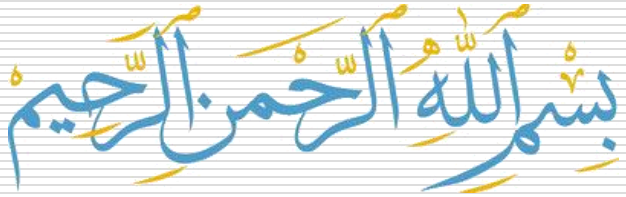


میکرو کنترلرهای AVR

برنامه ریزی تایمر

دانشکده برق و رباتیک

دانشگاه صنعتی شاهرود

حسین خسروی

Timers

- Many computer applications require accurate timing.
- Examples include
  - recording the time when an event occurs,
  - calculating the time difference between events,
  - performing tasks at specific or periodic time instants,
  - creating accurate time delays,
  - generating waveforms of certain shape, period or duty cycle.

Overview of Timers in ATmega32

- ATmega32 has 3 timers
  - Timer 0 and Timer 2 are **8 bit** timers/counters
  - Timer 1 is **16 bit** timer/counter

- Each timer is associated with a counter and a clock signal.

- The counter is incremented by 1 in every period of the timer's clock signal.

- The clock signal of a timer can come from
  - the internal system clock or
  - an external clock source.

Overview of Timers in ATmega32

- **When the internal system clock is used, a prescaler can be used make the timer count at a slower rate.**

- **Example:**
  - **Suppose the system clock rate = 1Mhz (1µs per cycle).**
  - **Suppose a timer prescaler of 64 is used.**
    - Then, timer will increment every 64µs.
    - Using (**CSn2:0 from TCCRn**) 1/8, 1/64, 1/256, 1/1024 of CLK is possible

Timer terminology

- **Input Capture:**
  - An input signal is connected to a pin, designated as input capture pin, of the timer.
  - When a preset event (rising edge, falling edge, change) occurs on the input capture pin, the current value of the timer is stored in a register.
  - Example
    - PD6: ICP1 (Timer/Counter1 Input Capture Pin)
- **Output Compare:**
  - A timer usually has a pin designated as output compare pin.
  - When the timer reaches a preset value, the output compare pin can be automatically changed to logic 0 or logic 1.
  - Example
    - PD7: OC2 (Timer/Counter2 Output Compare Match Output)

Overview of Timers in ATmega32

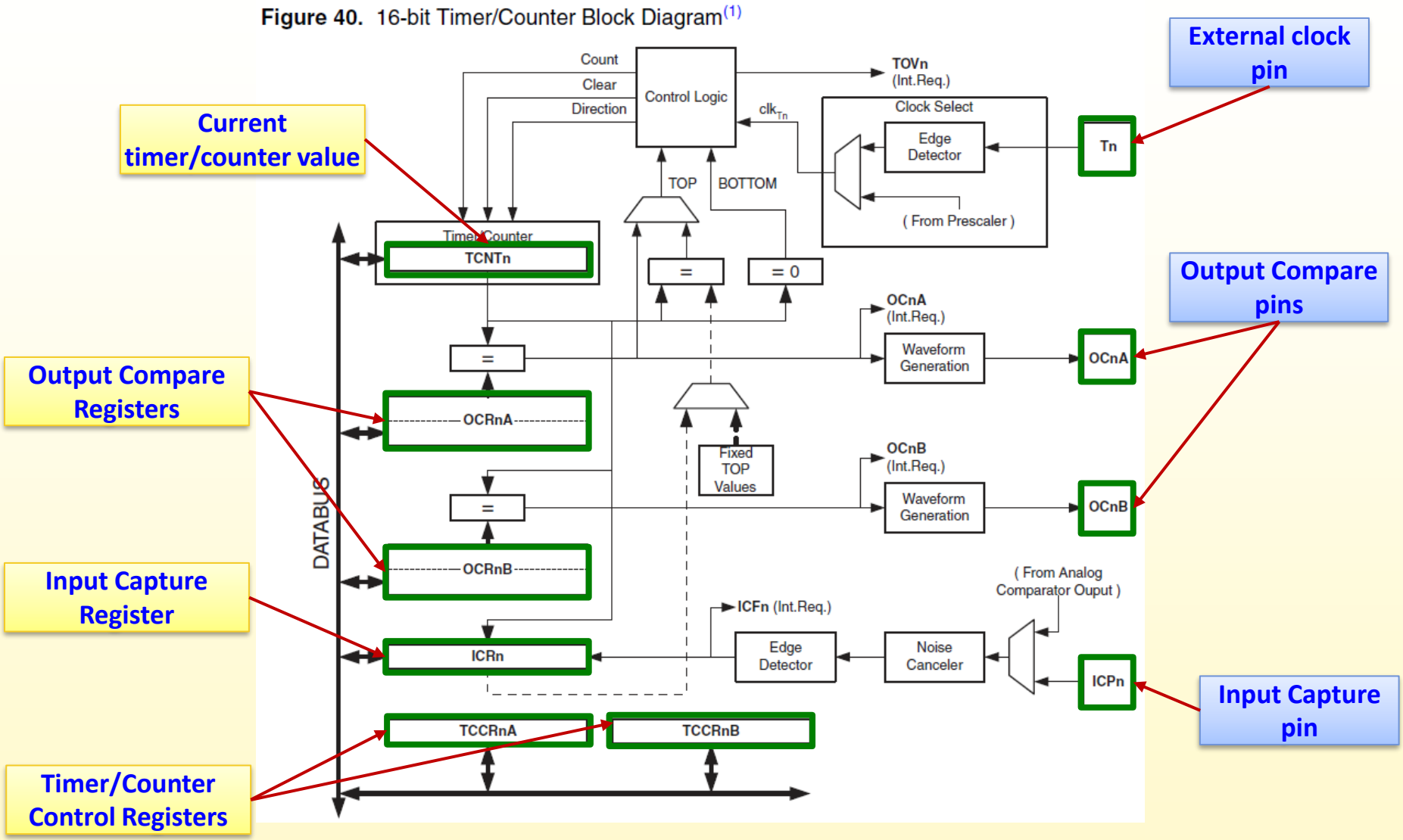
	Timer 0	Timer 1	Timer 2
Overall	- 8-bit counter - 10-bit prescaler	- 16-bit counter - 10-bit prescaler	- 8-bit counter - 10-bit prescaler
Functions	- PWM - Frequency generation - Event counter - Output compare	- PWM - Frequency generation - Event counter - Output compare: 2 channels - Input capture	- PWM - Frequency generation - Event counter - Output compare
Operation modes	- Normal mode - Clear timer on compare match - Fast PWM - Phase correct PWM	- Normal mode - Clear timer on compare match - Fast PWM - Phase correct PWM	- Normal mode - Clear timer on compare match - Fast PWM - Phase correct PWM

