Mechatronics Engineering

Tutorial #4

DC, Servo & Stepper Motors
Input Capture Module

Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

Tutorial Contents

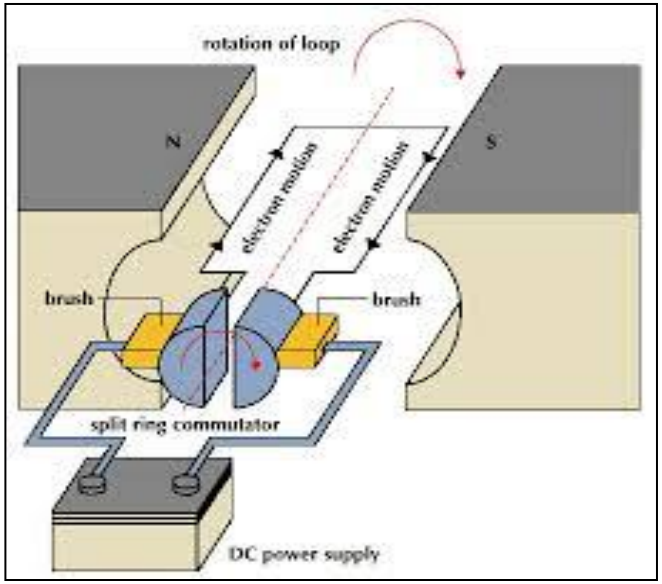
- DC, Servo & Stepper Motors
- Input Capture Module in ATmega328P

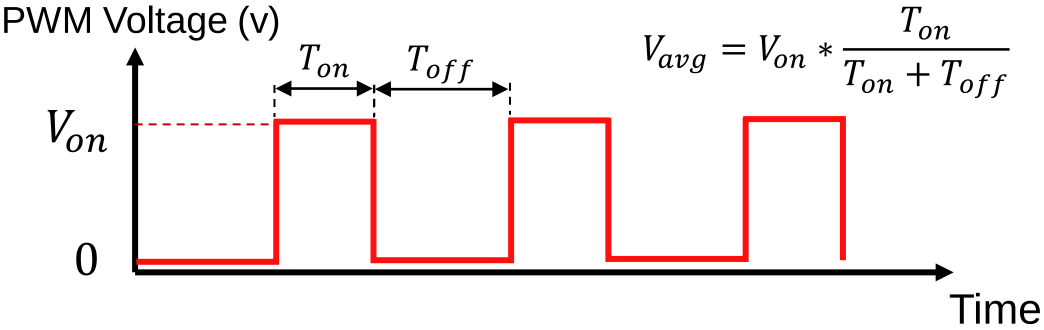
Tutorial Contents

- **DC, Servo & Stepper Motors**
- Input Capture Module in ATmega328P

DC, Servo & Stepper Motors

- DC Motors:
  - Continuous Rotation motors
  - When you supply power, a DC motors rotates until that power is removed
  - The speed of DC motors is controlled using pulse width modulation (PWM)

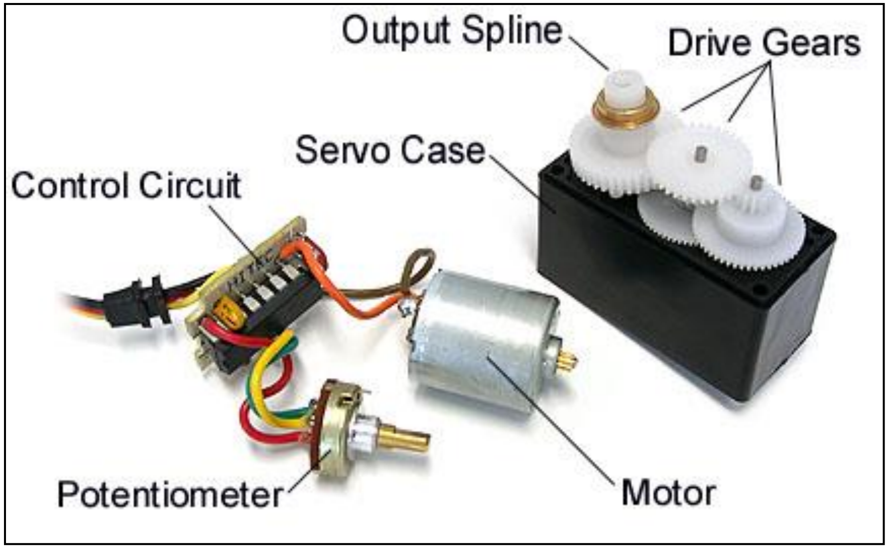




$$V_{avg} = V_{on} * \frac{T_{on}}{T_{on} + T_{off}}$$

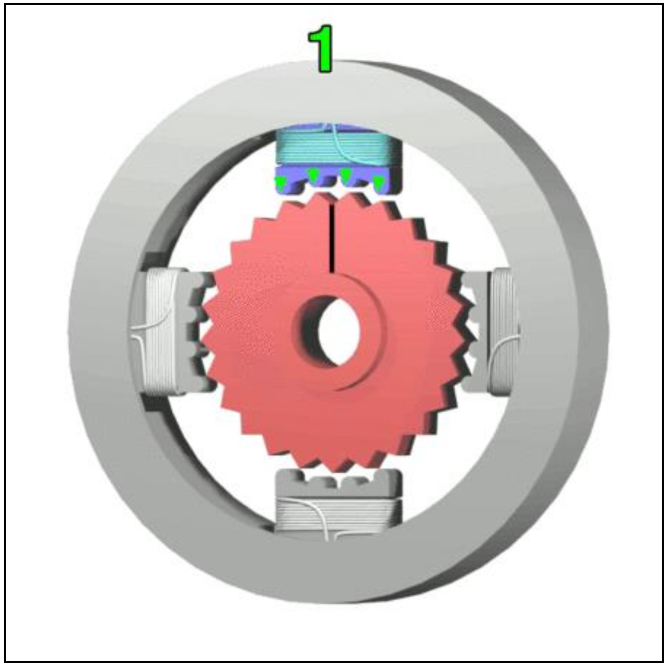
DC, Servo & Stepper Motors

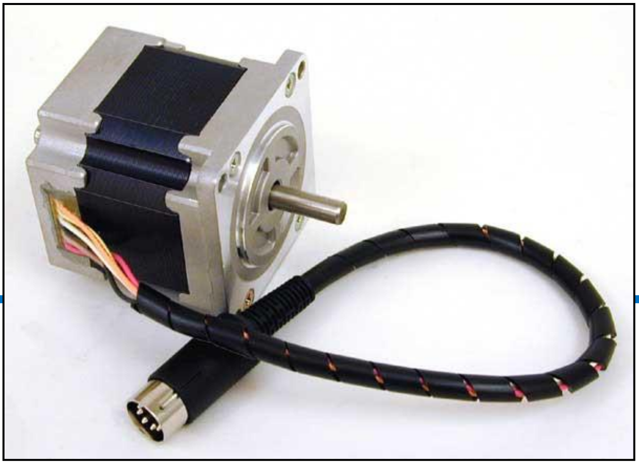
- Servo Motors:
  - It is an assembly of:
    - DC motor
    - A gearing set
    - A Control circuit
    - A Position sensor (usually potentiometer)
  - the angle of rotation is limited to ~ 180 degrees
  - have three wires (Control, Power & Ground)



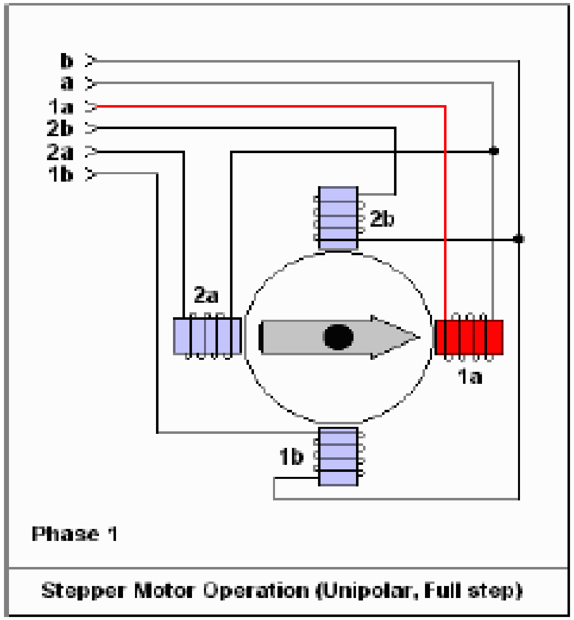
DC, Servo & Stepper Motors

- Stepper Motors:
  - Consist of multiple toothed electromagnets arranged around a central gear to define position
  - Stepper motors require an external control circuit or micro controller to individually energize each electromagnet and make the motor shaft turn

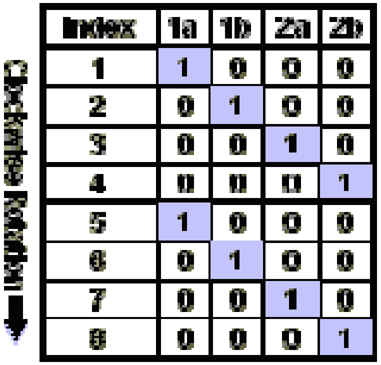




Stepper Motors

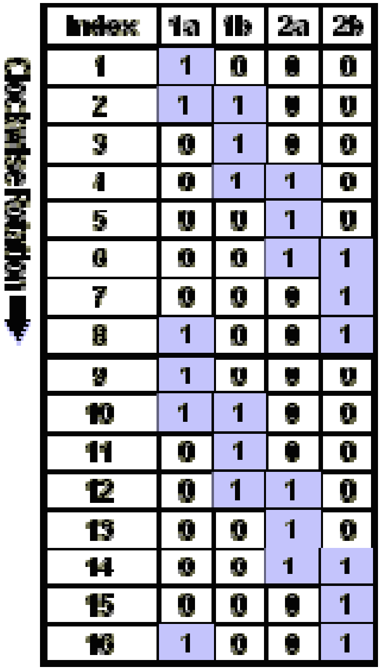


Stepper Motor Operation (Unipolar, Full step)

