Microprocessor System Design
AVR Microcontroller
Part 3 - Timer

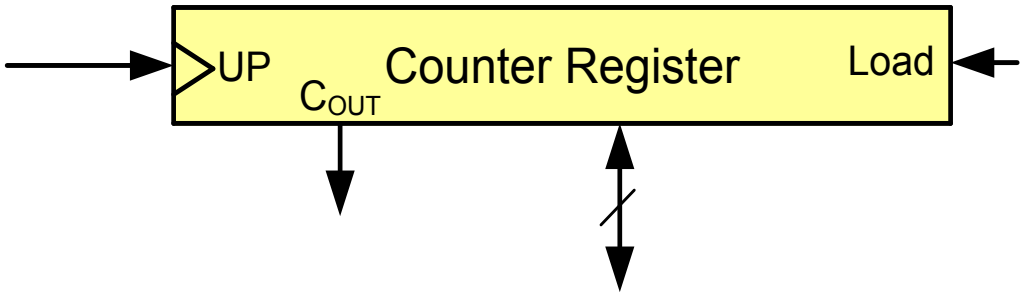
Omid Fatemi

Contents

Timer/counter

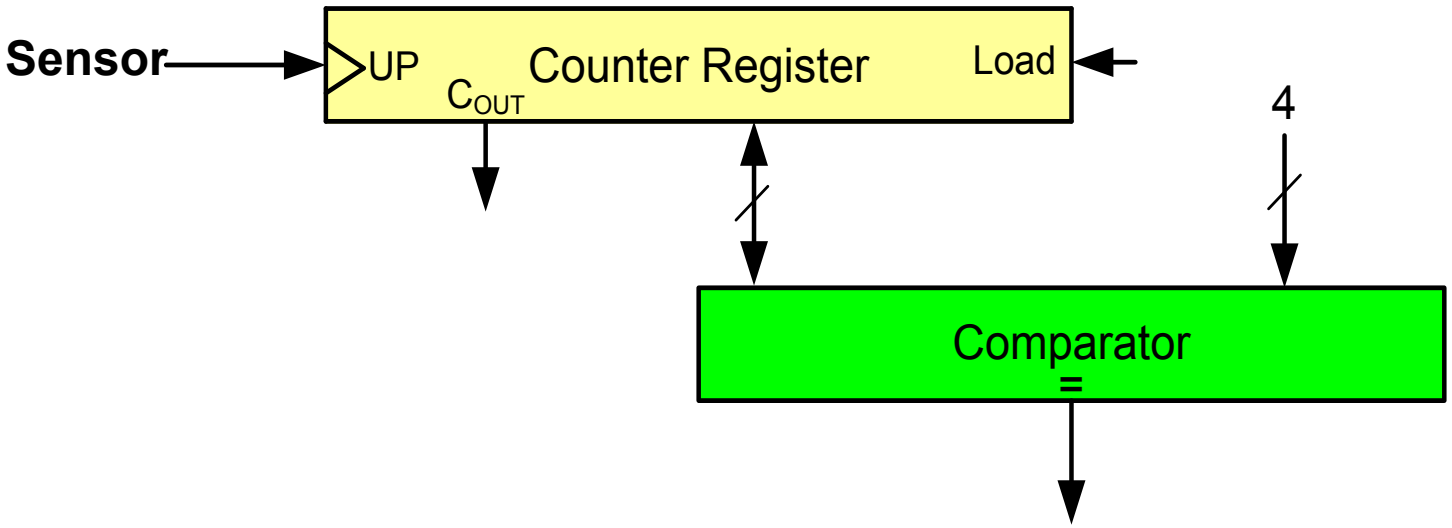
The AVR microcontroller
and embedded
systems
using assembly and c

A counter register



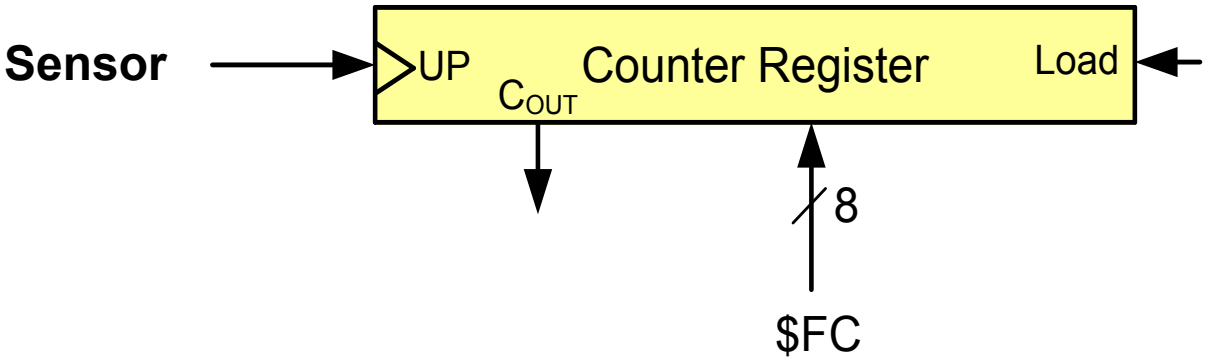
A simple design (counting people)

First design

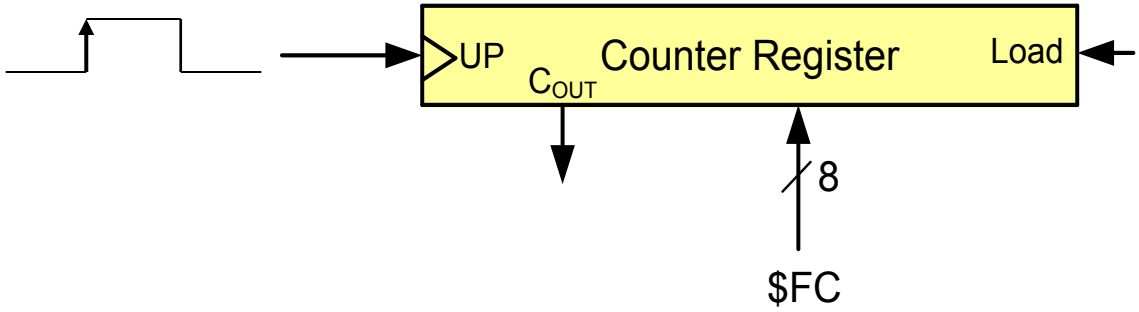


A simple design (counting people)

Second design

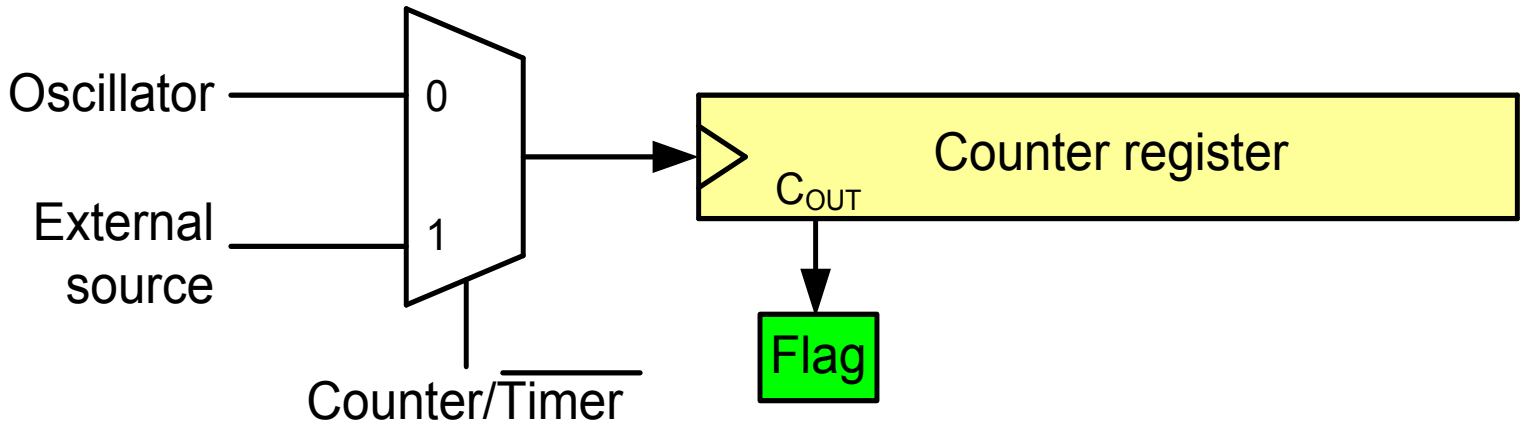


A simple design (making delay)



A generic timer/counter

- **Delay generating**
- **Counting**
- **Wave-form generating**
- **Capturing**

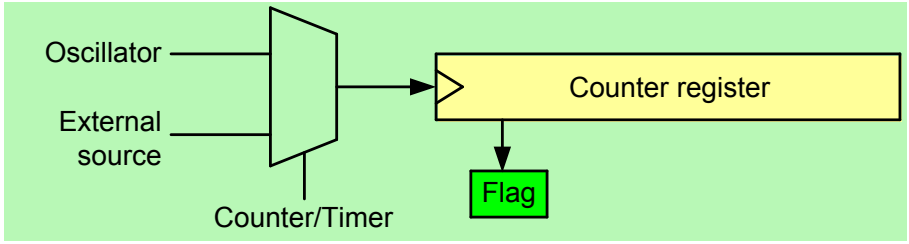


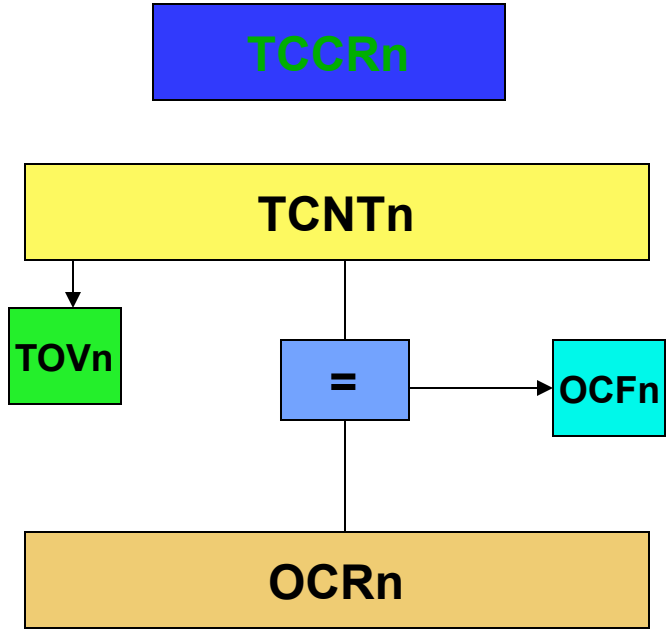
Timers in AVR

- 1 to 6 timers
  - 3 timers in ATmega32
- 8-bit and 16-bit timers
  - two 8-bit timers and one 16-bit timer in ATmega32

Timer in AVR

- TCNTn (Timer/Counter register)
- TOVn (Timer Overflow flag)
- TCCRn (Timer Counter control register)
- OCRn (output compare register)
- OCFn (output compare match flag)





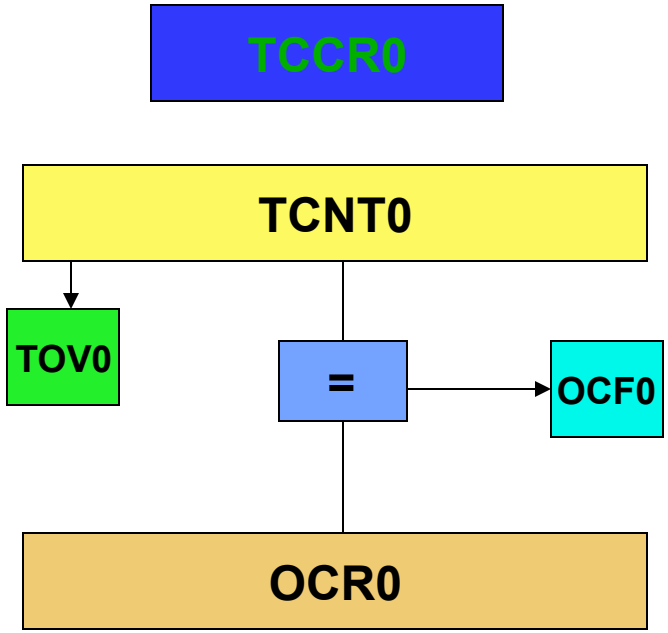
Comment:

All of the timer registers are byte-addressable I/O registers

Timer 0 (an 8-bit timer)

The AVR microcontroller
and embedded
systems
using assembly and c

Timer 0



OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

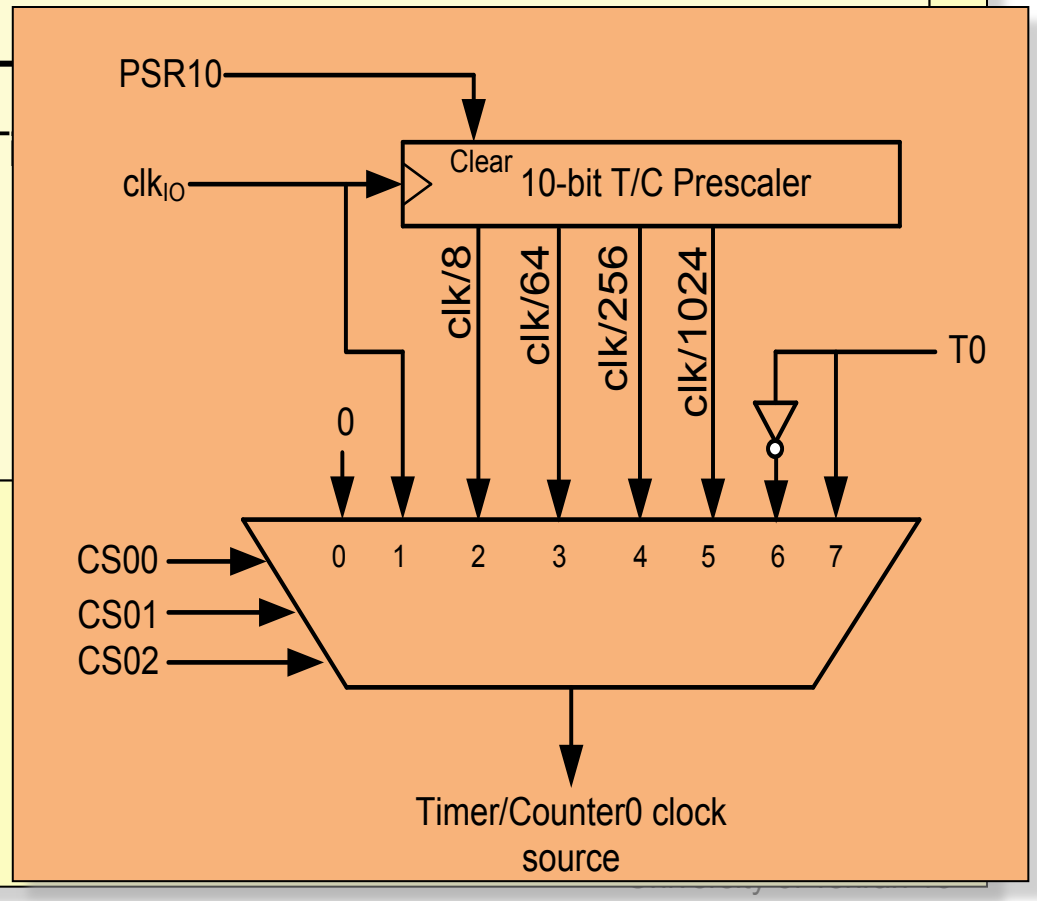
TIFR

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

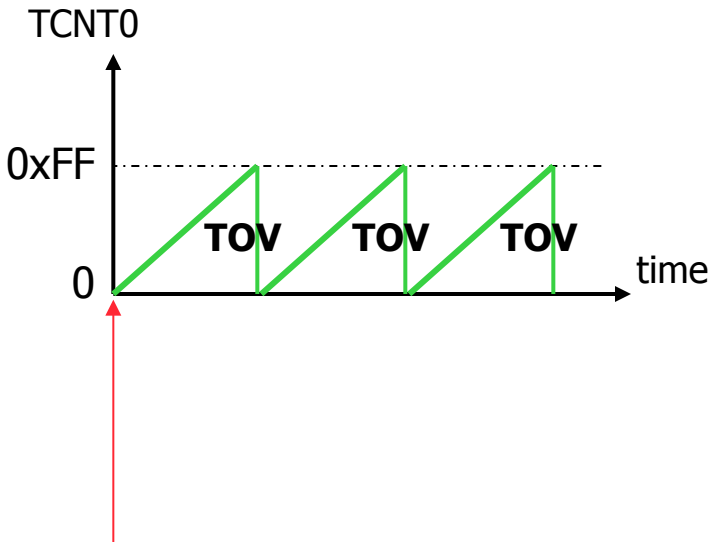
TCCR0

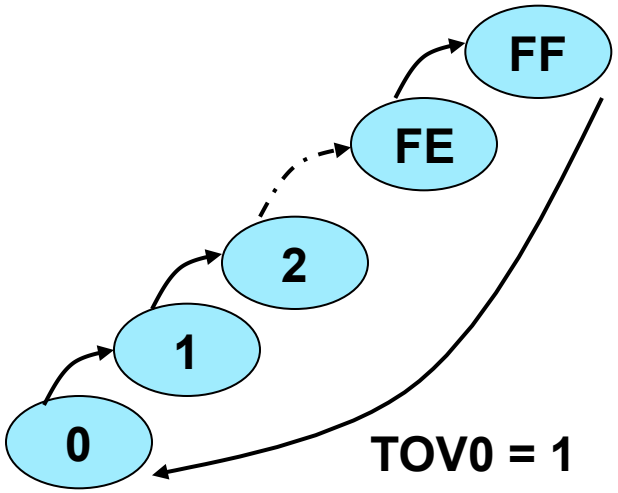
Timer Mode (WGM)

WGM00	WGM01	Comment
0	0	Normal
0	1	CTC (Clear T
1	0	PWM, phase
1	1	Fast PWM



Normal mode

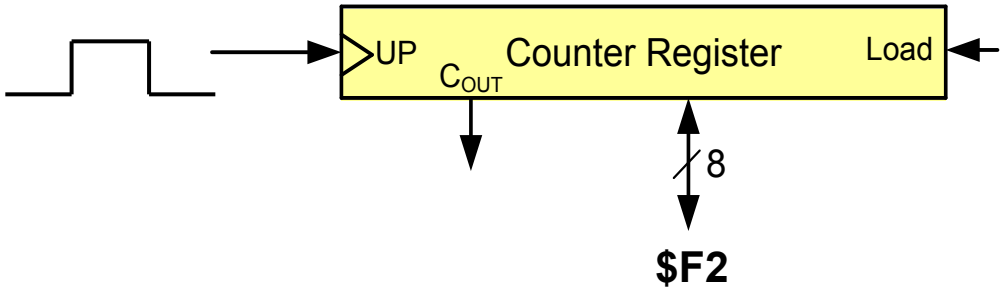




TOV0: **1**

Example 1: Write a program that waits 14 machine cycles in Normal mode.

