ICT 5307 : Embedded System Design

Lecture 9
Timer1, Square Wave & PWM Generation

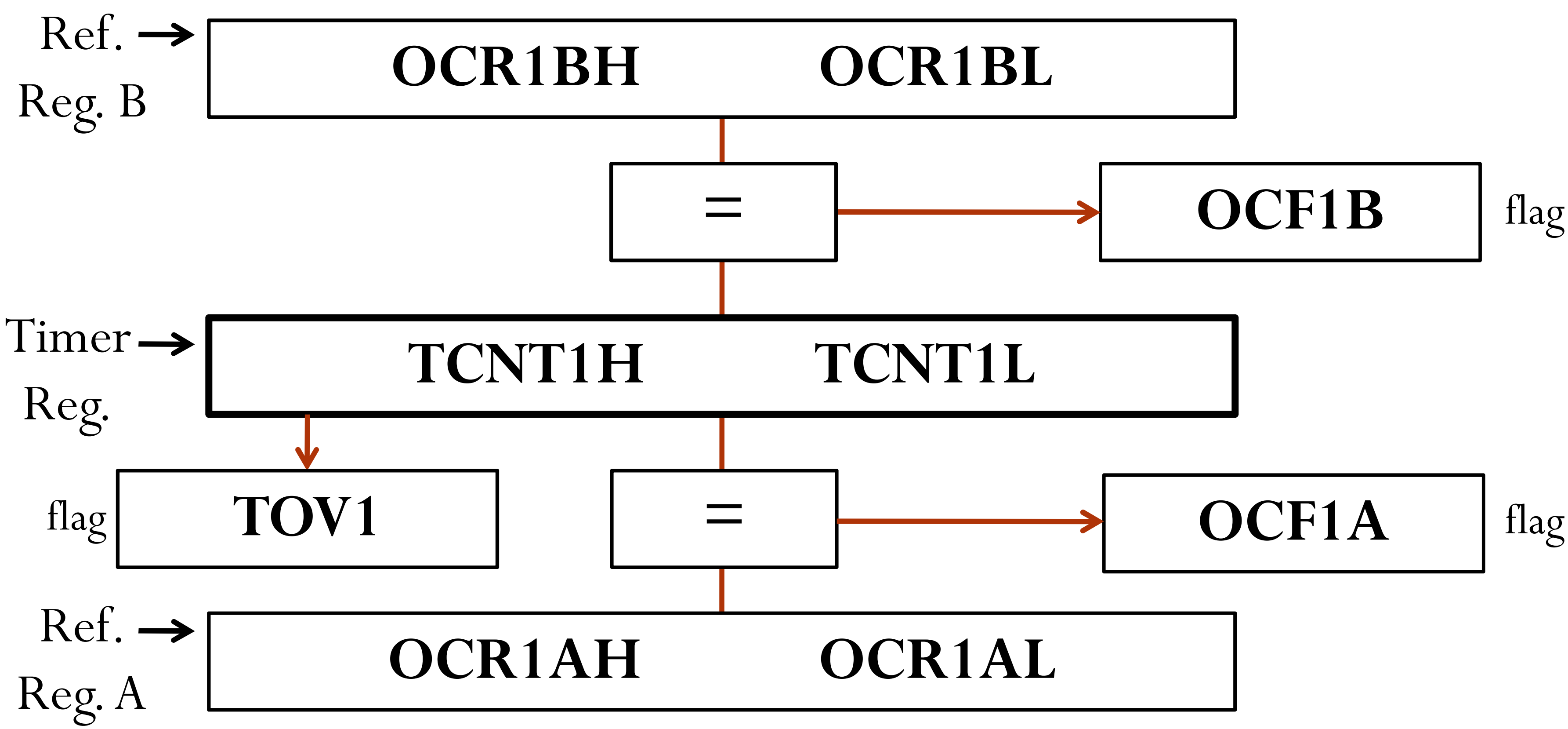
Professor S.M. Lutful Kabir

BUET

Timer1 Programming

- Timer1 is a 16-bit timer and has lots of capabilities.
- It is split into two bytes. These are referred to TCNT1L and TCNT1H.
- Timer1 has two control registers, namely TCCR1A (8-bit) and TCCR1B (8-bit).
- TOV1 flag bit goes high when overflow occurs.
- There are two OCR registers, namely OCR1A(16-bit) and OCR1B(16-bit).
- There are two separate flags for each of two OCR registers, which acts independently. The figure in the next slide explains how they work.

Comparisons and Overflow in Timer1



TIFR (Timer/Counter) Interrupt Flag Register

TIFR Register

OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

Timer 1

TOV1	Timer1 overflow flag bit;
OCF1B	Timer 1 output compare B match flag
OCF1A	Timer 1 output compare A match flag
ICF1	Input Capture flag

TCCR1A & TCCR1B (Timer Counter Control Registers)