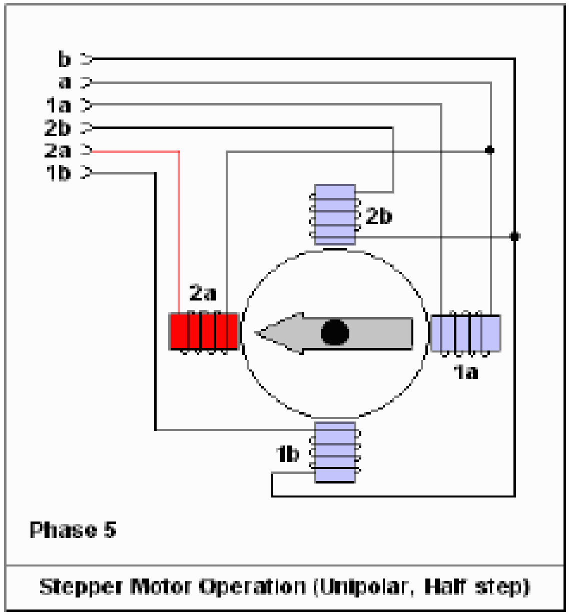


Index	1a	1b	2a	2b
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	1	0	0	0
6	0	1	0	0
7	0	0	1	0
8	0	0	0	1

Full step sequence showing how binary numbers can control the motor

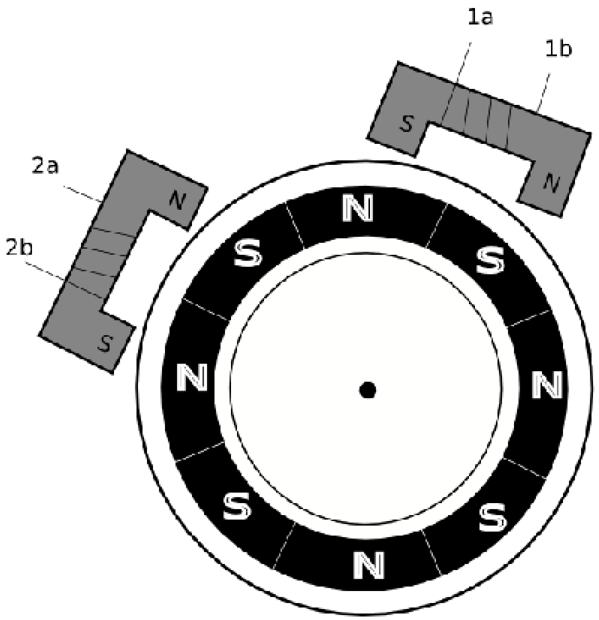


Index	1a	1b	2a	2b
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1
9	1	0	0	0
10	1	1	0	0
11	0	1	0	0
12	0	1	1	0
13	0	0	1	0
14	0	0	1	1
15	0	0	0	1
16	1	0	0	1



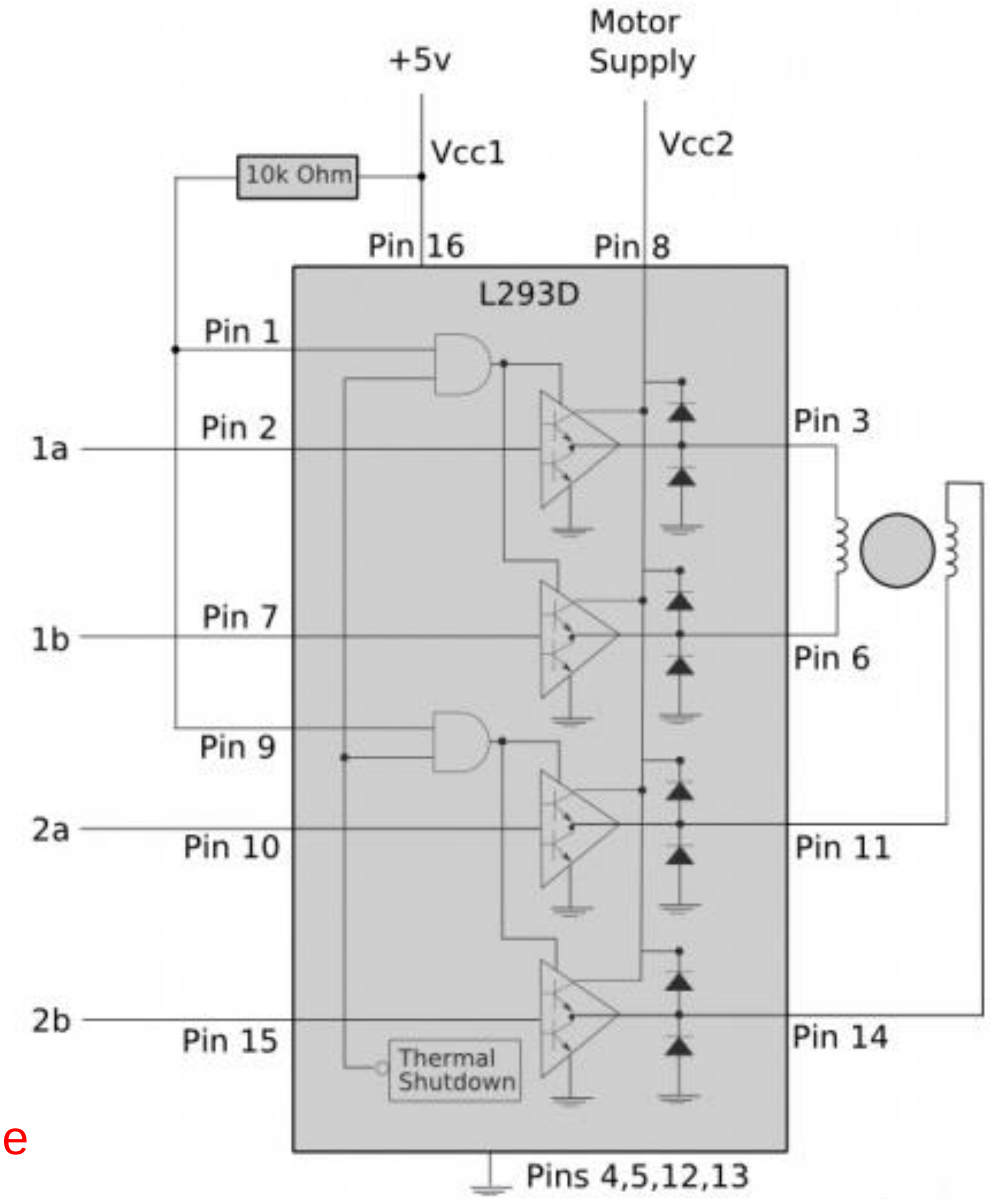
Stepper Motor Operation (Unipolar, Half step)

- Half step sequence of binary control numbers



Step Angle	Steps per Revolution
0.72	500
1.8	200
2.0	180
2.5	144
5.0	72
7.5	48
15	24

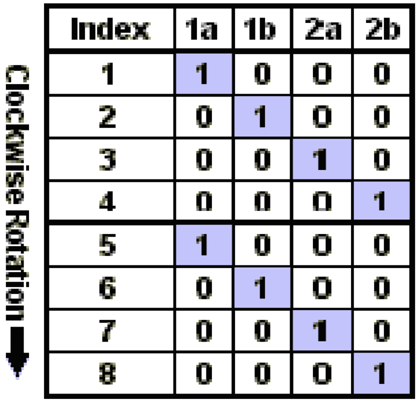
Stepper motors with different step angle



Problem 1:

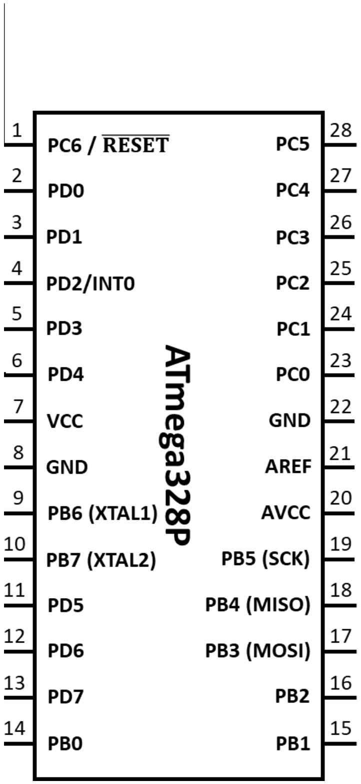
Construct the hardware connection and the C code (for an ATmega328P Microcontroller) that operates a stepper motor (connected to PORTC Lower bits) in clockwise and counterclockwise upon the state of two active high switches connected to PD0 and PD4 pins

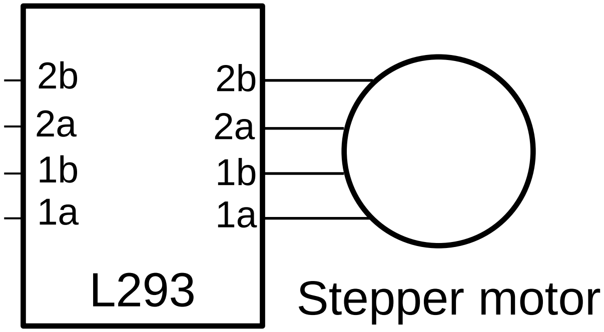
PD0	PD4	Operation
1	0	C.W.
0	1	C.C.W.