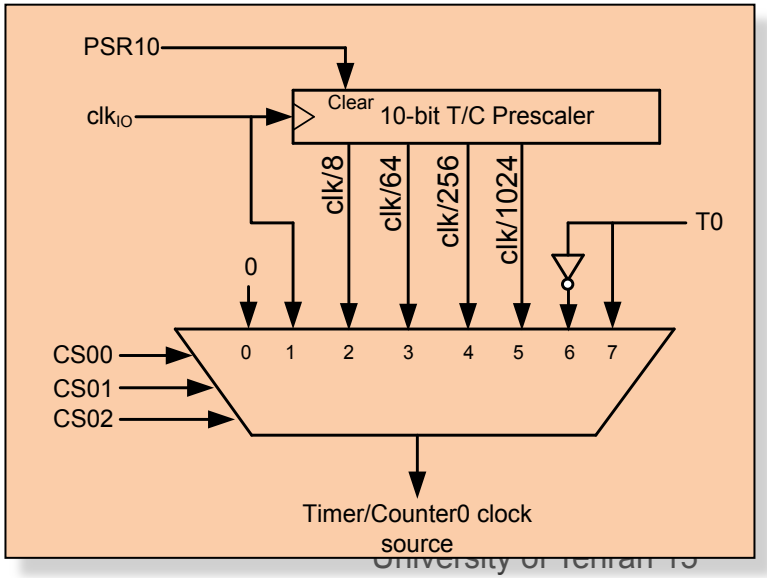
```
14 = $0E

  $100
  -$0E
 ------
   $F2
```



FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
	0			0	0	0	1

TCCR0

WGM00	WGM01	Comment
0	0	Normal
0	1	CTC
1	0	PWM, phase correct
1	1	Fast PWM



Example 1: Write a program that waits 14 machine cycles in Normal mode.

```
$100
-$0E
-----
 $F2
```

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR

0 0 0 0 0 0 0 1

```
.INCLUDE "M32DEF.INC"

        LDI     R16,0x20
        SBI     DDRB,5      ;PB5 as an output
        LDI     R17,0
        OUT     PORTB,R17
BEGIN:  LDI     R20,0xF2
        OUT     TCNT0,R20       ;load timer0
        LDI     R20,0x01
        OUT     TCCR0,R20 ;Ti
AGAIN:  IN      R20,TIFR
        SBRS    R20,0 ;if TOV
        RJMP    AGAIN
        LDI     R20,0x0
        OUT     TCCR0,R20
        LDI     R20,(1<<TOV0)
        OUT     TIFR,R20

        EOR     R17,R16
        OUT     PORTB,R17
        RJMP    BEGIN
```

set byte immediate

toggle

SKIP

```
DDRB = 1<<5;
PORTB &= ~(1<<5);   //PB5=0
while (1)
{
```

Question: How to calculate the delay generated by the timer?

Answer:

1) Calculate how much a machine clock lasts.
   T= 1/f
2) Calculate how many machine clocks it waits.
3) Delay = T * number of machine cycles

In example 1 calculate the delay. XTAL = 10 MHz.

Solution 1 (inaccurate):

1) **Calculating T:**

T = 1/f = 1/10M = 0.1µs

2) **Calculating num of machine cycles:**

```
$100
-$F2
  $0E = 14
```

3) **Calculating delay**

14 * 0.1µs = 1.4 0µs

```
.INCLUDE "M32DEF.INC"

          LDI     R16,0x20
          SBI     DDRB,5   ;PB5 as an output
          LDI     R17,0
          OUT     PORTB,R17
BEGIN:    LDI     R20,0xF2
          OUT     TCNT0,R20          ;load timer0
          LDI     R20,0x01
          OUT     TCCR0,R20 ;Timer0,Normal mode,int clk
AGAIN:    IN      R20,TIFR           ;read TIFR
          SBRS    R20,0 ;if TOV0 is set skip next inst.
          RJMP    AGAIN
          LDI     R20,0x0
          OUT     TCCR0,R20          ;stop Timer0
          LDI     R20,0x01
          OUT     TIFR,R20           ;clear TOV0 flag

          EOR     R17,R16            ;toggle D5 of R17
          OUT     PORTB,R17          ;toggle PB5
          RJMP    BEGIN
```

Accurate calculating

Other than timer, executing the instructions consumes time; so if we want to calculate the accurate delay a program causes we should add the delay caused by instructions to the delay caused by the timer

```
        LDI     R16,0x20
        SBI     DDRB,5
        LDI     R17,0
        OUT     PORTB,R17
BEGIN:  LDI     R20,0xF2        1
        OUT     TCNT0,R20       1
        LDI     R20,0x01        1
        OUT     TCCR0,R20       1
AGAIN:  IN      R20,TIFR        1
        SBRS    R20,0           1 / 2
        RJMP    AGAIN           2
        LDI     R20,0x0         1
        OUT     TCCR0,R20       1
        LDI     R20,0x01        1
        OUT     TIFR,R20        1
        EOR     R17,R16         1
        OUT     PORTB,R17       1
        RJMP    BEGIN           2
                                18
```

Delay caused by timer = 14 * 0.1µs = 1.4 µs **Delay caused by instructions = 18 * 0.1µs = 1.8**

Total delay = 3.2 µs ➔ wave period = 2*3.2 µs = 6.4 µs ➔ wave frequency = 156.25 KHz

Finding values to be loaded into the timer

1. Calculate the period of clock source.
   - Period = 1 / Frequency
     - » E.g. For XTAL = 8 MHz ➔ T = 1/8MHz
2. Divide the desired time delay by period of clock.
3. Perform 256 - n, where n is the decimal value we got in Step 2.
4. Set TCNT0 = 256 - n

Example 2: Assuming that XTAL = 10 MHz, write a program to generate a square wave with a period of 10 ms on pin PORTB.3.

- For a square wave with T = 10 µs we must have a time delay of 5 µs. Because XTAL = 10 MHz, the counter counts up every 0.1 µs. This means that we need 5 µs / 0.1 µs = 50 clocks. 256 - 50 = 206.

```
.INCLUDE "M32DEF.INC"

         LDI     R16,0x08
         SBI     DDRB,3   ;PB3 as an output
         LDI     R17,0
         OUT     PORTB,R17
BEGIN:   LDI     R20,206
         OUT     TCNT0,R20         ;load timer0
         LDI     R20,0x01
         OUT     TCCR0,R20 ;Timer0,Normal mode,int clk
AGAIN:   IN      R20,TIFR          ;read TIFR
         SBRS    R20,TOV0 ;if TOV0 is set skip next
         RJMP    AGAIN
         LDI     R20,0x0
         OUT     TCCR0,R20         ;stop Timer0
         LDI     R20,0x01
         OUT     TIFR,R20          ;clear TOV0 flag
         EOR     R17,R16           ;toggle D3 of R17
         OUT     PORTB,R17         ;toggle PB3
         RJMP    BEGIN
```

```
DDRB = 1<<3;
PORTB &= ~ (1<<3);
while (1)
{
  TCNT0 = 206;
  TCCR0 = 0x01;
  while((TIFR&0x01) == 0);
  TCCR0 = 0;
  TIFR = 1<<TOV0;
  PORTB = PORTB ^ (1<<3);
}
```

Example 3: Modify TCNT0 in Example 2 to get the largest time delay possible with no prescaler. Find the delay in µs. In your calculation, do not include the overhead due to instructions.

- **To get the largest delay we make TCNT0 zero. This will count up from 00 to 0xFF and then roll over to zero.**

```
.INCLUDE "M32DEF.INC"

         LDI     R16,1<<3
         SBI     DDRB,3      ;PB3 as an output
         LDI     R17,0
         OUT     PORTB,R17
BEGIN:   LDI     R20,0x0
         OUT     TCNT0,R20          ;load Timer0
         LDI     R20,0x01
         OUT     TCCR0,R20 ;Timer0,Normal mode,int clk
AGAIN:   IN      R20,TI
         SBRS    R20,TO
         RJMP    AGAIN
         LDI     R20,0x
         OUT     TCCR0,
         LDI     R20,0x
         OUT     TIFR,R
         EOR     R17,R1
         OUT     PORTB,
         RJMP    BEGIN
```

```
DDRB = 1 << 3;
PORTB &= ~(1<<3);
while (1)
{
   TCNT0 = 0x0;
   TCCR0 = 0x01;

while((TIFR&(1<<TOV0))==0);
   TCCR0 = 0;
   TIFR = 0x01;
   PORTB = PORTB^(1<<3);
}
```

Solution

1) **Calculating T:**

T = 1/f = 1/10MHz = 0.1µs

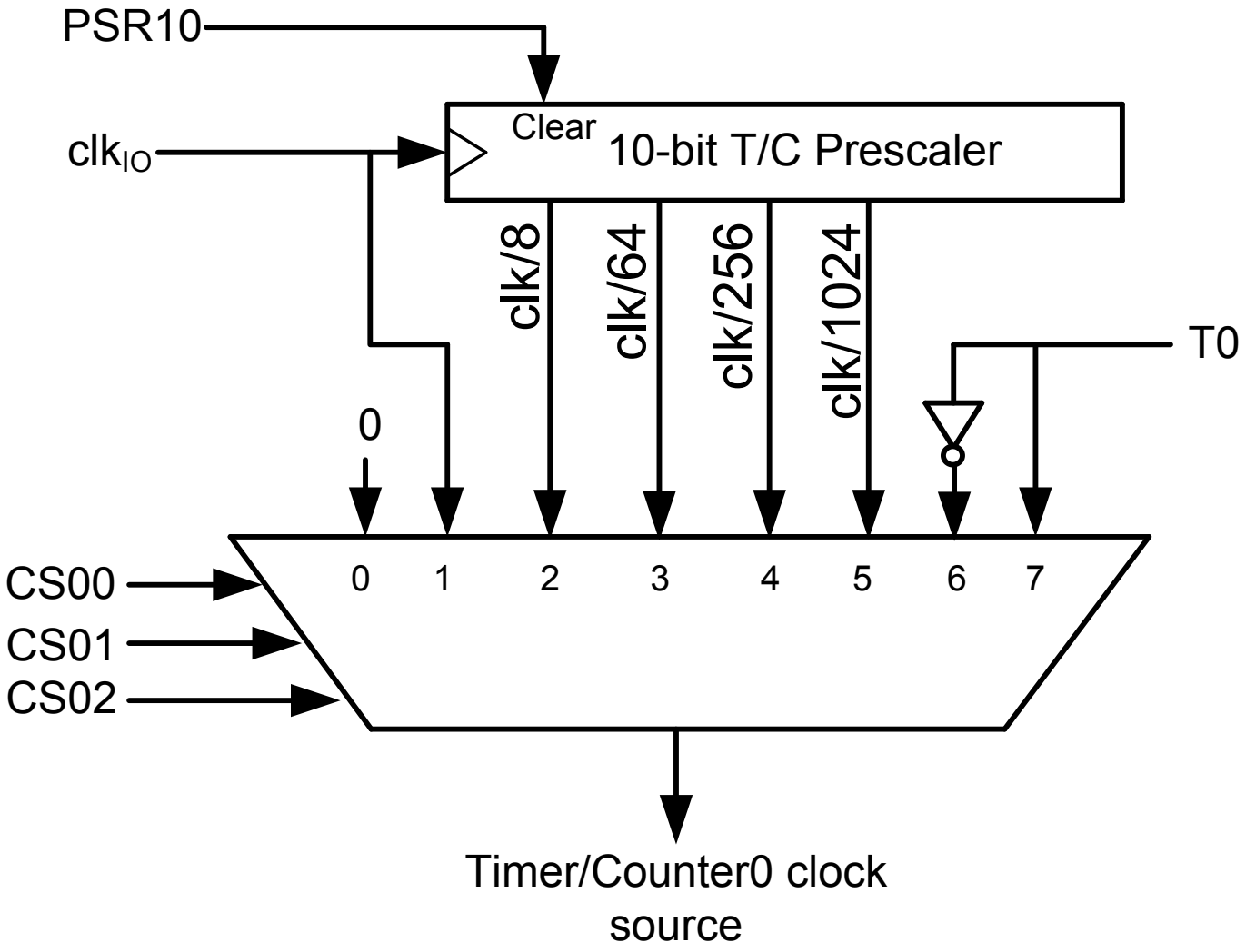
2) **Calculating delay**

256 * 0.1µs = 25.6µs

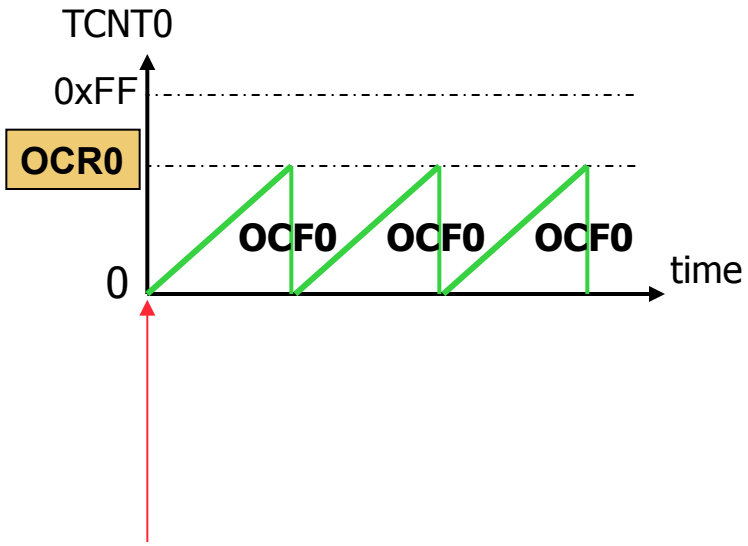
Generating Large Delays

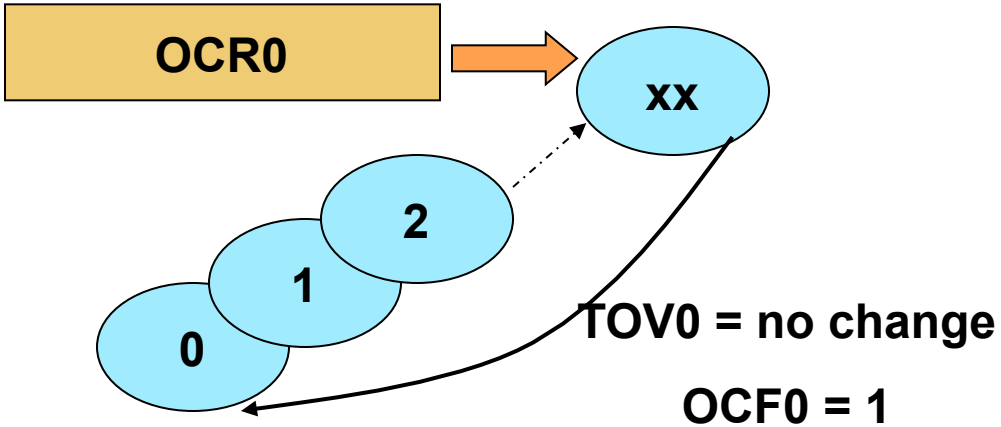
- **Using loop**
- **Prescaler**
- **Bigger counters**

Prescaler and generating a large time delay



CTC (Clear Timer on Compare match) mode





TOV0: 0

OCF0: 1

Rewrite example 2 using CTC

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR

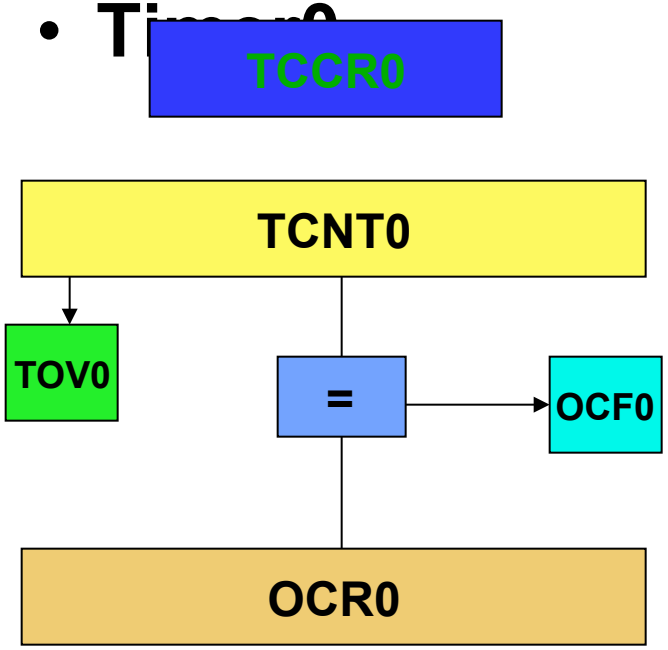
- **For a square wave with T = 10 µs we must have a time delay of 5 µs. Because XTAL = 10 MHz, the counter counts up every 0.1 µs. This means that we need 5 µs / 0.1 µs = 50 clocks. Therefore, we have OCR0= 49.**

```
.INCLUDE "M32DEF.INC"
        LDI     R16,0x08
        SBI     DDRB,3   ;PB3 as an output
        LDI     R17,0
        OUT     PORTB,R17
        LDI     R20,49
        OUT     OCR0,R20 ;load timer0
BEGIN:  LDI     R20,0x09
        OUT     TCCR0,R20 ;Timer0,CTC mode,int clk
AGAIN:  IN      R20,TIFR          ;read TIFR
        SBRS    R20,OCF0 ;if OCF0 is set skip next
        RJMP    AGAIN
        LDI     R20,0x0
        OUT     TCCR0,R20         ;stop Timer0
        LDI     R20,0x02
        OUT     TIFR,R20          ;clear TOV0 flag
        EOR     R17,R16           ;toggle D3 of R17
        OUT     PORTB,R17         ;toggle PB3
        RJMP    BEGIN
```

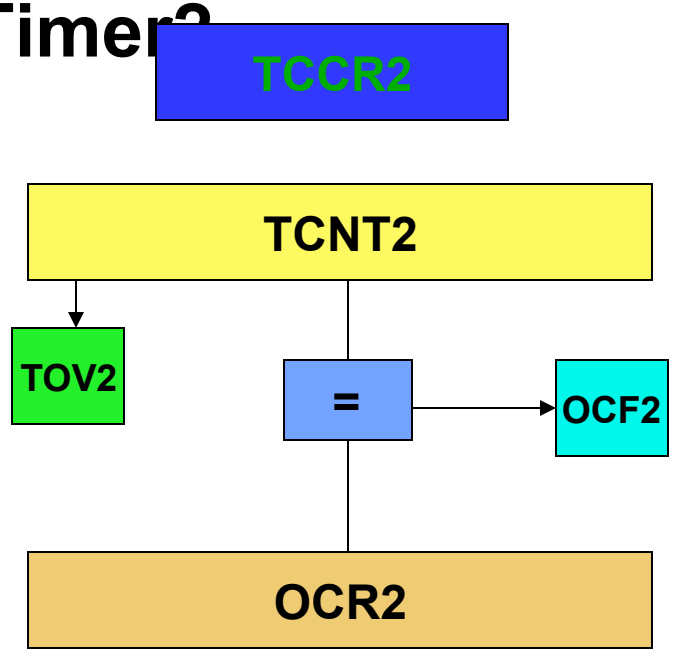
```
DDRB |= 1<<3;
PORTB &= ~(1<<3);
while (1)
{
  OCR0 = 49;
  TCCR0 = 0x09;

while((TIFR&(1<<OCF0))==0);
  TCCR0 = 0; //stop timer0
  TIFR = 0x02;
  PORTB.3 = ~PORTB.3;
}
```

