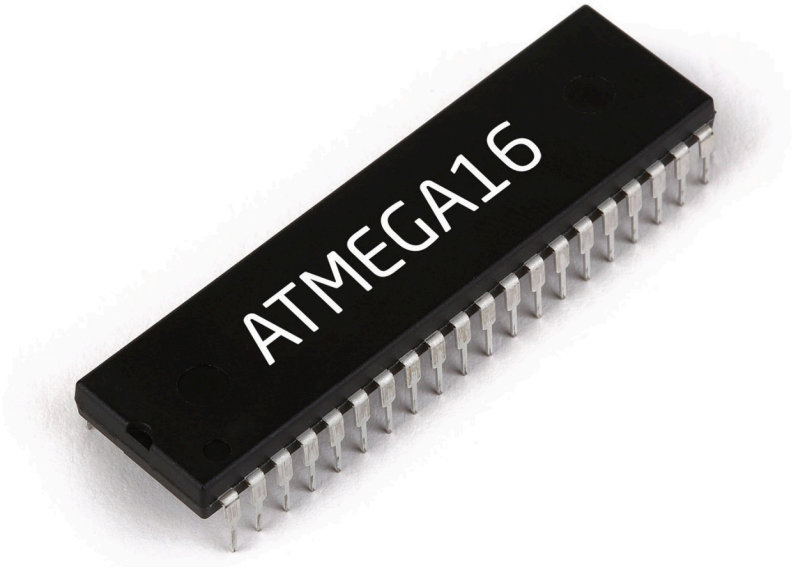
DEVELOPMENT KIT of AVR DOCUMENTATION



List of Contents

3 Examples

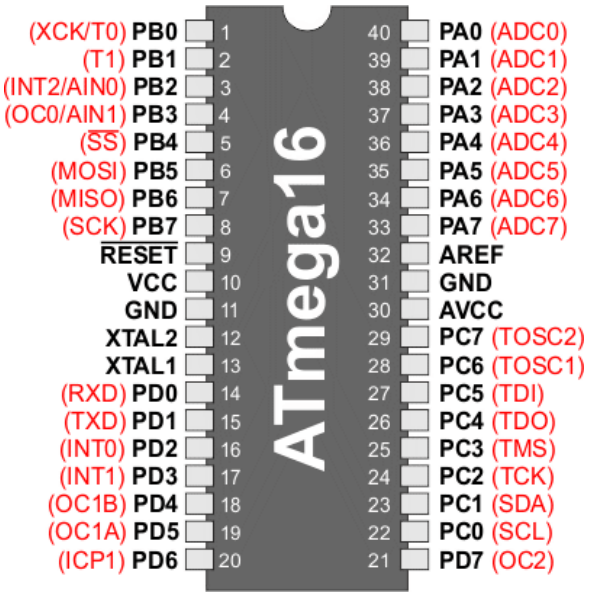
- *LED BLINKING*
- *LED CHASER*
- *BUZZER*
- *LCD*
- *RELAY*
- *SEVEN SEGMENT DISPLAY*
- *RTC*
- *KEYPAD*
- *BUTTON*
- *MULTIPLE DISPLAY*
- *SERIAL*
- *STEPPER*

Introduction

*The **ATmega16** is an 8-bit microcontroller from Microchip's AVR family, designed with a **RISC architecture** that allows it to execute most instructions in a single clock cycle, giving it a speed of **up to 1 million instructions per second per MHz**. It has **16 KB of flash memory** for program storage, **1 KB of SRAM** for temporary data, and **512 bytes of EEPROM** for permanent data storage. The microcontroller can run at speeds up to **16 MHz** and comes with built-in communication features like **USART, SPI, I²C, and JTAG**, which make it easy to connect with other devices and support debugging.*

*It also includes an **ADC, timers, PWM outputs, and 32 I/O pins**, making it flexible for many applications. With its efficiency and wide voltage range, the ATmega16 is commonly used in **robotics, IoT devices, automation, and educational projects**.*

Pin Diagram



Specifications

- Looks like the specifications you pasted are actually for the ATmega32, not the ATmega16 🙂.