Clockwise Rotation

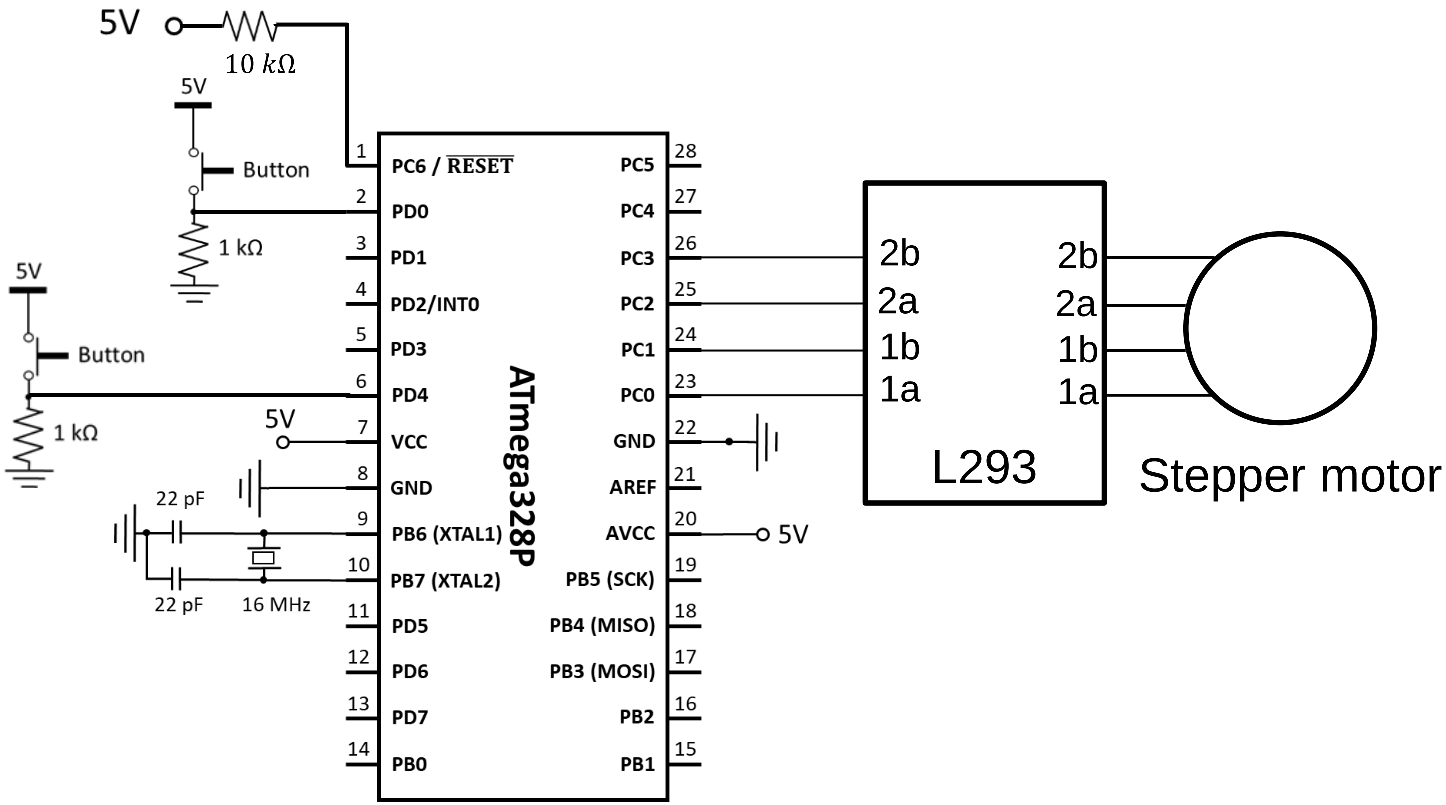
Index	1a	1b	2a	2b
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	1	0	0	0
6	0	1	0	0
7	0	0	1	0
8	0	0	0	1





PC3	PC2	PC1	PC0
2b	2a	1b	1a

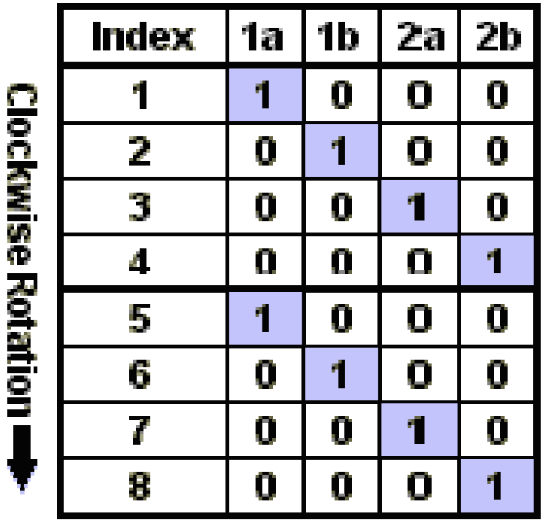
Solution:


5V
10 kΩ
5V
Button
1 kΩ
5V
Button
1 kΩ
5V
22 pF
22 pF
16 MHz
1 PC6 / RESET
2 PD0
3 PD1
4 PD2/INT0
5 PD3
6 PD4
7 VCC
8 GND
9 PB6 (XTAL1)
10 PB7 (XTAL2)
11 PD5
12 PD6
13 PD7
14 PB0
28 PC5
27 PC4
26 PC3
25 PC2
24 PC1
23 PC0
22 GND
21 AREF
20 AVCC
19 PB5 (SCK)
18 PB4 (MISO)
17 PB3 (MOSI)
16 PB2
15 PB1
5V
ATmega328P
2b
2a
1b
1a
2b
2a
1b
1a
L293
Stepper motor

Code:

```
#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRD &= 0b11101110; // set PD0 and PD4 as inputs
    DDRC |= 0b00001111; // set PC0-PC3 as outputs

    while(1)
    {
        if(PIND & 0b00000001) // PD0 pressed -> clockwise rotation
        {
            PORTC = 0b00010001;
            _delay_ms(100);
            PORTC = PORTC << 1;
            _delay_ms(100);
            PORTC = PORTC << 1;
            _delay_ms(100);
            PORTC = PORTC << 1;
            _delay_ms(100);

        }else if(PIND & 0b00010000)// PD4 pressed -> counterclockwise rotation
        {
            PORTC = 0b10001000;
            _delay_ms(100);
            PORTC = PORTC >> 1;
            _delay_ms(100);
            PORTC = PORTC >> 1;
            _delay_ms(100);
            PORTC = PORTC >> 1;
            _delay_ms(100);
        }
    }
}
```

PC3	PC2	PC1	PC0
2b	2a	1b	1a



Clockwise Rotation

Index	1a	1b	2a	2b
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	1	0	0	0
6	0	1	0	0
7	0	0	1	0
8	0	0	0	1

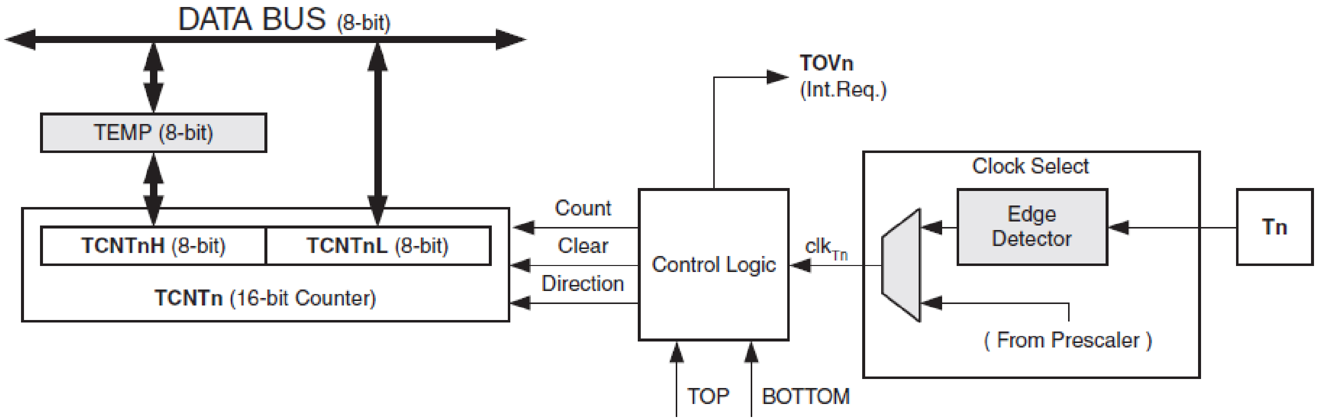
Tutorial Contents

- DC, Servo & Stepper Motors
- **Timer 1 Input Capture Module in ATmega328P**

Timer 1 Input Capture in ATmega328P:

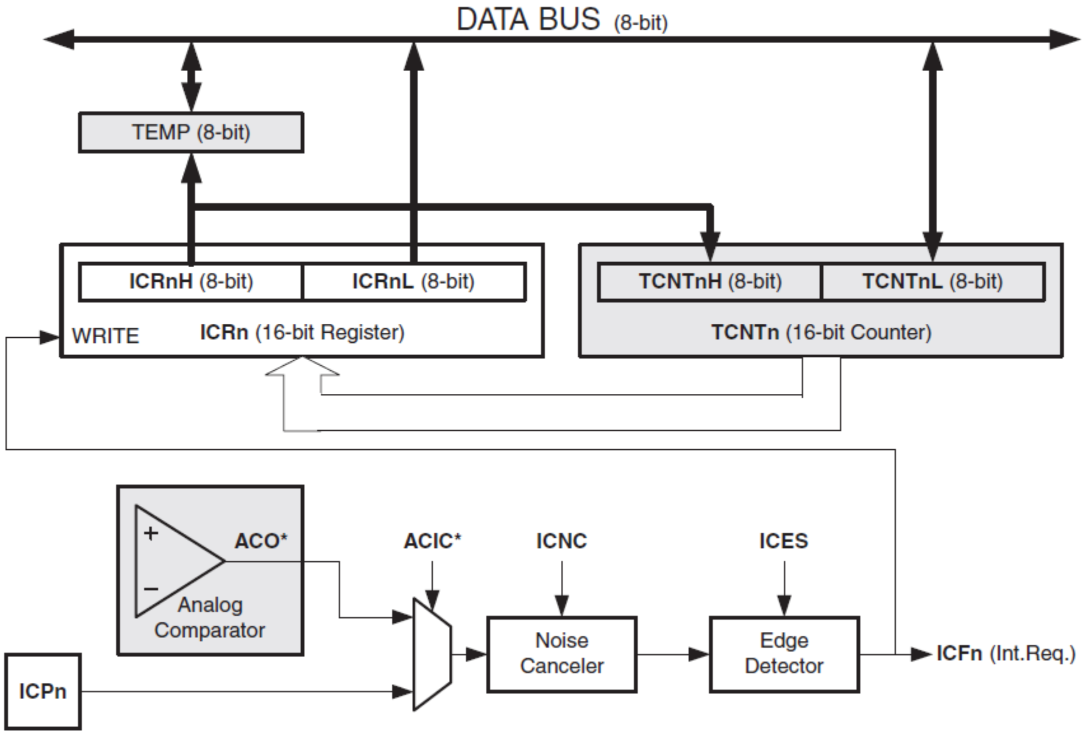
- Timer 1: 16-bit timer. Can count up to 65535!
- A 16-bit register values should be stored in a datatype of int or larger.
- The "C" Compiler handles the 16-bit registers access. For example, we can access TCNT1 directly in "C":

```
unsigned int CurrentCount;
CurrentCount = TCNT1;
```



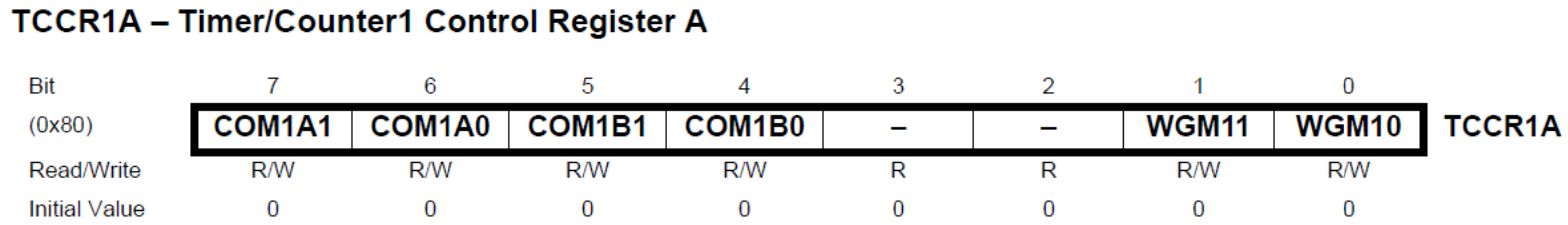
Timer 1 Input Capture in ATmega328P:

- Input Capture Module in ATmega328P:
- When an Edge is detected, the current count of TCNT1 is written into the ICR1 register.



