ICT 5307 – Embedded System Design

Lecture 8
Simulation Tools and Timer – Part 1

Professor S.M. Lutful Kabir

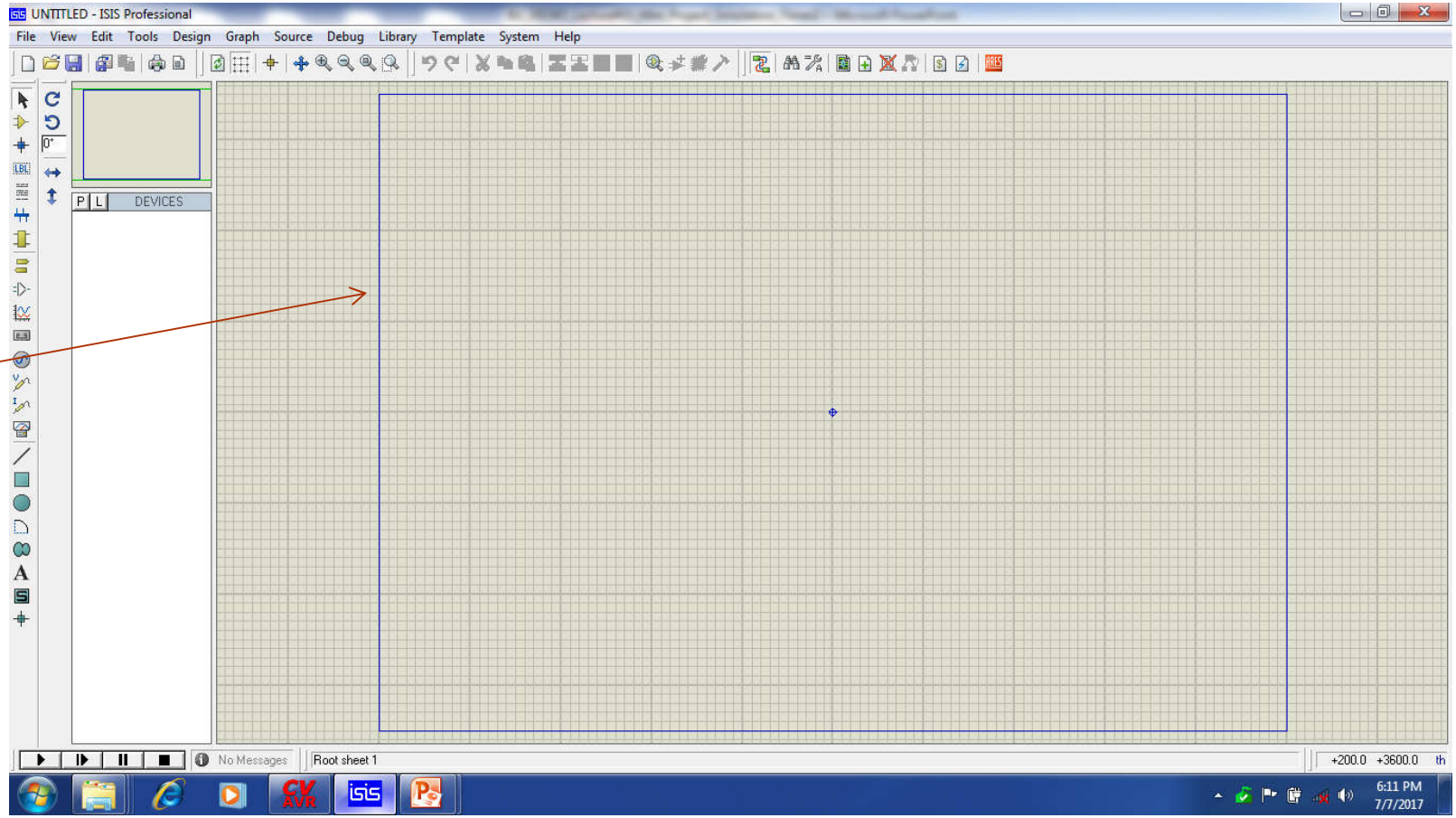
BUET

A Quick Development Strategy for your project

- Simulation of a Circuit in the software called Proteus.
- Development of the Program in CodeVisionAVR.
- The compiled code (the .hex file) can be linked from the Proteus.
- The program may be run in Proteus.
- Although there may be some problem in actual practical implementation but as an initial development we get a first hand feedback from the output of the simulation.
- Therefore all students must be able to use the two software and there project should be implemented at least in Proteus.

Use of Proteus for Simulating uC based Projects

Area to be used



In the vertical left and horizontal upper menu you will find different symbols indicating some options for drawing your circuit.

Different tools

Selecting a zone

Selecting a component

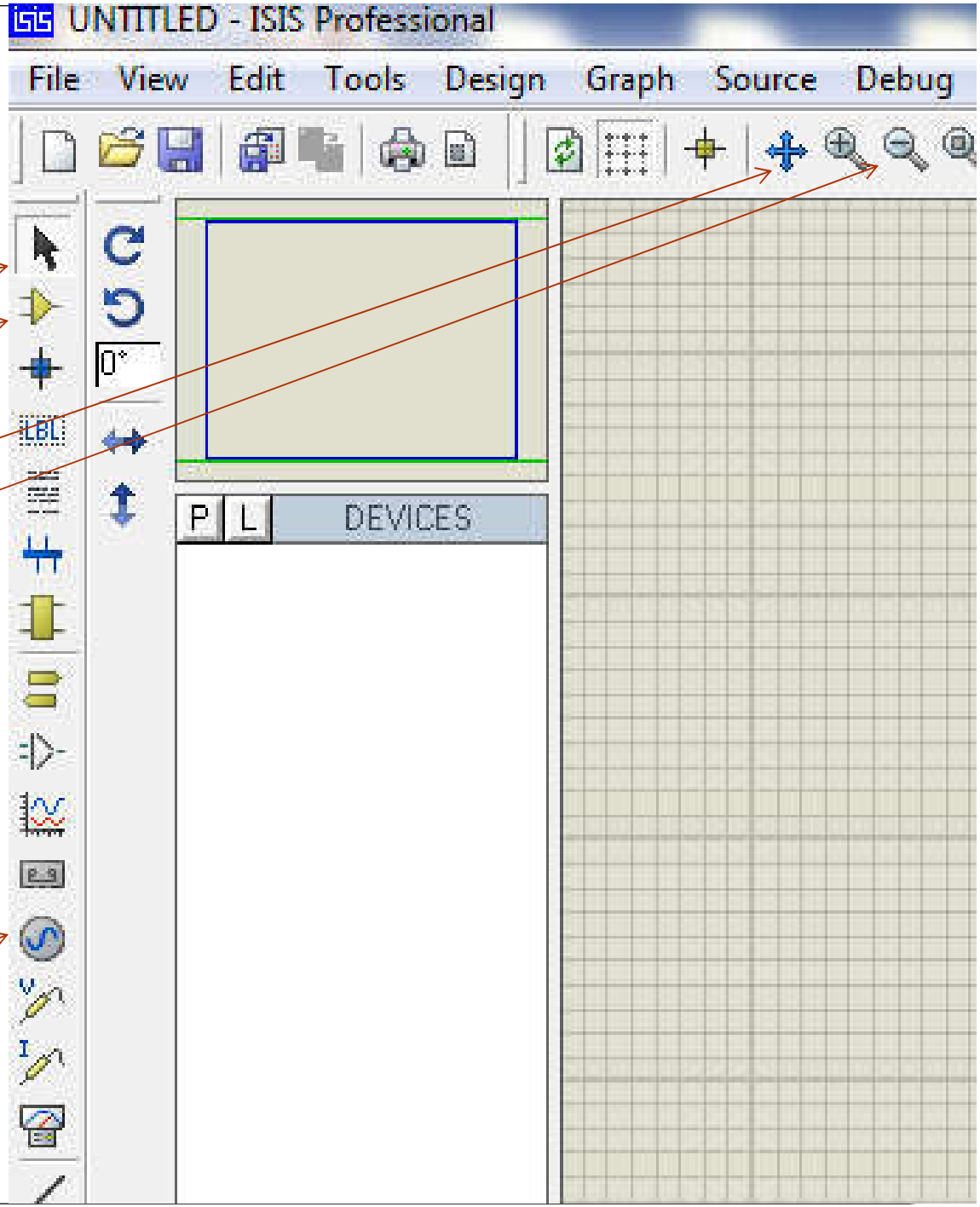
Centering your drawing

Zooming in and out

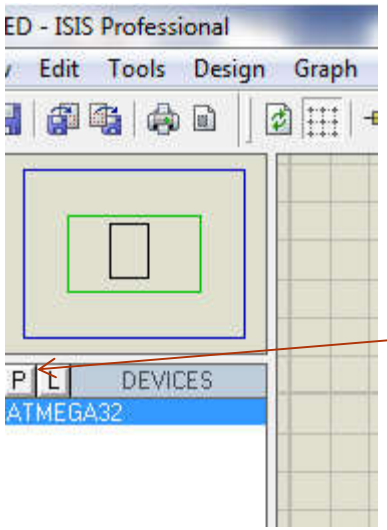
Adding terminals, POWER, GROUND

Adding Voltage Sources

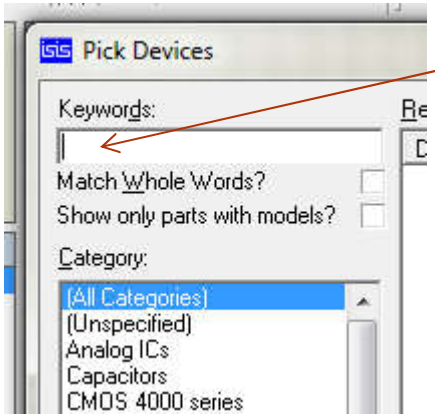
Adding Virtual Instruments, like OSCILLOSCOPE



