ZAGAZIG UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER & SYSTEMS ENGINEERING DEPT.

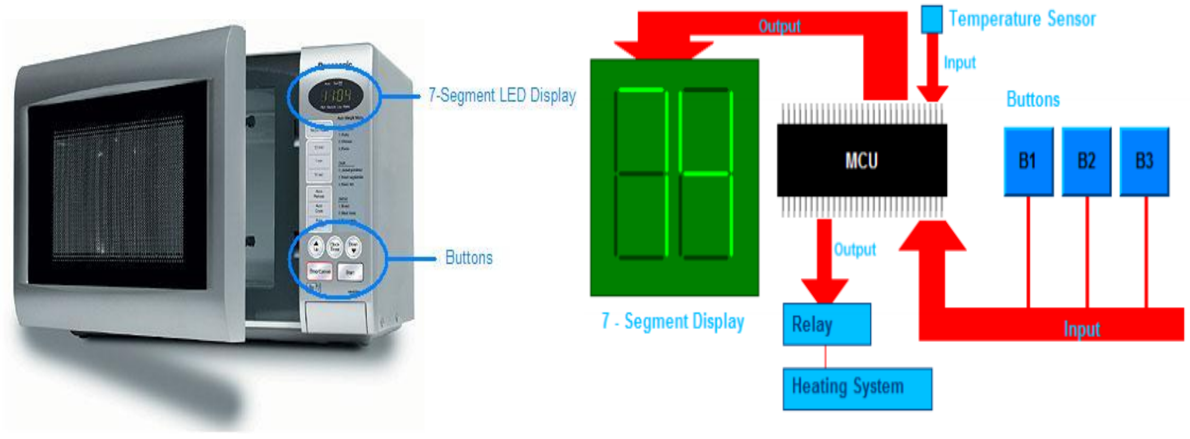
INTRODUCTION TO MICROCONTROLLER

OBJECTIVES

- At the end of this lab you should be able to:
  1. Identify the Atmel ATmega32 microcontroller, and associated hardware.
  2. Create a new project in CodeVision.
  3. Simulate the project in proteus.
  4. Run the program on microcontroller.

JUST WHAT IS A MICROCONTROLLER

- To get you understand quickly I define a microcontroller as a single chip computer. Yes it is a full computer in its own. It has a CPU. RAM, some amount of EEPROM (for secondary storage i.e. permanent storage without power), many on-chip peripherals (Timer, Serial communication, Analogue to Digital converters,..).
- But compared to a personal computer their resources (RAM, speed etc) are less. But that is what is required!!! Because P.C. is a general purpose computer, which means it can run thousands of different softwares that are available for specific needs. Like it can run a game. The same P.C. can run this browser in which you are reading this! It can run a custom solution for banks, railways and airways. It can run a 3D modeling, video editing & image editing software for a production company. Many of these are huge software, requiring lots of memory and CPU power. And a P.C. can run simultaneously many of this!!! So to run them the host computer should have enough RAM and CPU power so that it can run heaviest of them. But in case of a microcontroller(aka MCU) which is used for a specific purpose like switching a Microwave oven heating off after a preset time, and also when temperature exceed preset value.



- This design also involve controlling few indicator L.E.D. , 2 or 3 seven segment display, few switches and a relay controlling the oven. This doesn't need a monster with 3.2 GHz Intel Core 2 Quad extreme processor with 2 GB Dual channel DDR3 RAM, 320 GB HDD and Dedicated nVIDIA GEFORCE graphics !!!. What is of main concern is cost and size. What it will be running throughout its life is a simple program that is hardly more than 4KB in size, requiring less than 128 Bytes of RAM to store few variables and optionally a few bytes to store permanently the temperature and time set on last use so that it can read those on startup.

  Example where MCUs are used are:

  - Calculators, Electronic wending machines, Electronic weighing scales, Phones(digital with LCD and phonebook)
  - Anything that is small, but has great functions and is cheap!!!

ATMEGA32

- **PIN count:** Atmega32 have 40 pins. Two for Power (10: +5v, 11: ground), two for oscillator (pin 12, 13), one for reset (pin 9), three for providing necessary power and reference voltage to its internal ADC, and 32 (4×8) I/O pins.