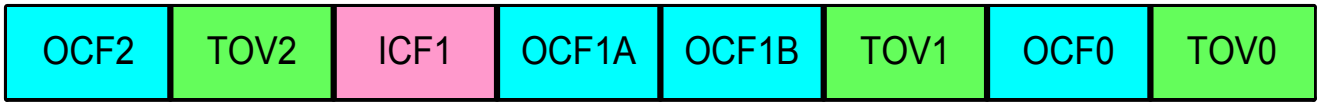
Timer2

- Timer0



- Timer2





The difference between Timer0 and Timer2

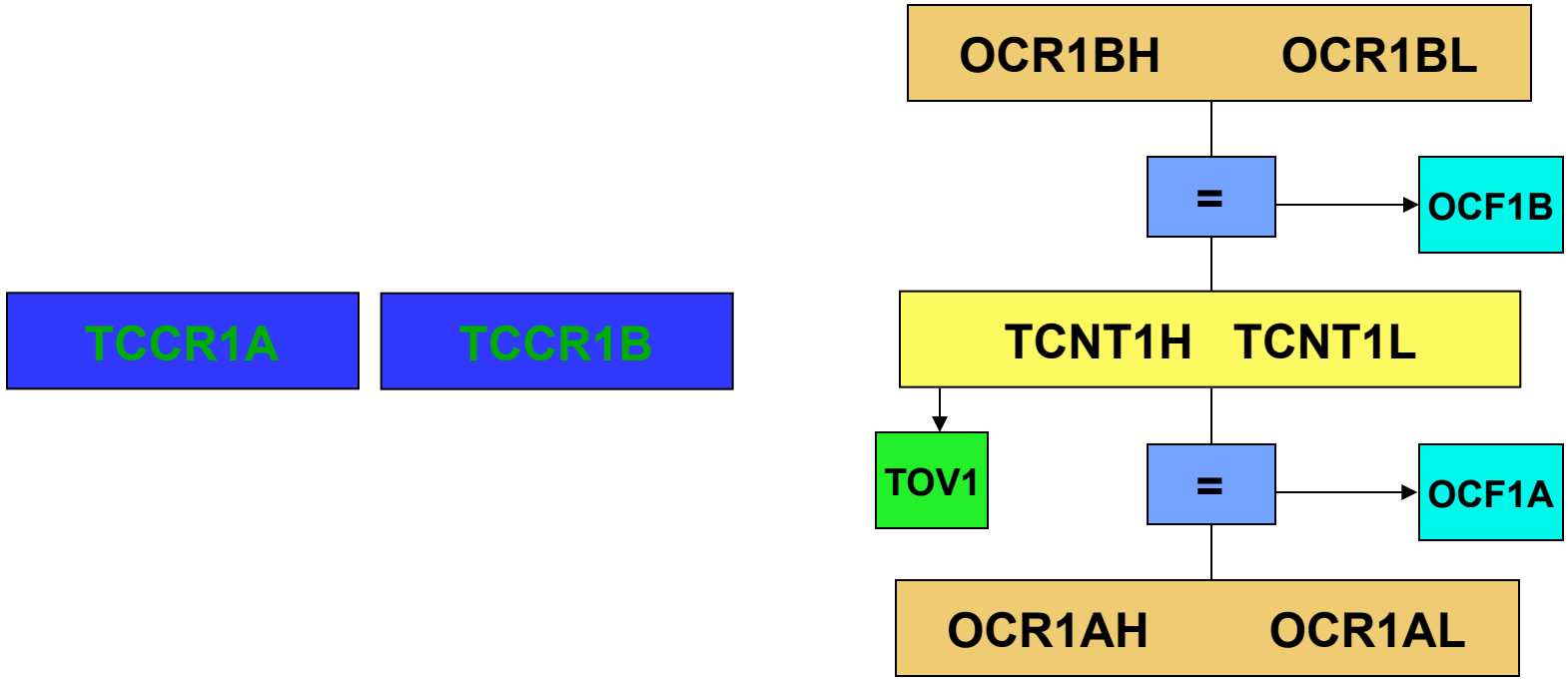
- **Timer0**

CS02	CS01	CS00	Comment
0	0	0	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 (falling edge)
1	1	1	External clock (rising edge)

- **Timer2**

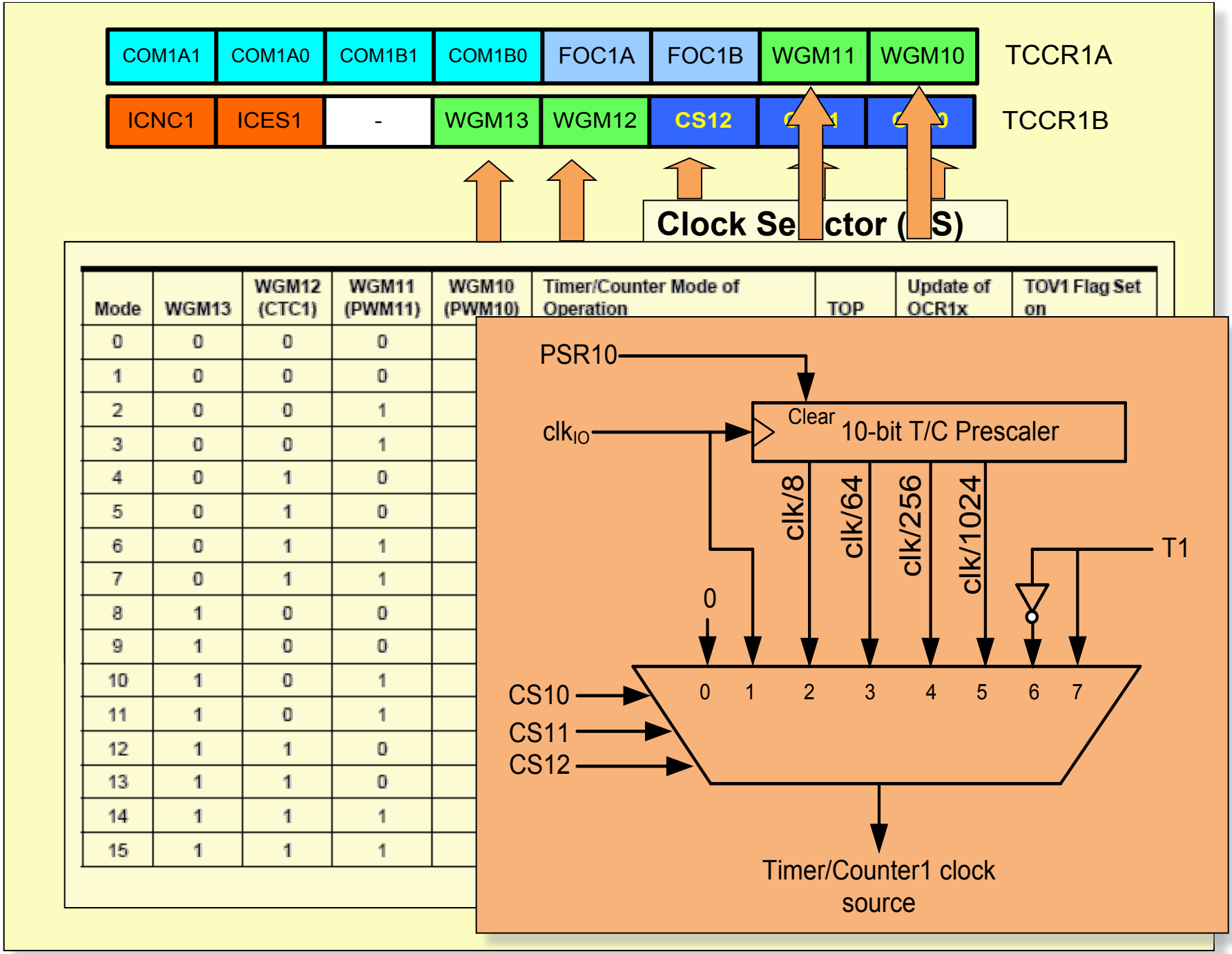
CS22	CS21	CS20	Comment
0	0	0	Timer/Counter stopped
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 32
1	0	0	clk / 64
1	0	1	clk / 128
1	1	0	clk / 256
1	1	1	clk / 1024

Timer 1



OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

TIFR

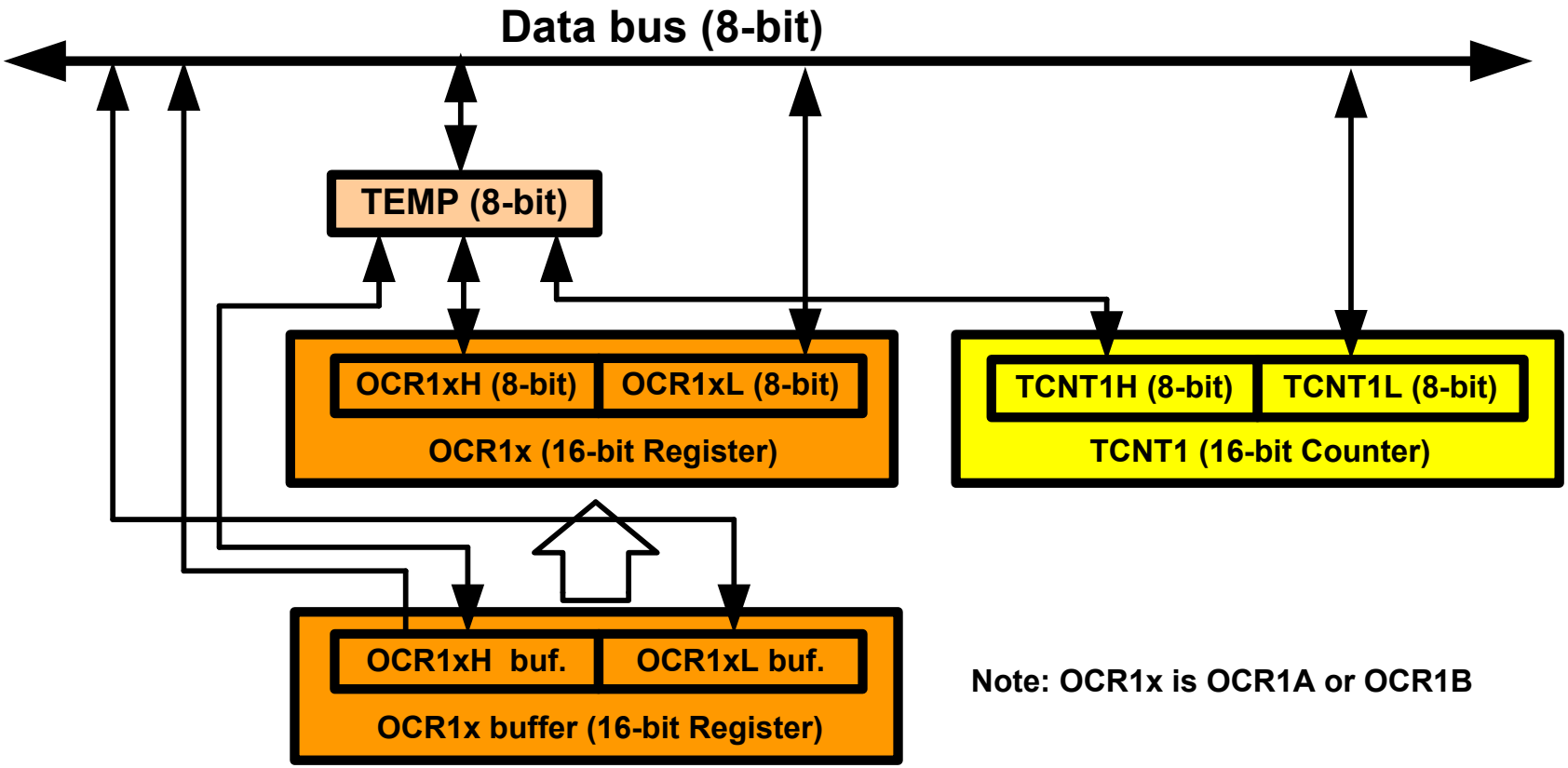


Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x	TOV1 Flag Set on
0	0	0	0					
1	0	0	0					
2	0	0	1					
3	0	0	1					
4	0	1	0					
5	0	1	0					
6	0	1	1					
7	0	1	1					
8	1	0	0					
9	1	0	0					
10	1	0	1					
11	1	0	1					
12	1	1	0					
13	1	1	0					
14	1	1	1					
15	1	1	1					

Assuming XTAL = 10 MHz write a program that toggles PB5 once per millisecond, using Normal mode.

```
.INCLUDE "M32DEF.INC"
    LDI    R16,HIGH(RAMEND)     ;init stack pointer
    OUT    SPH,R16
    LDI    R16,LOW(RAMEND)
    OUT    SPL,R16
    SBI    DDRB,5               ;PB5 as an output
BEGIN:SBI  PORTB,5              ;PB5 = 1
    RCALL  DELAY_1ms
    CBI    PORTB,5              ;PB5 = 0
    RCALL  DELAY_1ms
    RJMP   BEGIN

DELAY_1ms:
    LDI    R20,HIGH(-10000)
    OUT    TCNT1H,R20
    LDI    R20, ,LOW(-10000)
    OUT    TCNT1L,R20           ;Timer1 overflows after 10000 machine cycles
    LDI    R20,0x0
    OUT    TCCR1A,R20           ;WGM11:10=00
    LDI    R20,0x1
    OUT    TCCR1B,R20           ;WGM13:12=00,CS=CLK
AGAIN:IN   R20,TIFR             ;read TIFR
    SBRS   R20,TOV1             ;if OCF1A is set skip next instruction
    RJMP   AGAIN
    LDI    R20,1<<TOV1
    OUT    TIFR,R20             ;clear TOV1 flag
    LDI    R19,0
    OUT    TCCR1B,R19           ;stop timer
    OUT    TCCR1A,R19           ;
    RET
```

TEMP register



Note: OCR1x is OCR1A or OCR1B

```
LDI  R20,0xF3
OUT  TCNT1L,R20
LDI  R21,TCNT1H
OUT  TCNT1H,R20
LDI   R20,0x53
OUT   TCNT1L,R20
```

```
TCNT1L=0xF3;
TCNT1H=0x53;
```

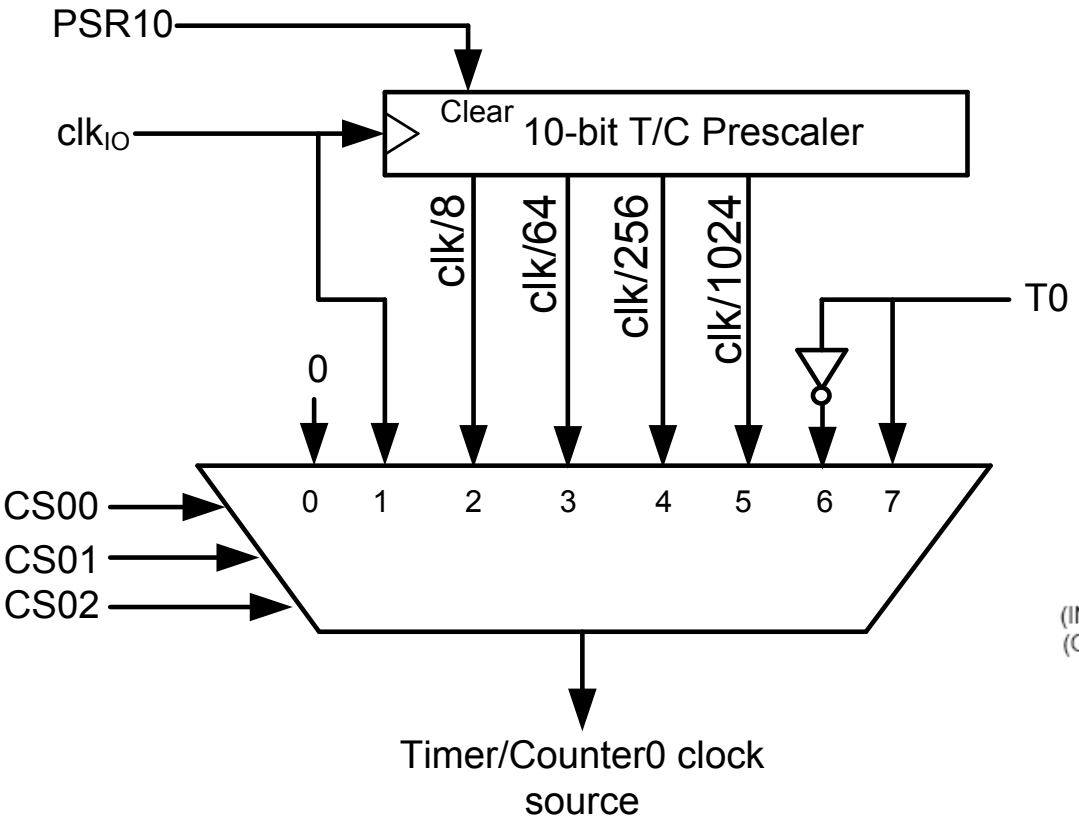
Assuming XTAL = 10 MHz write a program that toggles PB5 once per millisecond, using CTC mode.

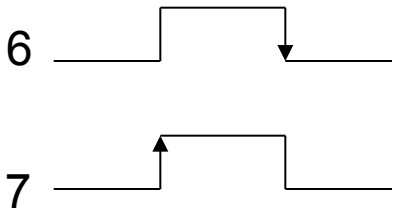
```
.INCLUDE "M32DEF.INC"
    LDI     R16,HIGH(RAMEND)
    OUT     SPH,R16
    LDI     R16,LOW(RAMEND)
    OUT     SPL,R16
    SBI     DDRB,5                  ;PB5 as an output
BEGIN:SBI   PORTB,5                 ;PB5 = 1
    RCALL   DELAY_1ms
    CBI     PORTB,5                 ;PB5 = 0
    RCALL   DELAY_1ms
    RJMP    BEGIN

DELAY_1ms:
    LDI     R20,0x00
    OUT     TCNT1H,R20              ;TEMP = 0
    OUT     TCNT1L,R20              ;TCNT1L = 0, TCNT1H = TEMP

    LDI     R20,0x27
    OUT     OCR1AH,R20              ;TEMP = 0x27
    LDI     R20,0x0F
    OUT     OCR1AL,R20              ;OCR1AL = 0x0F, OCR1AH = TEMP

    LDI     R20,0x3
    OUT     TCCR1A,R20              ;WGM11:10=11
    LDI     R20,0x19
    OUT     TCCR1B,R20              ;WGM13:12=11,CS=CLK
AGAIN:
    IN      R20,TIFR                ;read TIFR
    SBRS    R20,OCF1A               ;if OCF1A is set skip next instruction
    RJMP    AGAIN
    LDI     R20,1<<OCF1A
    OUT     TIFR,R20                ;clear OCF1A flag
    LDI     R19,0
    OUT     TCCR1B,R19              ;stop timer
    OUT     TCCR1A,R19              ;
    RET
```