Here’s a clean and short bullet-point version of the ATmega16 specifications, formatted properly like your list:

- ATmega16 Specifications
- 32 × 8 general-purpose working registers
- 16 KB in-system self-programmable Flash program memory
- 1 KB internal SRAM
- 512 bytes EEPROM
- Available in 40-pin DIP, 44-lead TQFP, 44-pad QFN/MLF packages
- 32 programmable I/O lines
- 8-channel, 10-bit ADC
- Two 8-bit timers/counters with prescalers and compare modes
- One 16-bit timer/counter with prescaler, compare, and capture modes
- 4 PWM channels
- In-system programming by on-chip bootloader
- Programmable watchdog timer with on-chip oscillator
- Programmable serial USART
- Master/Slave SPI serial interface
- JTAG interface for on-chip debugging and programming

Application

- It used in different temperature control systems.
- It used in the different analog signal calculation and management techniques.
- It used in different entrenched schemes like chocolate apparatus, peddling mechanism.
- It used for controlling the motor.
- It used for Numerical signal handling.
- It used for Marginal Interfacing scheme.

Material Required

- USB Cable
- Flat Ribbon Cable
- AVR Cable

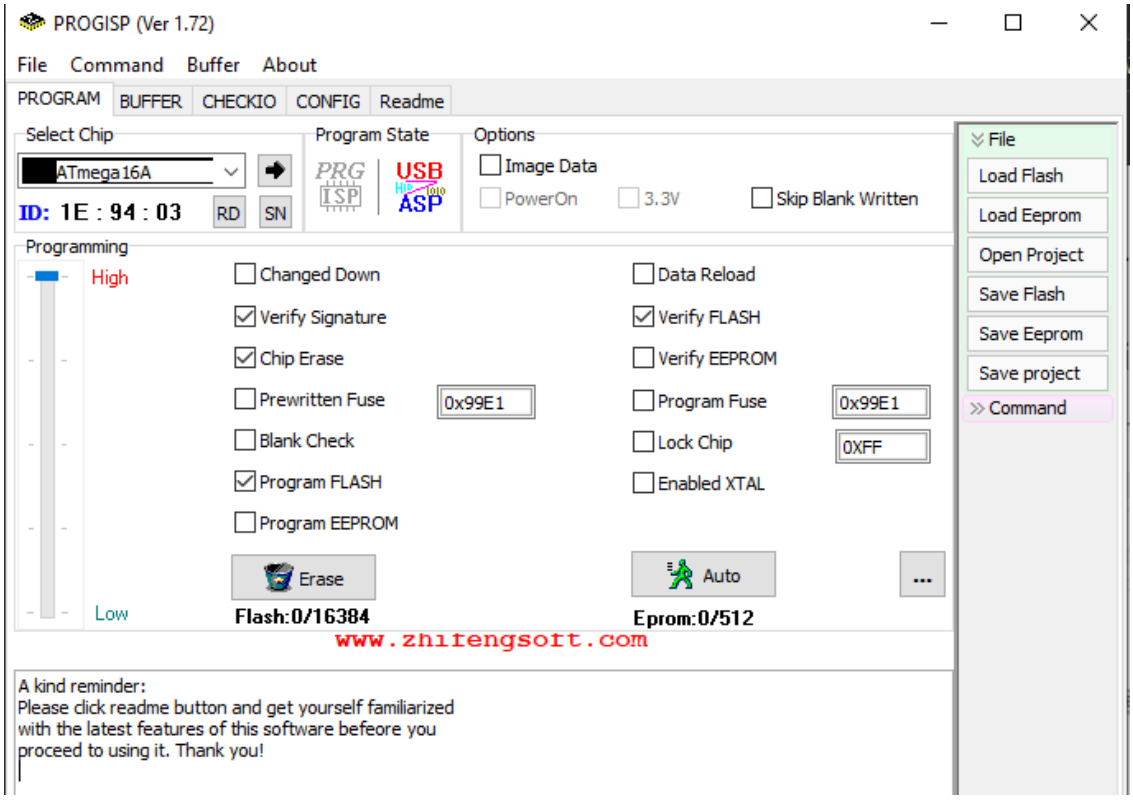
Imp Steps To Be Followed

First, download the file **usbasp.2011-05-28.tar.gz (519 KB)**.

After extracting, open the **AVR** folder and then go to the **Usbaspv2011_Proteus8.3Files** folder.