TCCR1A Register

COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

TCCR1B Register

ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

FOC1A, FOC1B – Related to force compare

ICNC1, ICES1 – Related to Input Capture

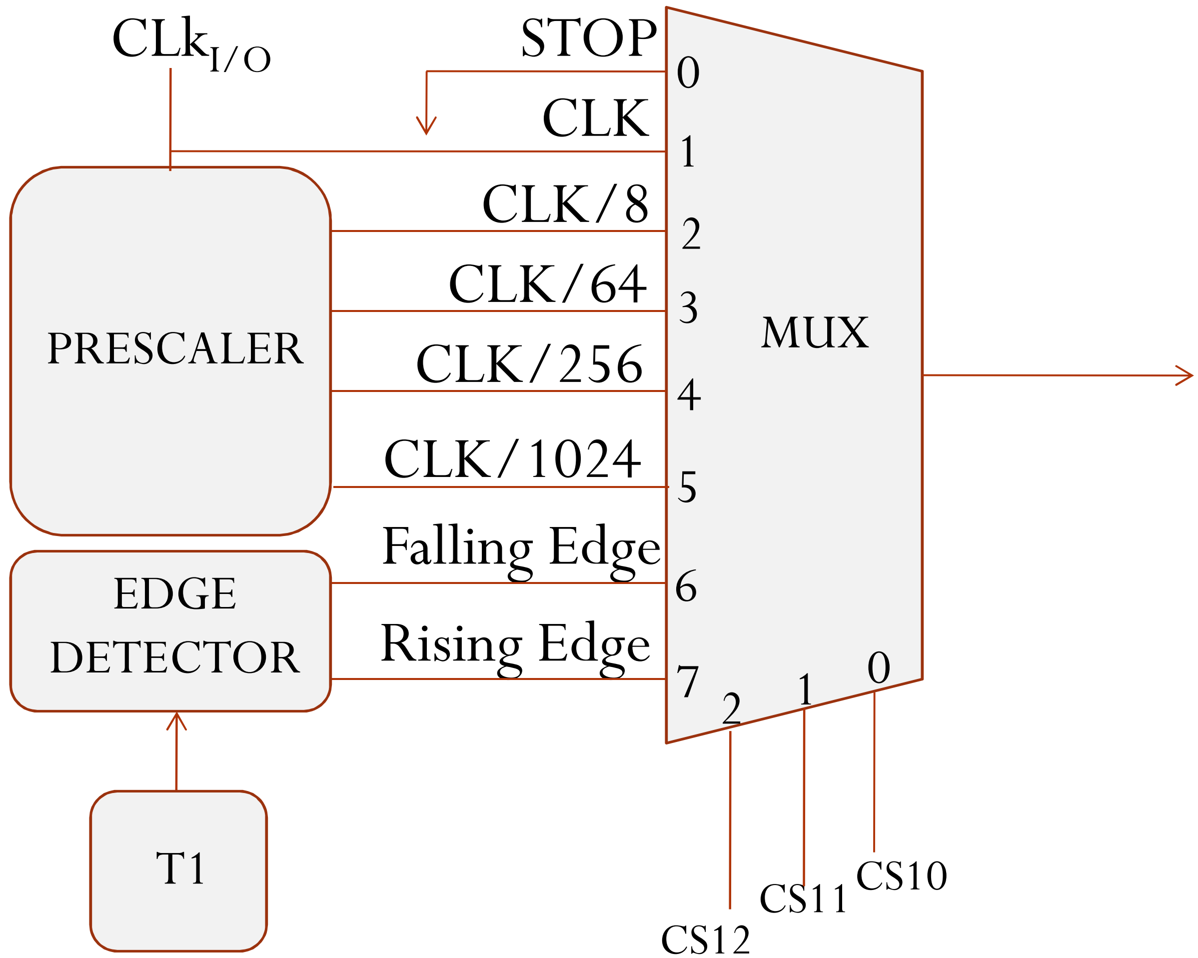
COM1A1, COM1A0, COM1B1, COM1B0

– Related to Waveform Generation

WGM13, WGM12, WGM11, WGM10 – Mode Selection

CS12, CS11, CS10 – Clock Selection

Block Diagram for Timer1



Mode Selection

Mode	WGM13	WGM12	WGM11	WGM10	Description of Mode
0	**0**	**0**	**0**	**0**	**Normal**
4	**0**	**1**	**0**	**0**	**CTC for OCR1A**
12	1	1	0	0	CTC for ICR1
Others	0/1	0/1	0/1	0/1	All related to PWM

An Exercise

- Write a program to toggle only the PORTB.5 bit continuously every second. Use timer 1, Normal mode, and 1:256 prescaler to create the delay. Assume XTAL=8 MHz.

Solution:

- XTAL=8 MHz → $T_{\text{machine cycle}}$ = 1/8 uSec=0.125 uS
- Prescaler=1:256 → T_{clock} = 256 X 0.125 = 32 uS
- So, number of clock necessary to make a delay of 1sec is (1s/32uS)=31250
- Therefore, the number to be loaded in the timer register is (65535-31250+1)=34286=0x85EE
- So, TCNT1L=0xEE and TCNT1H=0x85

The Program Using Timer1

```
#include ....
void T1Delay()
 int main() {
   DDRB=0xFF;
   while (1) {
        T1Delay();
        PORTB=~PORTB;
   }
}
```

```
void T1Delay ()
{
      TCNT1L=0XEE;
      TCNT1H=0X85;
      TCCR1A=0x00;
      TCCR1B=0x04;
      while ((TIFR&04)==0);
      TIFR=0x04;
}
```

Our Discussion So Far

- Our previous discussions of three timers were limited to NORMAL operation.
- All WGM and COM bits in the control register (TCCR) were set zero.
- Let us revisit the control register (for example TCCR0)

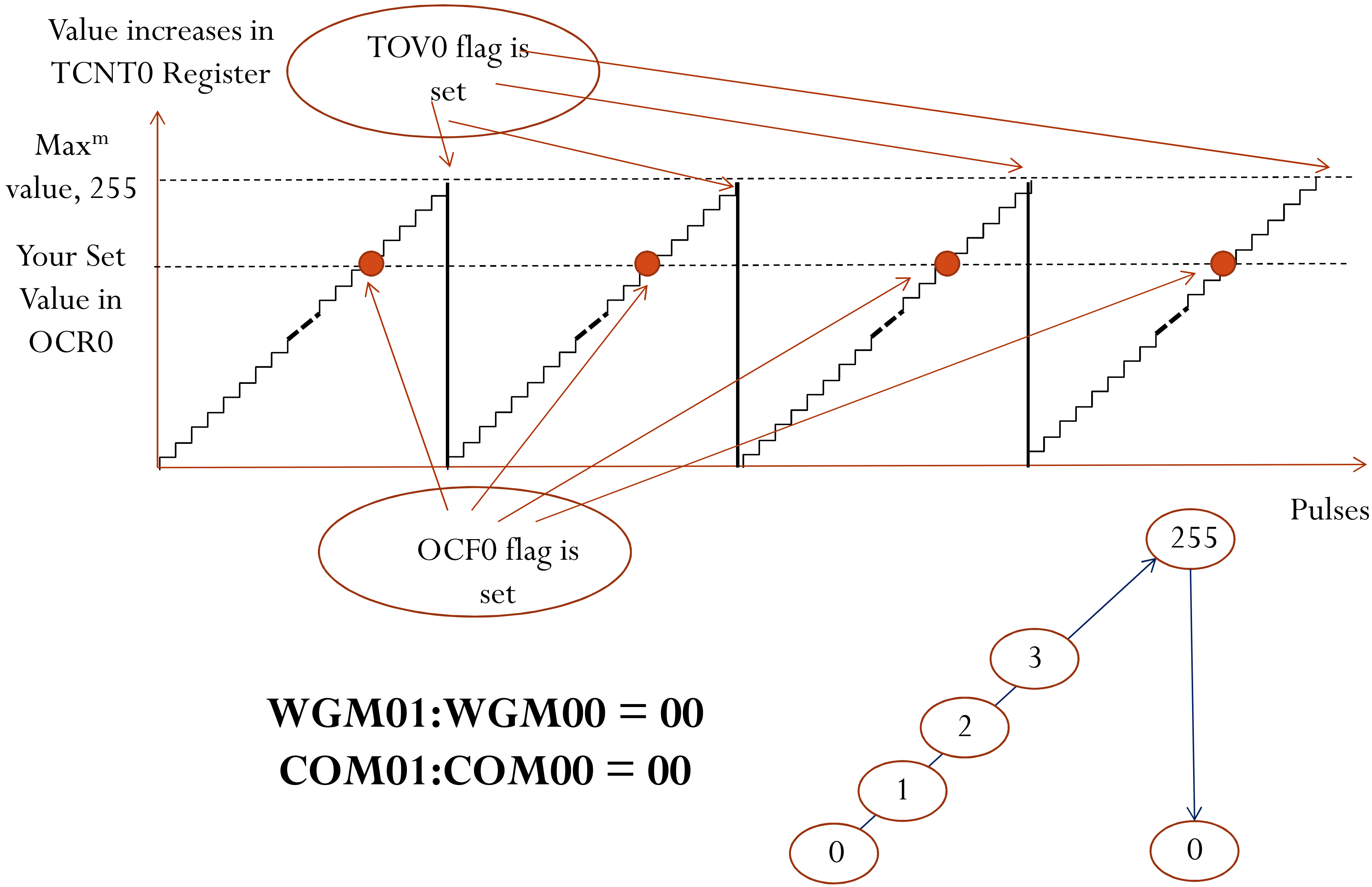
TCCR0 Register

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

	WGM00	WGM01
NORMAL	0	0
----------------	0	1
----------------	1	0
----------------	1	1

	COM01	COM00
NORMAL	0	0
----------------	0	1
----------------	1	0
----------------	1	1

NORMAL MODE : Value Increases in TCNT0 Register at Every Pulse and Resets When it Overflows