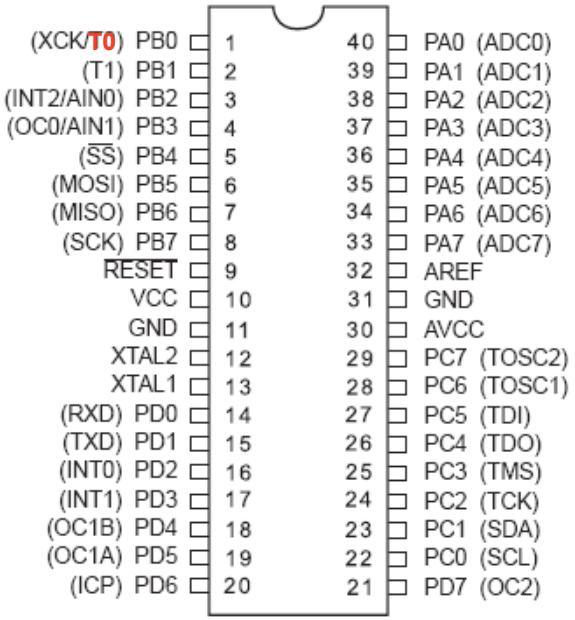
Counting

The AVR microcontroller
and embedded
systems
using assembly and c

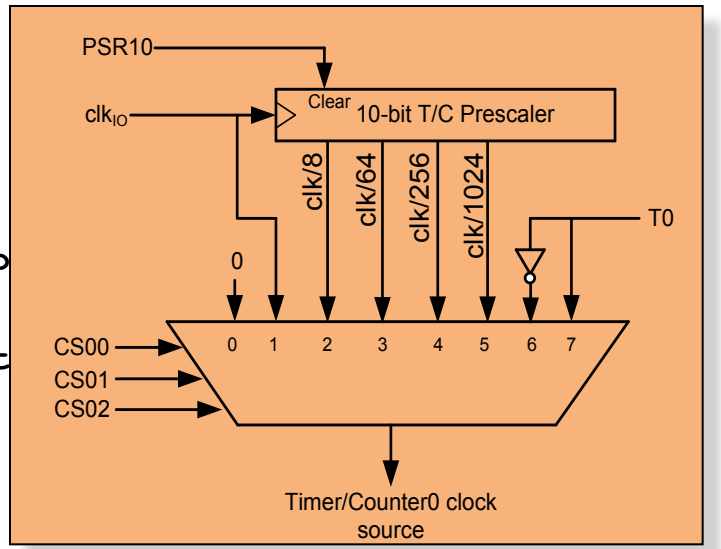
Counting


PSR10
clk$_{IO}$
Clear
10-bit T/C Prescaler
clk/8
clk/64
clk/256
clk/1024
T0
0
CS00
CS01
CS02
0 1 2 3 4 5 6 7
Timer/Counter0 clock source


6
7


(XCK/T0) PB0 1
(T1) PB1 2
(INT2/AIN0) PB2 3
(OC0/AIN1) PB3 4
(SS) PB4 5
(MOSI) PB5 6
(MISO) PB6 7
(SCK) PB7 8
RESET 9
VCC 10
GND 11
XTAL2 12
XTAL1 13
(RXD) PD0 14
(TXD) PD1 15
(INT0) PD2 16
(INT1) PD3 17
(OC1B) PD4 18
(OC1A) PD5 19
(ICP) PD6 20
40 PA0 (ADC0)
39 PA1 (ADC1)
38 PA2 (ADC2)
37 PA3 (ADC3)
36 PA4 (ADC4)
35 PA5 (ADC5)
34 PA6 (ADC6)
33 PA7 (ADC7)
32 AREF
31 GND
30 AVCC
29 PC7 (TOSC2)
28 PC6 (TOSC1)
27 PC5 (TDI)
26 PC4 (TDO)
25 PC3 (TMS)
24 PC2 (TCK)
23 PC1 (SDA)
22 PC0 (SCL)
21 PD7 (OC2)

Example Assuming that clock pulses are fed into pin T0, write a program for counter 0 in normal mode to count the pulses on falling edge and display the state of the TCNT0 count on PORTC.




```
.INCLUDE "M32DEF.INC"
   CBI    DDRB,0            ;make T0 (PB0) inp
   LDI    R20,0xFF
   OUT    DDRC,R20          ;make PORTC output
   LDI    R20,0x06
   OUT    TCCR0,R20         ;counter, falling
AGAIN:
   IN     R20,TCNT0
   OUT    PORTC,R20         ;PORTC = TCNT0
   IN     R16,TIFR
   SBRS   R16,TOV0
   RJMP   AGAIN             ;keep doing it
   LDI    R16,1<<TOV0
   OUT    TIFR, R16
   RJMP   AGAIN             ;keep doing it
```

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

TCCR0

Assuming that clock pulses are fed into pin T1. Write a program for counter 1 in CTC mode to make PORTC.0 high every 100 pulses.

```
.INCLUDE "M32DEF.INC"

   CBI    DDRB,1              ;make T1 (PB1) input

   SBI    DDRC,0              ;PC0 as an output

   LDI    R20,0x0
   OUT    TCCR1A,R20
   LDI    R20,0x0E
   OUT    TCCR1B,R20          ;CTC, counter, falling edge
AGAIN:
   LDI    R20,0
   OUT    OCR1AH,R20          ;TEMP = 0
   LDI    R20,99
   OUT    OCR1AL,R20          ;ORC1L = R20, OCR1H = TEMP
L1:       IN       R20,TIFR
   SBRS   R20,OCF1A
   RJMP   L1                  ;keep doing it
   LDI    R20,1<<OCF1A        ;clear OCF1A flag
   OUT    TIFR, R20

   SBI    PORTC,0             ;PC0 = 1
   CBI    PORTC,0             ;PC0 = 0
   RJMP   AGAIN               ;keep doing it
```

Wave generating and Capturing

The AVR microcontroller
and embedded
systems
using assembly and c

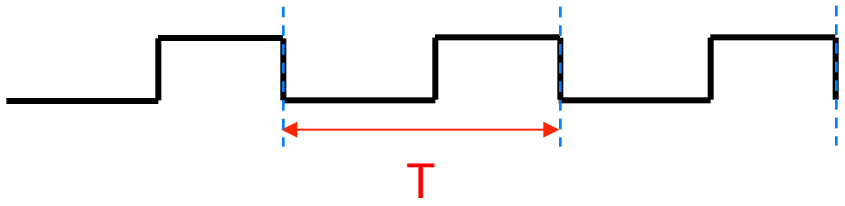
Topics

- Wave characteristics
  - Music and frequencies
- Timer0 review
- Wave generating using Timer0
- Wave generating using Timer2
- Wave generating using Timer1
- Capturing

Wave characteristics

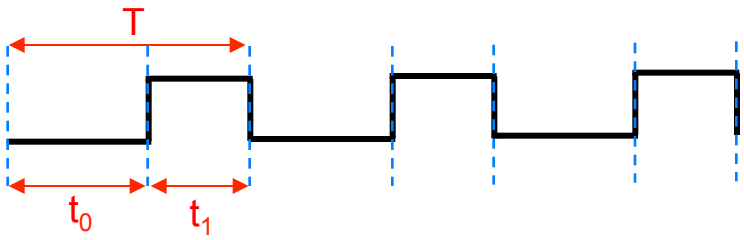
- **Period**
  - **Frequency**

$$f = \frac{1}{T}$$



- **Duty cycle**

$$\text{duty cycle} = \frac{t_0}{T} \times 100 = \frac{t_0}{t_0 + t_1} \times 100$$

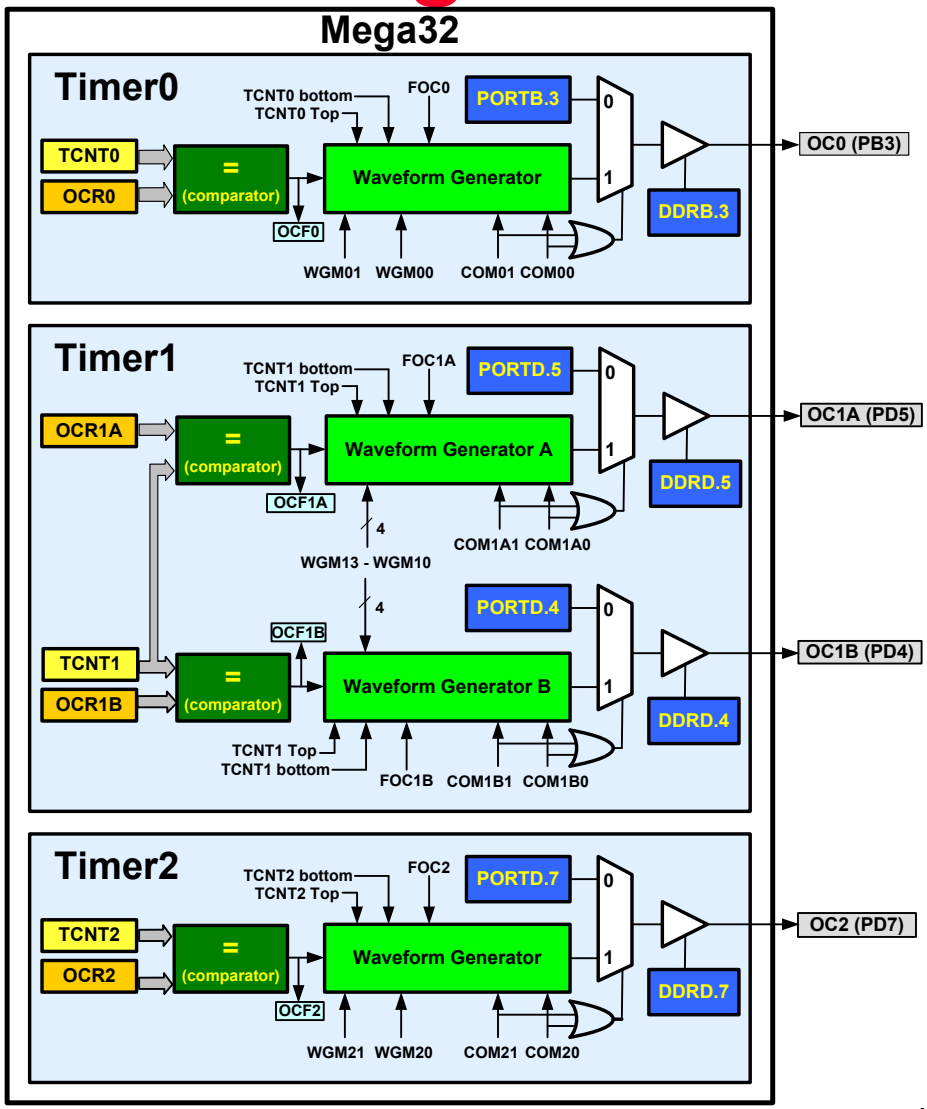


- **Amplitude**

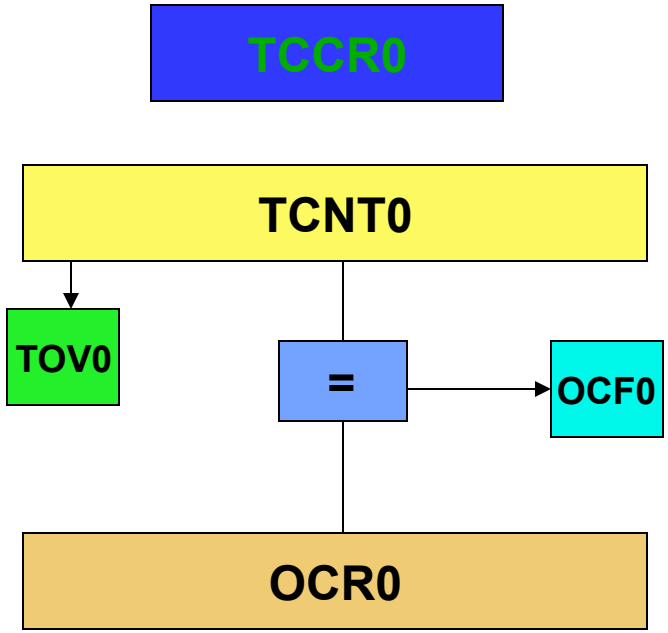
Music and Frequencies

Figure 13-5. Piano Note Frequencies

Waveform generators in ATmega32

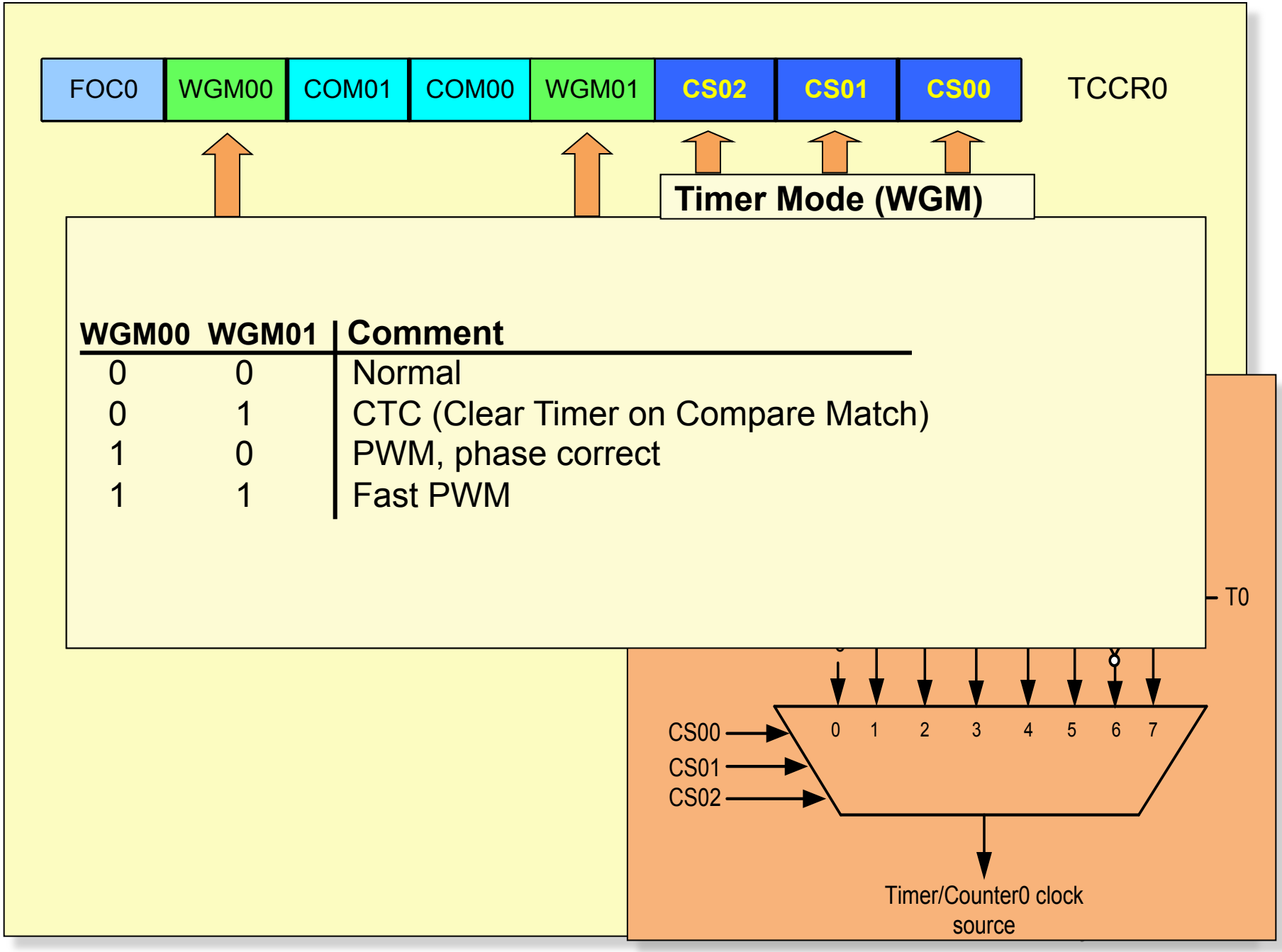


Timer0 Review



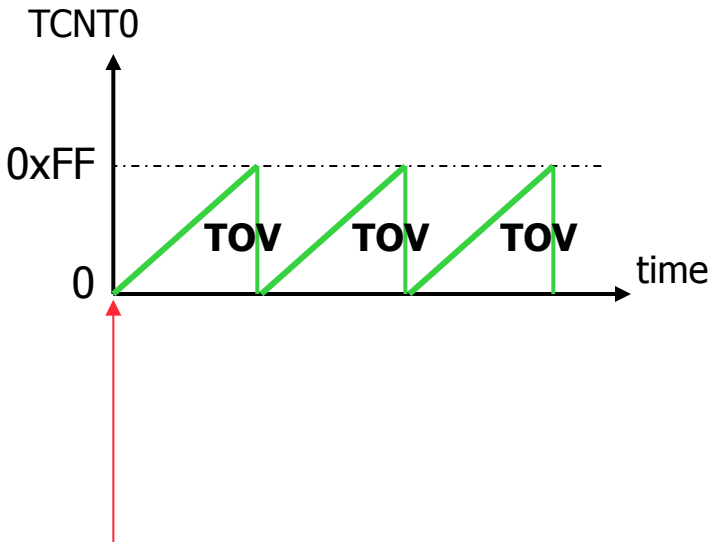
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

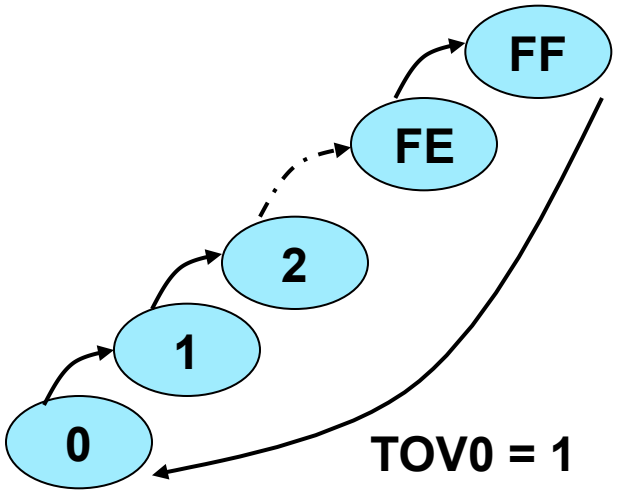
TIFR


FOC0
WGM00
COM01
COM00
WGM01
CS02
CS01
CS00
TCCR0
Timer Mode (WGM)
WGM00 WGM01 Comment
0 0 Normal
0 1 CTC (Clear Timer on Compare Match)
1 0 PWM, phase correct
1 1 Fast PWM
T0
CS00
CS01
CS02
0 1 2 3 4 5 6 7
Timer/Counter0 clock source