➢ **Timer 1 has the most capability among the three timers.**

Timer 1: An overview

- 16-bit counter.
- 10-bit prescaler: 8, 64, 256, 1024
- Can trigger a timer overflow interrupt when counter reaches MAX.
- Can trigger an input capture interrupt when an event occurs on the input capture pin.
  - timer value is stored automatically in a register.
  - input capture pin for Timer 1 is ICP1 (PD.6).
- can trigger an output compare match interrupt when timer reaches a preset value.
- There are two independent output compare channels A and B.

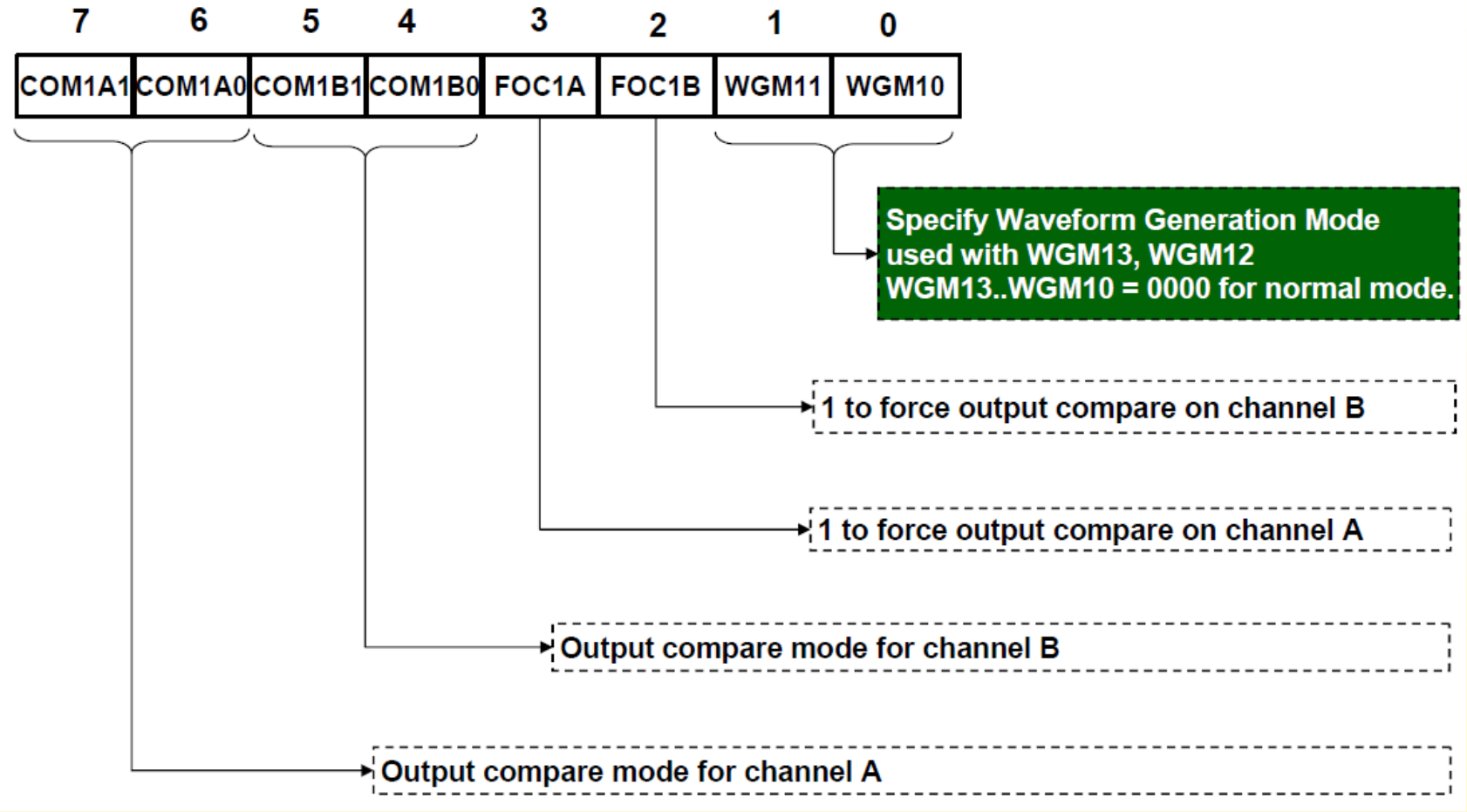
Timer 1: Block diagram



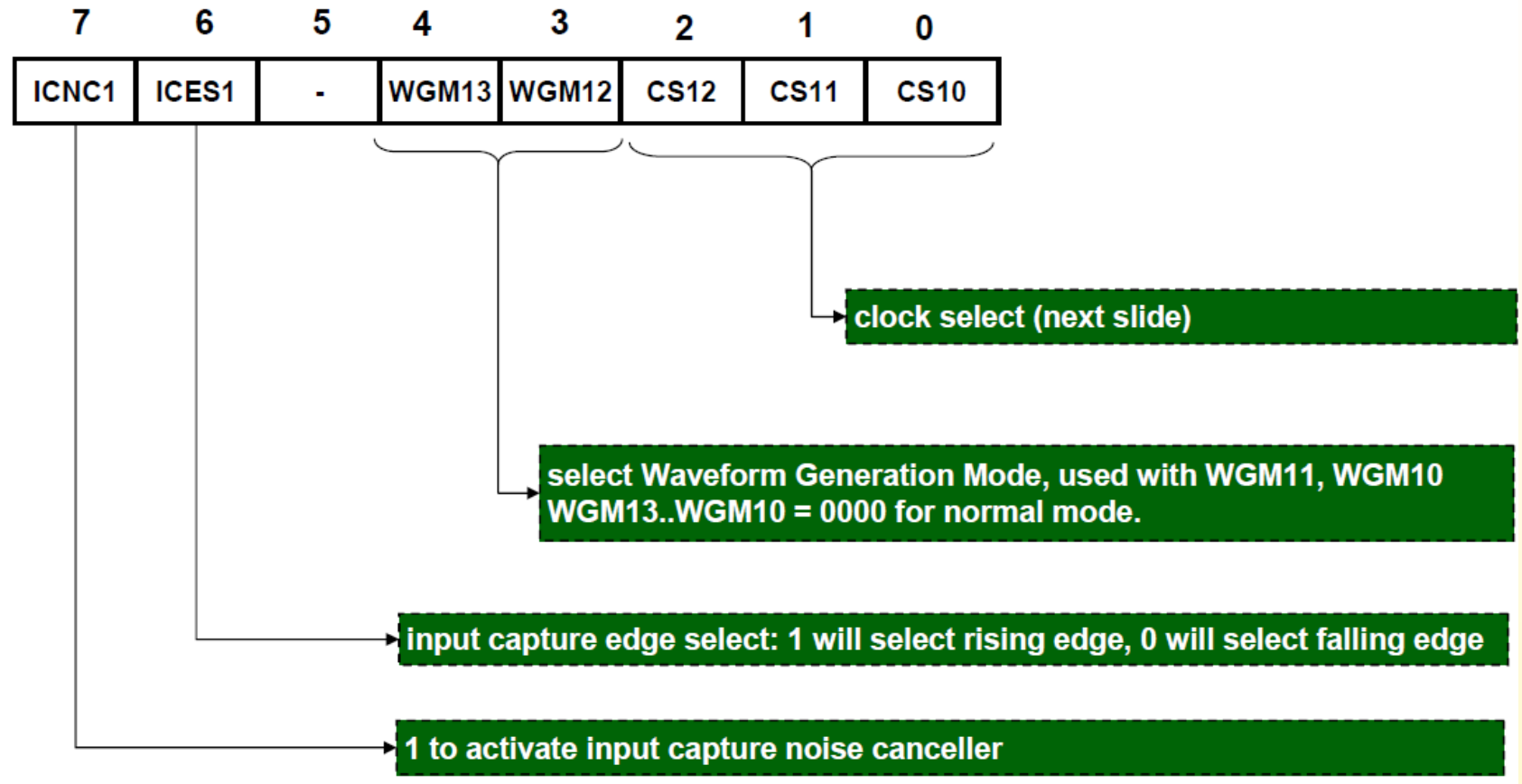
Timer 1 – Five groups of registers

- 1) Timer/Counter 1
  - TCNT1
  - 16-bit register that stores the current value of the timer.
- 2) Timer/Counter 1 Control Registers
  - TCCR1A and TCCR1B
  - To configure the operations of Timer 1.
- 3) Input Capture Register
  - ICR1
  - to store timer value when an event occurs on input capture pin.
- 4) Interrupt registers
  - TIMSK to enable timer interrupts
  - TIFR to monitor status of timer interrupts.
- 5) Output Compare Registers
  - OCR1A, OCR1B
  - To store the preset values for output compare.

Timer/Counter 1 Control Register A (TCCR1A)



Timer/Counter 1 Control Register B (TCCR1B)