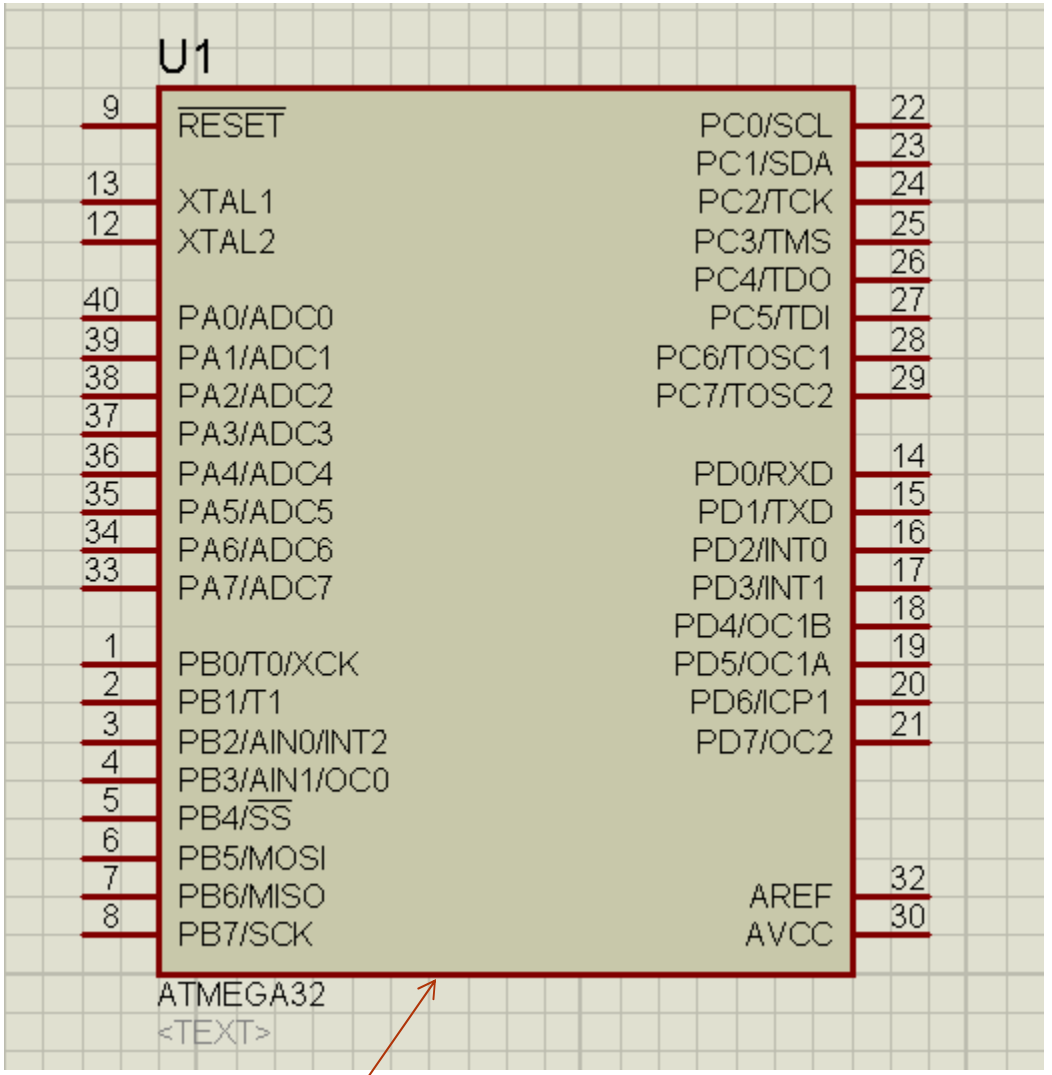
Adding a component



After choosing component mode, press **"P"**



A new window will appear and you should type few letters of your desired component

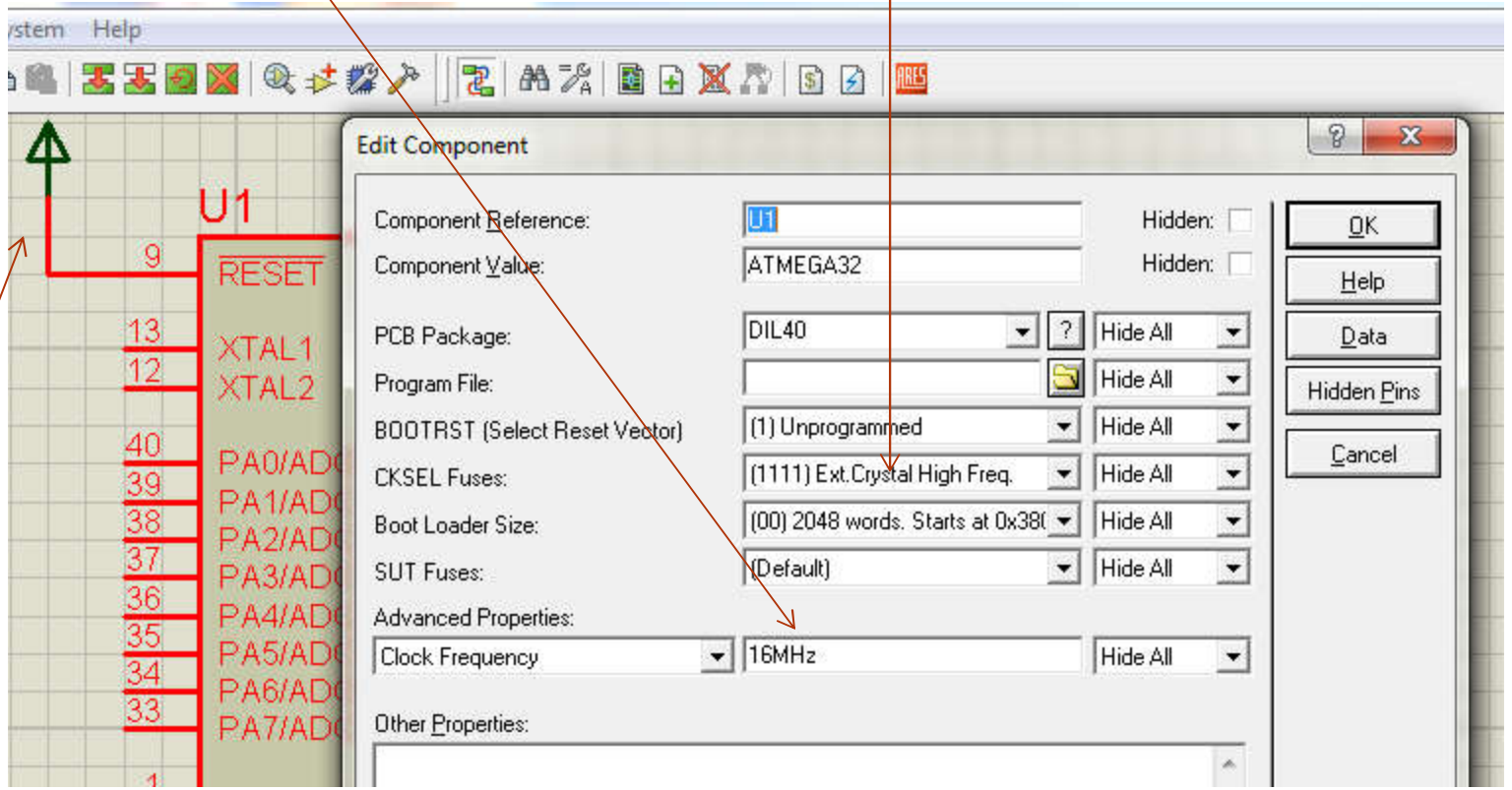


By clicking the cursor in the drawing are, the chosen component will be added

Adding some parameters of the uC

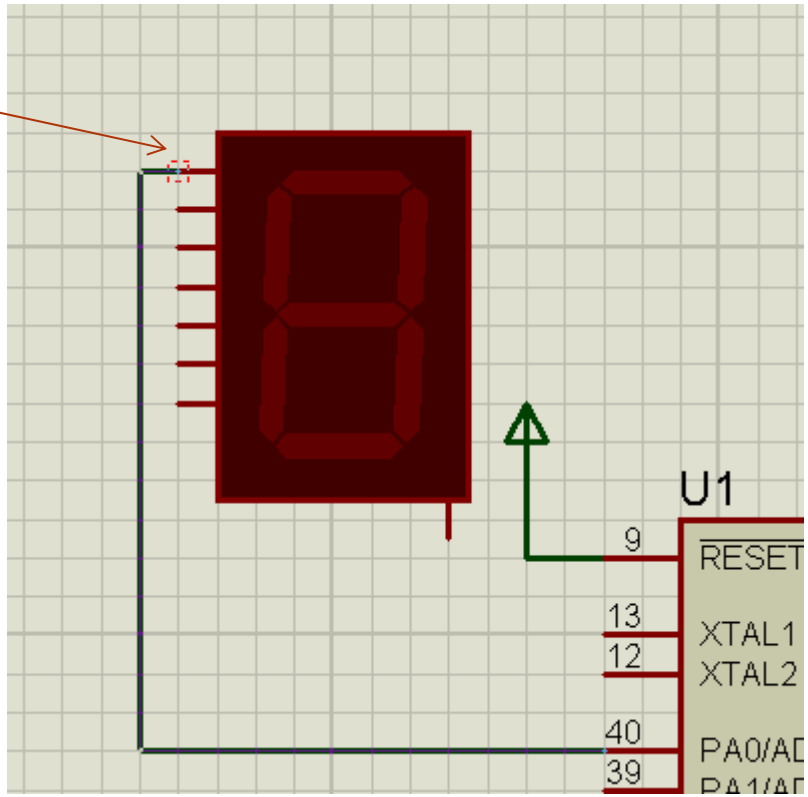
By double clicking on the uC, a new window will appear, two values have to be set there. 1. CKSEL Fuses -> Ext high freq. and 2. Clock freq -> 16 MHz.

Using terminal mode add VCC (POWER) at RESET Pin.