WGM01:WGM00 = 00
COM01:COM00 = 00

Variation # 1 of WGM and COM bits in TCCR Register

TCCR0 Register

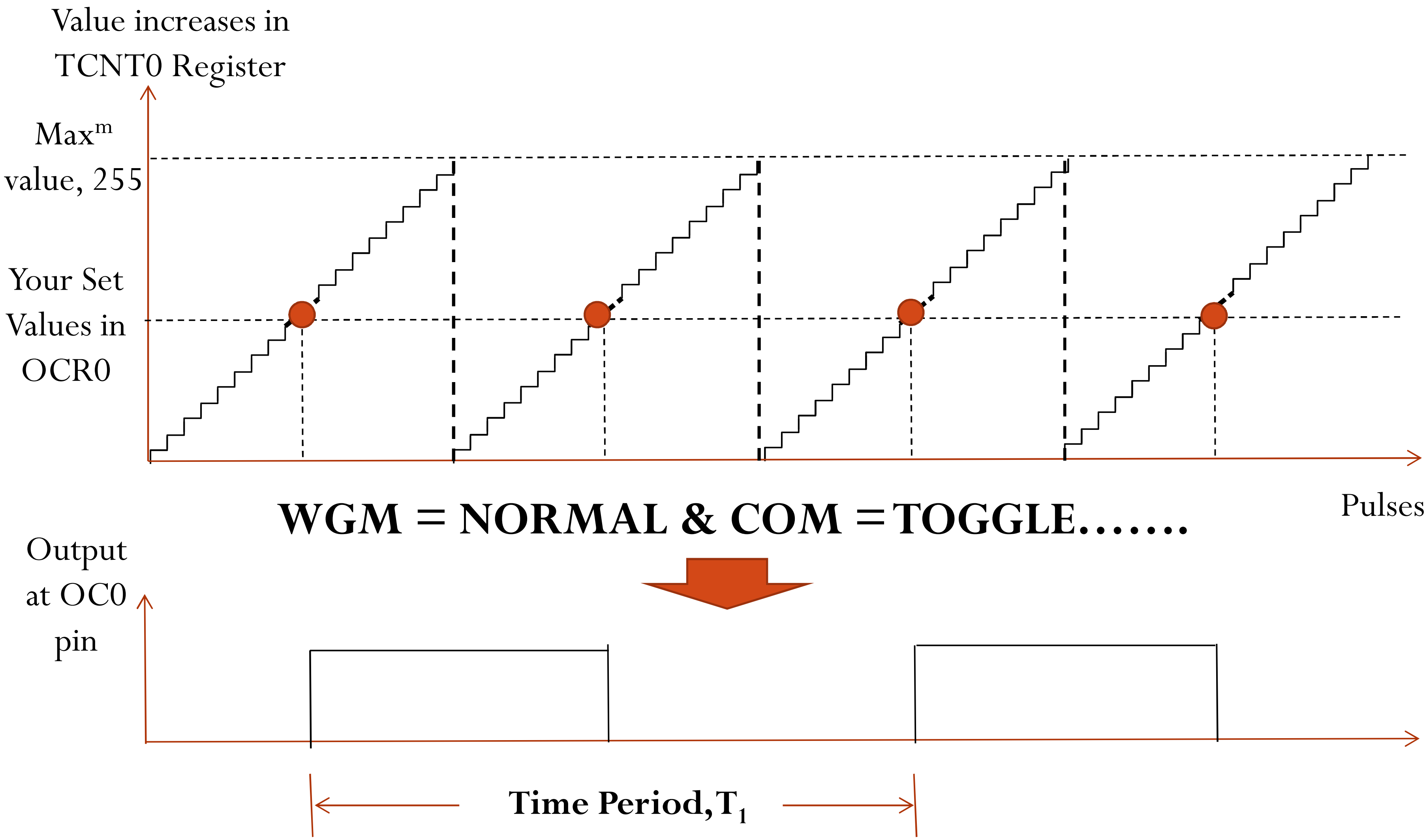
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

	WGM00	WGM01
NORMAL	0	0
----------------	0	1
----------------	1	0
----------------	1	1

	COM01	COM00
NORMAL	0	0
TOGGLE….	0	1
----------------	1	0
----------------	1	1

- **Variation 1**: If we keep WGM values at **NORMAL** and COM values in **"Toggles OC0 at Compare Match"** [2nd choice]
- WGM bits at NORMAL means Timer0 Counting Register, TCNT0 will reset normally. When its content becomes FF, it will roll over to 00 in next pulse.
- There is a pin in uC called OC0 pin. If you set that pin as output, the pin will automatically toggle at every COMPARE MATCH

Variation # 1 Explained



Note that The Time Period is Constant, It depends on the Clock frequency and Prescaler And The duty cycle is always fixed at 50%

Calculation of Time period

- Let us say we used a Prescaling factor of P.
- So, the frequency of timing will be $f=f_{osc}/P$
- Therefore each pulse will be of duration $T_{pulse}=1/f$
- Overflow will occur when FF+1 or 256 number of pulses appears.
- So, the time period of the output pulse will be $T_{out}=2*256*P/f_{osc}$.
- Say we use f_{osc}=16 MHz, P=1024, then T_{out}= 32.768 ms
- Output frequency=$1/T_{out}$=30.52 Hz
- For Timer0 and Timer1, the value of P can be 1, 8, 64, 256 or 1024, so the possible output frequencies are 31.25 kHz, 3.91 kHz, 488.28 Hz, 122.07 Hz and 30.52 Hz.

Code in CodeVisionAVR [NORMAL]