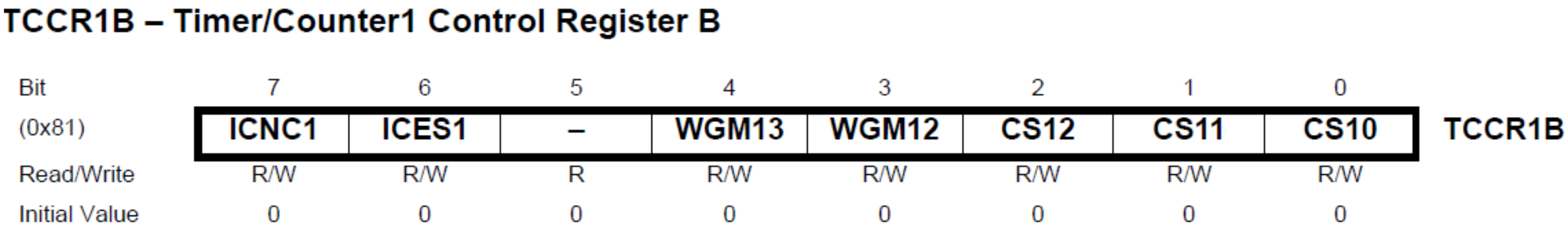
Timer 1 Input Capture in ATmega328P:

- Timer 1 Configuration Registers:



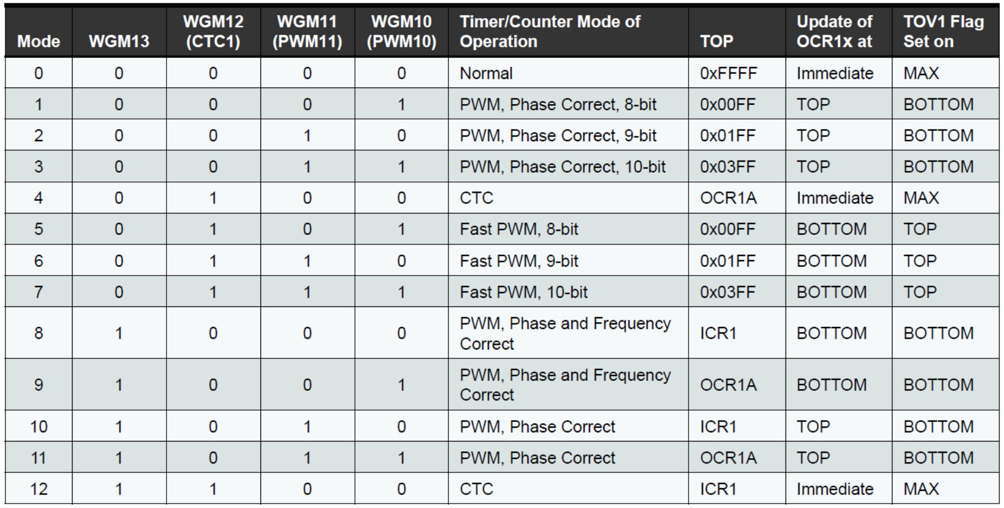
TCCR1A – Timer/Counter1 Control Register A

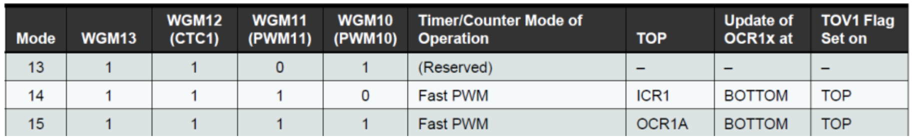
Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

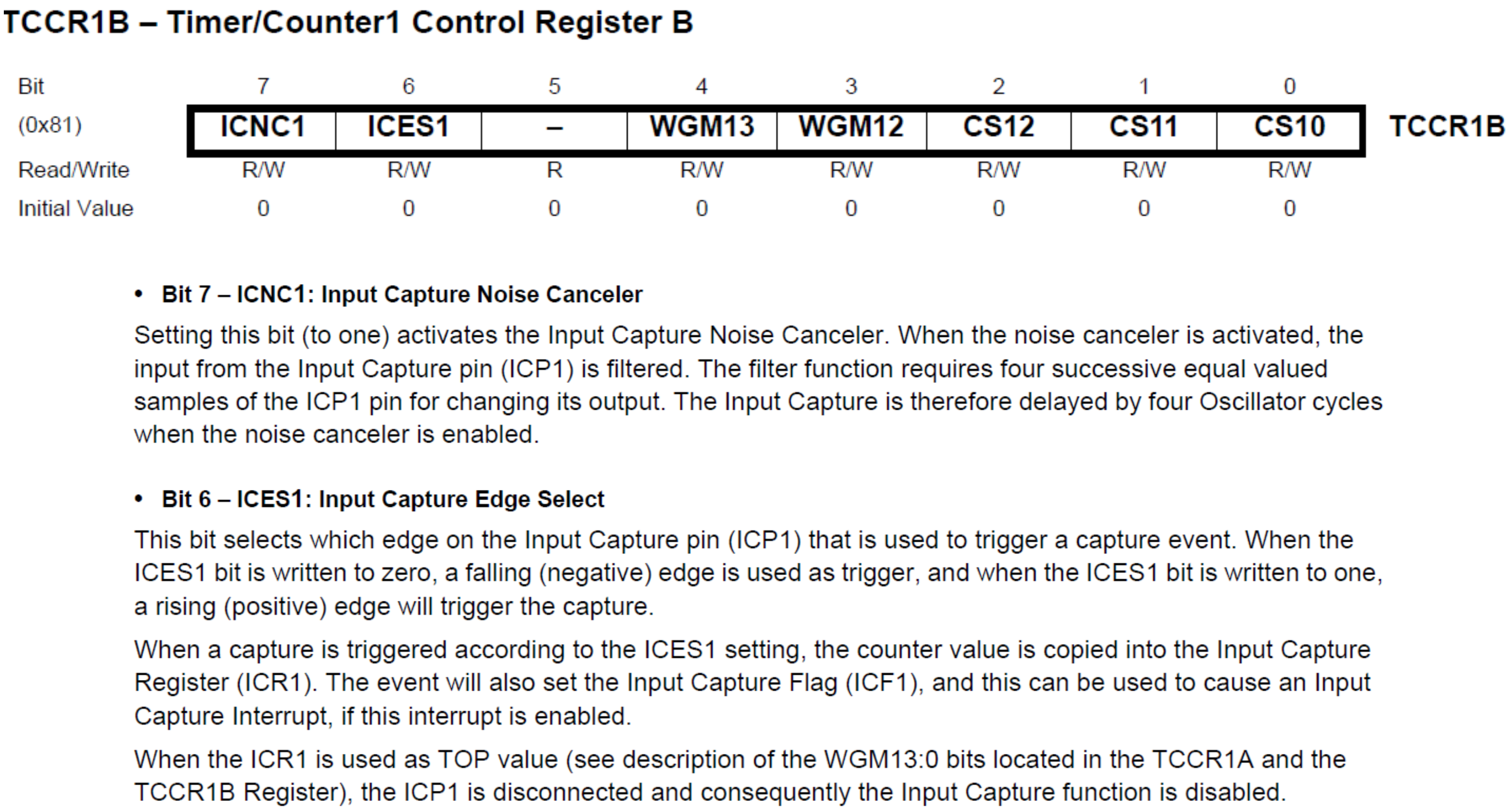




Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	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

Timer 1 Input Capture in ATmega328P:

- Timer 1 Configuration Registers:



TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ICNC1: Input Capture Noise Canceler**

Setting this bit (to one) activates the Input Capture Noise Canceler. When the noise canceler is activated, the input from the Input Capture pin (ICP1) is filtered. The filter function requires four successive equal valued samples of the ICP1 pin for changing its output. The Input Capture is therefore delayed by four Oscillator cycles when the noise canceler is enabled.

- **Bit 6 – ICES1: Input Capture Edge Select**

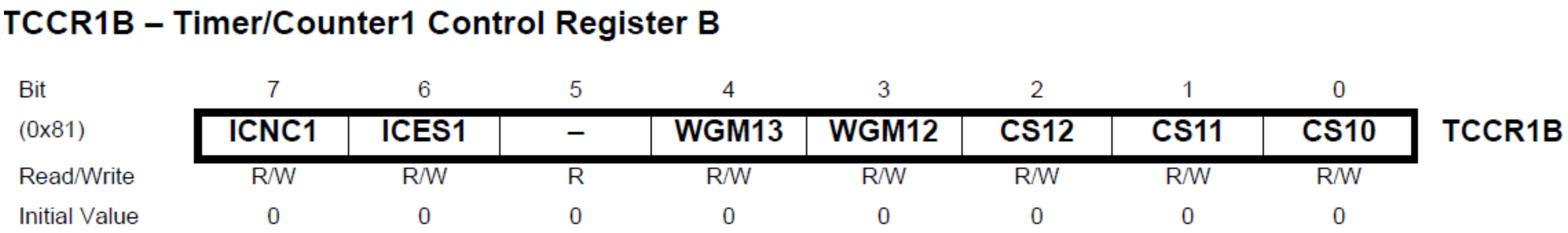
This bit selects which edge on the Input Capture pin (ICP1) that is used to trigger a capture event. When the ICES1 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICES1 bit is written to one, a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICES1 setting, the counter value is copied into the Input Capture Register (ICR1). The event will also set the Input Capture Flag (ICF1), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled.