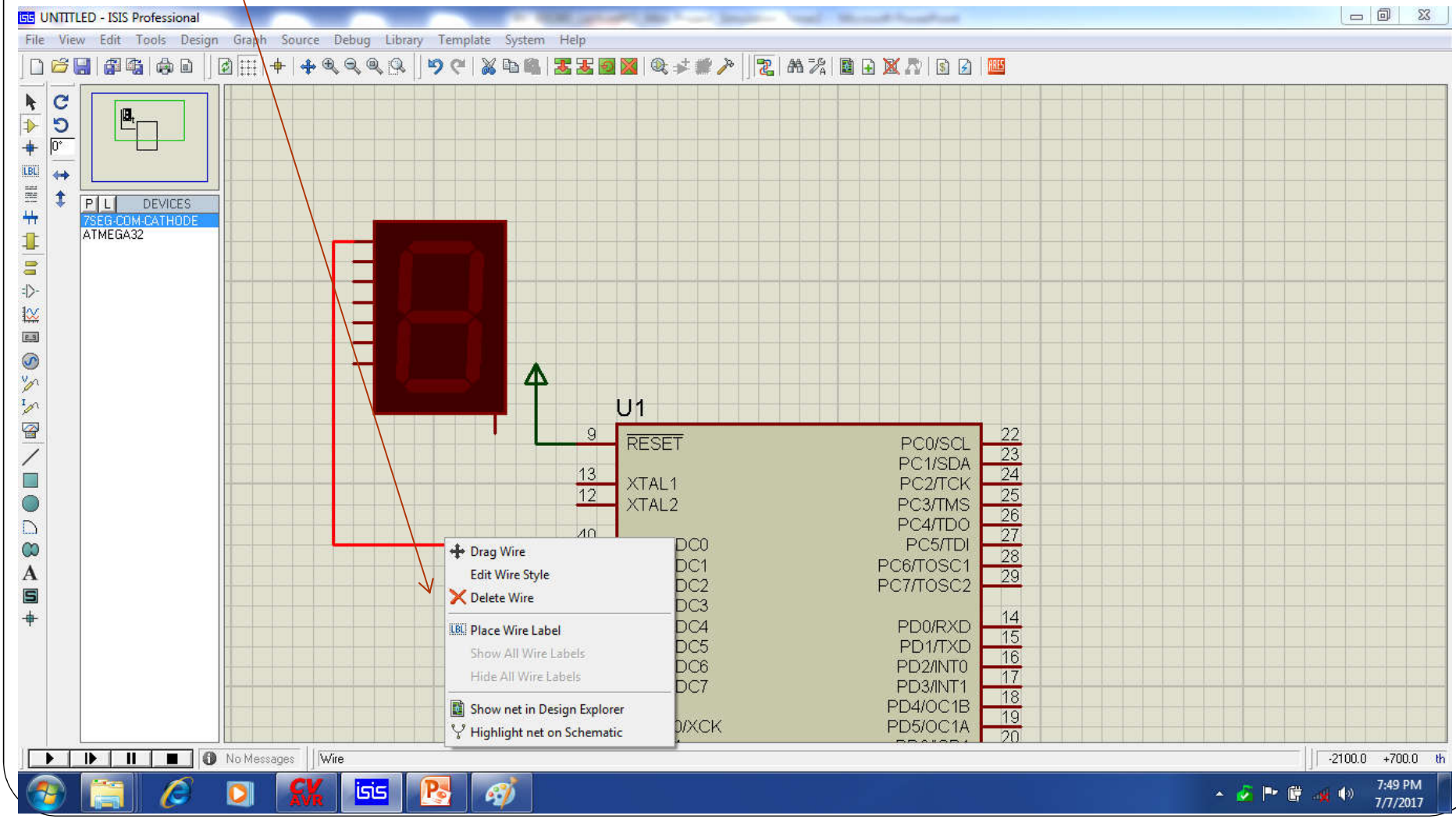
Connection a Line

A dotted square box appears in the terminal. Start from one end and drag your cursor to other pin. Similar dotted rectangular box will appear and click there. It ends.

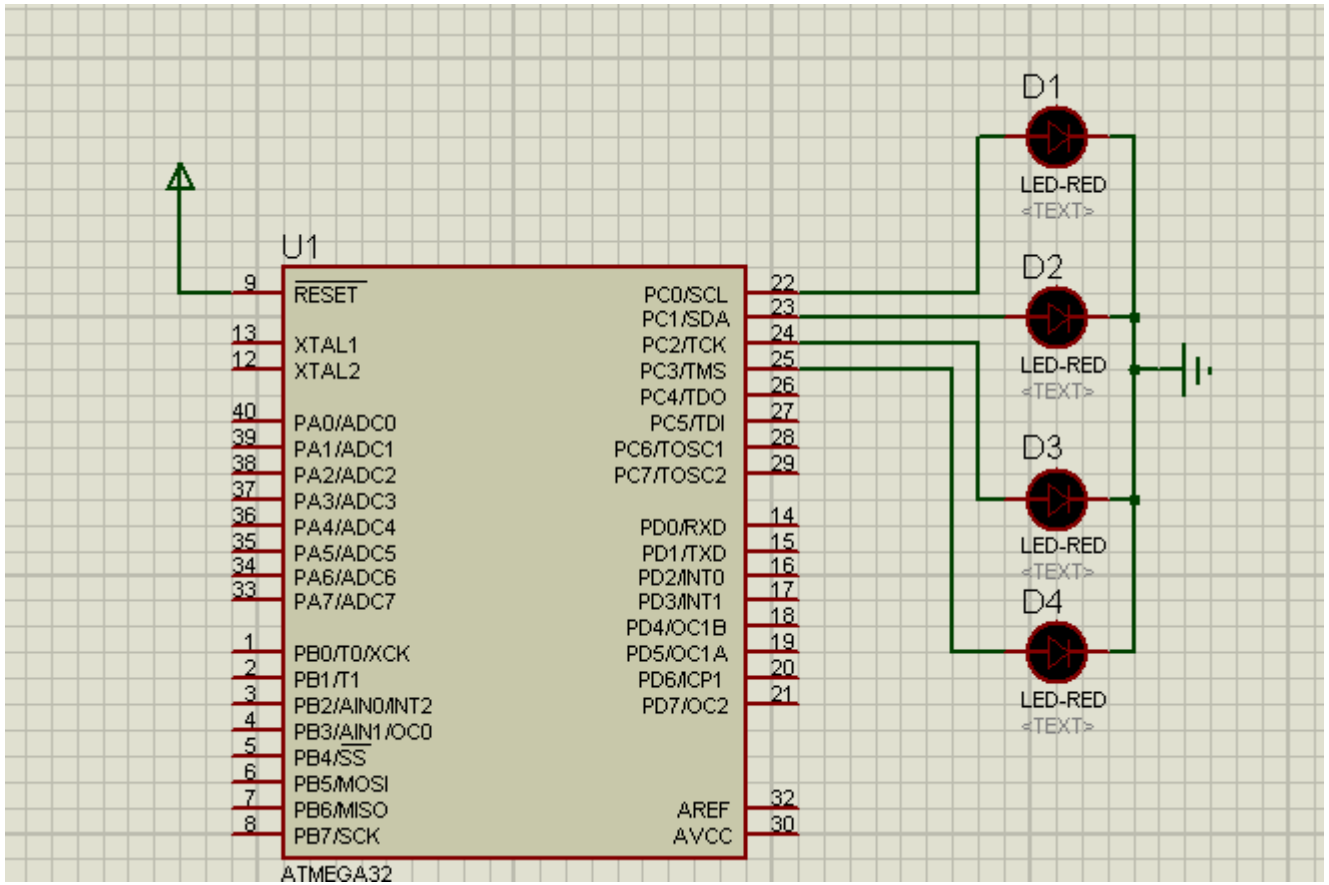


Removing a wire

If you put cursor over a wire a window will appear choose "Delete wire" option.



An Example Circuit : Four LEDs Connected in PortC0:3 are made ON and OFF at every 1 sec



The Program to be Developed in CodeVisionAVR

```
#include <mega32.h>
#include <delay.h>

void main(void)
{
DDRC=0x0F;
PORTC=0x0F;
while (1)
      {
       PORTC=~PORTC;
       delay_ms(1000);
      }
}
```

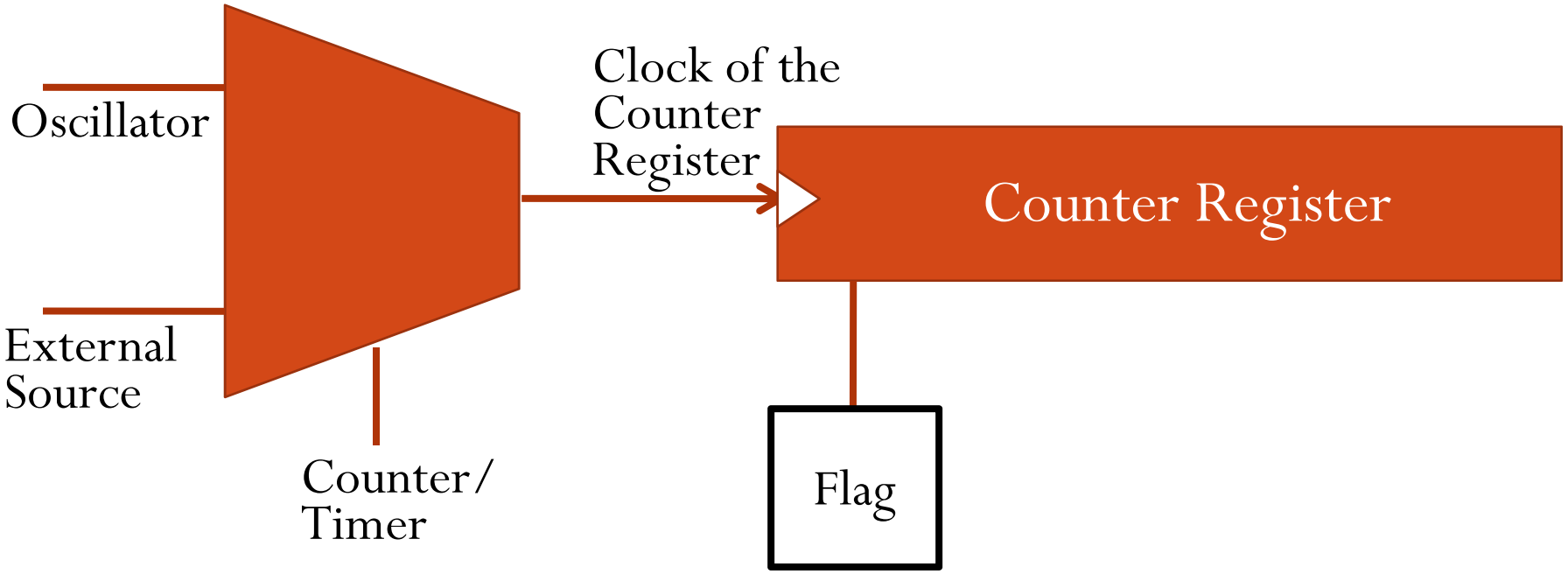
Timer/ Counter: Basic Concept

- There are counter registers in microcontrollers for counting events or measuring time.
- When we want to count an event, we connect the external event source to the clock pin of the counter register.
- Then when an event occurs externally, the content of the counter increments; in this way the content of the counter represents how many times an event has occurred.