Prescaling

WGM bits

FOC0	**WGM00**	**COM01**	**COM00**	**WGM01**	CS02	CS01	CS00

COM bits

TCCR0 Register

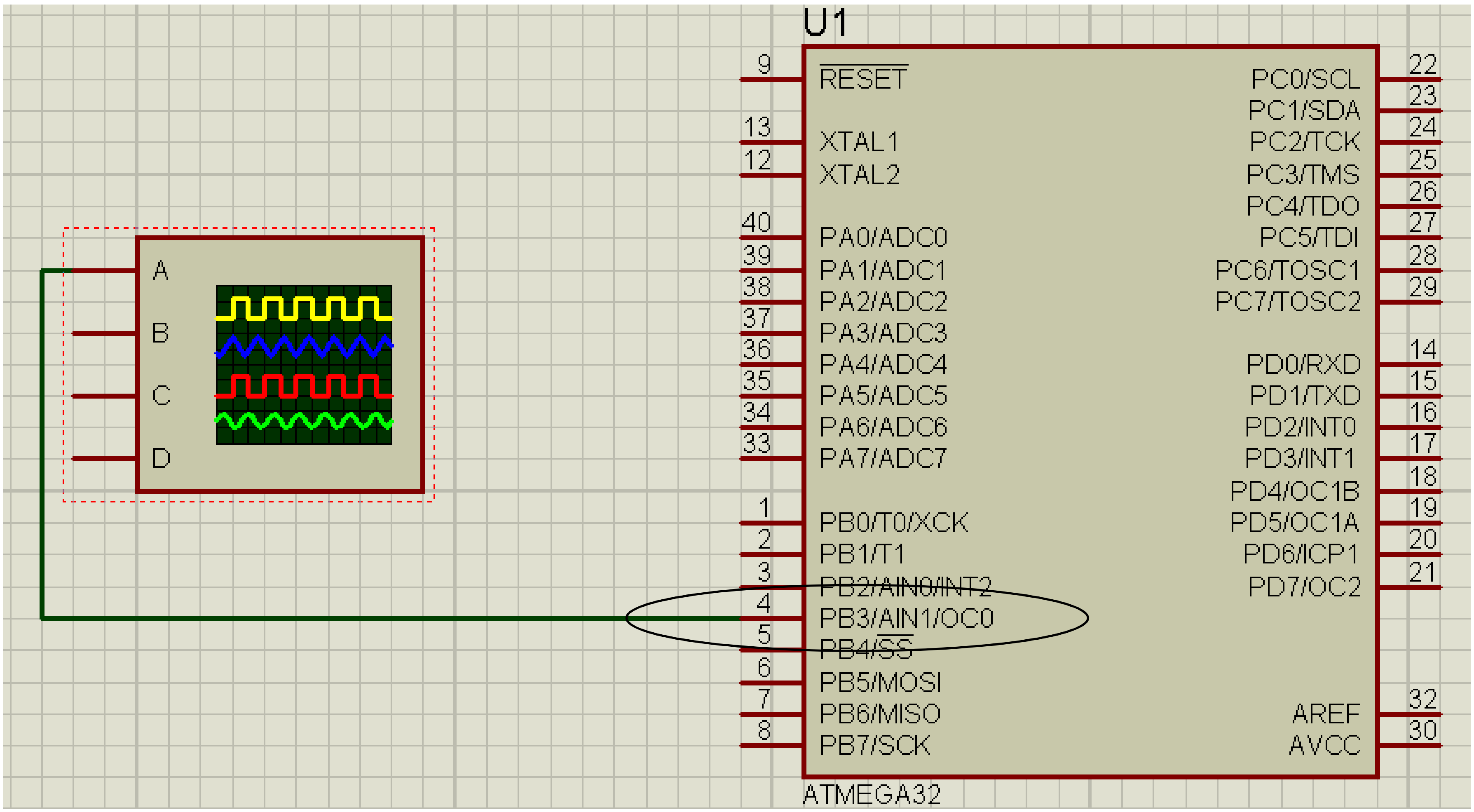
```
#include <mega32.h>
void main(void)
{
DDRB.3=1;
TCCR0=(0<<WGM00) | (0<<COM01) | (1<<COM00) | (0<<WGM01) |
(1<<CS02) | (0<<CS01) | (1<<CS00);
TCNT0=0x00;
OCR0=0x1B;
while (1)
    {
    }
}
```

WGM=00 – NORMAL

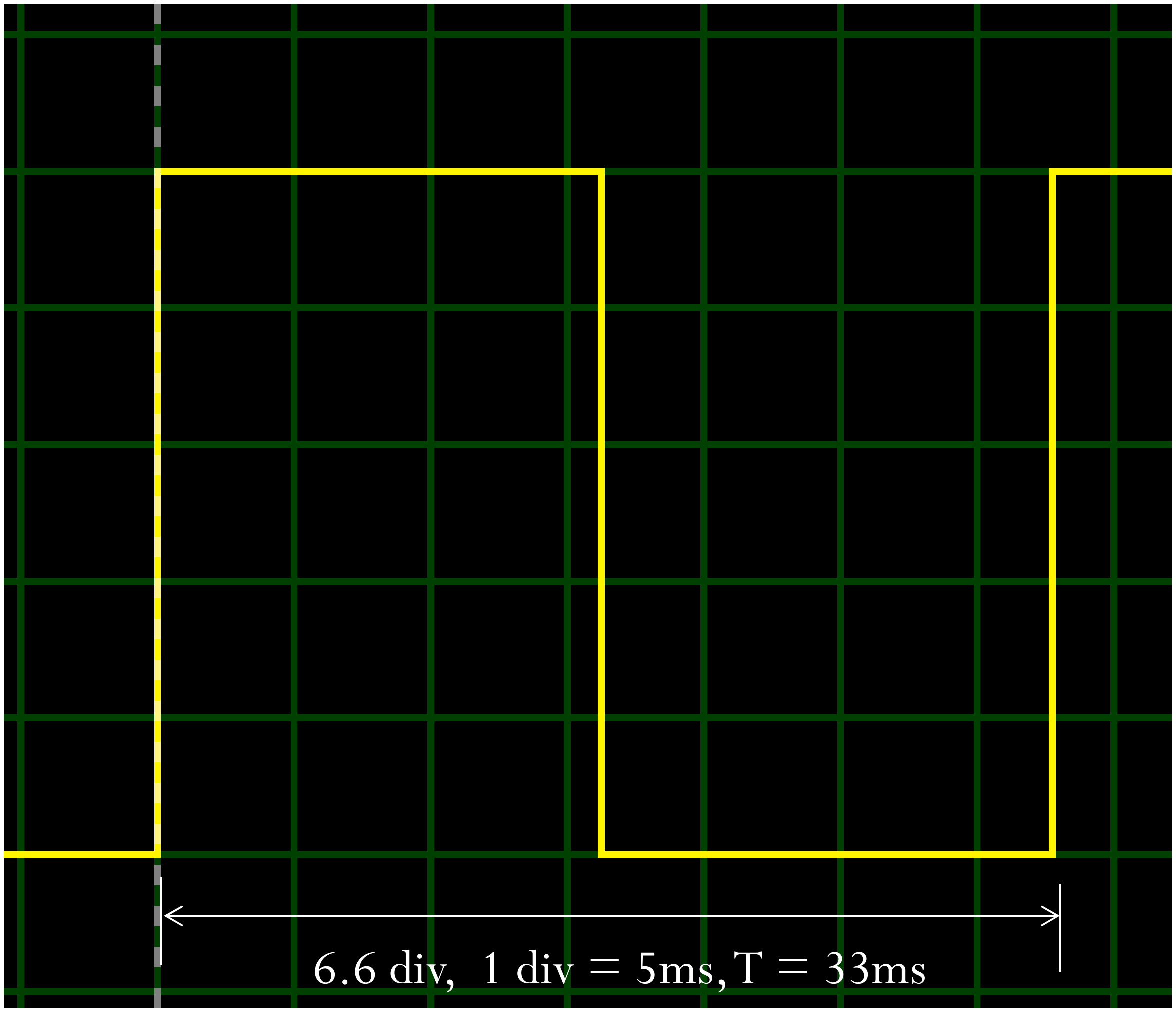
COM=01 – TOGGLE…..