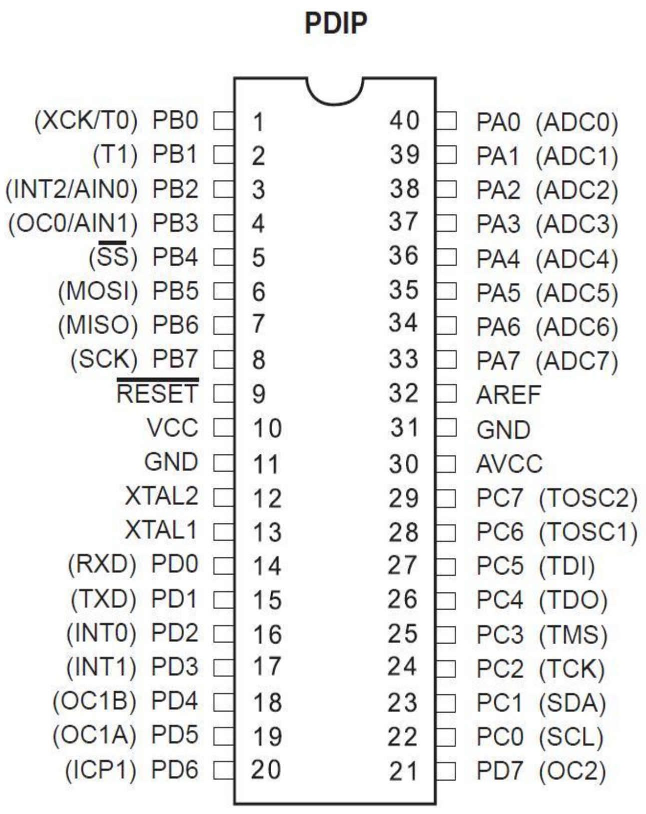
- **About I/O pins:** ATmega32 is capable of handling analogue inputs. Port A can be used either DIGITAL I/O Lines or each individual pin can be used as a single input channel to the internal ADC of ATmega32, plus a pair of pins (refer to ATmega32 datasheet) together can make a channel.
  - No pins can perform and serve for two purposes (for an example: Port A pins cannot work as a Digital I/O pin while the Internal ADC is activated) at the same time, it's the programmers responsibility to resolve the conflict in the circuitry and the program. Programmers are advised to have a look to the priority tables and the internal configuration from the datasheet.

- **Digital I/O pins:** ATmega32 has 32 pins (4portsx8pins) configurable as Digital I/O pins.

- **ADC:** It has one successive approximation type ADC, total 8 single channels are selectable. They can also be used as 7 (for TQFP packages) or 2 (for DIP packages) differential channels. Reference is selectable, either an external reference can be used or the internal 2.56V reference can be brought into action. There external reference can be connected to the Aref pin.

- **Communication Options:** ATmega32 have three data transfer modules embedded in it. They are
  - Two Wire Interface
  - USART
  - Serial Peripheral Interface

- **External Interrupt:** 3External interrupt is accepted. Interrupt sense is configurable.
- See datasheet for more information.



INPUT / OUTPUT

- You cannot imagine using microcontroller without using any of its I/O pins. Finally it's all about: taking input, processing it and generating output! Thus I/O registers and their correct settings is indispensable part while learning to program any microcontroller.

- We will learn how to use AVR ports and actually "code" for writing/reading data to/from port pins. It is slightly confusing for beginners, however once you understand it, you will certainly appreciate the way it is designed.

- **Registers: AVR** is 8 bit microcontroller. All its ports are 8 bit wide. Every port has 3 registers associated with it each one with 8 bits. Every bit in those registers configures pins of particular port. Bit0 of these registers is associated with Pin0 of the port; Bit1 of these registers is associated with Pin1 of the port… and likewise for other bits.

- These three registers are as follows :
  - DDRx register
  - PORTx register
  - PINx register

  (x can be replaced by A,B,C,D as per the AVR you are using)

- **DDRx register**

  DDRx (Data Direction Register) configures data direction of port pins. Means its setting determines whether port pins will be used for input or output. Writing 0 to a bit in DDRx makes corresponding port pin as input, while writing 1 to a bit in DDRx makes corresponding port pin as output.

  Example:

```
to make all pins of port A as input pins :
DDRA = 0b00000000;

to make all pins of port A as output pins :
DDRA = 0b11111111;

to make lower nibble of port B as output and higher nibble as input :
DDRB = 0b00001111;
```

- **PINx register**

  PINx (Port IN) used to read data from port pins. In order to read the data from port pin, first you have to change port’s data direction to input. This is done by setting bits in DDRx to zero. If port is made output, then reading PINx register will give you data that has been output on port pins.