Timer/ Counter: Basic Concept

- When we want to measure time, we connect the oscillator to the clock pin of the counter
- So, when the oscillator ticks, the content of the counter increases and since we know the time period of the clock so from the content of the counter we can measure how much time has been elapsed.

Block Diagrammatic Representation



To measure time

- One of the ways to generate a time delay is to first clear the counter register and wait until the counter reaches a certain number.
- For example, the oscillator frequency of a microcontroller is 1 MHz. It means that every clock is of 1 uSec width.
- In other words, the content of the counter register increments at every 1 uSec.
- If we want to measure 100 uSec then we have to clear the counter register and make the timer ON and wait until the register reaches 100.

Another method of time measurement

- In microcontrollers, there is a flag for each of the counters.
- The flag is set when the register overflows and it is cleared by software.
- The second method of generating time delay is to counter register with an appropriate value and wait until the counter register overflows and the flag is set.
- For example, if we want to make 3uS delay with a 1 MHz microcontroller we have to load the register with $FD.
- With first tick, it will become $FE, with the second tick, $FF and with the third tick, the register will overflow and the flag will be set.

Timers in ATmega32

- Different microcontrollers have different numbers of timers.
- In ATmega32, there are three timers, namely Timer0, Timer1 and Timer2
- Timer0 and Timer2 are 8-bit timers while Timer1 is a 16-bit timer.

Programming the timer

- In AVR, for each of the Timers, there is a counter register called TCNTn (**T**imer/**C**ou**NT**er). That is, in ATmega32, there are TCNT0, TCNT1 and TCNT2 registers.
- Upon reset TCNTn registers contains zero.
- It counts up with each pulse (external or internal)
- You can write values to TCNTn registers or you can also read them.

Programming the Timers

- Each timer/counter has a **T**imer **OV**erflow Flag called TOVn.
- When a Timer/Counter overflows, TOVn flag is set.
- Each Timer/Counter has also **T**imer/**C**ounter **C**ontrol **R**egister called **TCCRn** for setting modes of operation.
- For example, you can specify Timer0 to work as TIMER or COUNTER by loading proper values in TCCR0 register.

Programming the Timer

- Each timer has also OCRn (**O**utput **C**ompare **R**egister) and OCFn (**O**utput **C**ompare **F**lag) in TCCRn register.
- The content of TCNTn register is compared with OCRn register, when they are equal OCF flag is set.
  - Let us say TCNT0 register is cleared (set zero). And OCR0 register is loaded with 100.
  - If the timer is started, TCNT0 register will start increasing.
  - When TCNT0 register becomes equal to OCR0 register, OCF0 flag is set.

Timer0 Programming

TCCR0 Register

FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

- FOC0 – Bit 7 – Force Compare Match. This is a write only bit, writing 1 to it causes to act as if a compare match has occurred.

- WGM00, WGM01 – Bit 6 and 3 – Mode Selector
  0 0 – Normal
  0 1 – CTC (Clear Timer on Compare Match)
  1 0 – PWM, Phase Correct
  1 1 – Fast PWM

Timer0 Programming (Contd.)

TCCR0 Register

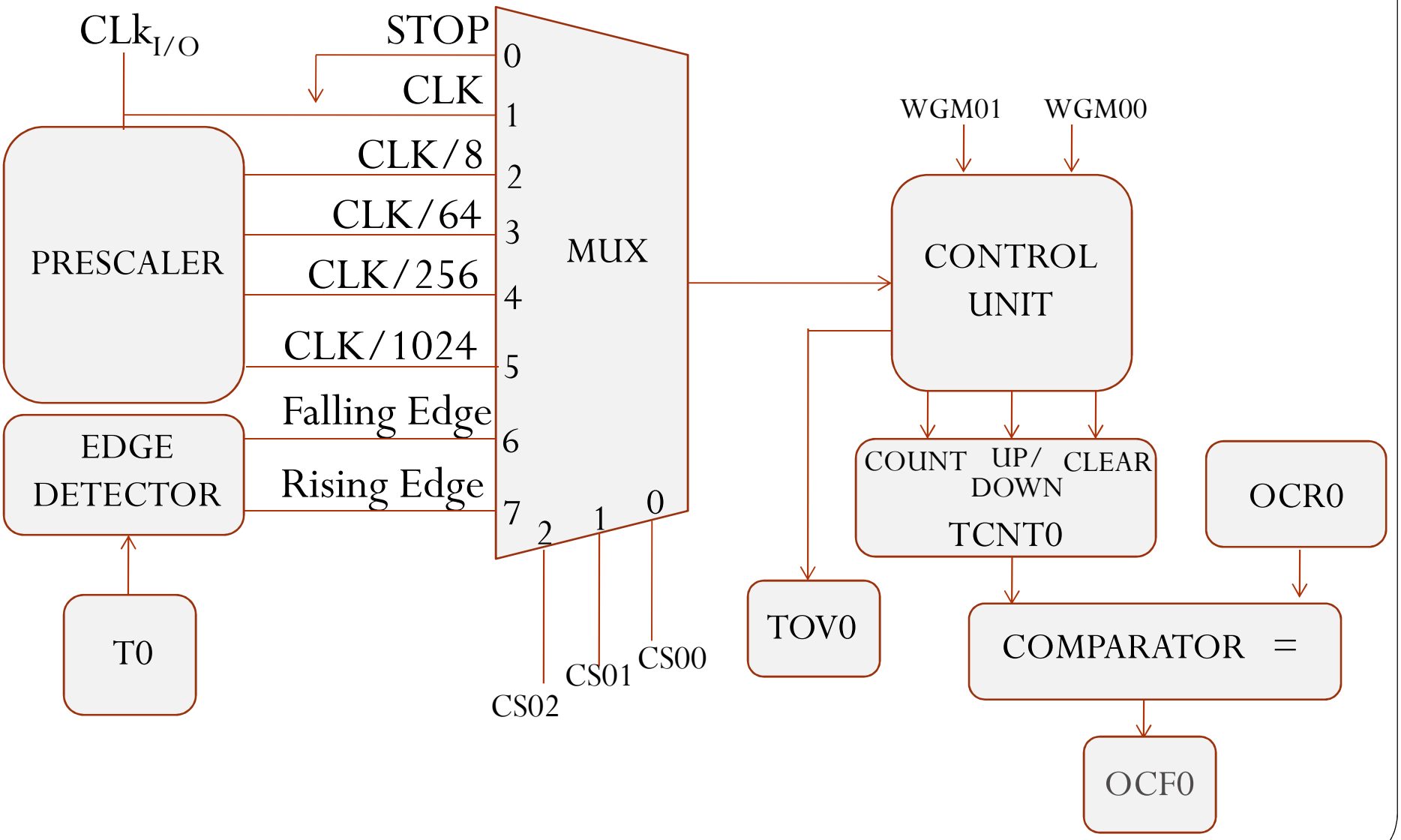
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

- COM 01:00 – Bit 5 and 4

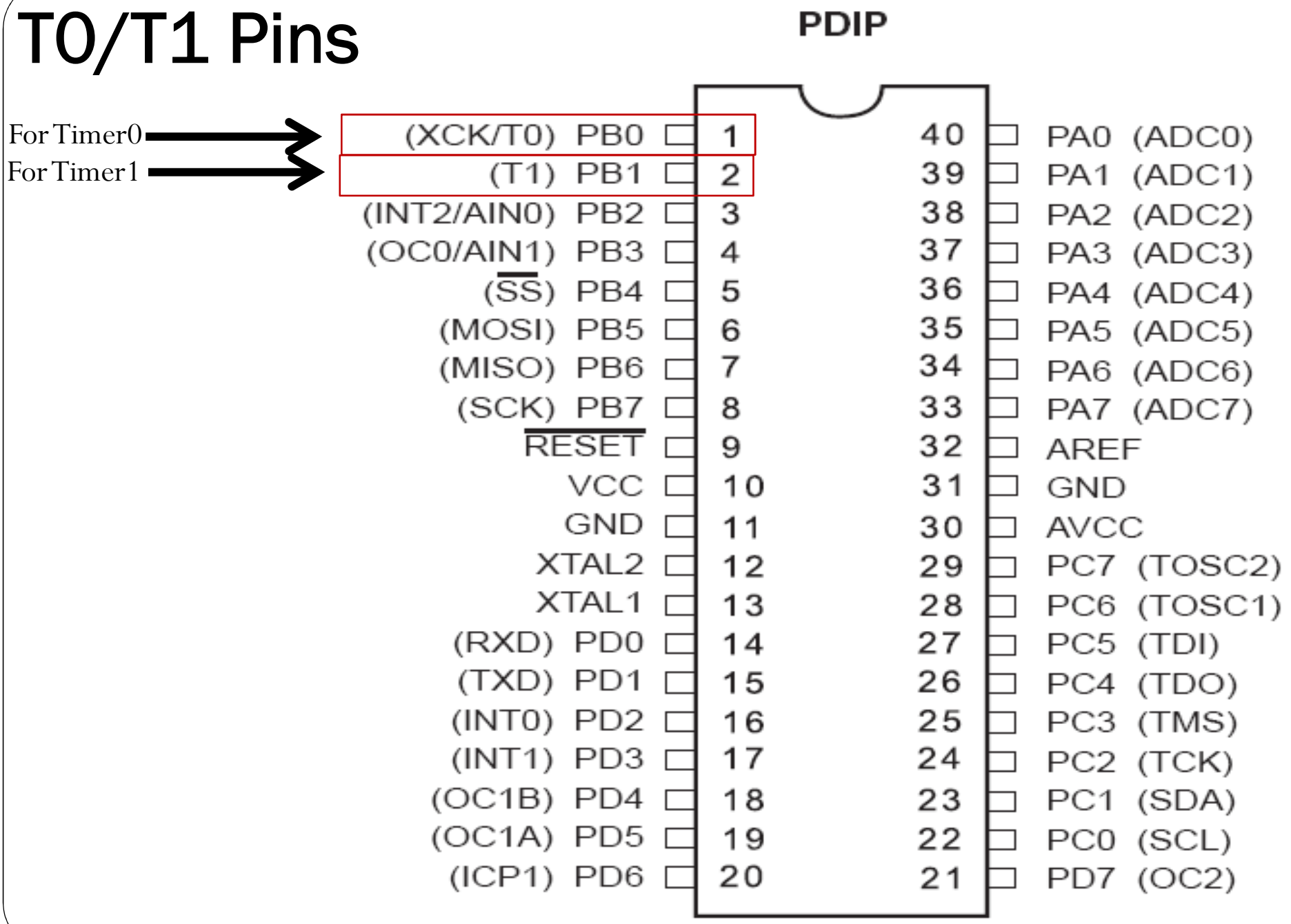
  Compare output mode This mode controls the waveform generator (will be discussed in another lecture)
- CS02:00 – Bit 2, 1 and 0 – Timer clock selector