Prescaling=101 – factor of 1024

Simulation in Proteus



Output in the Virtual Oscilloscope



Variation # 2 of WGM and COM bits in TCCR Register

TCCR0 Register

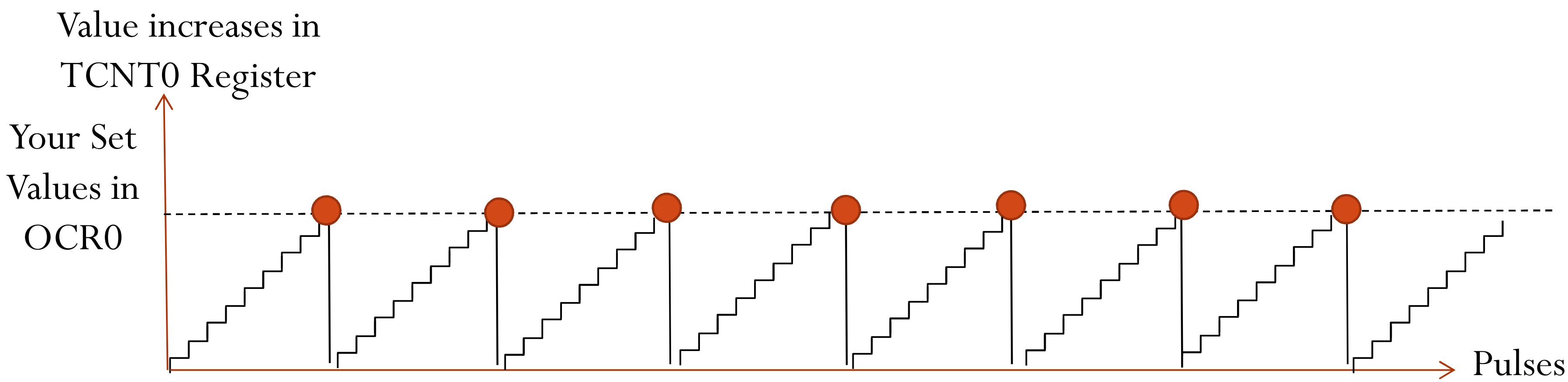
FOC0	**WGM00**	**COM01**	**COM00**	**WGM01**	CS02	CS01	CS00

	WGM00	WGM01
NORMAL	0	0
CTC	1	0
----------------	0	1
----------------	1	1

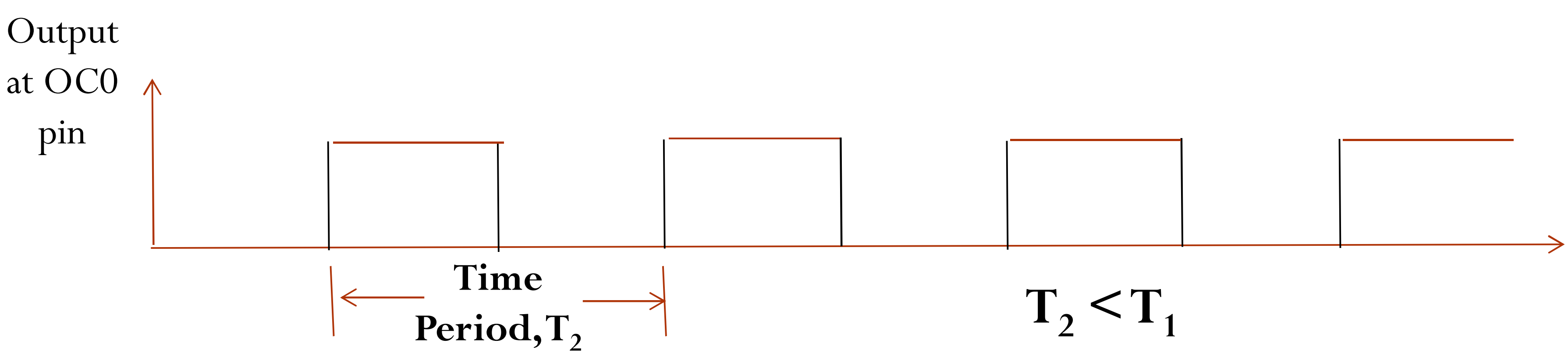
	COM01	COM00
NORMAL	0	0
TOGGLE….	0	1
----------------	1	0
----------------	1	1

- **Variation 2**: If we change WGM values at **CTC** [2nd choice] and COM values in **"Toggles OC0 at Compare Match"** [2nd choice]
- WGM bits at CTC (**Clear on Compare Match**) means Timer0 Counting Register, TCNT0 will reset at **COMPARE MATCH** Point. When its content becomes equal to the value of OCR0, it will roll over to 00 in next pulse.
- The OC0 pin will automatically toggle at every COMPARE MATCH

Variation # 2 Explained



WGM = CTC and COM = TOGGLE ……….



Note that The Time Period is Variable, depends on the Clock frequency, Prescaler and the value of OCR0. But The duty cycle is always fixed at 50%

Calculation of Time period

- Let us say we used a Prescaling factor of P.
- So, the frequency of timing will be $f=f_{osc}/P$
- Therefore each pulse will be of duration $T_{pulse}=1/f$
- Overflow will occur when ICR0 will be x in Hex or y in decimal
- So, the time period of the output pulse will be $T_{out}=2*(y+1)*P/f_{osc}$.
- Say we use $f_{osc}=16$ MHz, P=1024, then

$$T_{out}=2*(y+1)*1024*10^{-6}/16 \text{ sec}$$

- If we load 0x1B or 27D,

$$T_{out}=2*(27+1)*1024*10^{-6}/16 \text{ sec}=3.58 \text{ msec}$$

- Output frequency$=1/T_{out}=279.34$ Hz

Code in CodeVisionAVR [CTC]