43

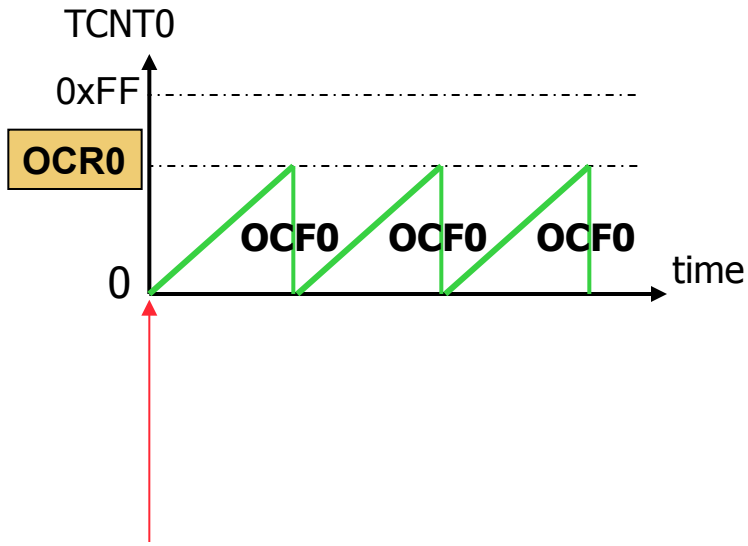
Normal mode

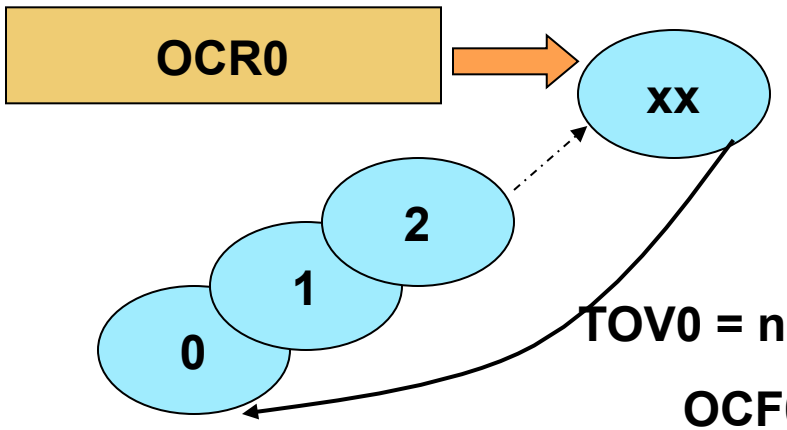




TOV0: **1**

CTC (Clear Timer on Compare match) mode





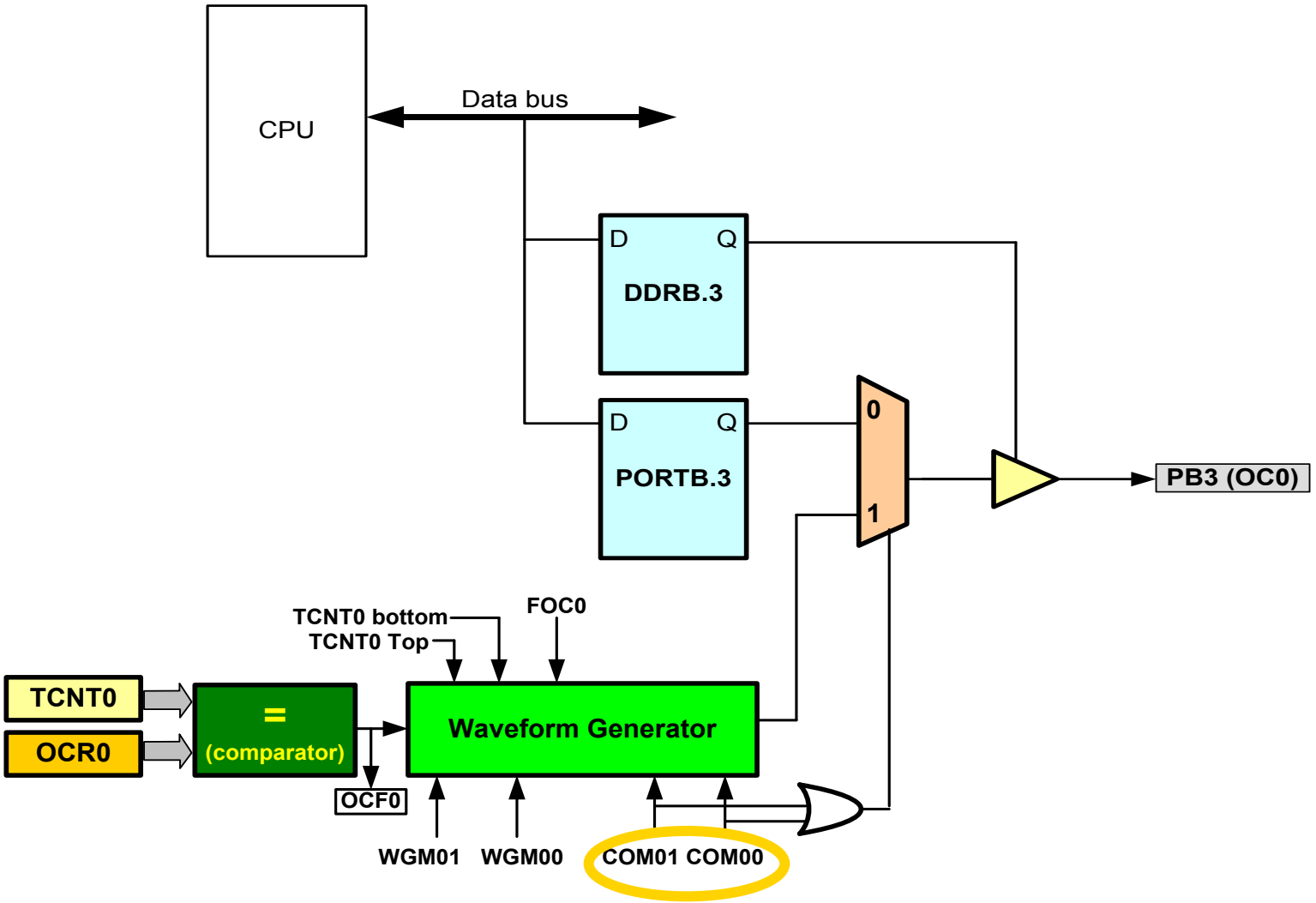
TOV0 = no change

OCF0 = 1

TOV0: 0

OCF0: 1

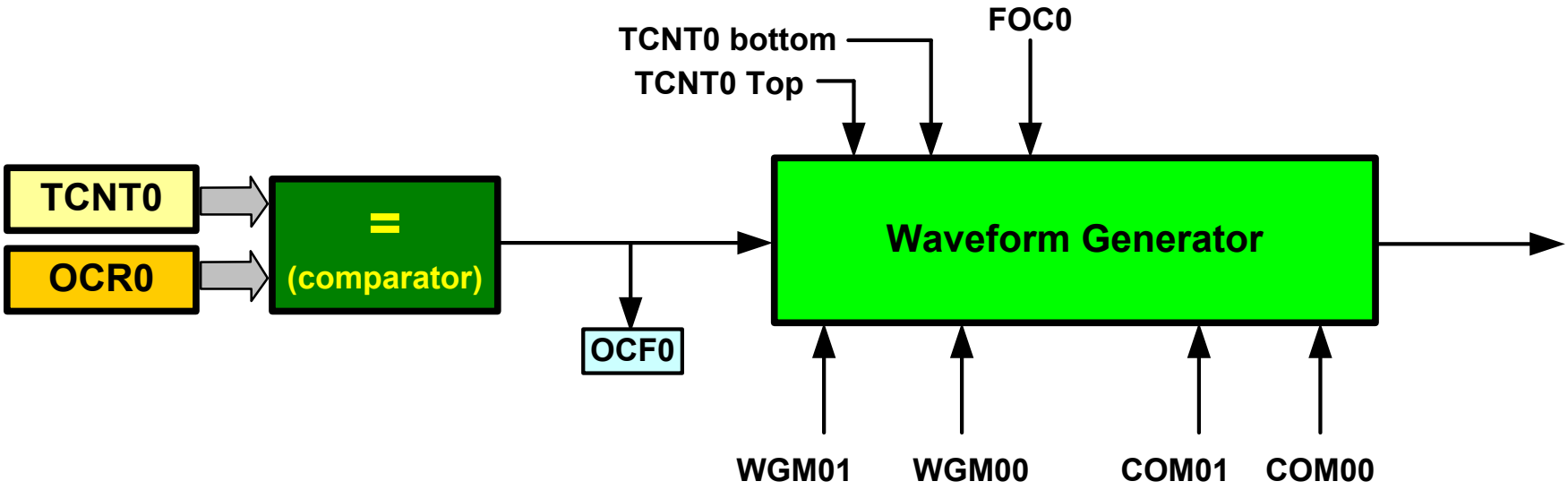
Waveform Generator



FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

TCCR0

Waveform Generator



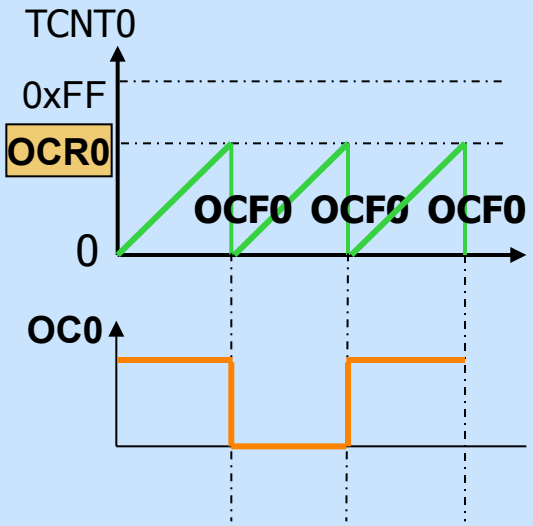
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

TCCR0

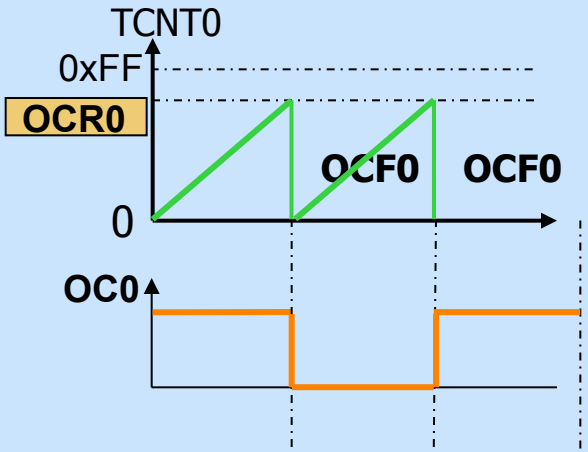
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

TCCR0

CTC, Toggle Mode
Duty cycle = 50%
Frequency = changeable



$$F_{OC0} = \frac{f_{clk}}{2N(OCR0+1)}$$



(COM)

COM
0
0
1
1

48

Assuming XTAL = 8 MHz, make a pulse with duty cycle = 50% and frequency = 500KHz

$$F_{OC0}=\frac{f_{clk}}{2N(OCR0+1)} \Rightarrow 500KHz=\frac{8MHz}{2N(OCR0+1)} \Rightarrow N(OCR0+1)=\frac{8MHz}{1MHz}$$

$$\Rightarrow N(OCR0+1)=8 \Rightarrow \begin{cases} N=1 \text{ and } OCR0=7 \\ N=8 \text{ and } OCR0=0 \end{cases}$$

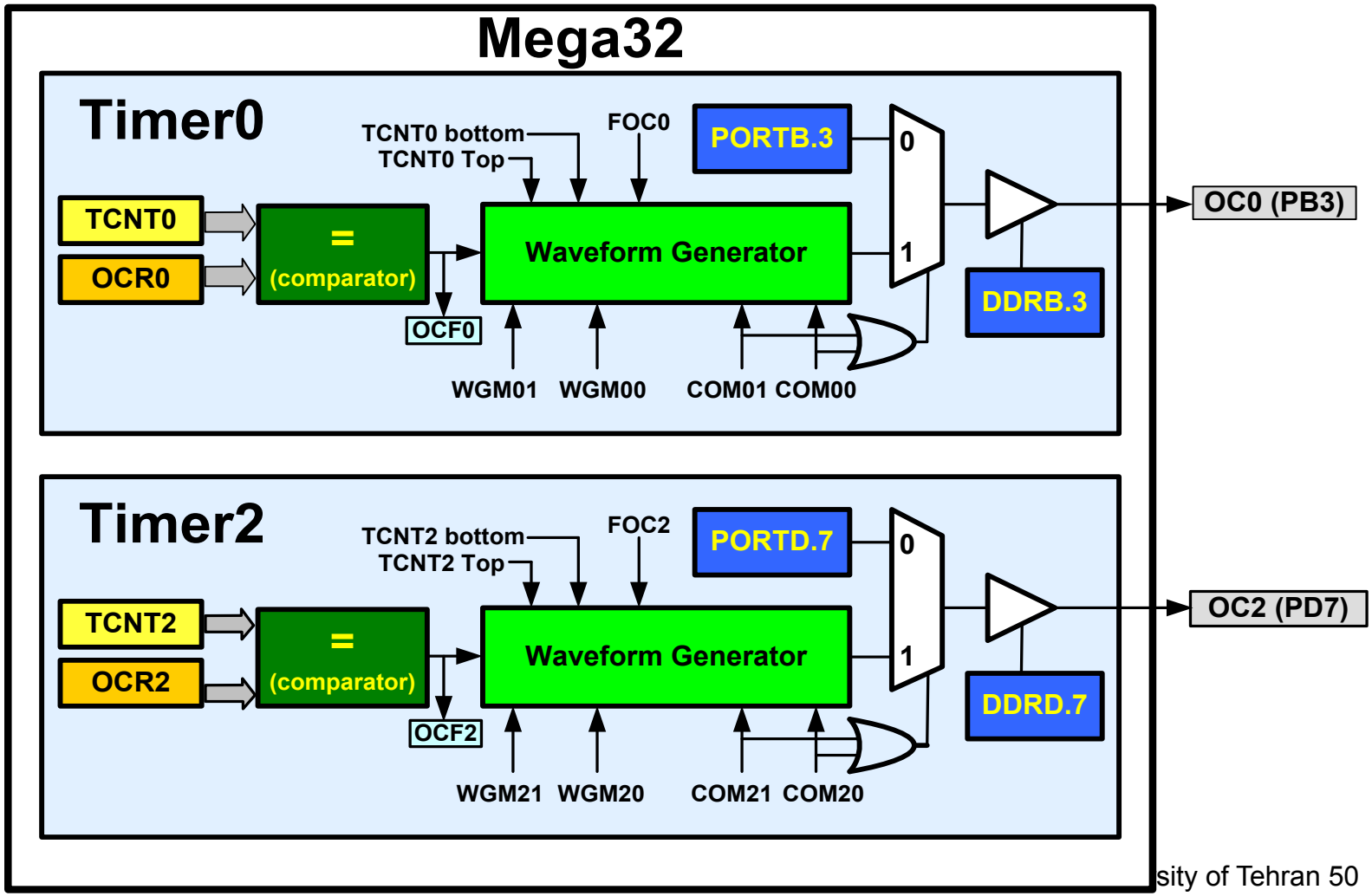
```
LDI R20,7
OUT OCR0,R20
LDI R20,0x19
OUT TCCR0,R20
```

```
OCR0 = 7;
TCCR0 = 0x19; //prescaler = 1
```

```
LDI R20,0
OUT OCR0,R20
LDI R20,0x1A
OUT TCCR0,R20
```

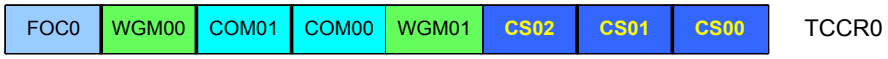
```
OCR0 = 0;
TCCR0 = 0x1A; //prescaler = 8
```

Wave generating in Timer2



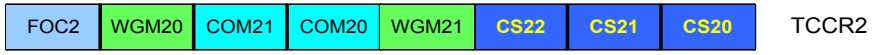
The difference between Timer0 and Timer2

- **Timer0**



CS02	CS01	CS00	Comment
0	0	0	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 (falling edge)
1	1	1	External clock (rising edge)

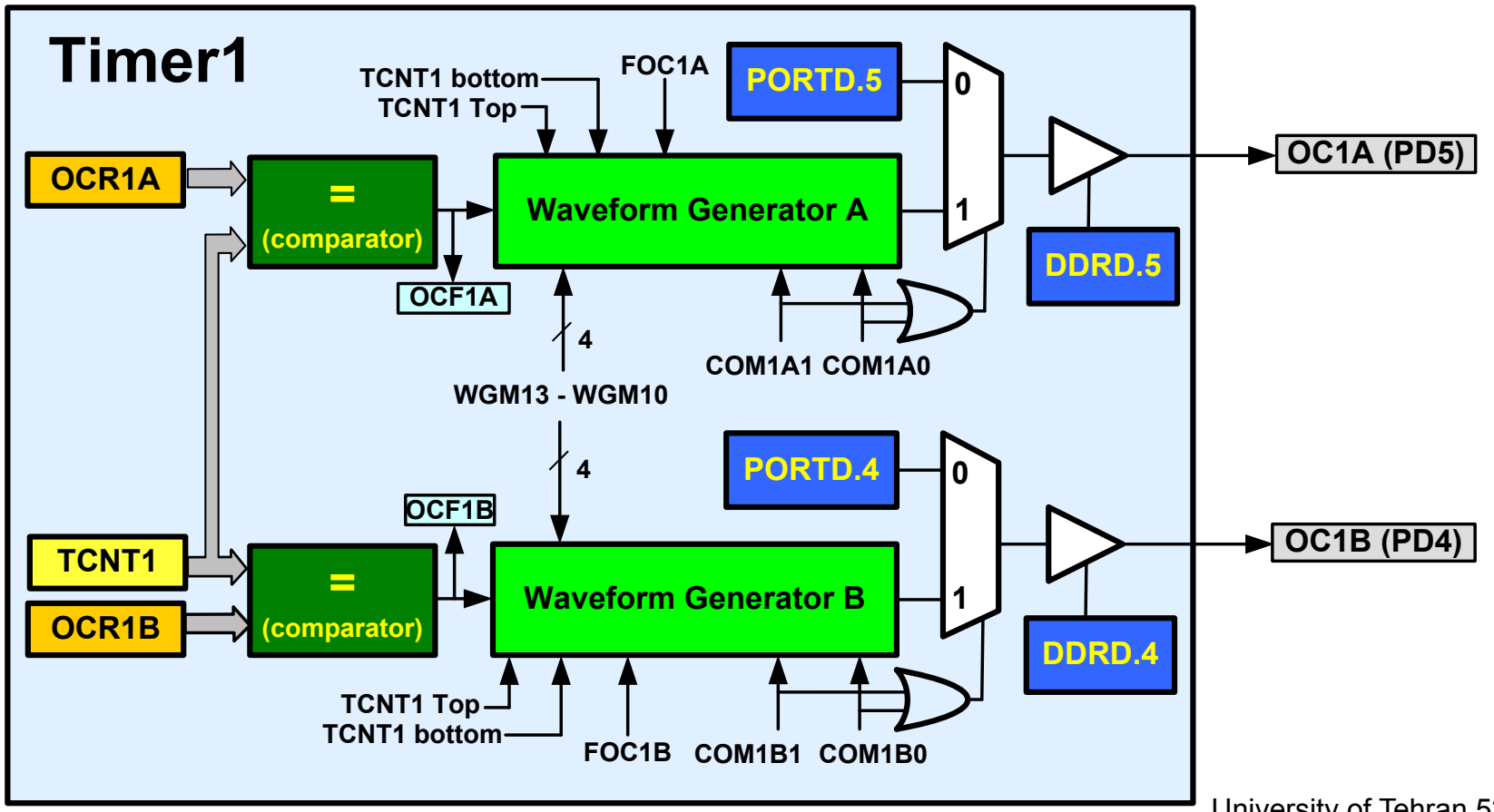
- **Timer2**

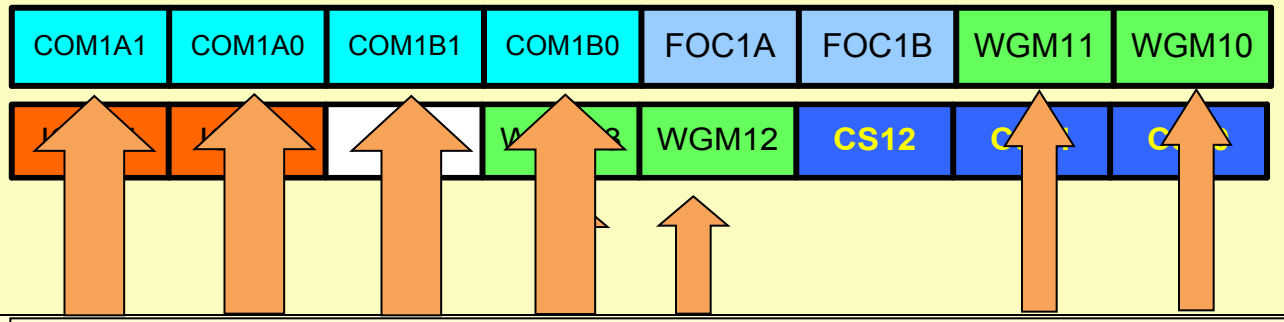


CS22	CS21	CS20	Comment
0	0	0	Timer/Counter stopped
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 32
1	0	0	clk / 64
1	0	1	clk / 128
1	1	0	clk / 256
1	1	1	clk / 1024