When the ICR1 is used as TOP value (see description of the WGM13:0 bits located in the TCCR1A and the TCCR1B Register), the ICP1 is disconnected and consequently the Input Capture function is disabled.

Timer 1 Input Capture in ATmega328P:

- Timer 1 Configuration Registers:



TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

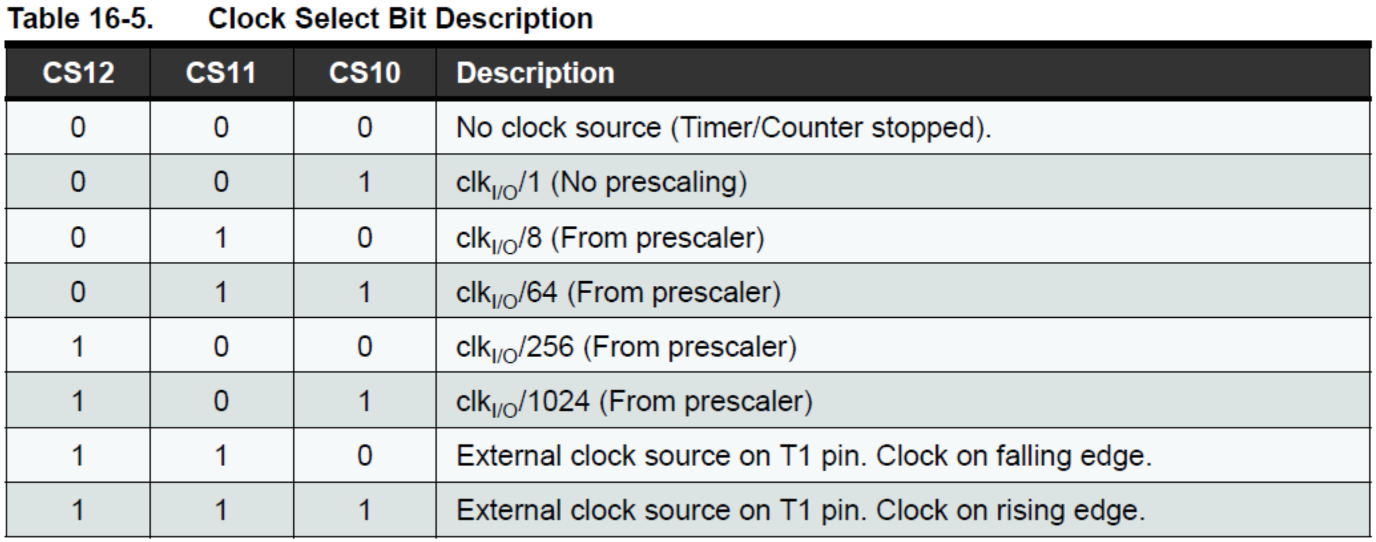


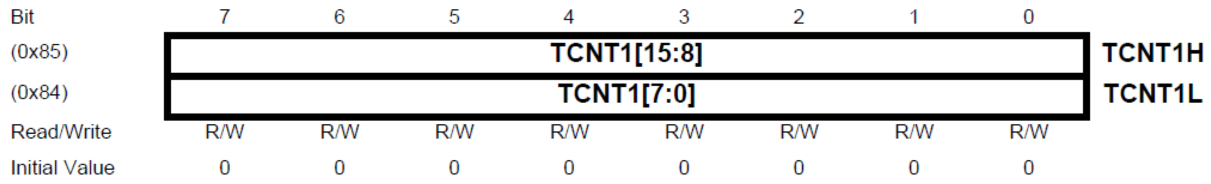
Table 16-5. Clock Select Bit Description

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.

Timer 1 Input Capture in ATmega328P:

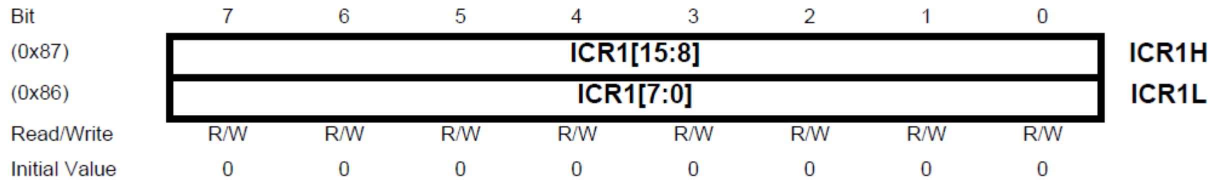
- Timer 1 Registers:

TCNT1H and TCNT1L – Timer/Counter1



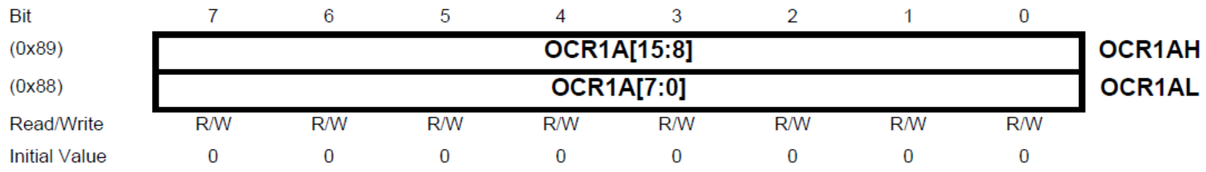
Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ICR1H and ICR1L – Input Capture Register 1



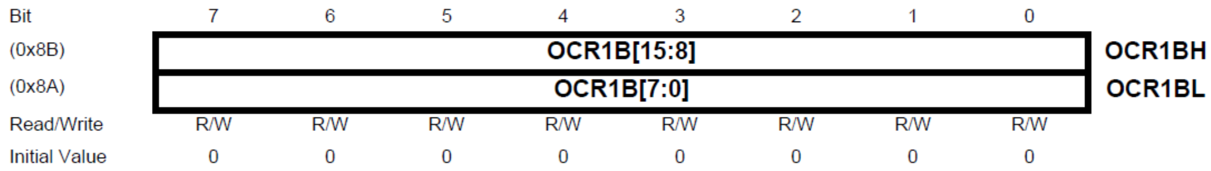
Bit	7	6	5	4	3	2	1	0	
(0x87)	ICR1[15:8]								ICR1H
(0x86)	ICR1[7:0]								ICR1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

OCR1AH and OCR1AL – Output Compare Register 1 A



Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A[15:8]								OCR1AH
(0x88)	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

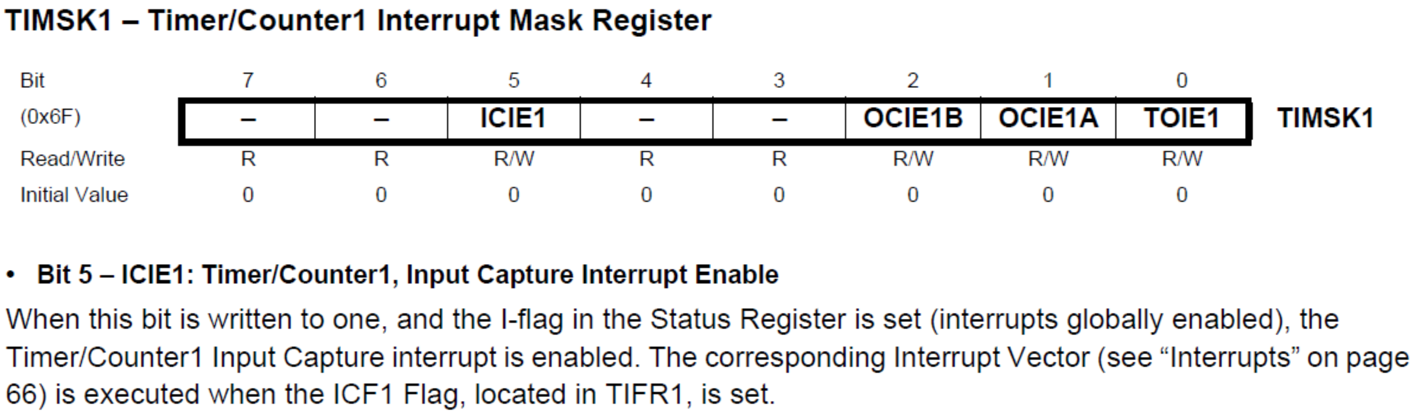
OCR1BH and OCR1BL – Output Compare Register 1 B



Bit	7	6	5	4	3	2	1	0	
(0x8B)	OCR1B[15:8]								OCR1BH
(0x8A)	OCR1B[7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer 1 Input Capture in ATmega328P:

- Timer 1 Interrupt Enable Register:

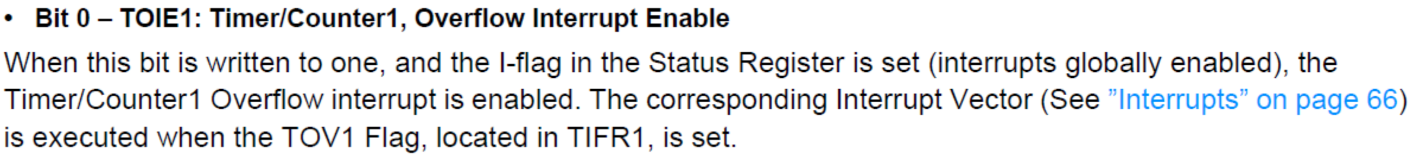


TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 5 – ICIE1: Timer/Counter1, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Input Capture interrupt is enabled. The corresponding Interrupt Vector (see “Interrupts” on page 66) is executed when the ICF1 Flag, located in TIFR1, is set.