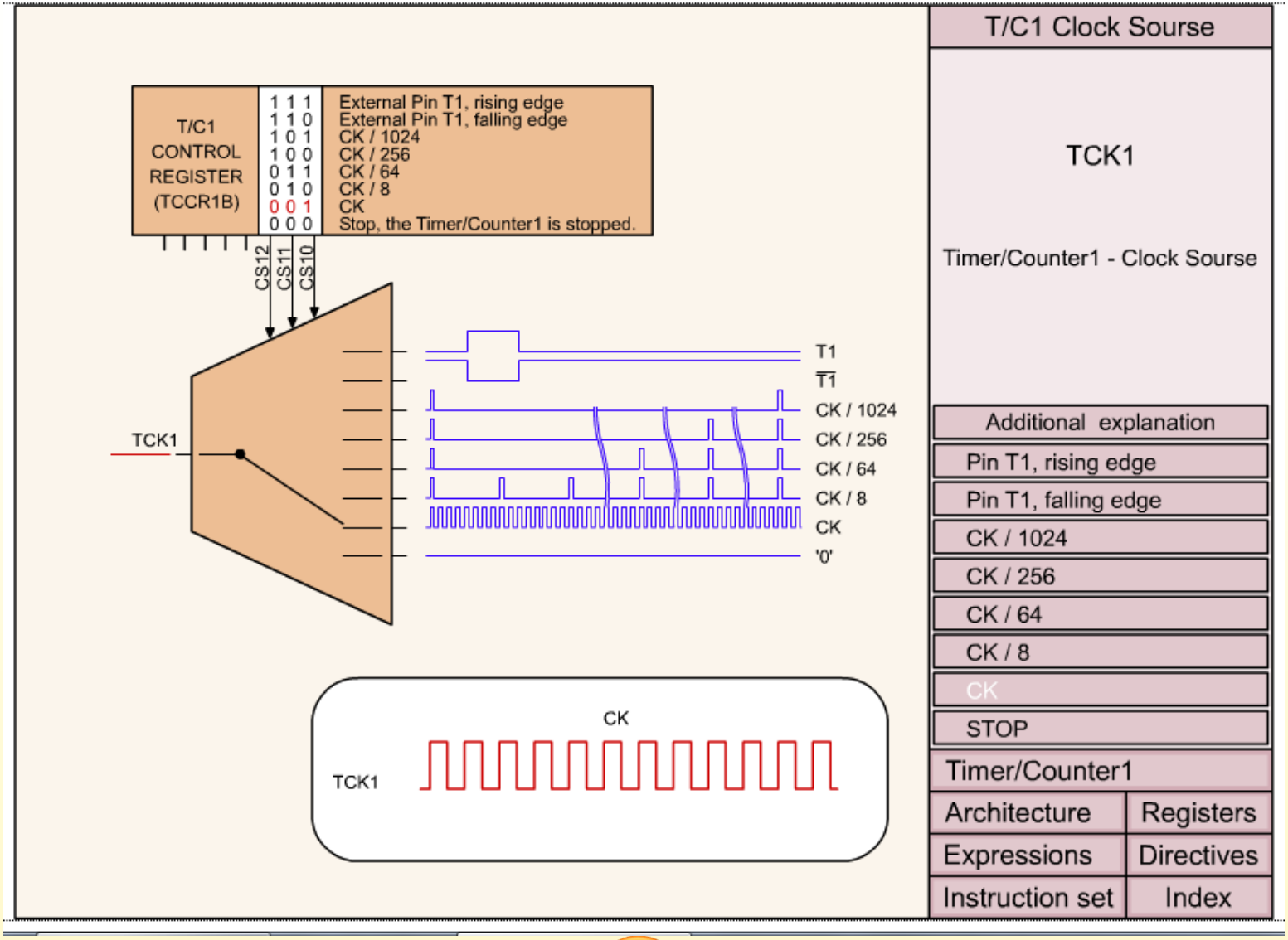


Clock select

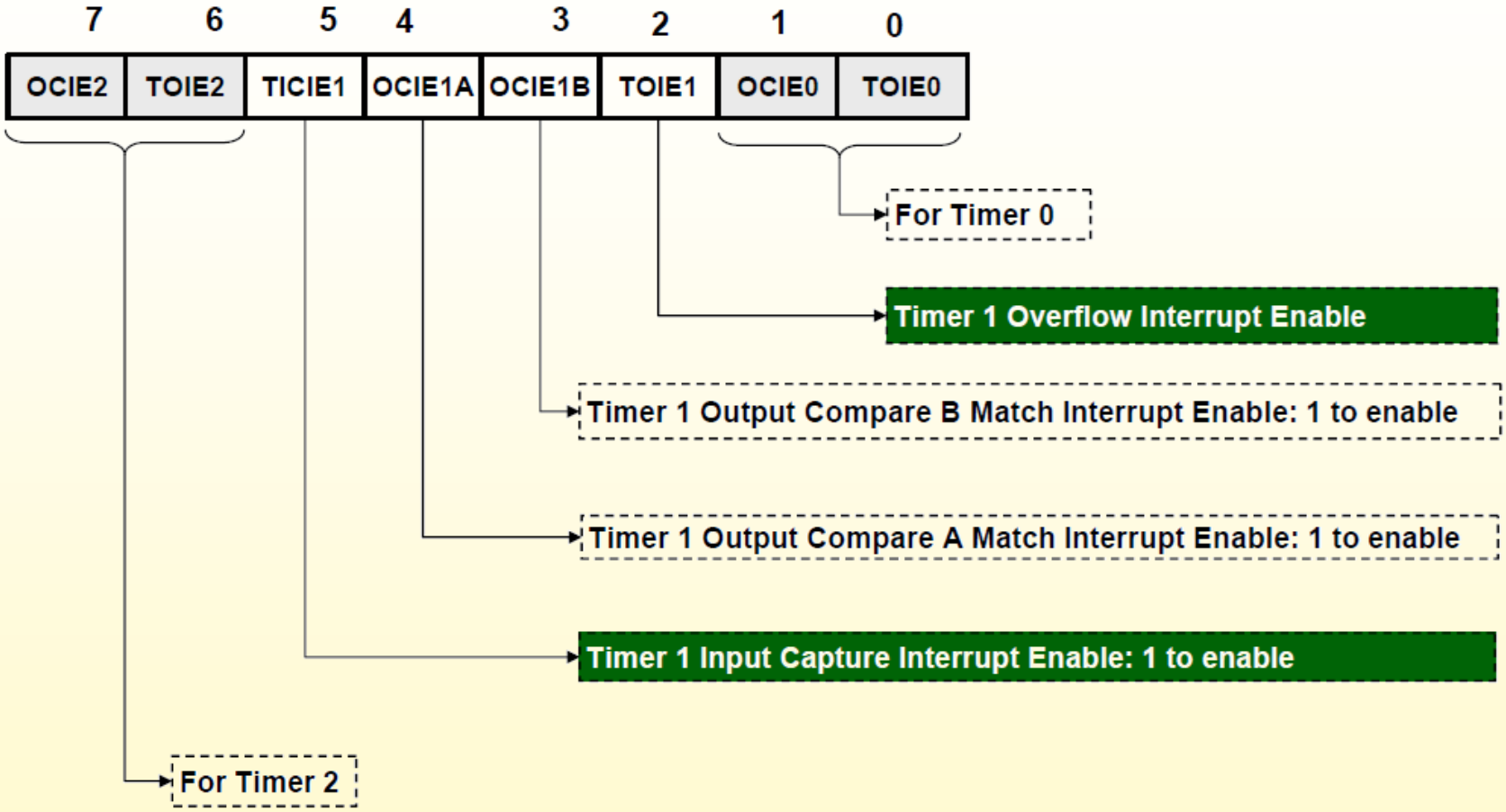
CS12	CS11	CS10	DESCRIPTION
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	**$clk_{I/O}/1$** (No prescaling)
0	1	0	**$clk_{I/O}/8$** (From prescaler)
0	1	1	**$clk_{I/O}/64$** (From prescaler)
1	0	0	**$clk_{I/O}/256$** (From prescaler)
1	0	1	**$clk_{I/O}/1024$** (From prescaler)
1	1	0	**External clock** source on T1 pin. Clock on **FALLING** edge.
1	1	1	**External clock** source on T1 pin. Clock on **RISING** edge.