WGM bits

Prescaling

FOC0	**WGM00**	**COM01**	**COM00**	**WGM01**	CS02	CS01	CS00

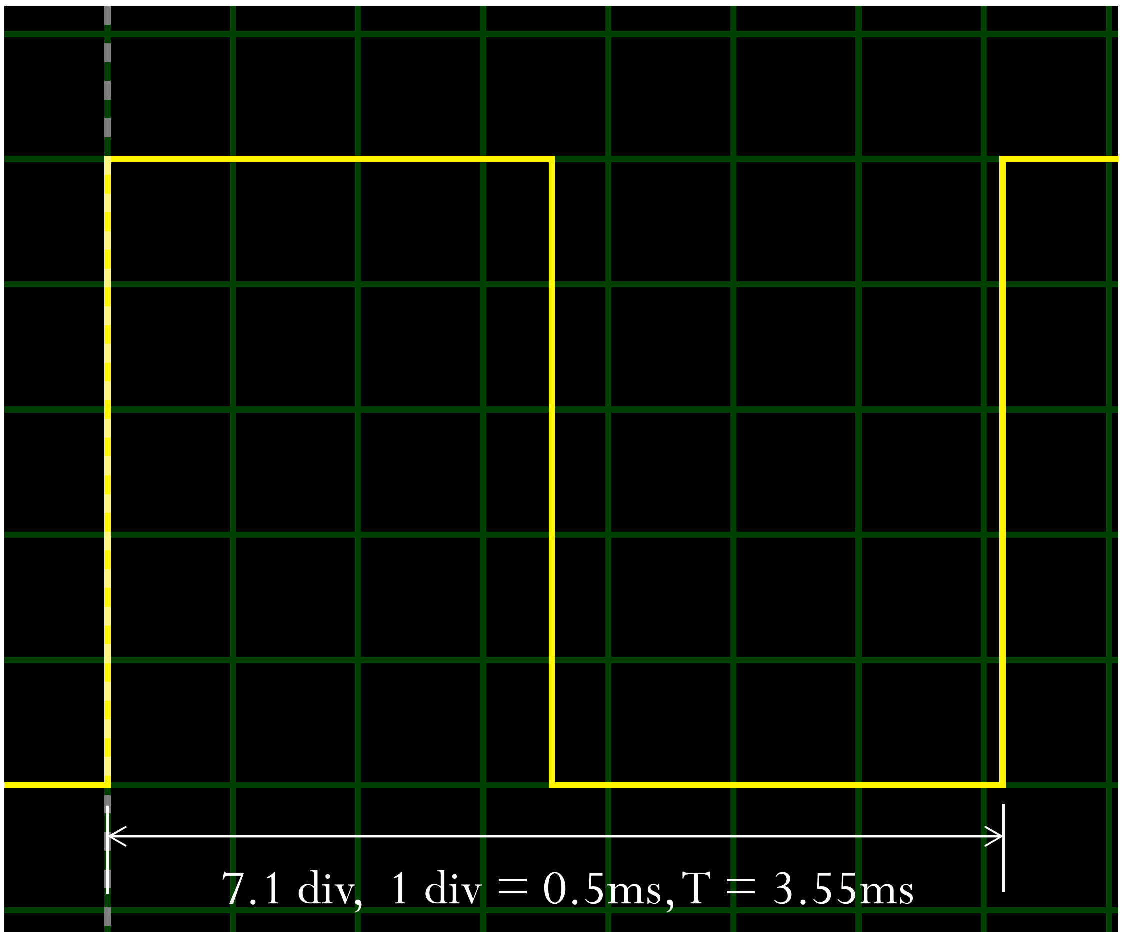
COM bits

TCCR0 Register

```
#include <mega32.h>
void main(void)
{
DDRB.3=1;
TCCR0=(0<<WGM00) | (0<<COM01) | (1<<COM00) | (1<<WGM01) |
(1<<CS02) | (0<<CS01) | (1<<CS00);
TCNT0=0x00;
OCR0=0x1B;
while (1)
      {
      }
}
```