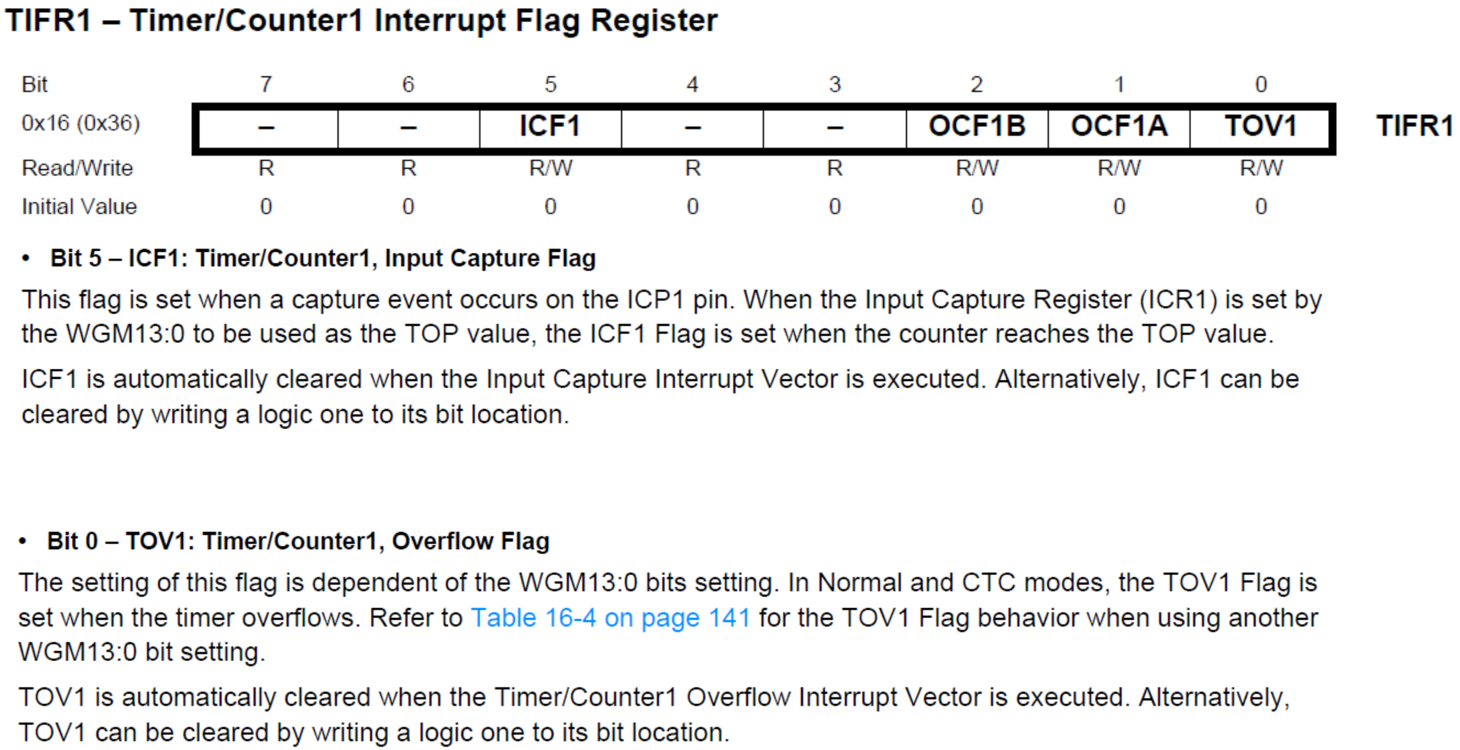


- **Bit 0 – TOIE1: Timer/Counter1, Overflow Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Overflow interrupt is enabled. The corresponding Interrupt Vector (See ”Interrupts” on page 66) is executed when the TOV1 Flag, located in TIFR1, is set.

Timer 1 Input Capture in ATmega328P:

- Timer 1 Interrupt Flag Register:



TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	**ICF1**	–	–	**OCF1B**	**OCF1A**	**TOV1**	**TIFR1**
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 5 – ICF1: Timer/Counter1, Input Capture Flag**

This flag is set when a capture event occurs on the ICP1 pin. When the Input Capture Register (ICR1) is set by the WGM13:0 to be used as the TOP value, the ICF1 Flag is set when the counter reaches the TOP value.

ICF1 is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICF1 can be cleared by writing a logic one to its bit location.

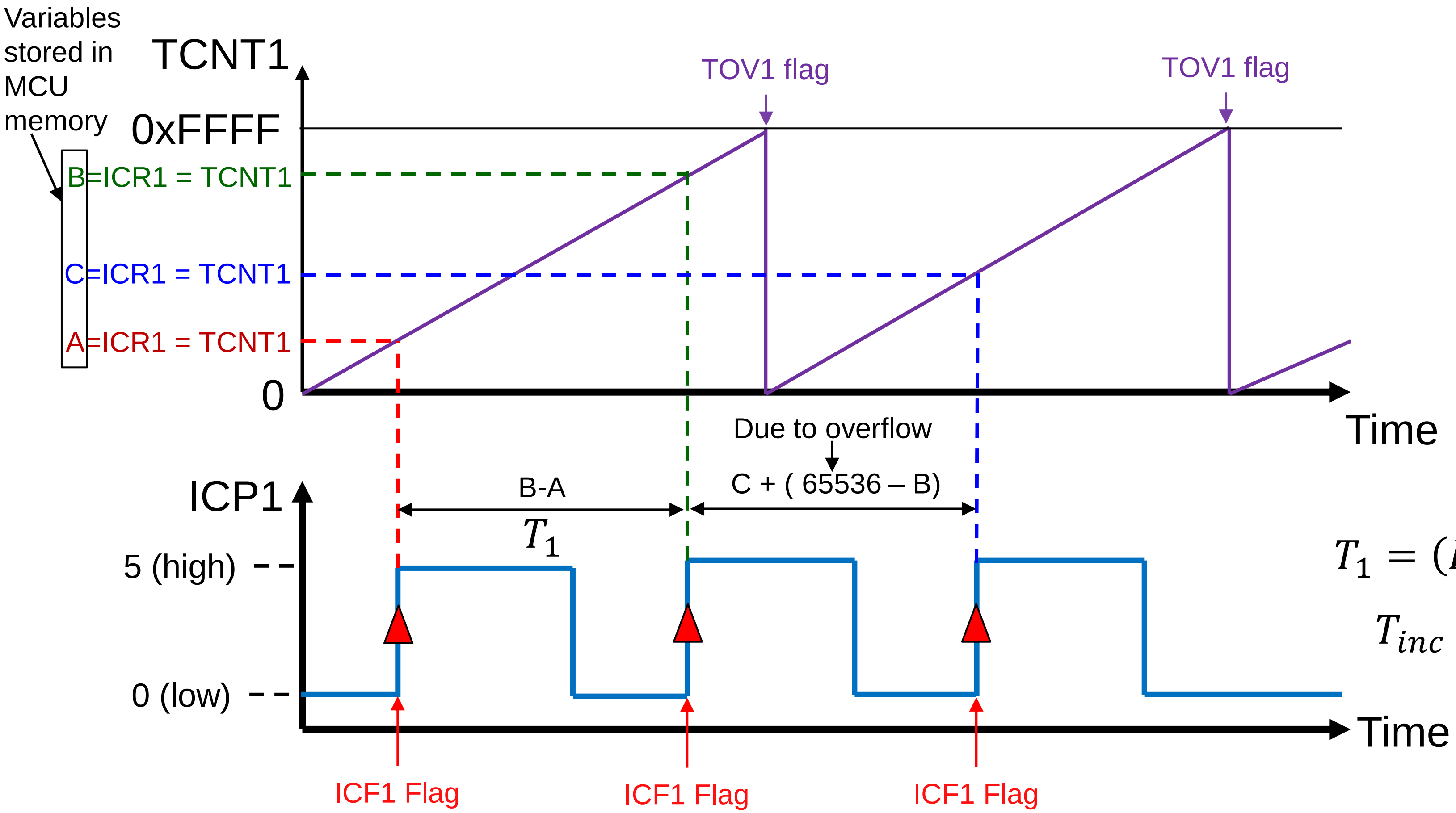
- **Bit 0 – TOV1: Timer/Counter1, Overflow Flag**

The setting of this flag is dependent of the WGM13:0 bits setting. In Normal and CTC modes, the TOV1 Flag is set when the timer overflows. Refer to Table 16-4 on page 141 for the TOV1 Flag behavior when using another WGM13:0 bit setting.

TOV1 is automatically cleared when the Timer/Counter1 Overflow Interrupt Vector is executed. Alternatively, TOV1 can be cleared by writing a logic one to its bit location.

Timer 1 Input Capture in ATmega328P:

- Example of using Timer 1 (normal mode) to capture period of a signal:
- ICES1 =1 (write to ICR1 on rising edge of ICP1)



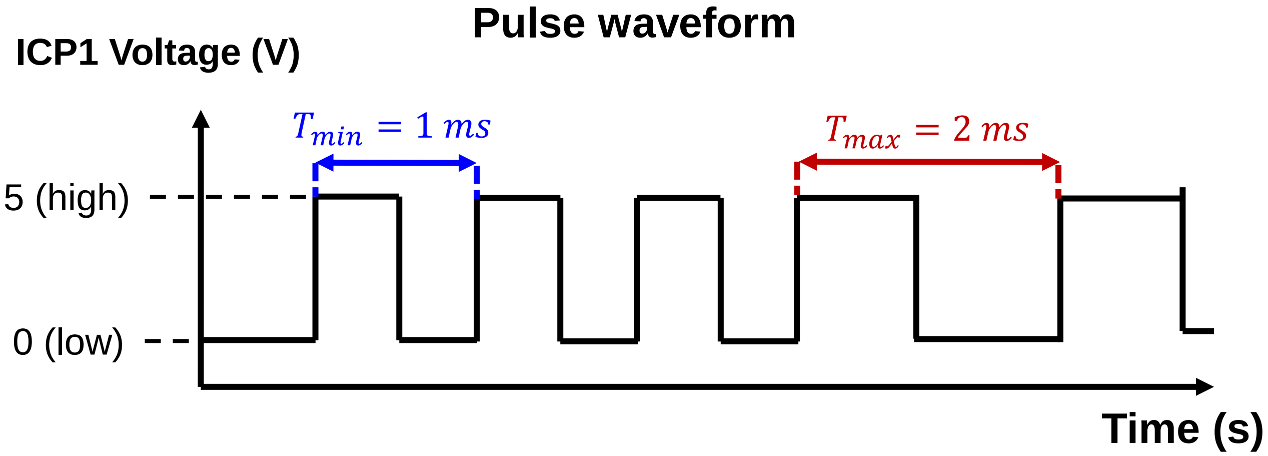
$$T_1 = (B - A) * T_{inc}$$

$$T_{inc} = N * T_{clk}$$

Problem 2:

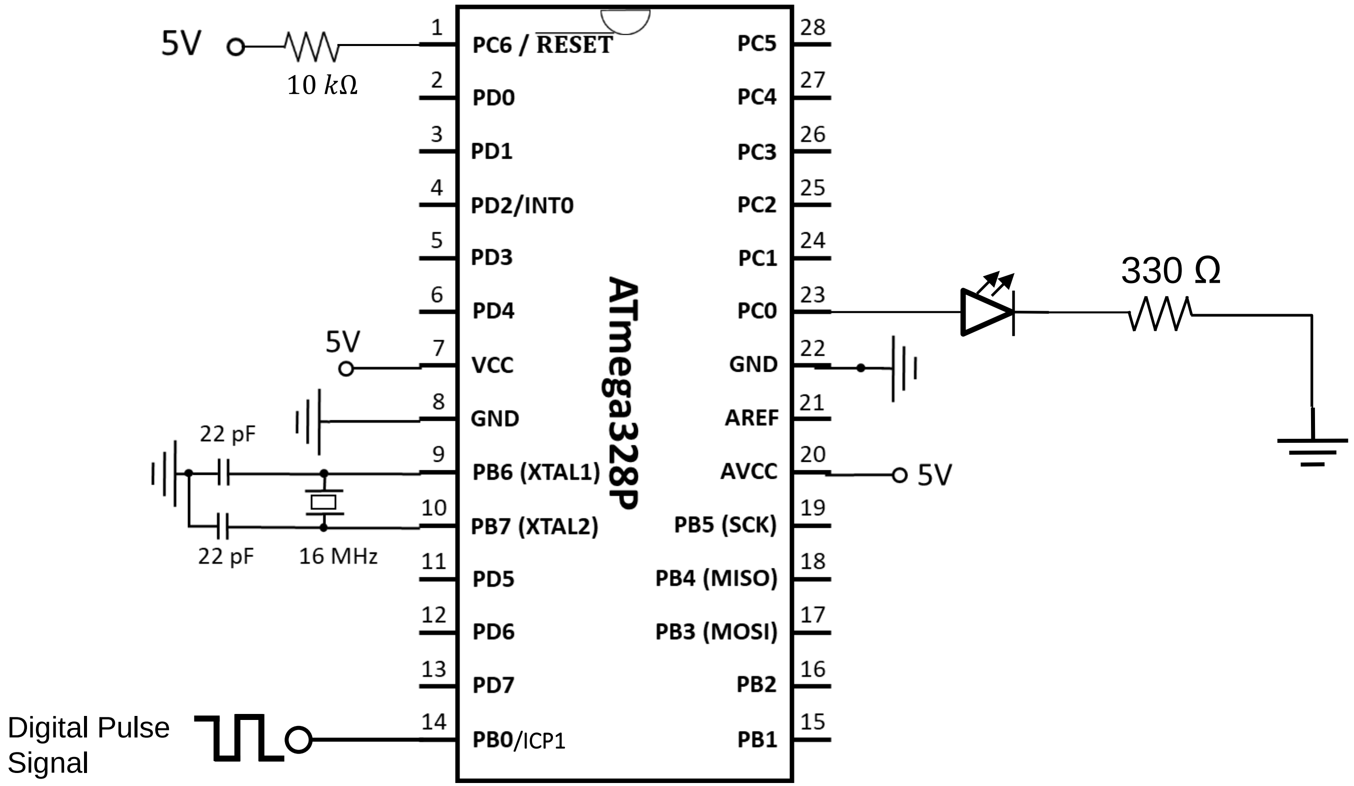
Given that a digital rectangular wave is detected by the ATmega328P through the ICP1 pin, the square wave period varies from 1 ms to 2 ms. It is required to measure the period using the input capture module of timer 1. If the period is greater than 1.5 ms then an LED should be turned on. Otherwise, the LED should be turned off.

Choose the prescaler such that the maximum period (2 ms) is recordable in a 16-bit register (since the 16-bit timer 1 is used).



Problem 2:

Circuit Diagram:



Solution:

- Timer 1 settings:
  - Normal Mode: **WGM10** = **WGM11** = **WGM12** = **WGM13** = 0
  - Capture Rising Edge: **ICES1** = 1
  - Prescaler calculation:

Overflow time in timer 1 is $T_{ovf} = (65536)N * T_{clk}$

$T_{inc} = N * T_{clk}$

Overflow time is required to be greater than 2 ms: $T_{ovf\ max} = 2\ ms$

Calculate required prescaler to achieve $T_{ovf} \geq T_{ovf\ max}$

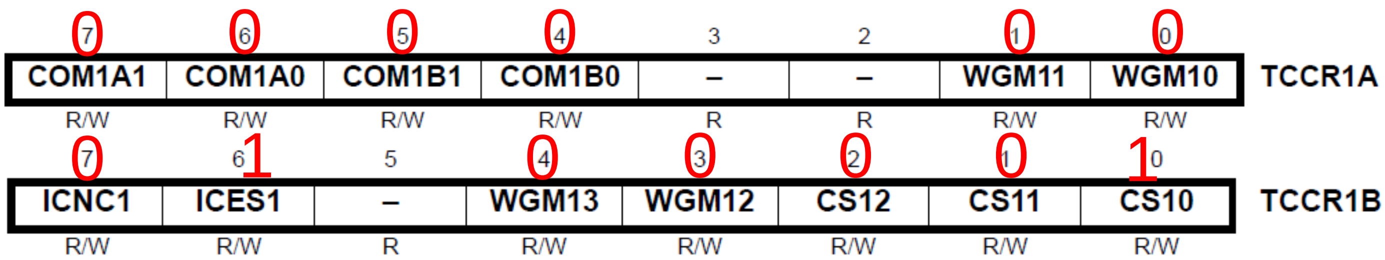
N :Prescaler

T_{clk}:Oscillator Period

$$N_{min} = \frac{T_{ovf\ max}}{65536\ * T_{clk}} = \frac{2\ * 10^{-3}}{65536\ * \frac{1}{16\ * 10^{6}}} = 0.49$$

$$N \geq 0.49\ \rightarrow N = 1$$

- Clock select bits (N = 1): **CS12** = **CS11** = 0, **CS10** = 1



	0	0	0	0			0	0	
	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
	R/W	R/W	R/W	R/W	R	R	R/W	R/W	

	0	1		0	0	0	0	1	
	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

Solution:

- Time equivalent to a single increment in Timer 1 (resolution):

$$T_{inc} = N \ * T_{clk} = 1 * 6.25 \ * 10^{-8} \, s = 62.5 \, ns$$

- Set Global Interrupt Enable bit "**I**"
- Enable input capture interrupt:

ICIE1 = 1

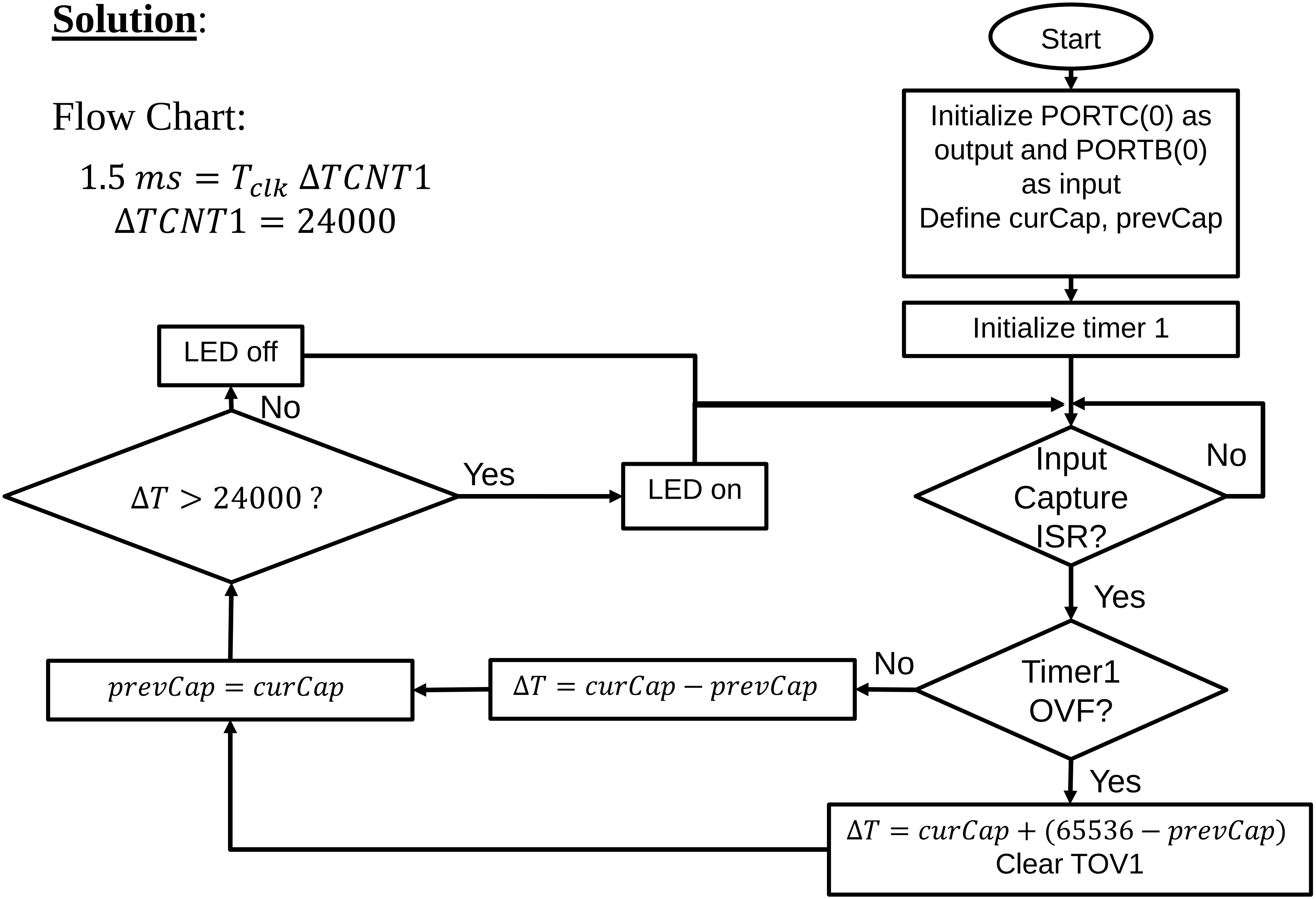
- The period of 1.5 ms is related to the timer 1 increments as follows:

$$1.5 \, ms = T_{clk} \, \Delta TCNT1$$
$$\Delta TCNT1 = 24000$$

Solution:

Flow Chart:

$$1.5\ ms = T_{clk}\ \Delta TCNT1$$
$$\Delta TCNT1 = 24000$$



Start

Initialize PORTC(0) as output and PORTB(0) as input
Define curCap, prevCap

Initialize timer 1

Input Capture ISR?