- **For ATmega32, the internal clock is set by default at $clk_{I/O}$= 1MHz.**
- **Timer 1 can run using the internal or external clock.**
- **If using the internal clock, we can set Timer 1 to run at a speed that is 8, 64, 256 or 1024 times slower than the internal clock.**
- **In Timer 2, prescalers 32 and 128 are also available but External clock is not.**

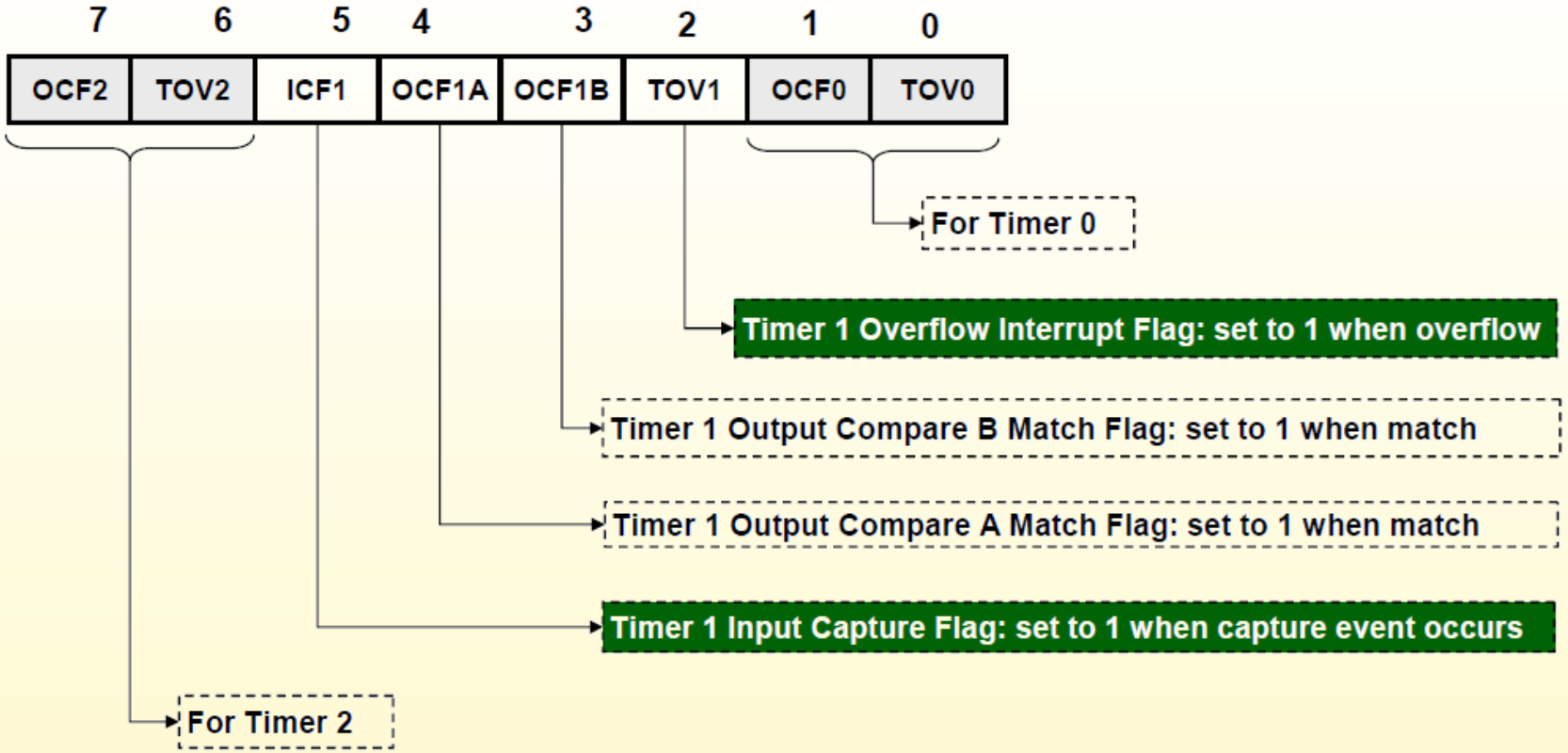
Clock Select



Timer/Counter Interrupt Mask Register (TIMSK)



Timer/Counter Interrupt Flag Register (TIFR)



This register has flags that indicate when a timer interrupt occurs.