  (see the next slide)

Block Diagram for Timer0



T0/T1 Pins


PDIP
For Timer0
For Timer1
(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
(ICP1) 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)

Timer0 Clock Selector (CS02 : 00)

D2	D1	D0	
0	0	0	- No clock source (Timer/ Counter Stopped)
0	0	1	- Clock (No Prescaling)
0	1	0	- Clock/8
0	1	1	- Clock/64
1	0	0	- Clock/256
1	0	1	- Clock/1024
1	1	0	- Ext. Clk on T0 pin, Clk on Falling Edge
1	1	1	- Ext. Clk on T0 pin, Clk on Rising Edge

TIMSK Register

OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

TOIE0	Timer0 Overflow Interrupt Enable
OCIE0	Timer0 Output Compare Interrupt Enable
TOIE1	Timer1 Overflow Interrupt Enable;
OCIE1B	Timer 1 Output Compare B match Interrupt Enable
OCIE1A	Timer 1 Output Compare A match Interrupt Enable
TICIE1	Timer/Counter1, Input Capture Interrupt Enable
TOIE2	Timer2 Overflow interrupt Enable
OCIE2	Timer2 output compare Interrupt Enable

TIFR (Timer/Counter) Interrupt Flag Register

TIFR Register

TOV0 Timer0 flag bit; 0- did not overflow, 1-has overflowed (FF->00)
OCF0 Timer0 output compare flag bit; 0-did nor match, 1-matched
TOV1 Timer1 flag bit;
OCF1B Timer 1 output compare B match flag
OCF1A Timer 1 output compare A match flag
ICF1 Input Capture flag
TOV2 Timer2 flag bit
OCF2 Timer2 output compare flag bit

Clearing TOV0 flag

- When Timer0 rolls over from 0xFF to 0x00, the TOV0 flag is set to 1.
- It remains set until the software clears it.
- The strange thing about this flag is that in order to clear it, we need to write 1 to it.
- Indeed, this rule applies to all flags of AVR chip.
- In AVR, if we want to clear a given flag of a certain register, we write 1 to it and 0 to other bits. For example, the following code clears TOV0 flag.

```
TIFR=0x01;
```

How to generate time delay with Timer0

- Load the TCNT0 register with the initial count value.
- Load the value in TCCR0 register, indicating mode, and prescaler option.
- When you select the clock source, the timer/counter starts to count, and each tick causes the content of the timer/counter to increment by 1.
- Keep monitoring the timer overflow flag (TOV0) flag to check whether it is raised.
- If raised, get out of the monitoring loop.
- Stop timer by disconnecting the clock source.
- Clear TOV0 flag for the next round

Finding Values to be loaded into the TCNT0

- Calculate the period of the Timer clock using the following formula,

  $$T_{clock} = 1/f_{clock},$$

- T_{clock} is the time at which timer increments.
- Divide the desired time delay by Tclock.
- This gives the number of clocks we need to achieve that time delay.
- Subtract that number from 256 (0xFF) and add 1, i.e., find 256-n+1.
- Convert the subtracted number in hex, say xx.
- Load TCNT0 with xx

Delay Time Generation

- We want to generate 1.0 sec time delay.
- After 100 loops, this time will be generated.
- So, each loop will be of 1.0/100 sec or 0.01 sec duration.
- The clock is 15.625 kHz. So, time period is 1/15.625 mS.
- Number of pulse required = $0.01/[(1/15.625)\ X10^3] = 156$
- In output compare mode, the value to be loaded in OCR0 =156D=9CH
- In overflow mode, the value to be loaded in TCNT0 = 256-156+1 = 101D=65H

The Program with Timer0 [Polling: OVERFLOW (Part-1)]

```
#include <mega32.h>
int i=0;
void main(void)
{
DDRC=0xFF;
PORTC=0xFF;
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 15.625 kHz
// Mode: Normal top=0xFF
// OC0 output: Disconnected
// Timer Period: 16.384 ms
TCCR0=(0<<WGM00) | (0<<COM01) | (0<<COM00) | (0<<WGM01) |
   (1<<CS02) | (0<<CS01) | (1<<CS00);
TCNT0=0x65;
OCR0=0x00;
```

The Program with Timer0 [Polling: OVERFLOW (Part-2)]

```
while (1)
      {
       while ((TIFR&0x01)==0);
       TIFR=0x01;
       TCNT0=0x65;
       i++;
       if (i==100)
       {
          PORTC=~PORTC;
          i=0;
       }
      }
}
```

The Program with Timer0 [Polling: OUTPUT COMPARE (Part-1)]

```
#include <mega32.h>
int i=0;
void main(void)
{
DDRC=0xFF;
PORTC=0xFF;
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 15.625 kHz
// Mode: Normal top=0xFF
// OC0 output: Disconnected
// Timer Period: 16.384 ms
TCCR0=(0<<WGM00) | (0<<COM01) | (0<<COM00) | (0<<WGM01) |
   (1<<CS02) | (0<<CS01) | (1<<CS00);
TCNT0=0x00;
OCR0=0x9C;
```

The Program with Timer0 [Polling: OUTPUT COMPARE (Part-2)]

```
while (1)
      {
       while ((TIFR&0x02)==0);
       TIFR=0x02;
       TCNT0=0x00;
       i++;
       if (i==100)
       {
          PORTC=~PORTC;
          i=0;
       }
      }
}
```

The Program with Timer0 [Interrupt: OVERFLOW (Part-1)]

```
#include <mega32.h>
int i=0;
// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
i++;
if (i==100)
   {
     PORTC=~PORTC;
     i=0;
   }
// Reinitialize Timer 0 value
TCNT0=0x65;
```

} **No need to reset the Interrupt flag, it is reset by the uC within ISR**

The Program with Timer0 [Interrupt: OVERFLOW (Part-2)]

```
void main(void)
{
DDRC=0xFF;
PORTC=0xFF;
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 15.625 kHz
// Mode: Normal top=0xFF
// OC0 output: Disconnected
// Timer Period: 9.984 ms
TCCR0=(0<<WGM00) |
   (0<<COM01) |
   (0<<COM00) |
   (0<<WGM01) | (1<<CS02)
   | (0<<CS01) | (1<<CS00);
```

```
TCNT0=0x65;
OCR0=0x00;
// Interrupt(s) initialization
TIMSK=(0<<OCIE2) | (0<<TOIE2) |
   (0<<TICIE1) | (0<<OCIE1A) |
   (0<<OCIE1B) | (0<<TOIE1) |
   (0<<OCIE0) | (1<<TOIE0);
#asm("sei")
while (1)
      {
      }
}
```

The Program with Timer0 [Interrupt: OUTPUT COMPARE (Part-1)]

```
#include <mega32.h>
int i=0;
// Timer 0 output compare interrupt service routine
interrupt [TIM0_COMP] void timer0_comp_isr(void)
{
   i++;
I  if (i==100)
   {
        PORTC=~PORTC;
        i=0;
   }
   TCNT0=0x00;
}  // No need to reset the Interrupt flag, it is reset by the uC within ISR
```

The Program with Timer0 [Interrupt: OUTPUT COMPARE (Part-2)]

```
void main(void)
{
DDRC=0xFF;
PORTC=0xFF;
// Timer/Counter 0 i
// Clock source: System Clock
// Clock value: 15.625 kHz
// Mode: Normal top=0xFF
// OC0 output: Disconnected
// Timer Period: 16.384 ms
TCCR0=(0<<WGM00) |
   (0<<COM01) | (0<<COM00) |
   (0<<WGM01) | (1<<CS02) |
   (0<<CS01) | (1<<CS00);
```

```
TCNT0=0x00;
OCR0=0x9C;
//Interrupt(s) initialization
TIMSK=(0<<OCIE2) | (0<<TOIE2)
   | (0<<TICIE1) | (0<<OCIE1A) |
   (0<<OCIE1B) | (0<<TOIE1) |
   (1<<OCIE0) | (0<<TOIE0);

#asm("sei")
while (1)
      {

      }
}
```

The Same Task of Generating “Delay” with Timer2 [not by delay_ms() function]

- Time2 is very similar to Timer0 as this is also a 8-bit Timer.
- The only difference is that Timer2 cannot be used as Counter [i.e. for counting external event].
- Rather, it can count the pulses of external clock.
- The external clock has to be connected to specific two pins of the microcontroller.
- The frequency of the external clock is set as such a value that making real time clock is facilitated.
- And the frequency of the crystal has to be 32.768 kHz.