  Example:

```
to read data from port A:
DDRA = 0x00;    //Set port a as input
 x = PINA;      //Read contents of port a
```

- **PORTx register**

  PORTx is used for two purposes:

1. To output data: when port is configured as output

   →When you set bits in DDRx to 1, corresponding pins becomes output pins. Now you can write data into respective bits in PORTx register. This will immediately change state of output pins according to data you have written.

   Example:

```
to output 0xFF data on port b
DDRB = 0b11111111;        //set all pins of port b as outputs
PORTB = 0xFF;             //write data on port
```

2. To input data

   When you set bits in DDRx to 0, i.e. make port pins as inputs, then corresponding bits in PORTx register are used to activate/deactivate pull-up registers associated with that pin. In order to activate pull-up resister, set bit in PORTx to 1, and to deactivate (i.e. to make port pin tri stated) set it to 0.
   In input mode, when pull-up is enabled, default state of pin becomes '1'. So even if you don't connect anything to pin and if you try to read it, it will read as 1. Now, when you externally drive that pin to zero (i.e. connect to ground / or pull-down), only then it will be read as 0.

   However, if you configure pin as tri state. Then pin goes into state of high impedance. We can say, it is now simply connected to input of some OpAmp inside the microcontroller and no other circuit is driving it from microcontroller. Thus pin has very high impedance. In this case, if pin is left floating (i.e. kept unconnected) then even small static charge present on surrounding objects can change logic state of pin. If you try to read corresponding bit in pin register, its state cannot be predicted. This may cause your program to go haywire, if it depends on input from that particular pin.

   - Thus while, taking inputs from pins using switches to take input, always enable pull-up resistors on input pins.
   - while using on chip ADC, ADC port pins must be configured as tri stated input.

   Example:

```
to make port a as input with pull-up enabled and read data from port a
DDRA = 0x00;        //make port a as input
PORTA = 0xFF;       //enable all pull-ups
y = PINA;           //read data from port a pins
```

```
to make port b as tri stated input
DDRB  = 0x00;        //make port b as input
PORTB = 0x00;        //disable pull-ups and make it tri state
```

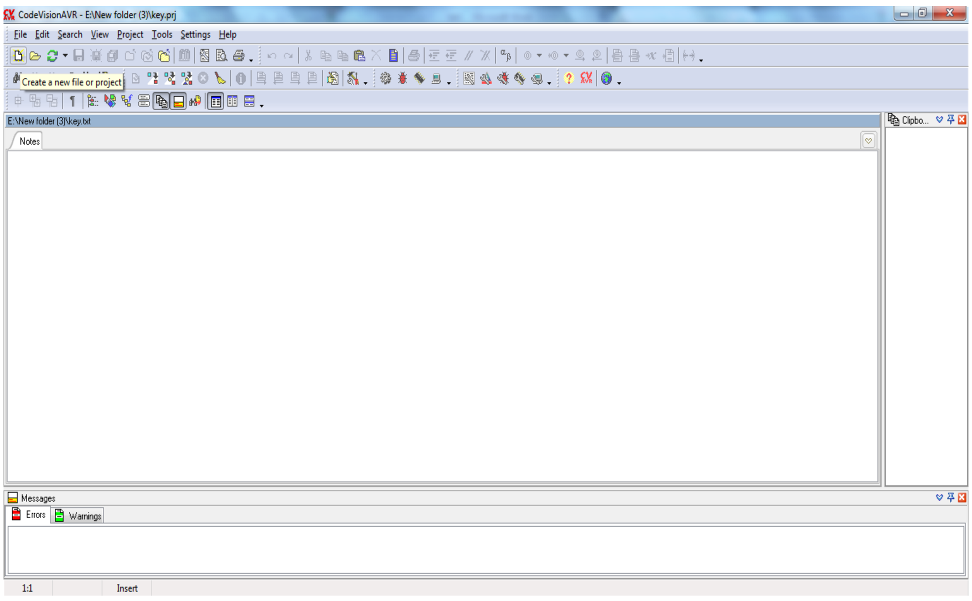
```
 to make lower nibble of port a as output, higher nibble as input with pull-up enabled
DDRA  = 0x0F;        //lower nib> output, higher nib> input
PORTA = 0xF0;       //lower nib> set output pins to 0, higher nib> enable pull-ups
```