نکته

TOVn

- وقتی شمارنده سرریز کند و از مقدار FF به 00 (برای تایمرهای ۸ بیتی) برود پرچم TOVn یک شده و باید به صورت نرم‌افزاری صفر شود.
- موضوع عجیب اینکه برای صفر شدن باید در آن مقدار یک را بنویسیم
  - این قاعده برای تمام پرچمها در AVR صادق است!

Timer applications

- **In this section, we consider three applications of Timer 1.**
  - Creating an accurate delay using timer overflow interrupt.
  - Measuring elapsed time between two events.
  - Measuring the period of a square signal using input capture interrupt.

Example

Create Delay using Timer 0

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

void T0Delay();
int main()
{
    DDRB = 0xFF;
    PORTB = 0xAA;
    while(1){
        PORTB = ~PINB;
        T0Delay();
    }
}

void T0Delay()
{
    TCNT0 = 0x05;//count 0xFF - 0x05 = 250 times
    TCCR0 = 0x01;//normal mode, clk/1
    while ( (TIFR & 0x1) == 0); //wait until overflow
    TCCR0 = 0;//stop timer
    TIFR = 0x1;//writing 1 on a flag will clear it!!!
}
```

Creating an accurate delay

Write a C program for ATmega32 to toggle PORTB every 2 seconds. It should use timer 1 overflow interrupt to create delays of 2s each.

- **Analysis**
- Internal system clock: 1MHz.
- With no prescaler, Timer 1 will increment every 1 µs.
- Because Timer 1 is 16-bit counter, it will overflow every 2^{16}µs (65536 µs).
- To have a 2s delay, we need Timer 1 to overflow for 2s/2^{16} µs = 31 times.

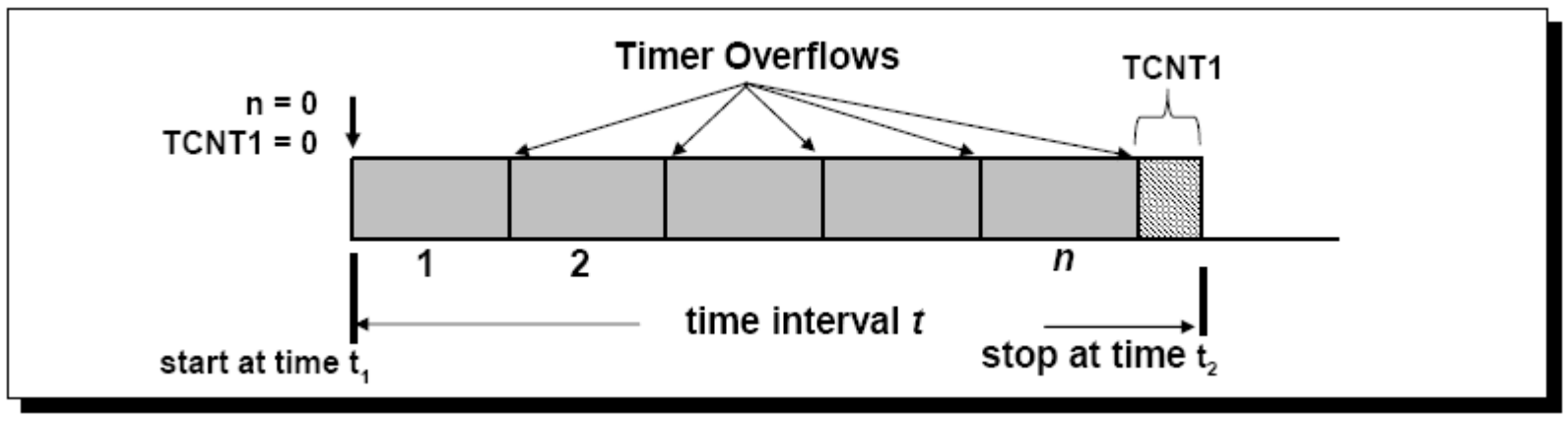
- **Coding**
- Write code to enable & intercept Timer 1 overflow interrupt.
- Use interrupt handler to count the number of overflows.
- When number of overflows = 31, toggle port B.

Creating an accurate delay: timer_delay.c

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

volatile int overflow_count;

ISR(TIMER1_OVF_vect){              // handler for Timer1 overflow interrupt
  overflow_count++;                // increment overflow count
  if (overflow_count >= 31)        // when 2s has passed
  {
    overflow_count = 0;            // start new count
    PORTB = ~PORTB;                // toggle port B
  }
}

int main(void)
{
  DDRB = 0xFF;                     // set port B for output
  PORTB = 0x00;                    // initial value of PORTB
  overflow_count = 0;              // initialise overflow count
  TCCR1A = 0b00000000;             // normal mode
  TCCR1B = 0b00000001;             // no prescaler, internal clock
  TIMSK = 0b00000100;              // enable Timer 1 overflow interrupt
  sei();                           // enable interrupt subsystem globally
  while (1){;}                     // infinite loop
  return 0;
}
```