Registers Related to Timer0

- **TCNT2** register - the number of pulse Timer2 counts

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

- **OCR2** register - the value with which the comparison is made

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

- **TIMSK** register – all INTERRUPT ENABLE BITS (**all Timers**)

OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

- **TIFR** registers - all INTERRUPT FLAGS (**all Timers**)

OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

TIFR (Timer/Counter) Interrupt Flag Register

TIFR Register

TOV0 Timer0 flag bit; 0- did not overflow, 1-has overflowed (FF->00)
OCF0 Timer0 output compare flag bit; 0-did nor match, 1-matched
TOV1 Timer1 flag bit;
OCF1B Timer 1 output compare B match flag
OCF1A Timer 1 output compare A match flag
ICF1 Input Capture flag
TOV2 Timer2 flag bit
OCF2 Timer2 output compare flag bit

TIMSK Register

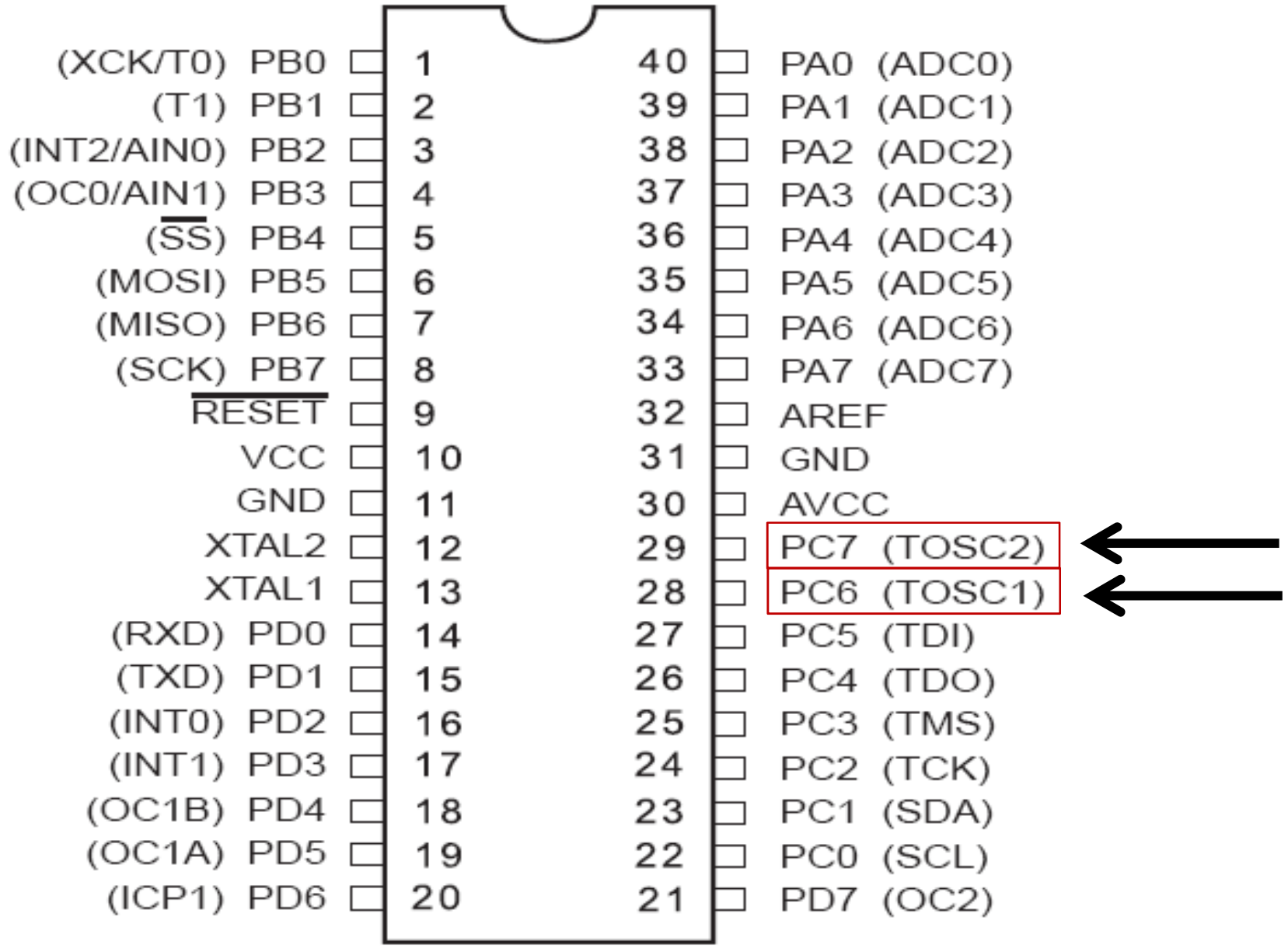
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

TOIE0	Timer0 Overflow Interrupt Enable
OCIE0	Timer0 Output Compare Interrupt Enable
TOIE1	Timer1 Overflow Interrupt Enable;
OCIE1B	Timer 1 Output Compare B match Interrupt Enable
OCIE1A	Timer 1 Output Compare A match Interrupt Enable
TICIE1	Timer/Counter1, Input Capture Interrupt Enable
TOIE2	Timer2 Overflow interrupt Enable
OCIE2	Timer2 output compare Interrupt Enable

Timer2 Programming

- Timer2 is very similar to Time0, both being 8-bit timers.
- But there are two distinct differences between the two.
  - Timer2 can be connected to real time counter. For that, we should connect a crystal of 32.768 kHz to TOSC1 and TOSC2 terminals and set AS2 bit of ASSR register.
  - Timer2 can NOT be used as counter of external events. Rather all setting of CS22:CS20 (3bits of TCCR2 register) are for internal frequency select. Of course the meaning of some of the combinations have different meaning than that of Timer0.

TOSC1 & TOSC2 Pins



Timer2 Programming

TCCR2 Register

FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20

- FOC2 – Bit 7 – Force Compare Match. This is a write only bit, writing 1 to it causes to act as if a compare match has occurred.

- WGM20, WGM21 – Bit 6 and 3 – Mode Selector
  0 0 – Normal
  0 1 – CTC (Clear Timer on Compare Match)
  1 0 – PWM, Phase Correct
  1 1 – Fast PWM

Timer2 Programming (Contd.)

TCCR2 Register

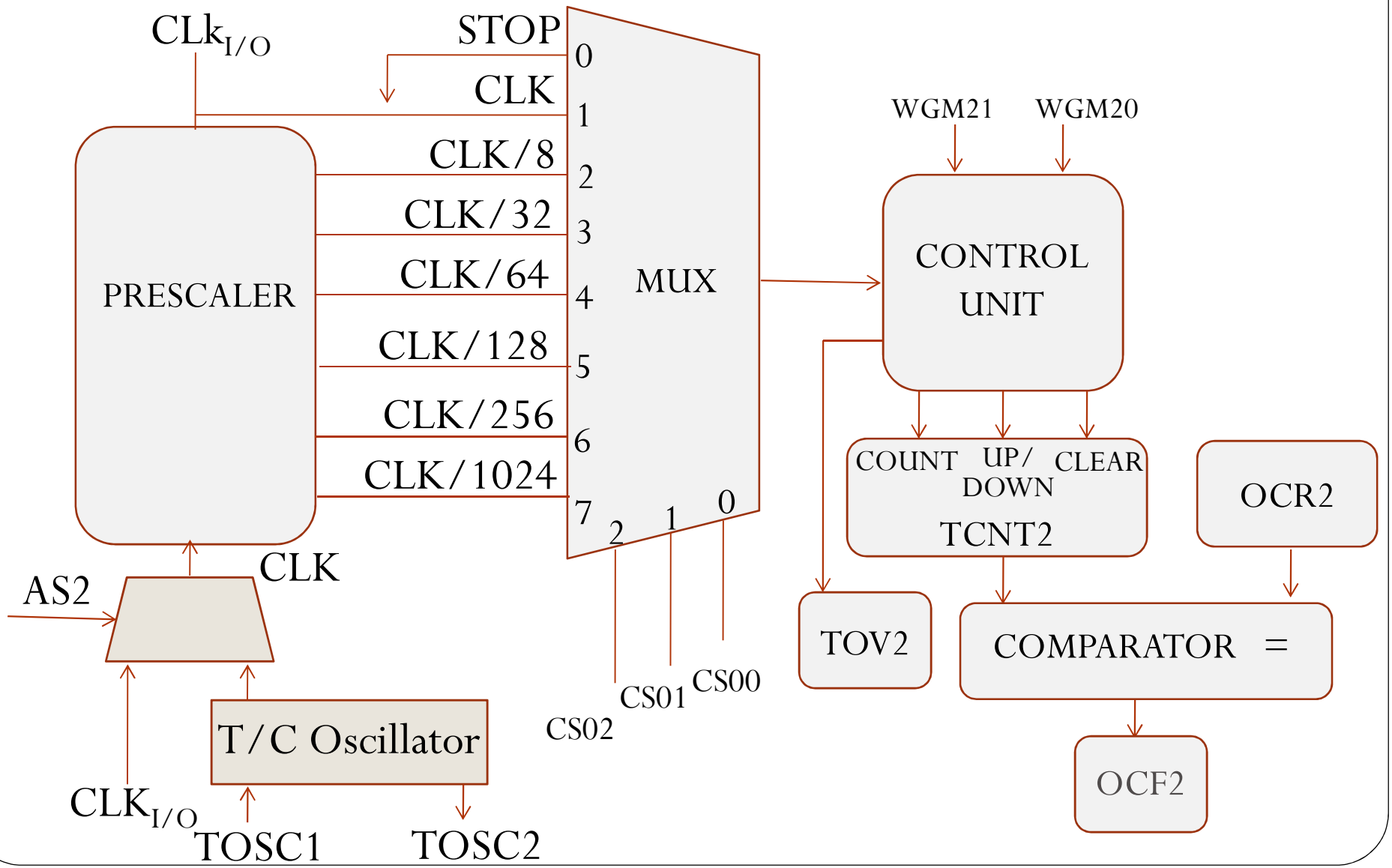
FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20

- COM 21:20 – Bit 5 and 4

  Compare output mode This mode controls the waveform generator (will be discussed in another lecture)
- CO22:20 – Bit 2, 1 and 0 – Timer clock selector

  (see the next slide)