Following table lists register bit settings and resulting function of port pins

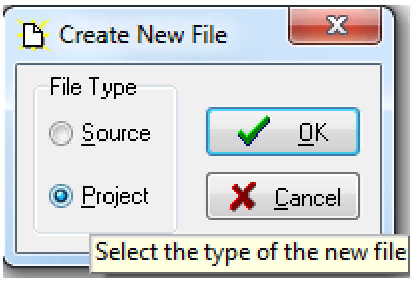
register bits → pin function ↓	DDRx.n	PORTx.n	PINx.n
tri stated input	0	0	read data bit x = PINx.n;
pull-up input	0	1	read data bit x = PINx.n;
output	1	write data bit PORTx.n = x;	n/a

START PROGRAMING

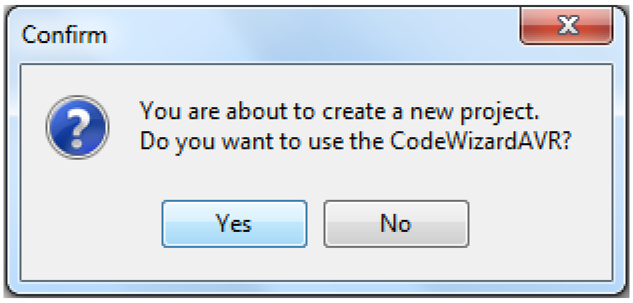
First, run CodeVision and create a new project:


CodeVisionAVR - E:\New folder (3)\key.prj
File Edit Search View Project Tools Settings Help
Create a new file or project
E:\New folder (3)\key.txt
Notes
Clipbo...
Messages
Errors
Warnings
1:1
Insert

Then you will be asked if you want to create project or create a source.



Select to project and then click ok then you will be asked if you want to use codewizardAVR automatic program generator, it allow you to easily write code.



Confirm

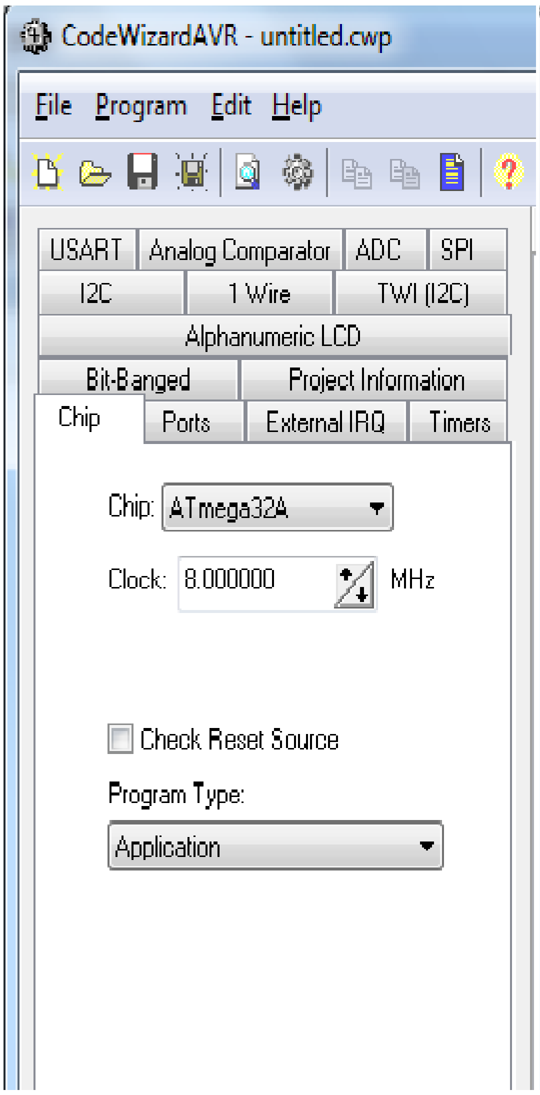
You are about to create a new project.
Do you want to use the CodeWizardAVR?