WGM=01 – CTC
COM=01 – TOGGLE…..
Prescaling=101 – factor of 1024

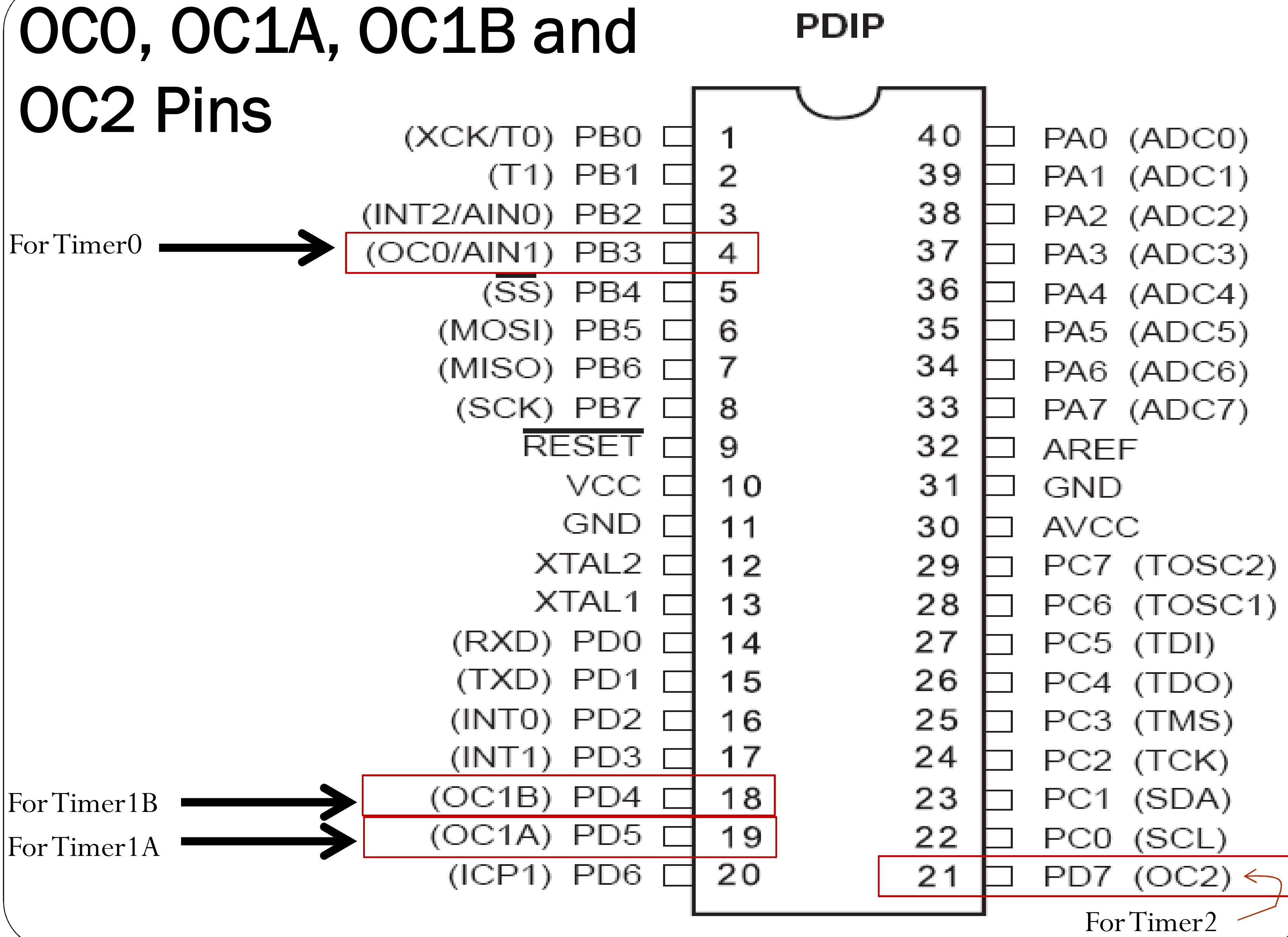
Output in the Virtual Oscilloscope



Physical Output PIN Related to All Timers

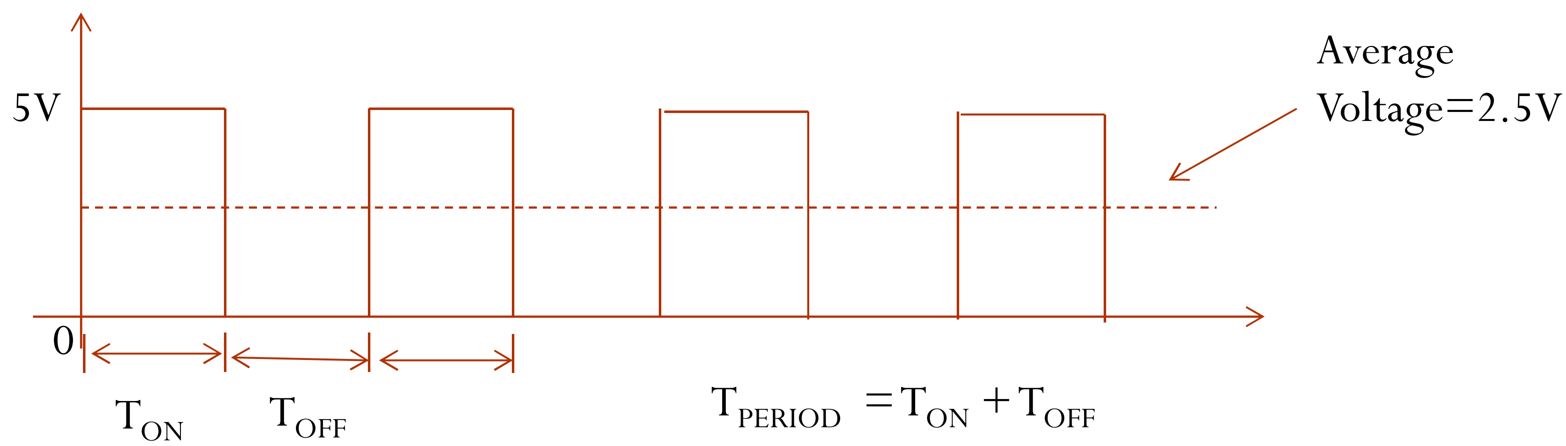
- Each Timer (Timer0, Timer1 and Timer2) has a waveform generator (WG).
- The waveform generator gives output to different pins of the microcontroller for different timers.
  - OC0 (PB.3) ← Timer 0, Output Compare
  - OC1A (PD.5) ← Timer 1, Output Compare A
  - OC1B (PD.4) ← Timer 1, Output Compare B
  - OC2 (PD.7) ← Timer 2, Output Compare
- WGMn and COMn bits of TCCR register determine how the waveform generator will work.

OC0, OC1A, OC1B and OC2 Pins

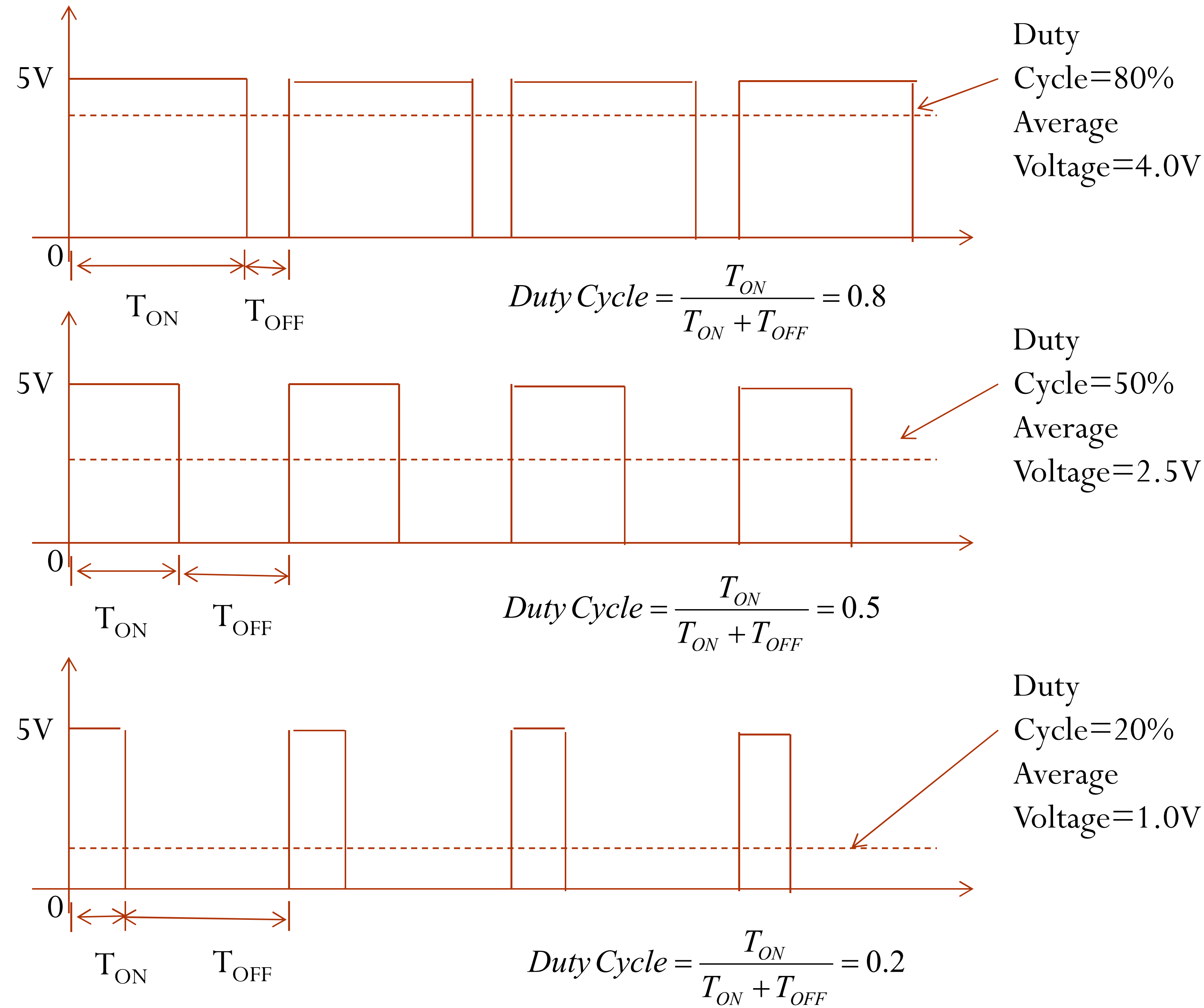


Generation of PWM Wave

- **PWM** stands for **P**ulse **W**idth **M**odulation.
- Let us say we apply continuous 5V DC to a motor. The motor will rotate at a certain speed.
- If you apply 5V DC for 50% of the time and 0V for another 50% time and repeat it. The average value of the voltage will be 2.5 volt. So the speed will be halved.



- By modifying the ON/OFF time within a time period, one can vary the average value and the speed of the motor may be varied.