Block Diagram for Timer2



TIFR (Timer/Counter) Interrupt Flag Register

TIFR Register

TOV2 Timer2 Overflow flag bit

OCF2 Timer2 output compare flag bit

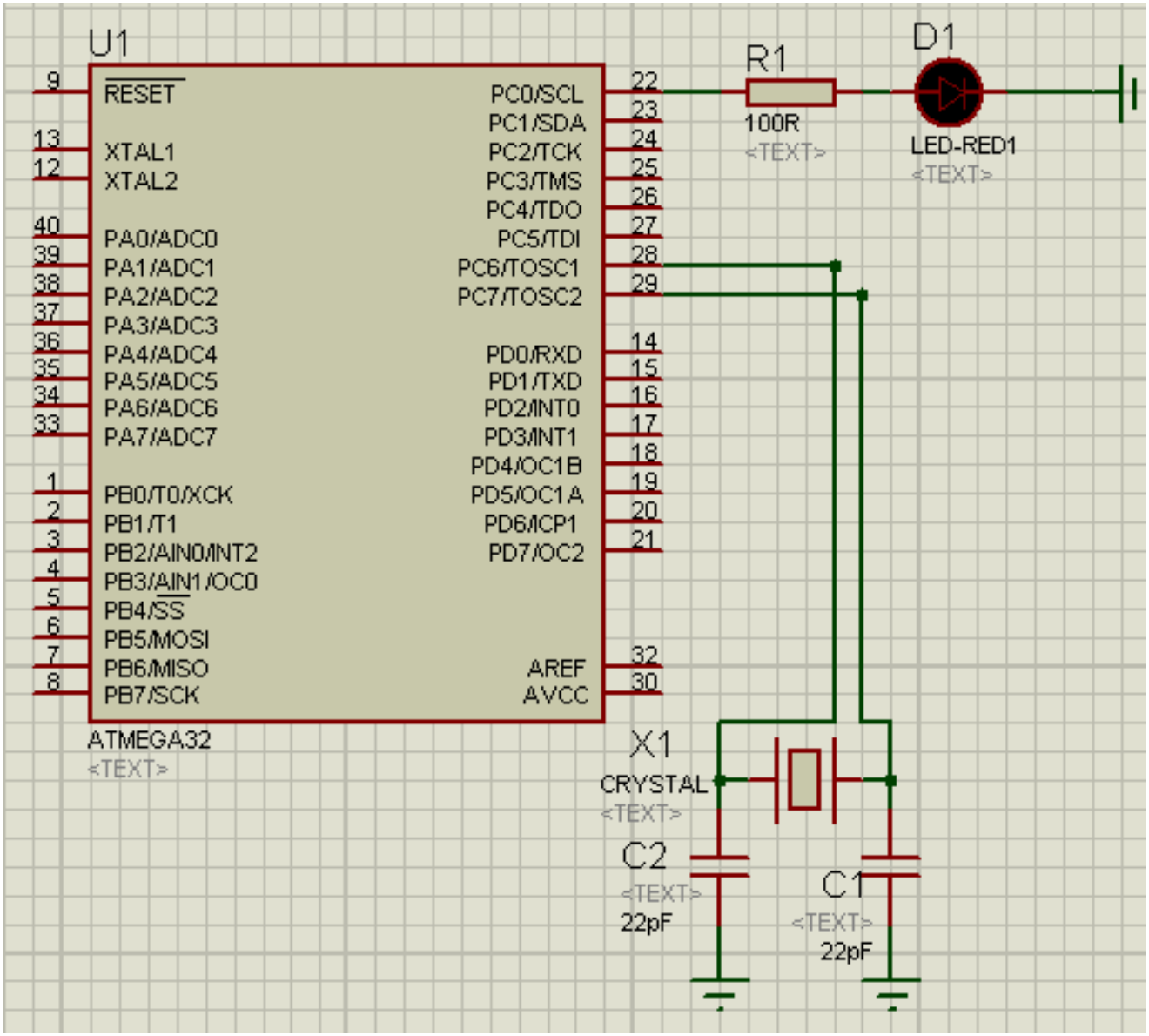
ASSR (Asynchronous Status Register)

ASSR Register

	AS2	TCN2UB	OCR2UB	TCR2UB

AS2 – When it is zero, Timer2 is clocked from $CLK_{I/O}$,
When it is set Timer2 works as RTC

The Circuit Developed in Proteus



Generation of a Train of Rectangular Pulses of a Fixed Frequency Using Timer2

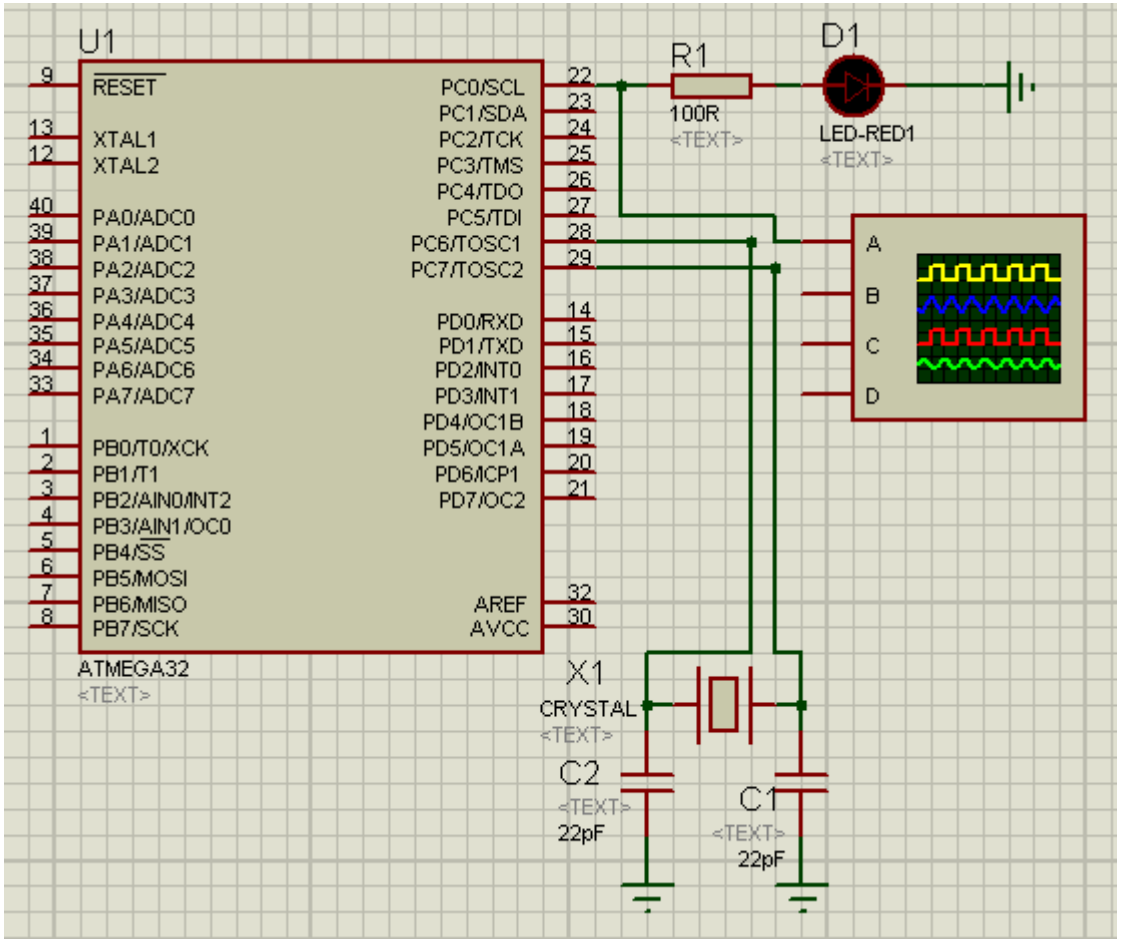
- Let us say, we want to generate a train of rectangular pulses having a frequency of 0.5 Hz.
- Therefore, the time period will be (1/0.5) sec or 2 sec.
- So the time in half cycle will be 1 sec.
- If we send high signal to a port pin for 1 Sec and then low for next 1 Sec and repeat this sequence, a train of pulse of 0.5 Hz will be generated.
- If we configure TIMER2 to be interrupted at every 1 Sec, and if we toggle a Port pin we shall obtain desired wave shape.
- Let us use external clock of 32.768 kHz with a Prescaling factor of 1024. Therefore duration of each pulse will be of (1024/32768) Sec.
- In other words, we need 32768/1024 or 32 (or 0x20) number of pulses for a time of 1 Sec.
- Therefore, in compare method, TCNT2=0x00; OCR2=0x20;
- And in TCCR2 > CS22:CS21:CS20=111;