Click yes to use codewizardAVR as shown in the next figure, here you can do so many things! But we will only change two things:

→Select the right chip

→Change Portc to be output

As shown in these pictures



CodeWizardAVR - untitled.cwp

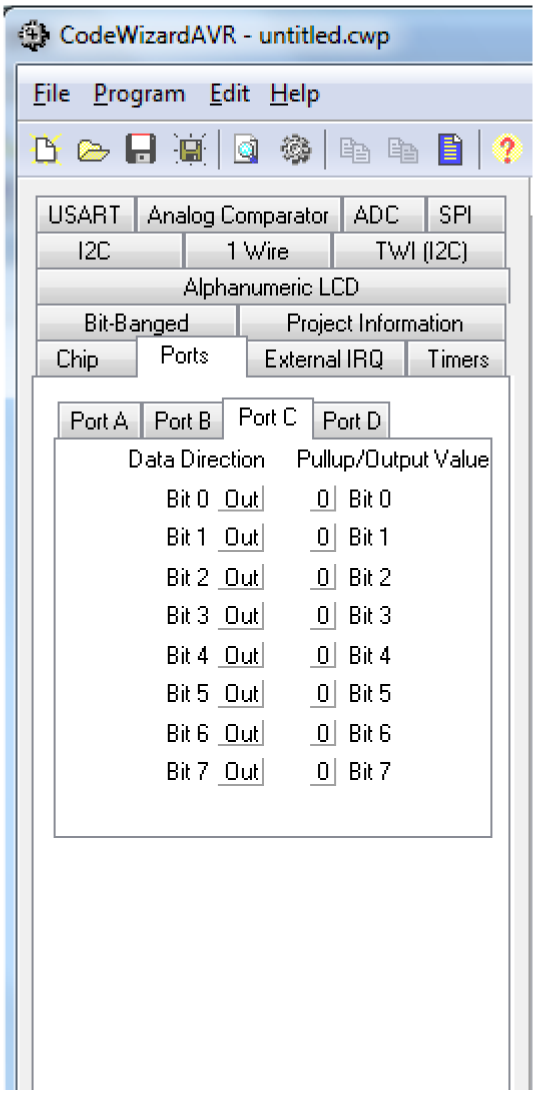
File Program Edit Help
USART Analog Comparator ADC SPI
I2C 1 Wire TWI (I2C)
Alphanumeric LCD
Bit-Banged Project Information
Chip Ports External IRQ Timers

Chip: ATmega32A

Clock: 8.000000 MHz

☐ Check Reset Source

Program Type: Application

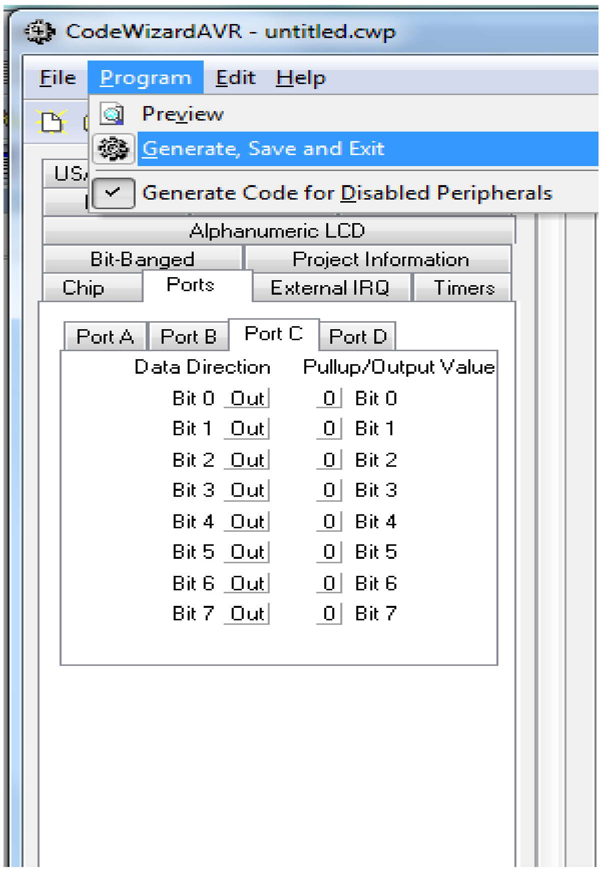


CodeWizardAVR - untitled.cwp

File Program Edit Help
USART Analog Comparator ADC SPI
I2C 1 Wire TWI (I2C)
Alphanumeric LCD
Bit-Banged Project Information
Chip Ports External IRQ Timers
Port A Port B Port C Port D