Measuring elapsed time

- To measure a time interval using Timer 1, we must keep track of both
  - the number of times that Timer 1 has overflowed: n
  - the current counter value: TCNT1
- If we reset n and TCNT1 at the beginning of the interval, then the time elapse is (assuming no prescaler, 1MHz clock)
- t = n x 65536 + TCNT1 (μs)



Measuring elapsed time

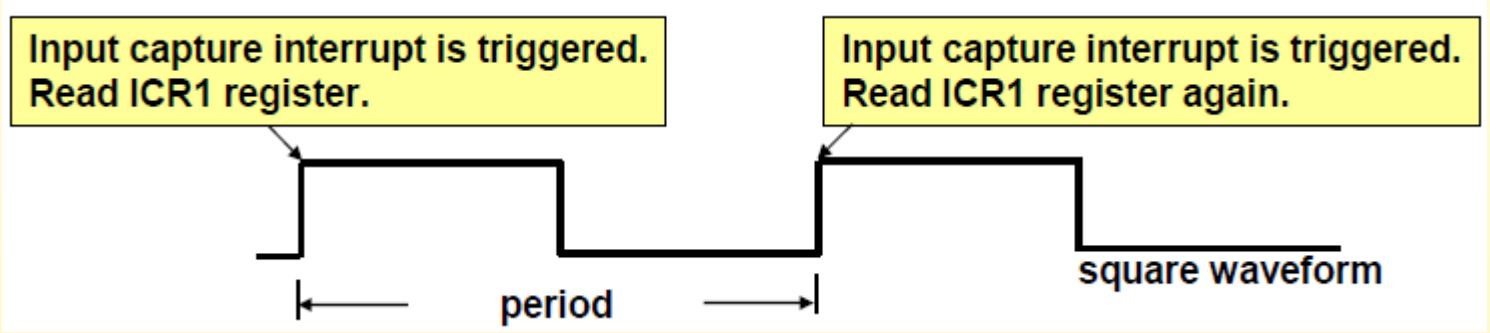
- **Use Timer 1 to measure the execution time of some custom C code.**
- **Approach:**
  - **Clear Timer 1 when the code starts.**
  - **Record Timer 1 when the code finishes.**
  - **Also, use Timer 1 Overflow Interrupt to keep track of how many times it has overflowed.**

Measuring elapsed time: measure_time.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>
volatile uint32_t n;
ISR(TIMER1_OVF_vect){     // handler for Timer1 overflow interrupt
    n++;                  // increment overflow count
}
int main(void) {
   int i, j;
   uint32_t elapse_time;

   TCCR1A = 0b00000000; // normal mode
   TCCR1B = 0b00000001; // no prescaler, internal clock
   TIMSK  = 0b00000100; // enable Timer 1 overflow interrupt
   n = 0;               // reset n
   TCNT1 = 0;           // reset Timer 1
   sei();               // enable interrupt subsystem globally
   // -----  start code --------------
   for (i = 0; i < 100; i++)
    for (j = 0; j < 1000; j++){;}
   // -----  end code ----------------
   elapse_time = n * 65536 + TCNT1;
   cli();               // disable interrupt subsystem globally
   return 0;
}
```

Measuring period of a square signal

- How can we measure the period of a square signal?
- Use Timer 1 input capture interrupt to measure the period of a square signal.
- **Analysis:**
  - **The period of a square wave = the time difference between two consecutive rising edges.**
  - **Connect the square wave to input capture pin of Timer 1.**
  - **Configure input capture module to trigger on a rising edge.**



Measuring period of a square signal

- **Assumption**:
  - The input signal has a high frequency, hence timer overflow can be ignored.
  - Can anyone explain why? and what is the safe range?

- **Implementation**:
  - **Select timer operations**: normal, no prescaler, internal clock 1MHz, noise canceller enabled, input capture for rising edges. ±
    - `TCCR1A = 0b00000000;`
    - `TCCR1B = 0b11000001;`
  - **Enable input capture interrupt**:
    - `TIMSK = 0b00100000`

Measure Period

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>
uint16_t period;
ISR(TIMER1_CAPT_vect){ // handler for Timer1 input capture interrupt
   period = ICR1; // period = value of Timer 1 stored in ICR1
   TCNT1 = 0; // reset Timer 1
}
int main(void) {
  // setup LCD
  TCCR1A = 0b00000000; // normal mode
  TCCR1B = 0b11000001; // no prescaler, rising edge, noise canceller
  TIMSK = 0b00100000; // enable Timer 1 input capture interrupt
  sei(); // enable interrupt subsystem globally
  while (1){
     // display period on LCD
  } // infinite loop
  return 0;
}
```

Run Period_meter_codevision