No

Yes

Timer1 OVF?

No

$\Delta T = curCap - prevCap$

Yes

$\Delta T = curCap + (65536 - prevCap)$
Clear TOV1

$prevCap = curCap$

$\Delta T > 24000\,?$

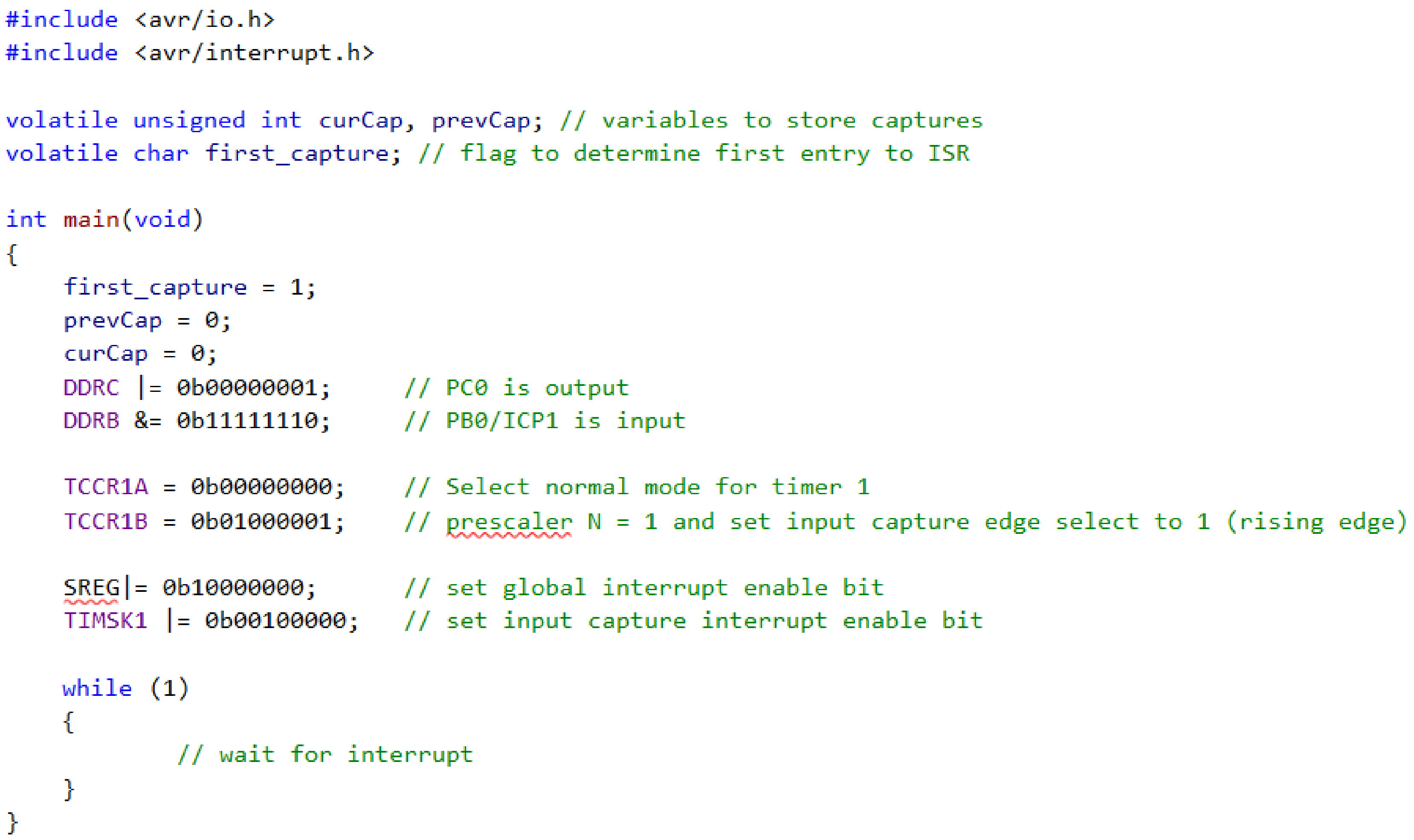
Yes

LED on

No

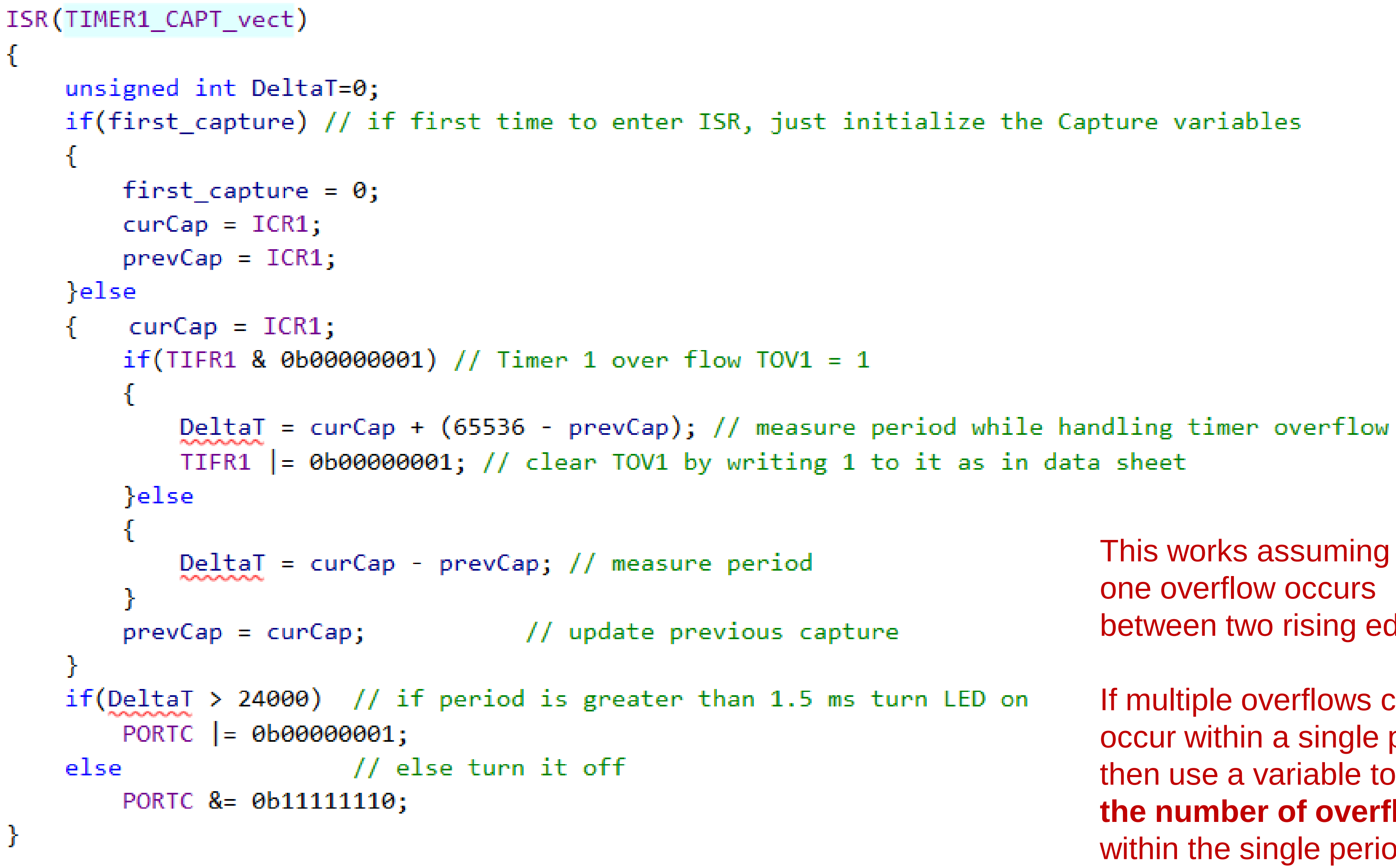
LED off

Code:



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

volatile unsigned int curCap, prevCap; // variables to store captures
volatile char first_capture; // flag to determine first entry to ISR

int main(void)
{
    first_capture = 1;
    prevCap = 0;
    curCap = 0;
    DDRC |= 0b00000001;      // PC0 is output
    DDRB &= 0b11111110;      // PB0/ICP1 is input

    TCCR1A = 0b00000000;     // Select normal mode for timer 1
    TCCR1B = 0b01000001;     // prescaler N = 1 and set input capture edge select to 1 (rising edge)

    SREG|= 0b10000000;       // set global interrupt enable bit
    TIMSK1 |= 0b00100000;    // set input capture interrupt enable bit

    while (1)
    {
            // wait for interrupt
    }
}
```

Code, Continued:



```
ISR(TIMER1_CAPT_vect)
{
    unsigned int DeltaT=0;
    if(first_capture) // if first time to enter ISR, just initialize the Capture variables
    {
        first_capture = 0;
        curCap = ICR1;
        prevCap = ICR1;
    }else
    {   curCap = ICR1;
        if(TIFR1 & 0b00000001) // Timer 1 over flow TOV1 = 1
        {
            DeltaT = curCap + (65536 - prevCap); // measure period while handling timer overflow
            TIFR1 |= 0b00000001; // clear TOV1 by writing 1 to it as in data sheet
        }else
        {
            DeltaT = curCap - prevCap; // measure period
        }
        prevCap = curCap;           // update previous capture
    }
    if(DeltaT > 24000)  // if period is greater than 1.5 ms turn LED on
        PORTC |= 0b00000001;
    else                // else turn it off
        PORTC &= 0b11111110;
}
```

This works assuming only one overflow occurs between two rising edges.

If multiple overflows can occur within a single period, then use a variable to **count the number of overflows** within the single period.