The Program Code

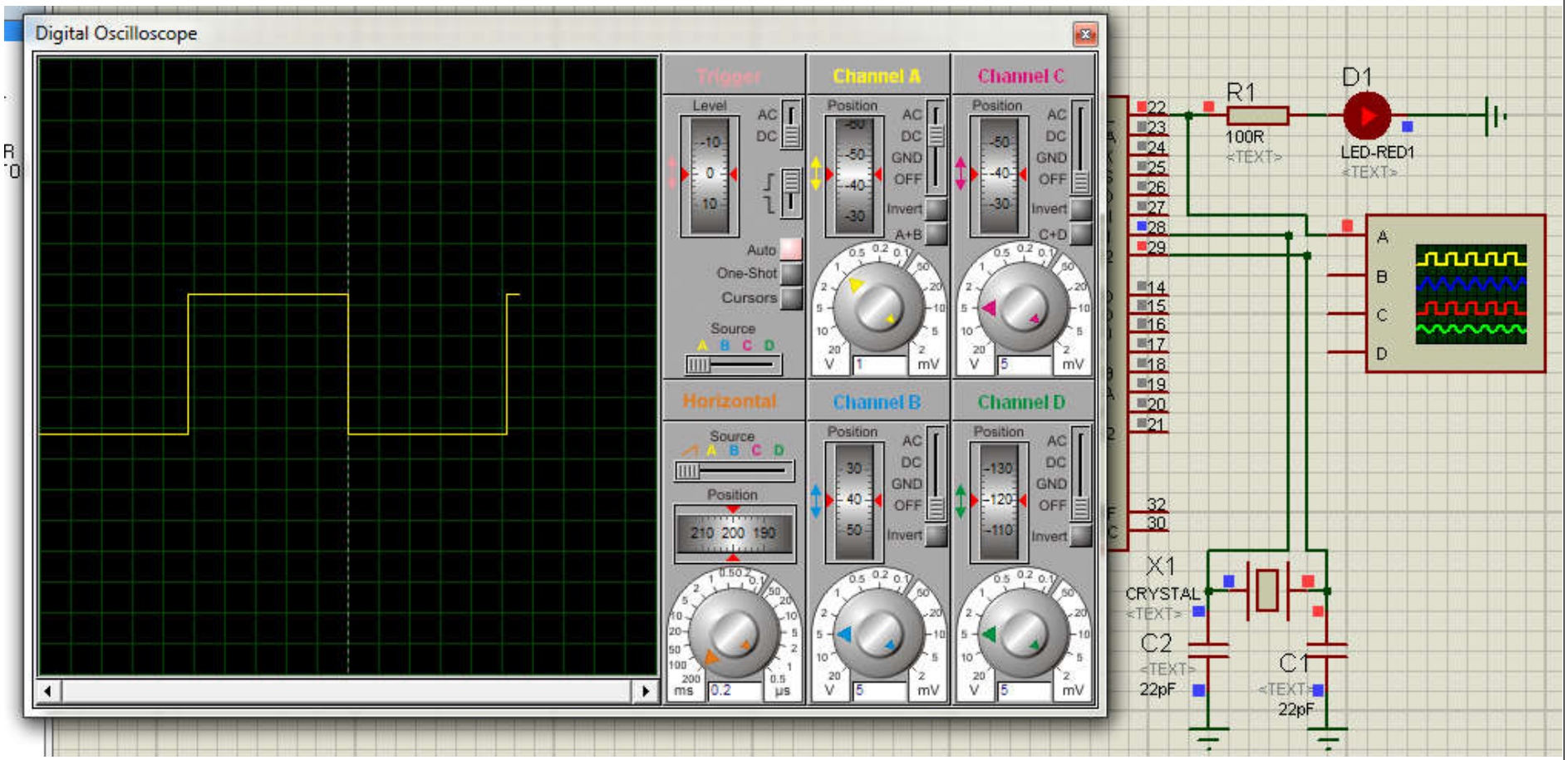
```
#include <mega32.h>
// Timer2 output compare interrupt service routine
interrupt [TIM2_COMP] void timer2_comp_isr(void)
{
PORTC.0=~PORTC.0;
TCNT2=0x00;
}
```

```
void main(void)
{
DDRC.0=1;
// Timer/Counter 2 initialization
// Clock source: TOSC1 pin
// Clock value: PCK2/1024
// Mode: Normal top=0xFF
// OC2 output: Disconnected
ASSR=1<<AS2;
TCCR2=(0<<PWM2) | (0<<COM21) | (0<<COM20) |
(0<<CTC2) | (1<<CS22) | (1<<CS21) | (1<<CS20);
TCNT2=0x00;
OCR2=0x20;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=(1<<OCIE2) | (0<<TOIE2) | (0<<TICIE1) |
(0<<OCIE1A) | (0<<OCIE1B) | (0<<TOIE1) | (0<<OCIE0) |
(0<<TOIE0);
// Global enable interrupts
#asm("sei")
PORTC=0x00;
while (1)
      {
      }
}
```

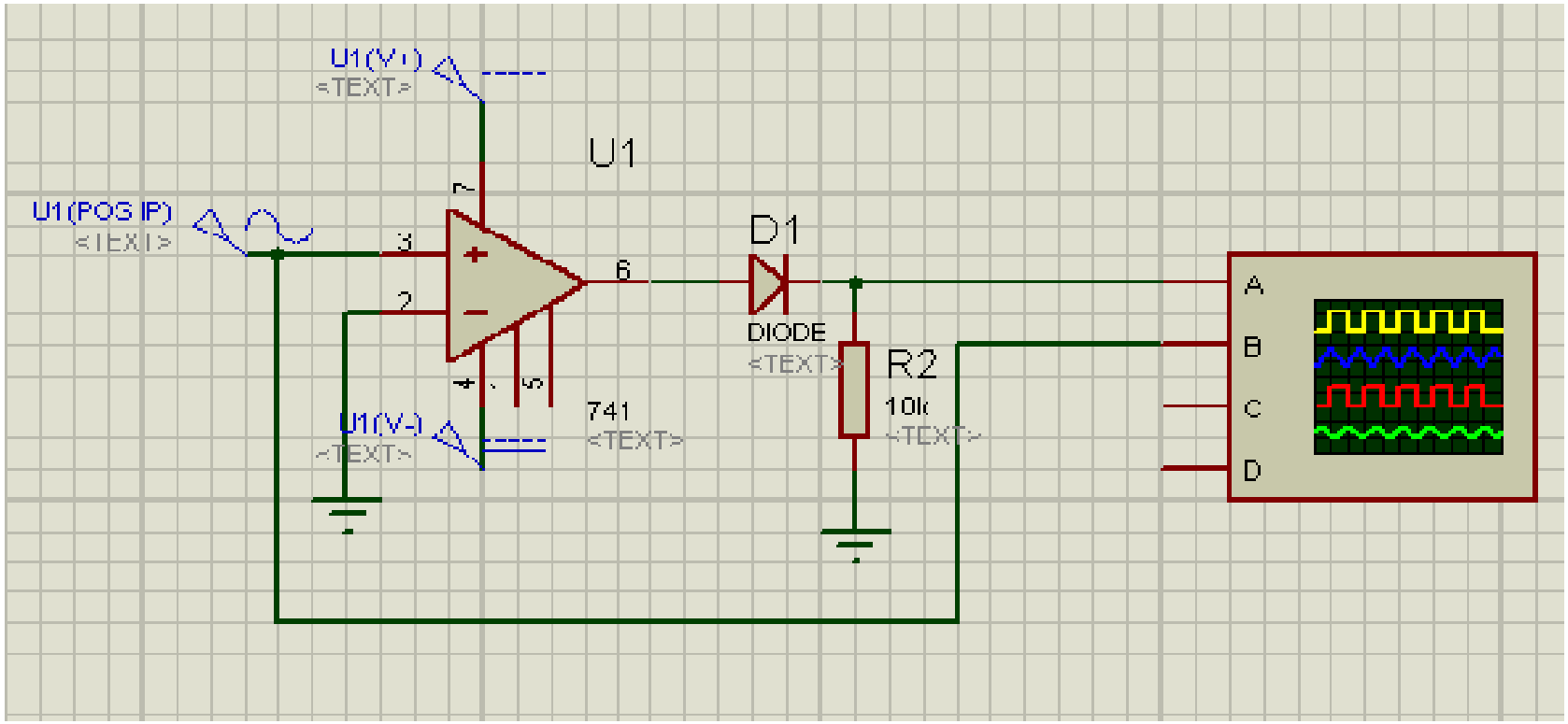
Adding Virtual Instrument : Oscilloscope



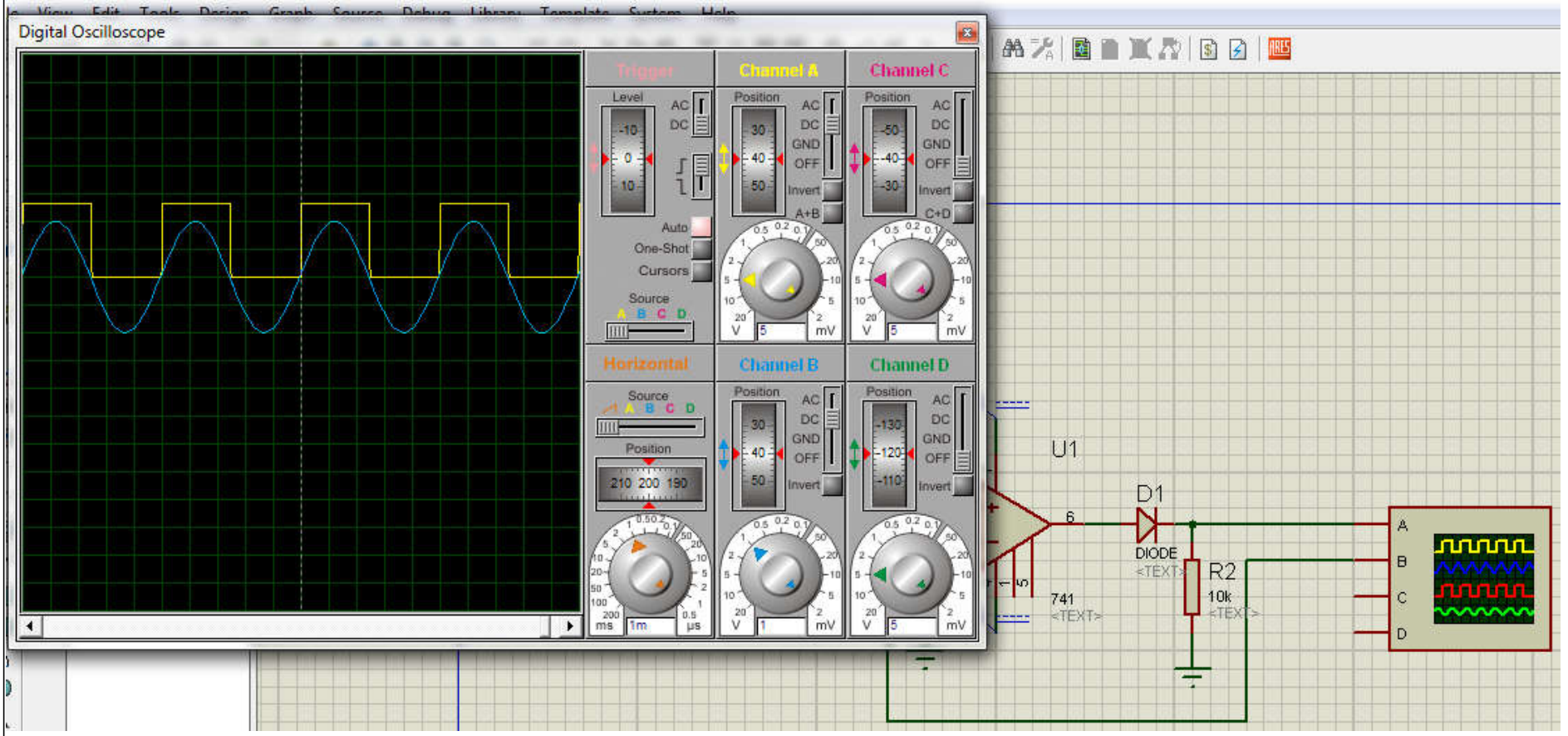
Output in the Oscilloscope



Adding Signal Generators



The Outputs


Digital Oscilloscope
Trigger
Channel A
Channel C
Horizontal
Channel B
Channel D
Level
Position
AC
DC
GND
OFF
Invert
A+B
C+D
Auto
One-Shot
Cursors
Source
A B C D
210 200 190
U1
741
D1
DIODE
R2
10k
A
B
C
D

Thanks