution will be much faster than a bunch of mod operations applied for many time checks. Case is usually implemented as a relative jumps or a jump table in assembly.