Timer1

- **Timer1 has two waveform generators**





In non PWM modes

COM1A1:COM1A0 D7 D6 Compare Output Mode for Channel A

COM1A1	COM1A0	Description
0	0	Normal port operation, OC1A disconnected
0	1	Toggle OC1A on compare match
1	0	Clear OC1A on compare match
1	1	Set OC1A on compare match

COM1B1:COM1B0 D5 D4 Compare Output Mode for Channel B

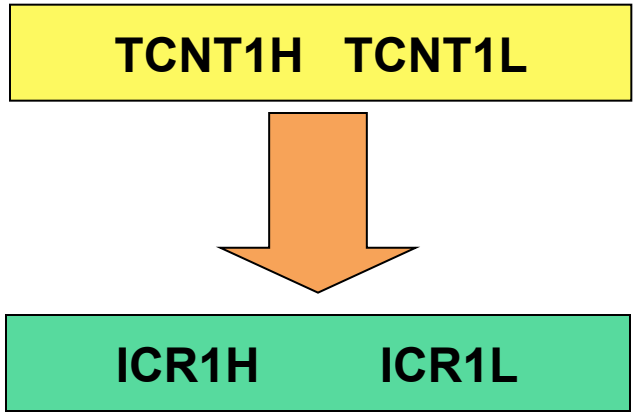
COM1B1	COM1B0	Description
0	0	Normal port operation, OC1B disconnected
0	1	Toggle OC1B on compare match
1	0	Clear OC1B on compare match
1	1	Set OC1B on compare match

Capturing in Timer/counter 1

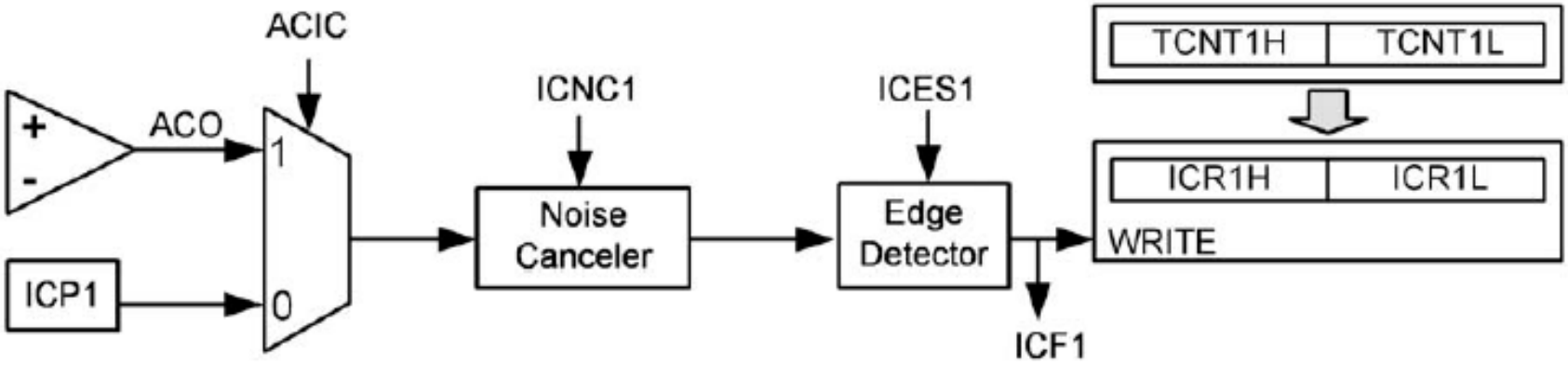
The AVR microcontroller
and embedded
systems
using assembly and c

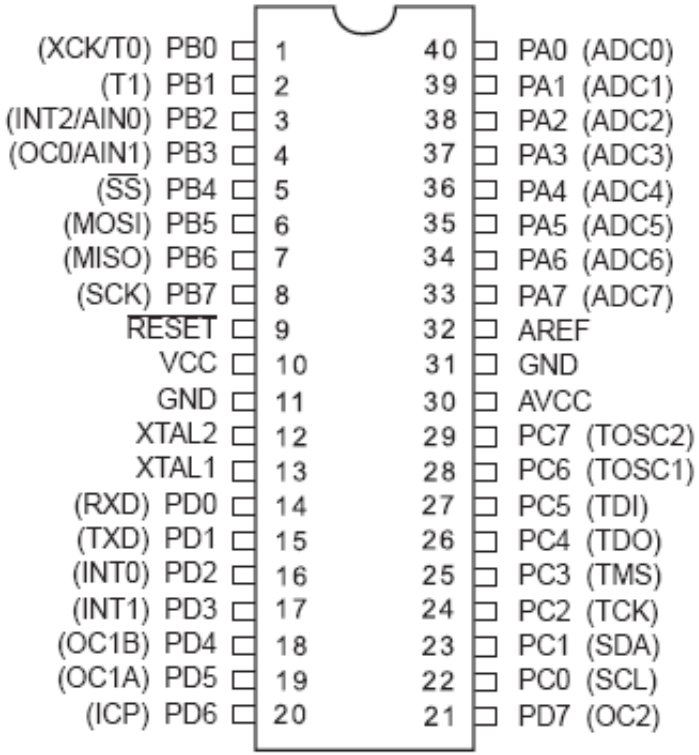
Capturing

- **Usages**
  - **Measuring duty cycle**
  - **Measuring period**



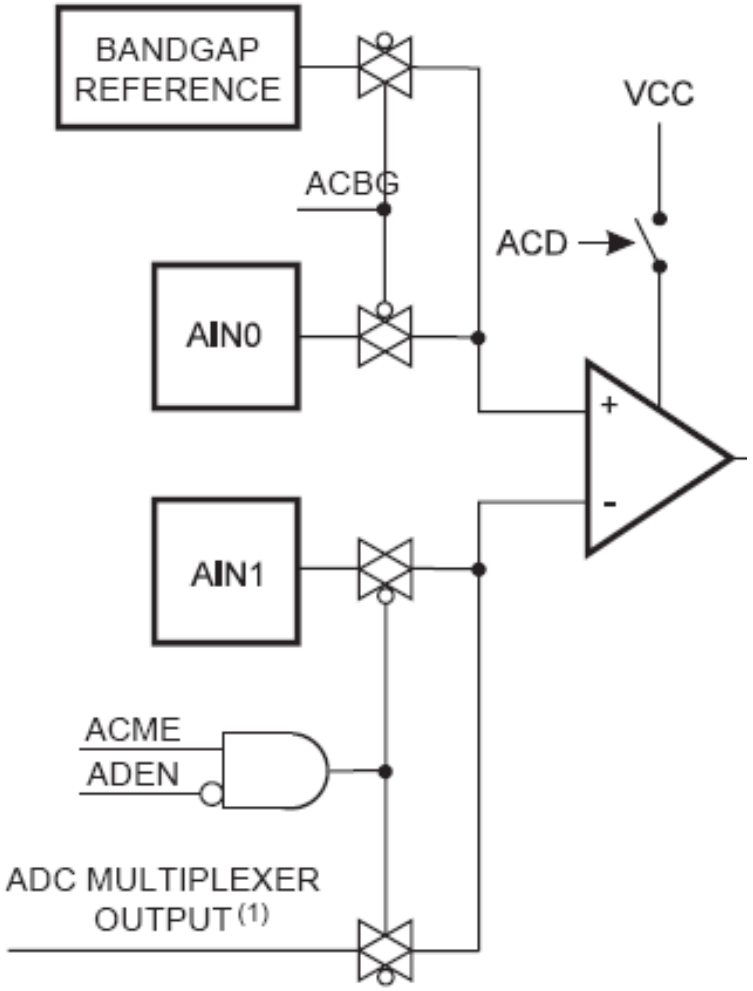
Capturing



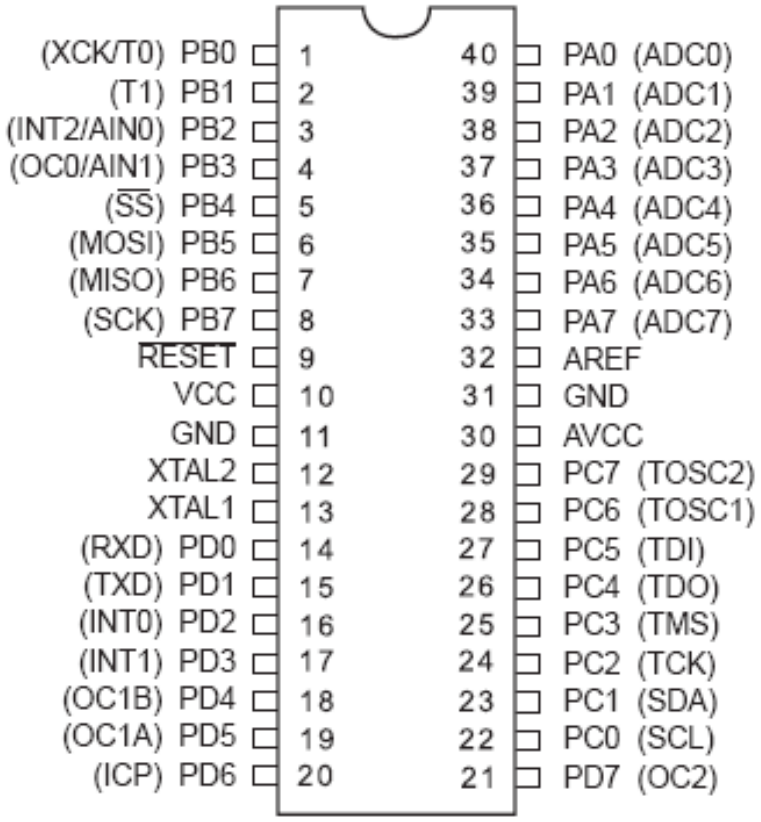


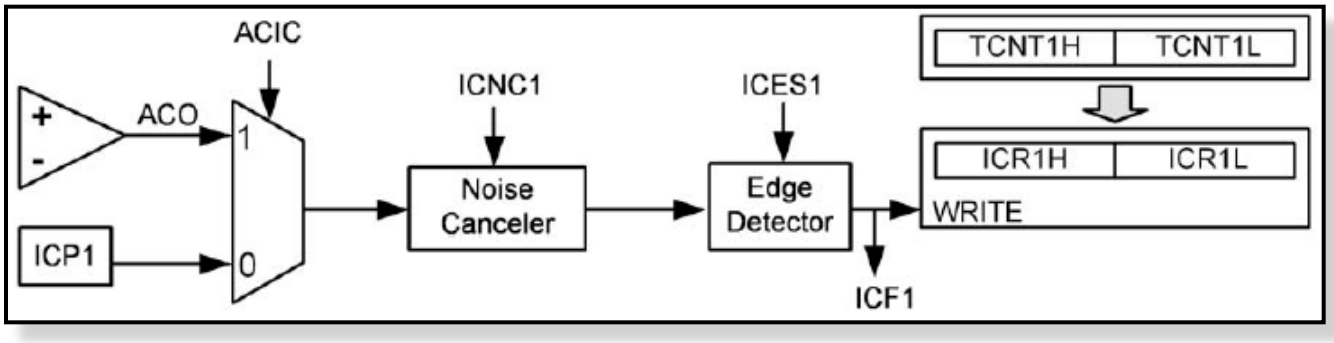
56

Comparator



Copied from ATmega32 datasheet page 196





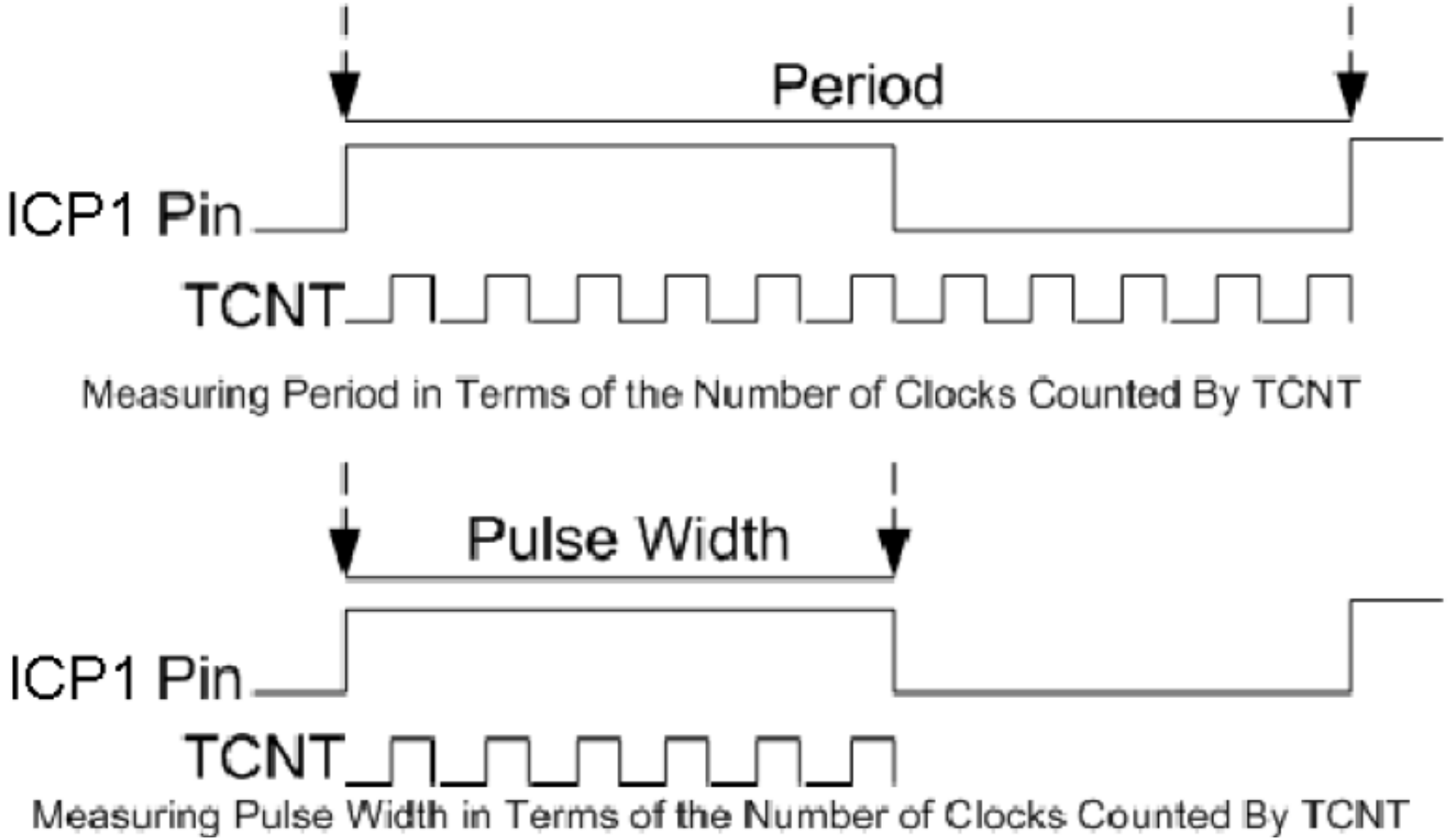
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B

ICNC1: Input Capture Noise Canceller
0:disabled
1:Enabled (captures after 4 successive equal valued samples)
ICSES1: Input Capture Edge Select
0: Falling edge
1: Rising edge

7	6	5	4	3	2	1	0	
ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

ACIC: Analog Comparator Input Capture Enable
0: ICP1 provides the capture signal
1: analog comparator is connected to the capturer

Measuring duty cycle and period



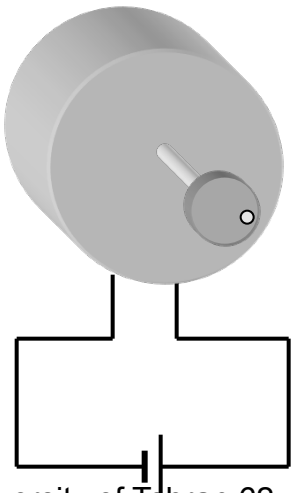
DC motor and PWM

The AVR microcontroller
and embedded
systems
using assembly and c

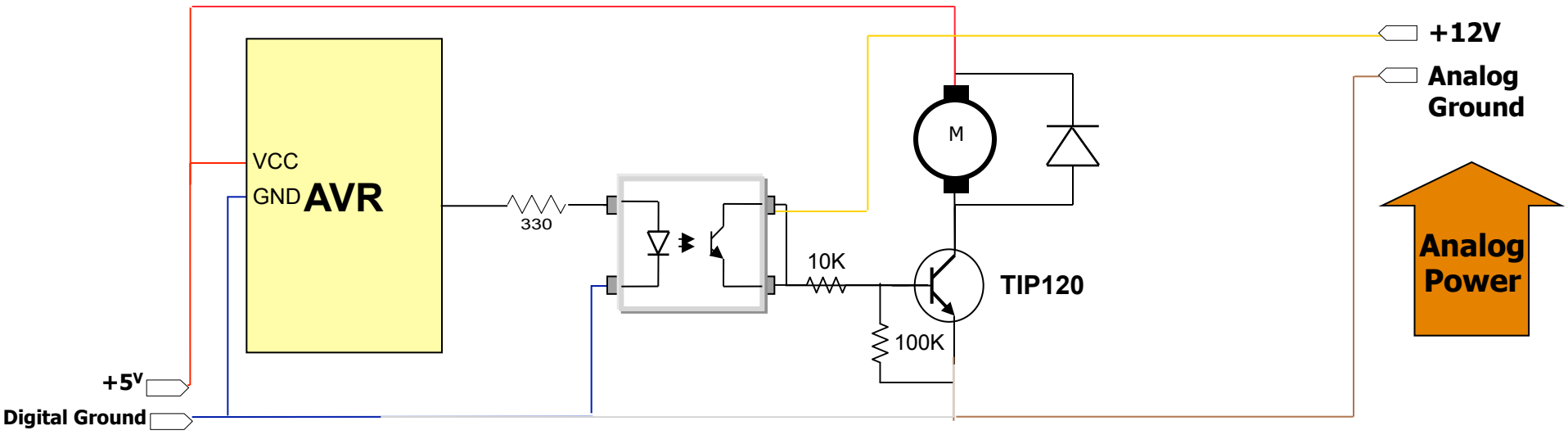
Topics

- DC motor
  - Unidirectional control
  - Bidirectional control
- PWM modes
  - Wave generating using Fast PWM
  - Wave generating using Phase correct PWM