5V
0
T_{ON}
T_{OFF}
Duty Cycle=80%
Average Voltage=4.0V
$Duty\ Cycle = \frac{T_{ON}}{T_{ON} + T_{OFF}} = 0.8$
5V
0
T_{ON}
T_{OFF}
Duty Cycle=50%
Average Voltage=2.5V
$Duty\ Cycle = \frac{T_{ON}}{T_{ON} + T_{OFF}} = 0.5$
5V
0
T_{ON}
T_{OFF}
Duty Cycle=20%
Average Voltage=1.0V
$Duty\ Cycle = \frac{T_{ON}}{T_{ON} + T_{OFF}} = 0.2$

Variation # 3 of WGM and COM bits in TCCR Register

TCCR0 Register

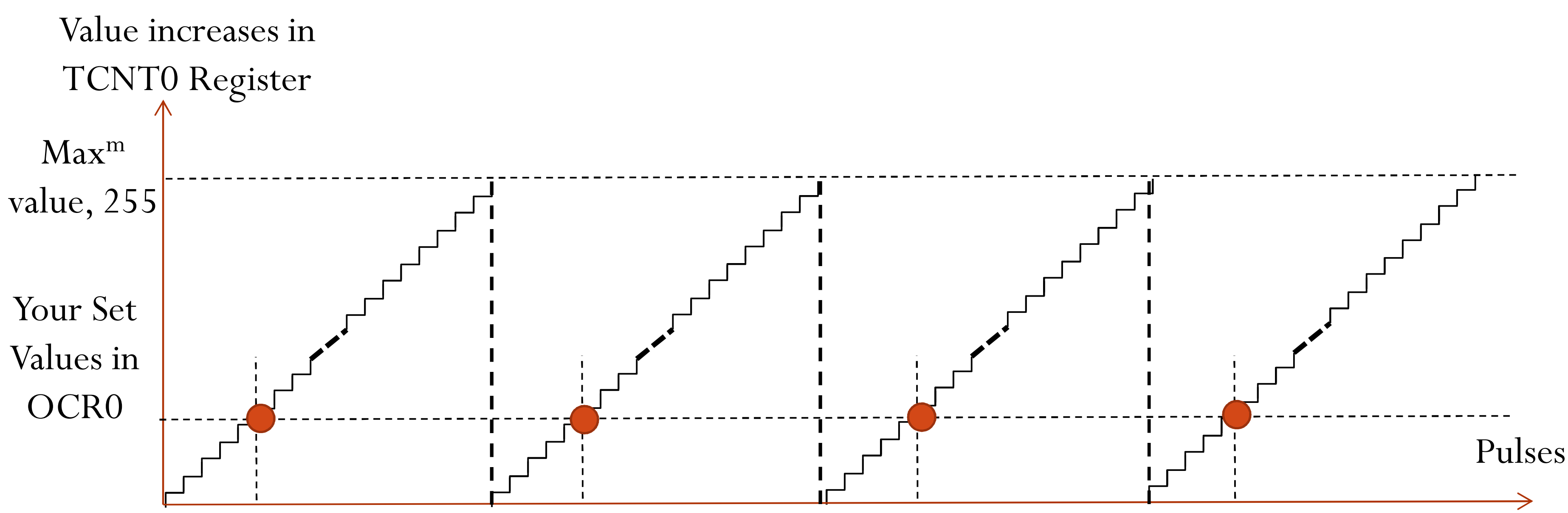
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

	WGM00	WGM01
NORMAL	0	0
CTC	0	1
P.C. PWM	1	0
FAST PWM	1	1

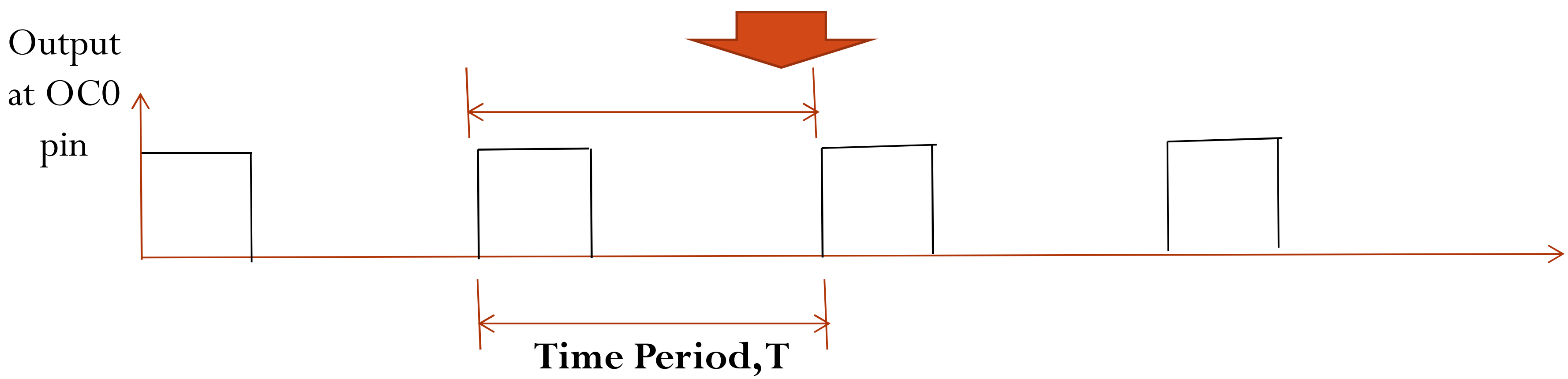
	COM01	COM00
NORMAL	0	0
Reserve	0	1
CLEARS….	1	0
SETS….	1	1

- **Variation 3**: If we choose WGM values at **FAST PWM** and COM values in **"CLEARS at COMPARE MATCH and sets at BOTTOM"** [3rd choice]
- With WGM bits at FAST PWM, Timer0 Counting Register, TCNT0 will reset normally. When its content becomes FF, it will roll over to 00 in next pulse.
- The OC0 pin will automatically CLEARS at every COMPARE MATCH and SETS at 00 in TCNT0 Register.

How PWM works in ATmega32



WGM = FAST PWM & COM = OC0 CLEARs at COMPARE MATCH & SETs at BOTTOM



The Calculation

- Crystal frequency = 16 MHz
- Prescaler, P=1024.
- Hence the clock frequency = (16/1024) uS
- Time of one clock (tick), Ttick =(1024/16) uS=64 uS.
- Time period of the output wave

  = 256 * Ttick =256*64 uS = 16.4 mS
- The value of OCR0 = duty cycle * 256/100
- For 20% duty cycle OCR0=51d = 33 h.

A Program for an LED Dimmer

- Let us connect an LED at OC0 pin and a square wave of gradually increasing and decreasing duty cycle will be applied at that pin.
- Let us choose P=1024, WGM at "FAST PWM" and COM at "Clears at COMPARE MATCH and sets at BOTTOM"
- It means CS02:00=101, WGM01:00=11 and COM01:00=10.

The Code [FIXED_FREQ]

Fast PWM

CLEARs at COMPARE MATCH & SETs at BOTTOM

```
#include <mega32.h>
#include <delay.h>
int duty=0;
float Fosc=0;
long int P=0;
float Ttick=0;
float Tp=0;
void main(void)
{
DDRB.3=1;
TCCR0=(1<<WGM00) | (1<<COM01) | (0<<COM00) | (1<<WGM01)
| (1<<CS02) | (0<<CS01) | (1<<CS00);
TCNT0=0x00;
Fosc=16e6;          // System clock=16MHz
P=1024;             // Prescaler=1024
Ttick=P/Fosc;       //Time period of each clock (tick)
Tp=256*Ttick;       //Time period of the output wave
```

The Code (continued)

```
while (1)
   {
   for (duty=10;duty<90;duty++)
    {
      OCR0=duty*256/100;   // 256 no. of clocks make 100% of  one time period
      delay_ms(50);
    }
   for (duty=90;duty>10;duty--)
    {
      OCR0=duty*256/100;;
      delay_ms(50);
    }
   }
}
```

Thanks