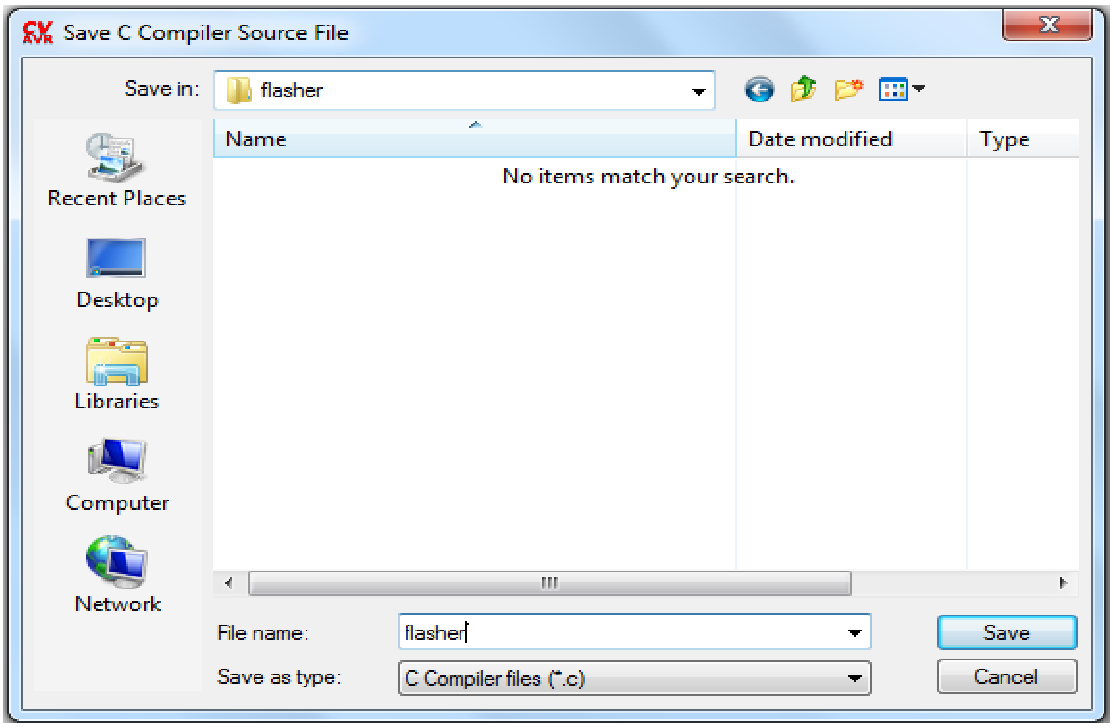
Data Direction		Pullup/Output Value	
Bit 0	Out	0	Bit 0
Bit 1	Out	0	Bit 1
Bit 2	Out	0	Bit 2
Bit 3	Out	0	Bit 3
Bit 4	Out	0	Bit 4
Bit 5	Out	0	Bit 5
Bit 6	Out	0	Bit 6
Bit 7	Out	0	Bit 7

Then we can generate the project and the code files.



Data Direction		Pullup/Output Value	
Bit 0	Out	0	Bit 0
Bit 1	Out	0	Bit 1
Bit 2	Out	0	Bit 2
Bit 3	Out	0	Bit 3
Bit 4	Out	0	Bit 4
Bit 5	Out	0	Bit 5
Bit 6	Out	0	Bit 6
Bit 7	Out	0	Bit 7

By doing this, CodeVision will ask you where to save the files and what names to use. put everything in the same folder and give the same name to each file .



Save in: flasher

No items match your search.

File name: flasher

Save as type: C Compiler files (*.c)

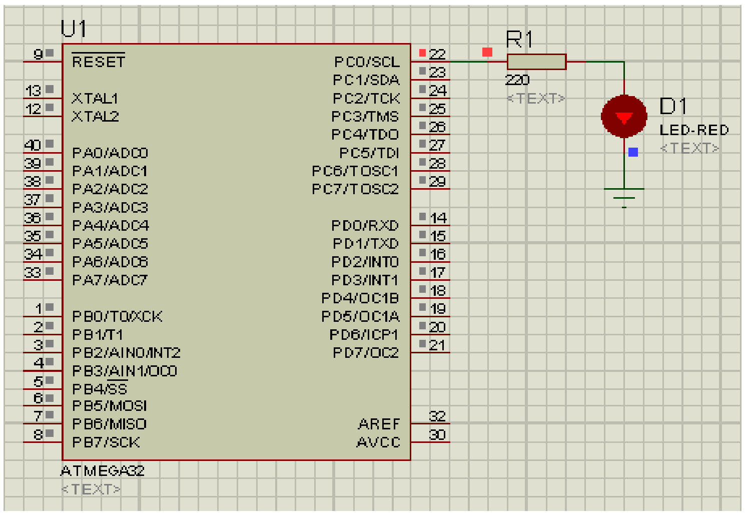
Now the code is generated

Example1: **Blink LED on PORTC**

Add the following code to the generated code.

```
PORTC =0x01;
delay_ms(500);
PORTC =0x00;
delay_ms(500);
```