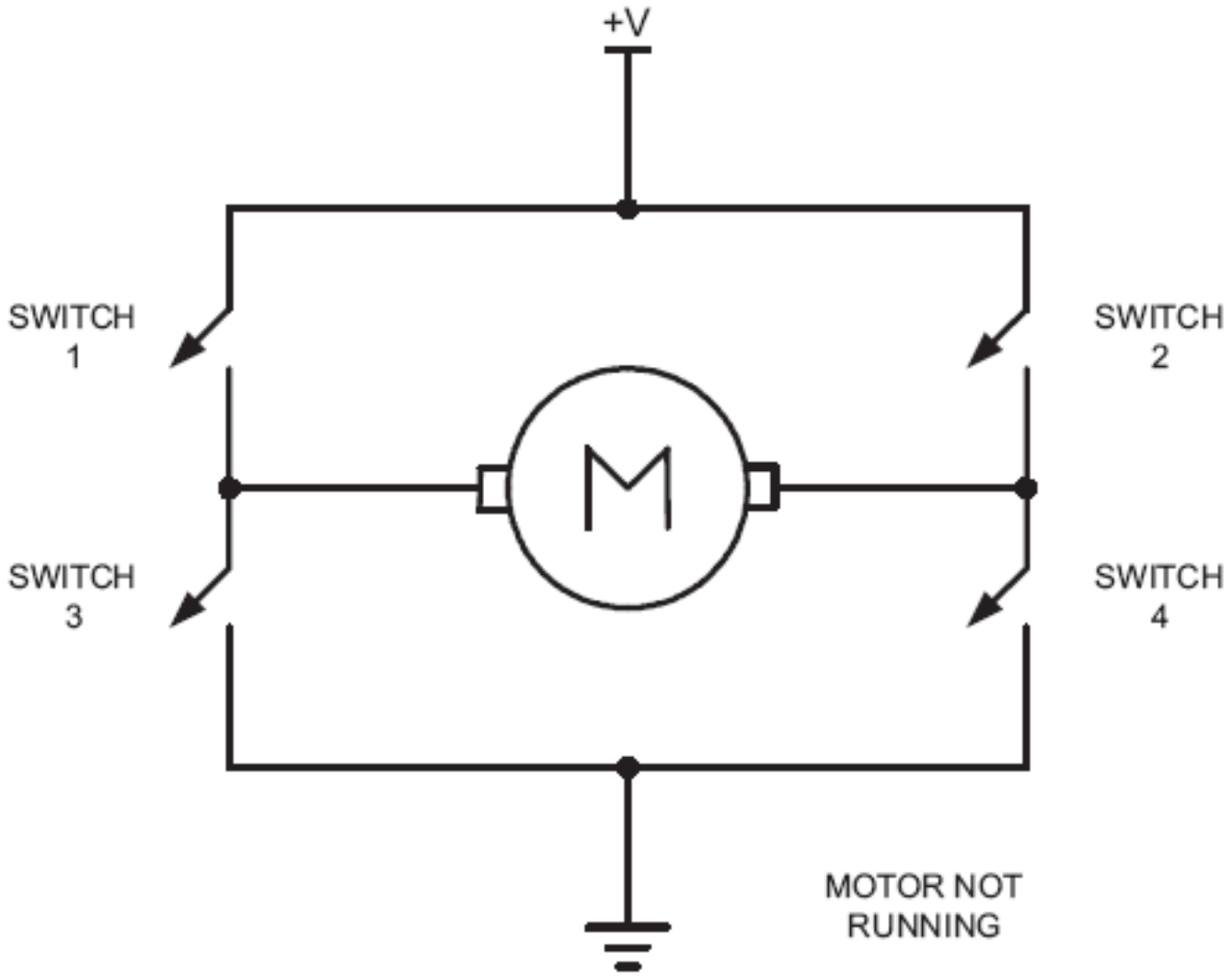
DC motor



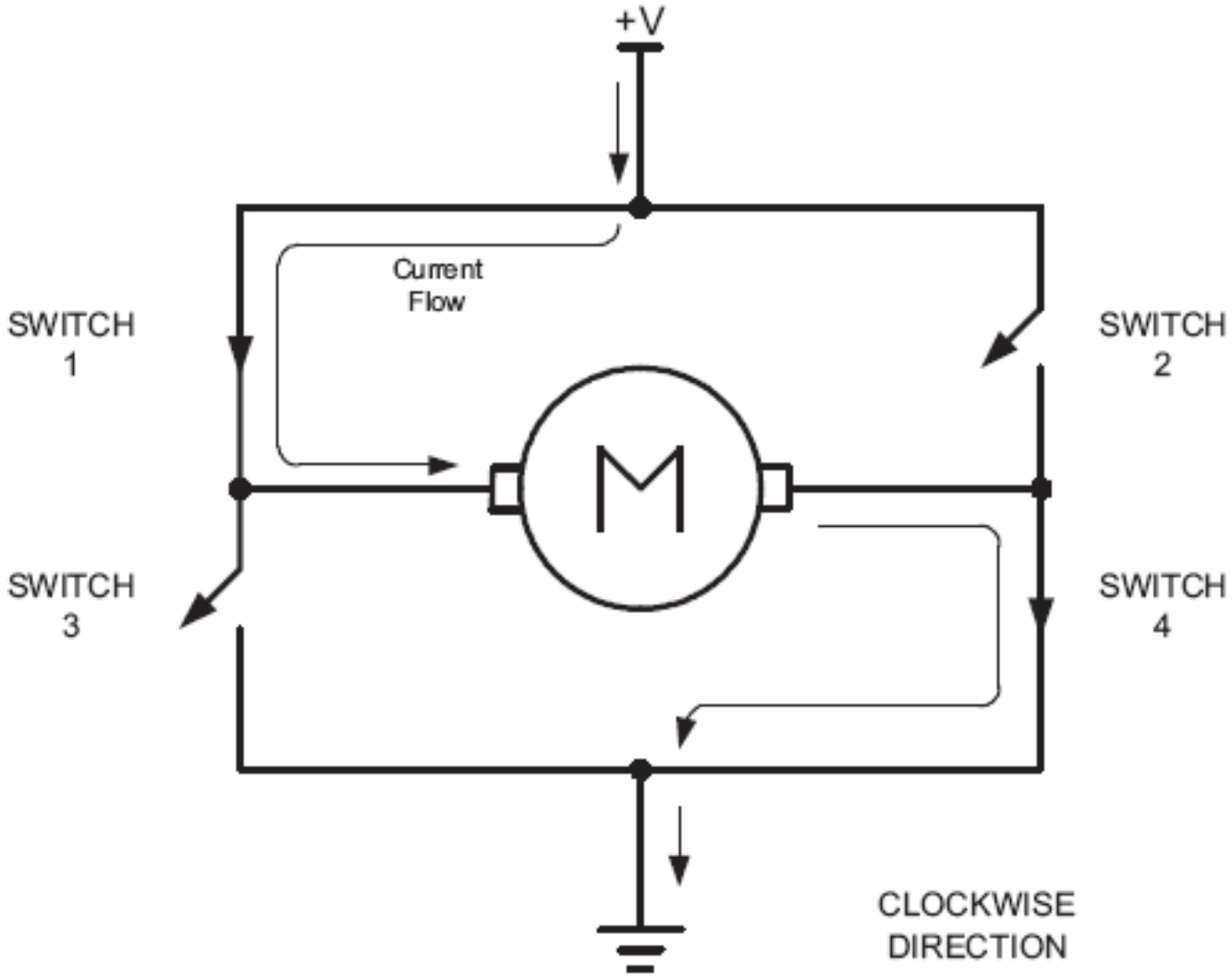
Unidirectional control



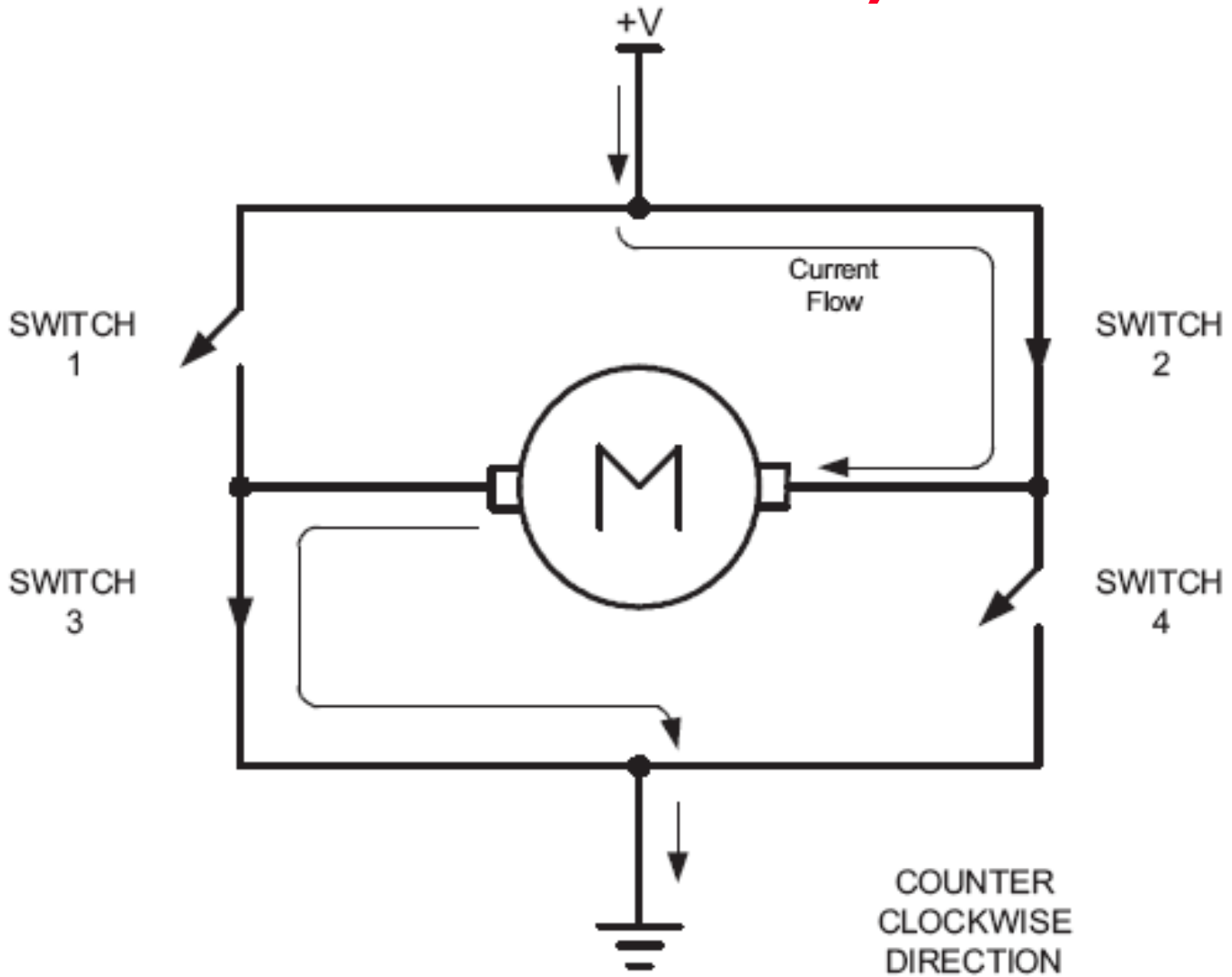
Bidirectional control



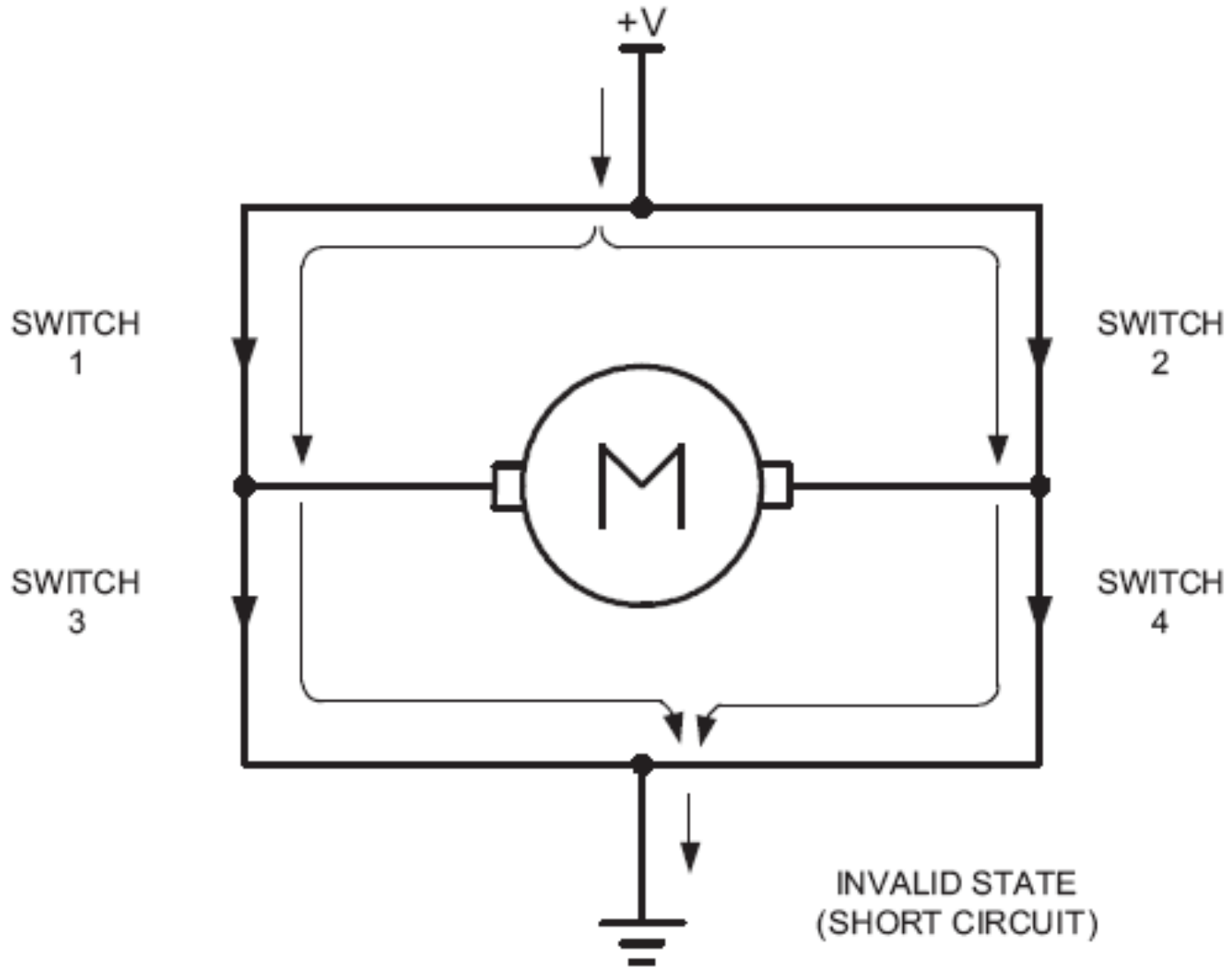
Bidirectional (clock wise)



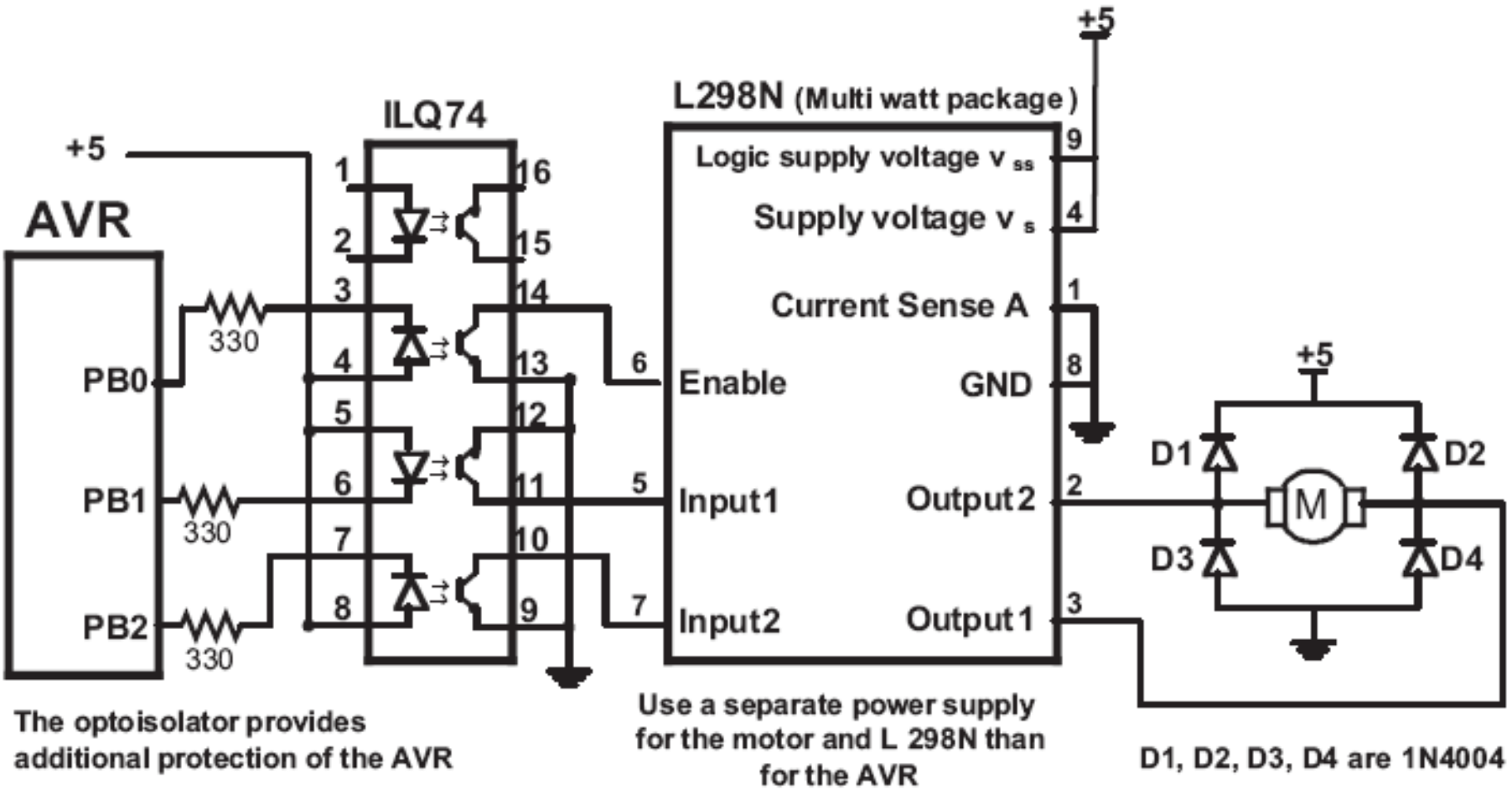
Bidirectional (counter clockwise)



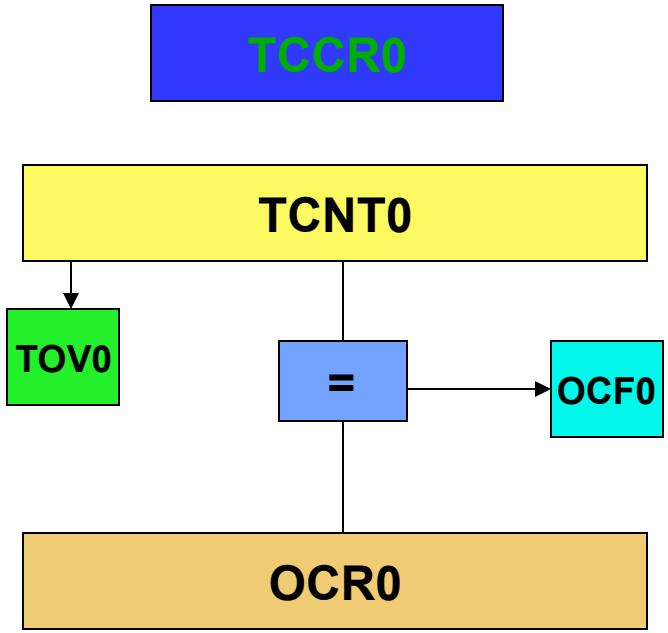
Bidirectional



Using L298N

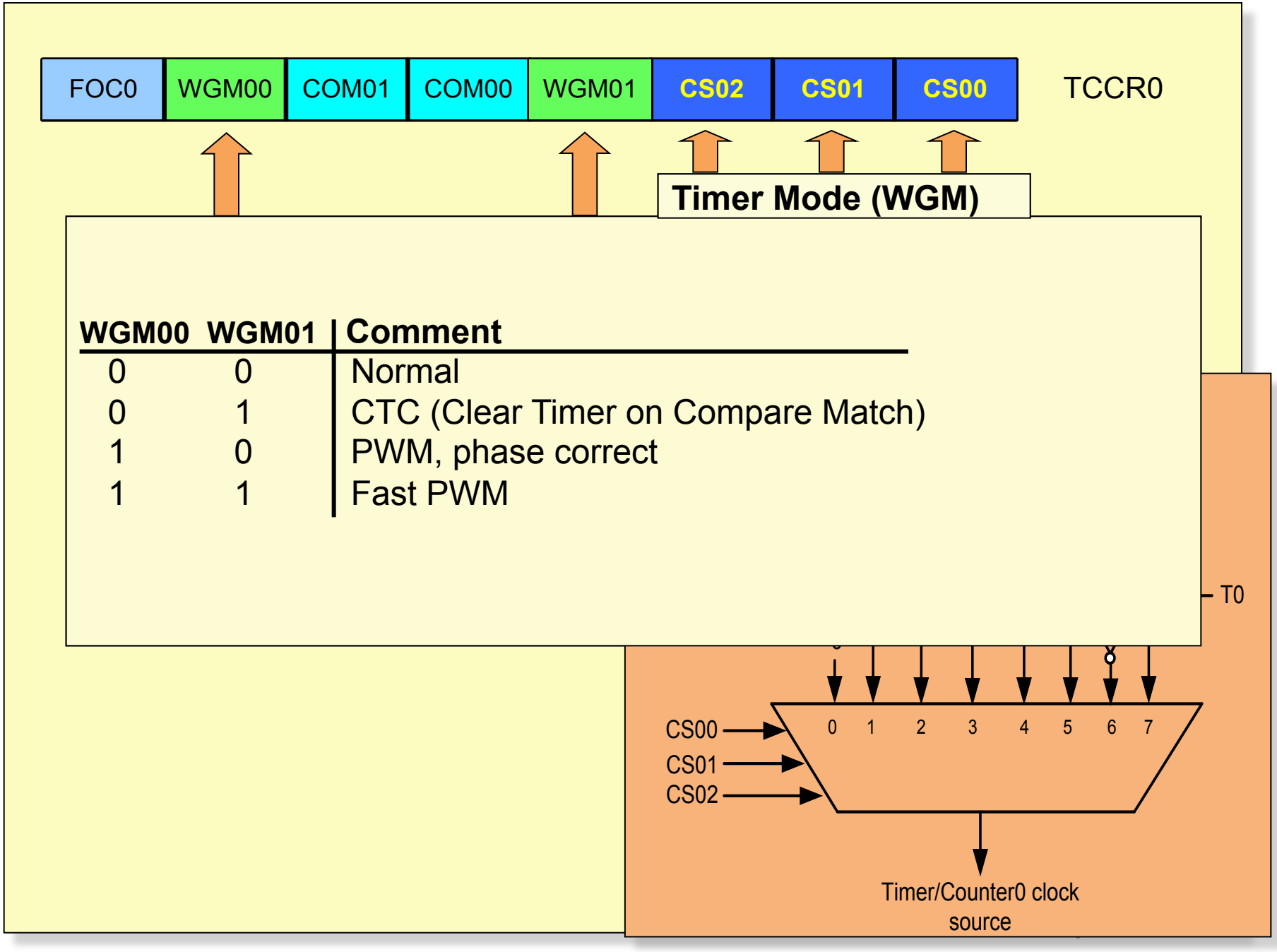

+5
ILQ74
L298N (Multi watt package)
AVR
Logic supply voltage v ss
Supply voltage v s
Current Sense A
Enable
GND
Input1
Output2
Input2
Output1
PB0
PB1
PB2
330
330
330
D1
D2
D3
D4
M
The optoisolator provides additional protection of the AVR
Use a separate power supply for the motor and L 298N than for the AVR
D1, D2, D3, D4 are 1N4004

Timer0 Review



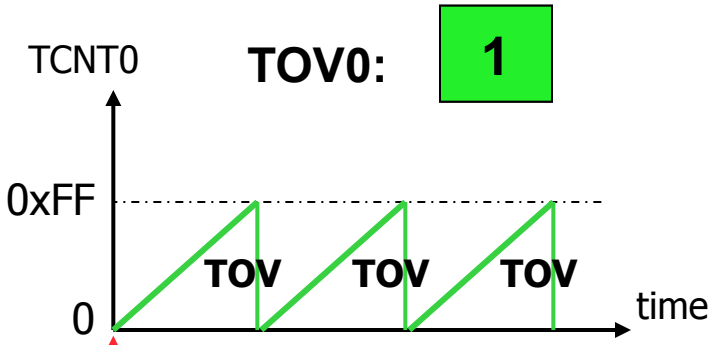
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

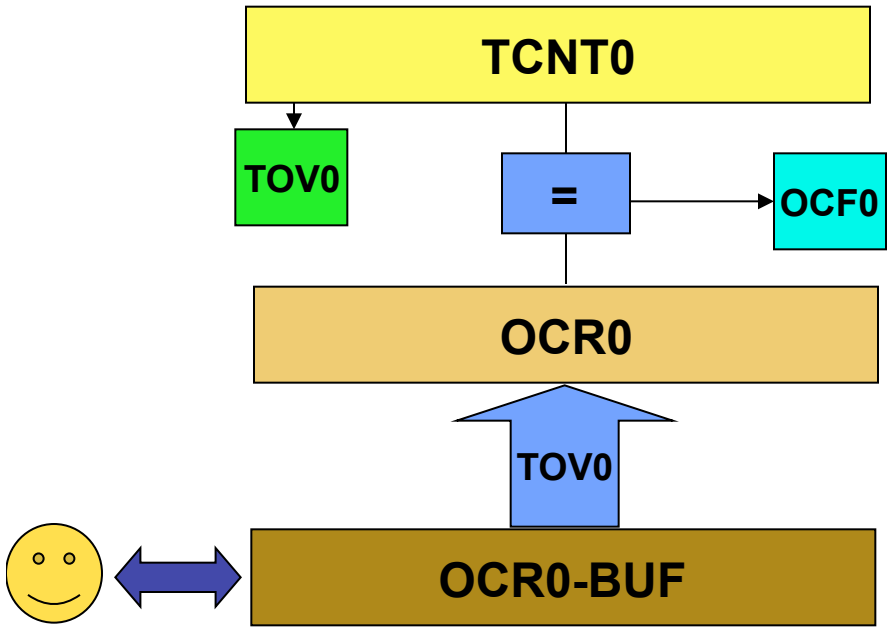
TIFR

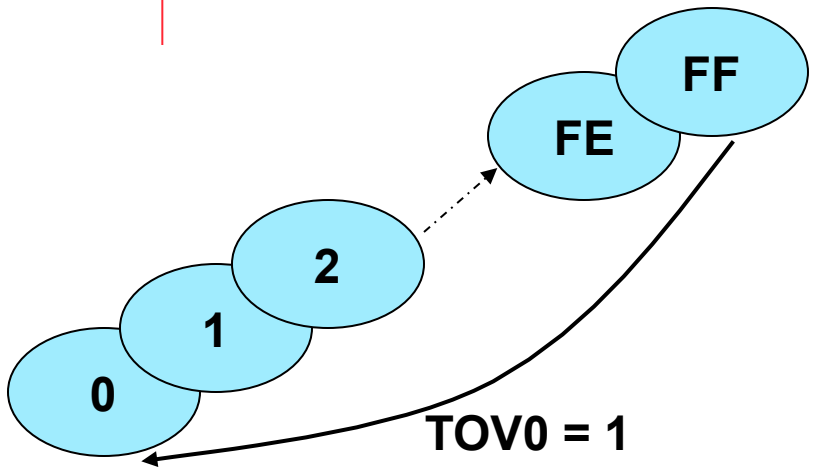

FOC0
WGM00
COM01
COM00
WGM01
CS02
CS01
CS00
TCCR0
Timer Mode (WGM)
WGM00 WGM01 Comment
0 0 Normal
0 1 CTC (Clear Timer on Compare Match)
1 0 PWM, phase correct
1 1 Fast PWM
T0
CS00
CS01
CS02
0 1 2 3 4 5 6 7
Timer/Counter0 clock source

Fast PWM mode

- **Similar to Normal mode but OCR0 is buffered.**

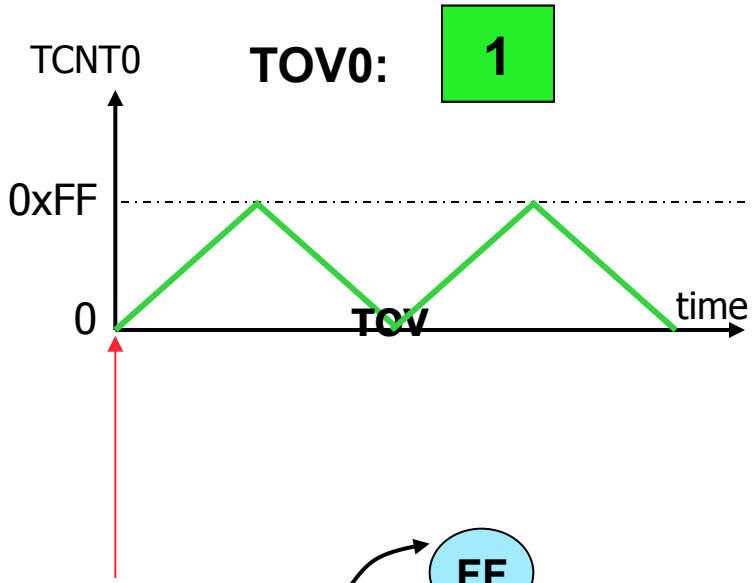


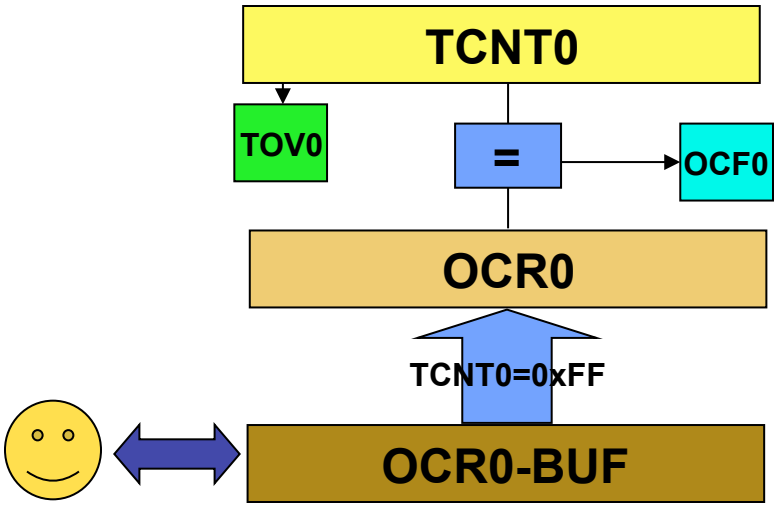


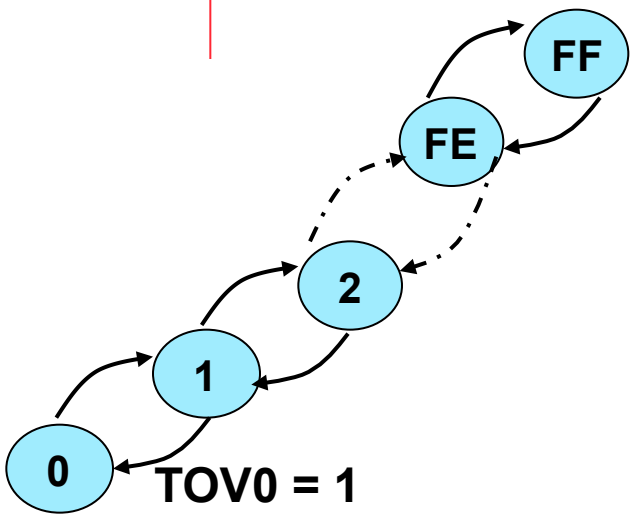


Phase Correct PWM mode

- Goes up and down like a yo-yo
- When TCNT becomes zero, the TOV0 flag sets.







FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

TCCR0

Compare Output Mode (COM)

CTC or Normal (Non PWM)

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Fast PWM

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at TOP
1	1	Set OC0 on compare match, clear OC0 at TOP

Note: 1. A special case occurs when OCR0 equals TOP and COM01 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See "Fast PWM Mode" on page 73 for more details.

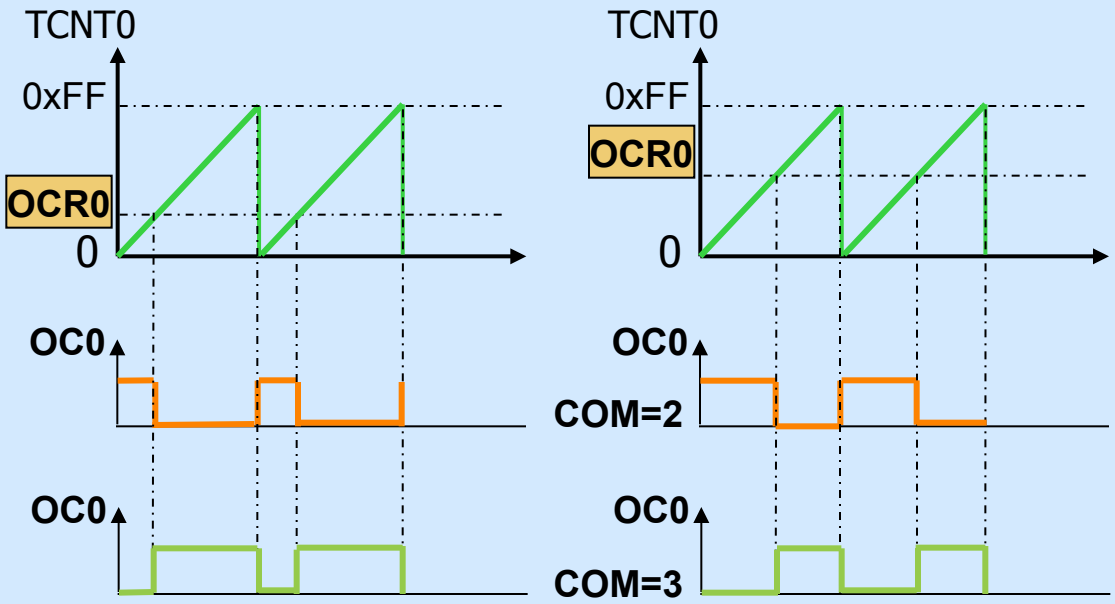
Phase Correct PWM

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

TCCR0

Fast PWM
Duty cycle = changeable (0% to 100%)
Frequency = selectable between limited choices



$$\text{Duty Cycle} = \frac{OCR0 + 1}{256} \times 100$$

$$\text{Duty Cycle} = \frac{255 - OCR0}{256} \times 100$$

$$F_{OC0} = \frac{f_{clk}}{N(256)}$$

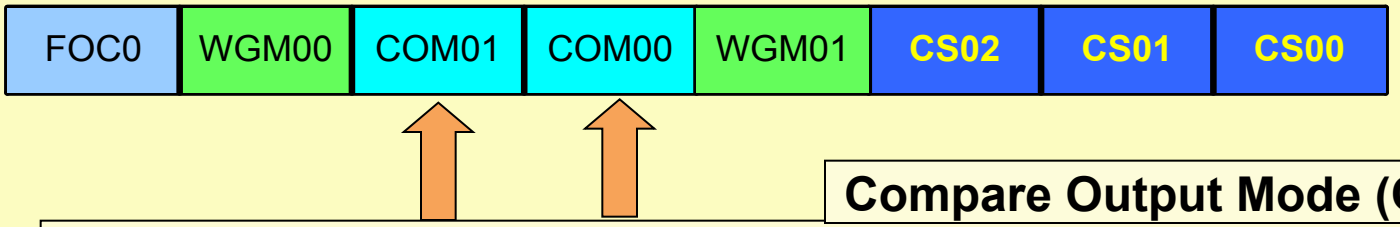
Assuming XTAL = 8 MHz, make the following pulse duty cycle = 75% and frequency = 31.250KHz

$$F_{OC0} = \frac{f_{clk}}{N(256)} \Rightarrow 31.250KHz = \frac{8MHz}{N(256)} \Rightarrow N = \frac{8MHz}{31.250K*256} = 1$$

75/100 = (OCR0+1)/255 ➔ OCR0+1 = 191 = 0xBF ➔ OCR0 = 0xBE

```
LDI R20,0xBE
OUT OCR0,R20
LDI R20,0x79
OUT TCCR0,R20
```

```
OCR0 = 0xBE;
TCCR0 = 0x79;
```



Compare Output Mode (COM)

CTC or Normal (Non PWM)

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Fast PWM

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at TOP
1	1	Set OC0 on compare match, clear OC0 at TOP

Note: 1. A special case occurs when OCR0 equals TOP and COM01 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See "Fast PWM Mode" on page 73 for more details.

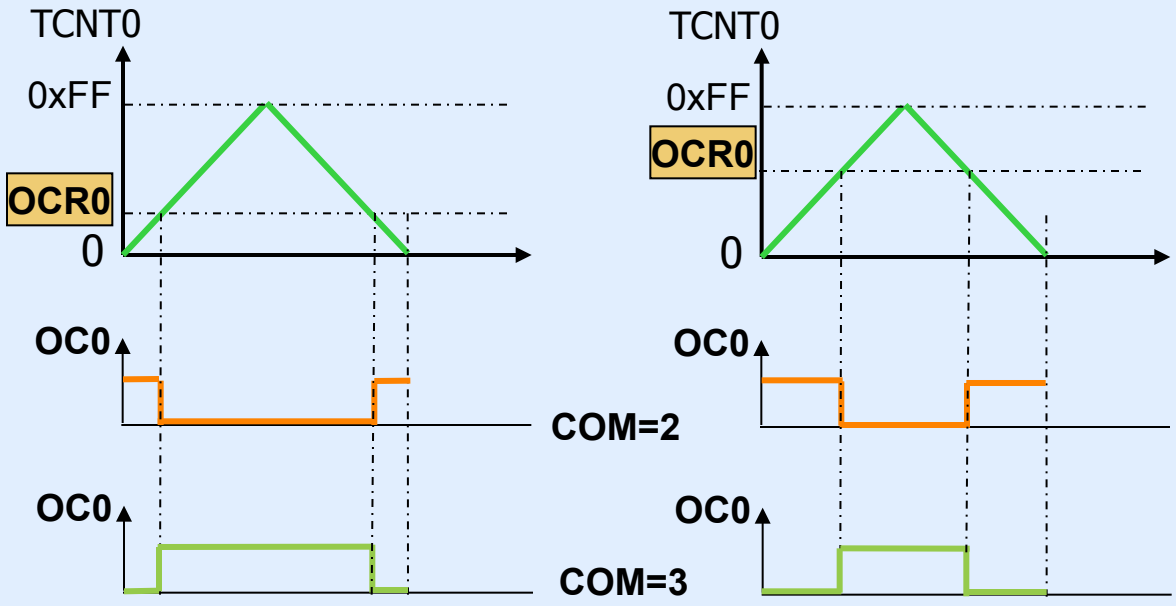
Phase Correct PWM

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

TCCR0

Phase Correct PWM
Duty cycle = changeable (0% to 100%)
Frequency = selectable between limited choices



$$\text{Duty Cycle} = \frac{OCRx}{255} \times 100$$

$$\text{Duty Cycle} = \frac{255 - OCR0}{255} \times 100$$

$$F_{OC0} = \frac{f_{clk}}{N(510)}$$

Alternate Pins for PDIP Package