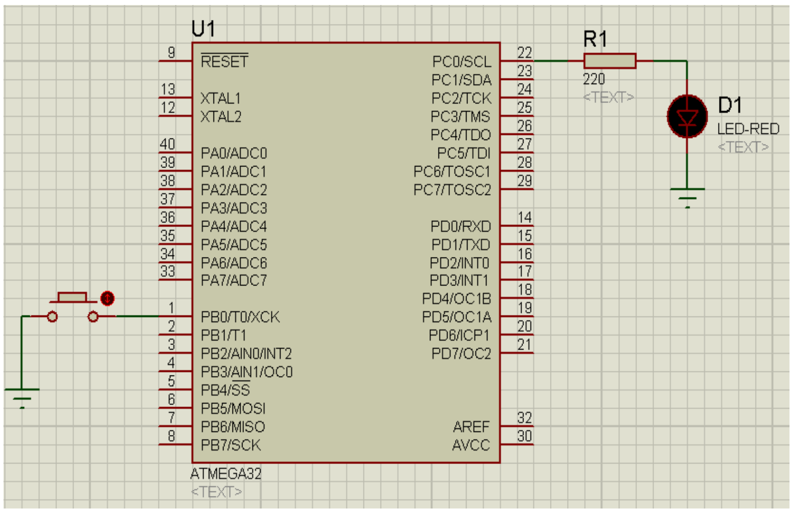
to call delay_ms() you must add delay.h library and the Circuit design is shown below



Example2: **Adding a Button to the Microcontroller and Making it do something**

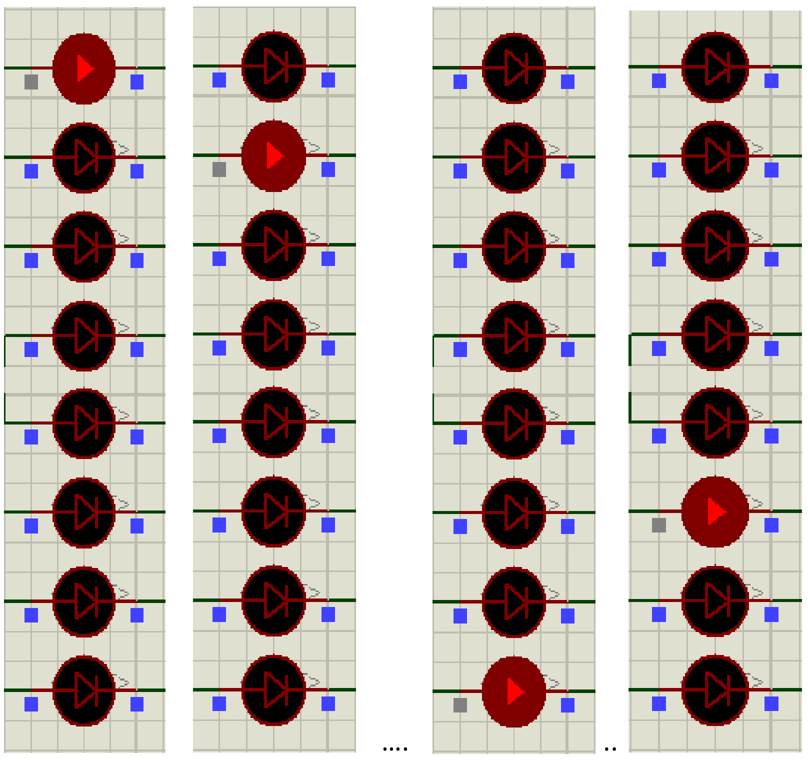
Adding a button or switch to the circuit enables the microcontroller to receive human input.
Add the following code:

```
key =PINB.0;
if(key == 1)
PORTC = 0x00;
else
{
PORTC = 0x01;
delay_ms(10);
}
```



REPORT

Q1: make the following flasher: the direction of the light shift right and left for ever.



Q2: output the decimal numbers on one seven-segment.