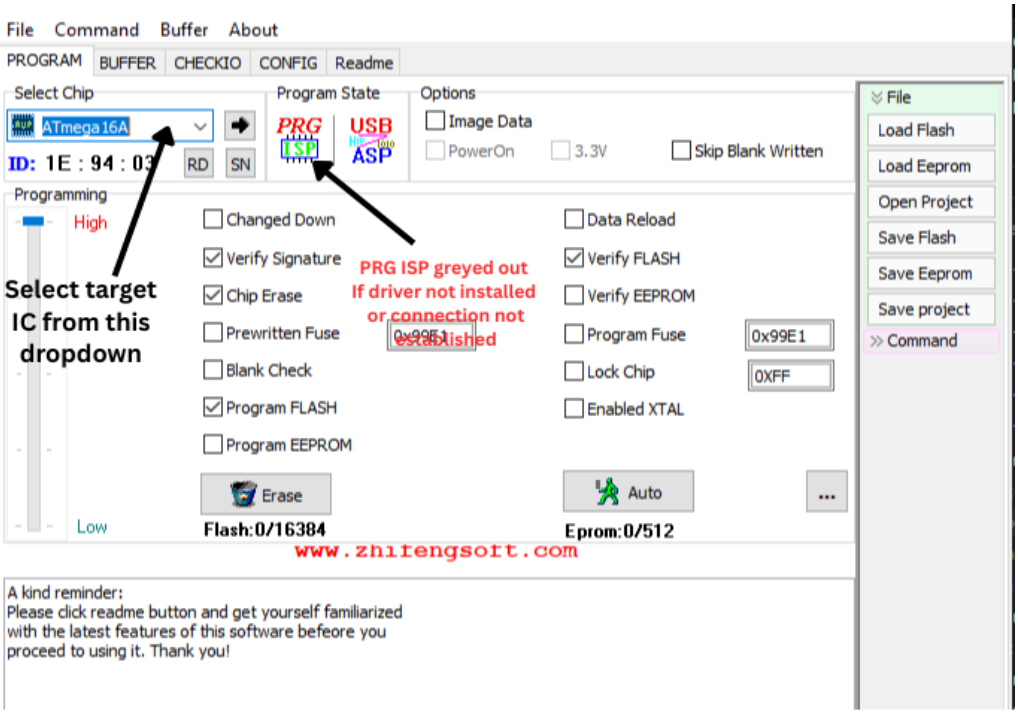
Inside this folder, locate and open the **ProgISP application (577 KB)**.

Opening this application will display the programming window used for loading and flashing the .hex file into the microcontroller.



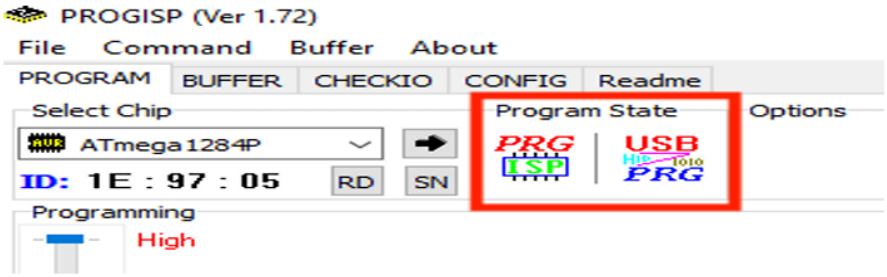
Steps :-

Step 1: First, connect the AVR kit to the computer using a USB cable. Once connected, the **Program State** indicator in **ProgISP** will be highlighted.



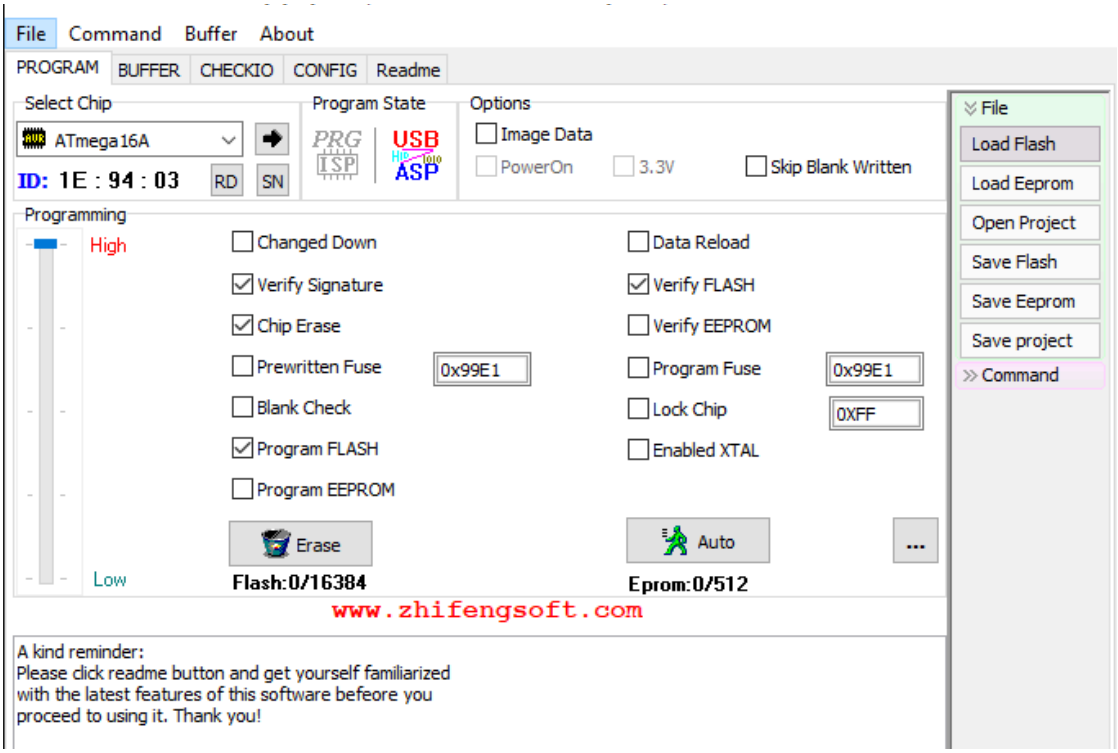
Step 2:

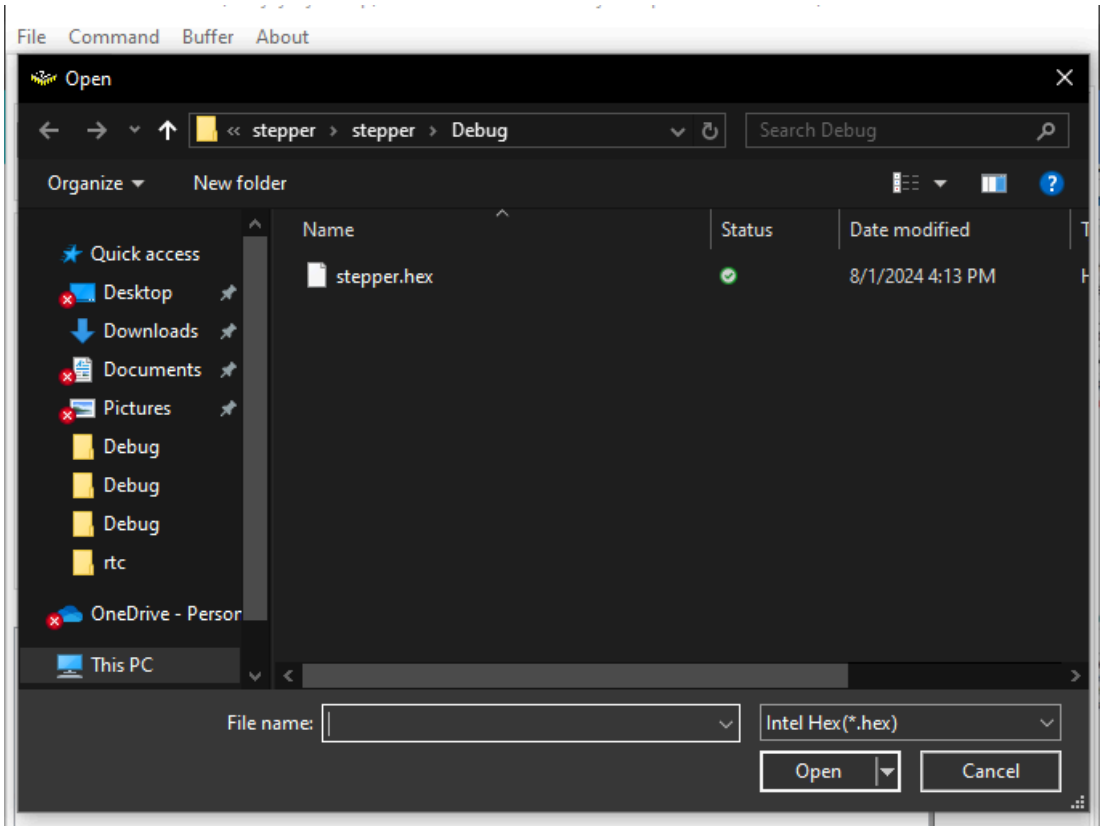
Move the cursor to **Select Chip** and choose the correct IC (for example, **ATmega32A**).



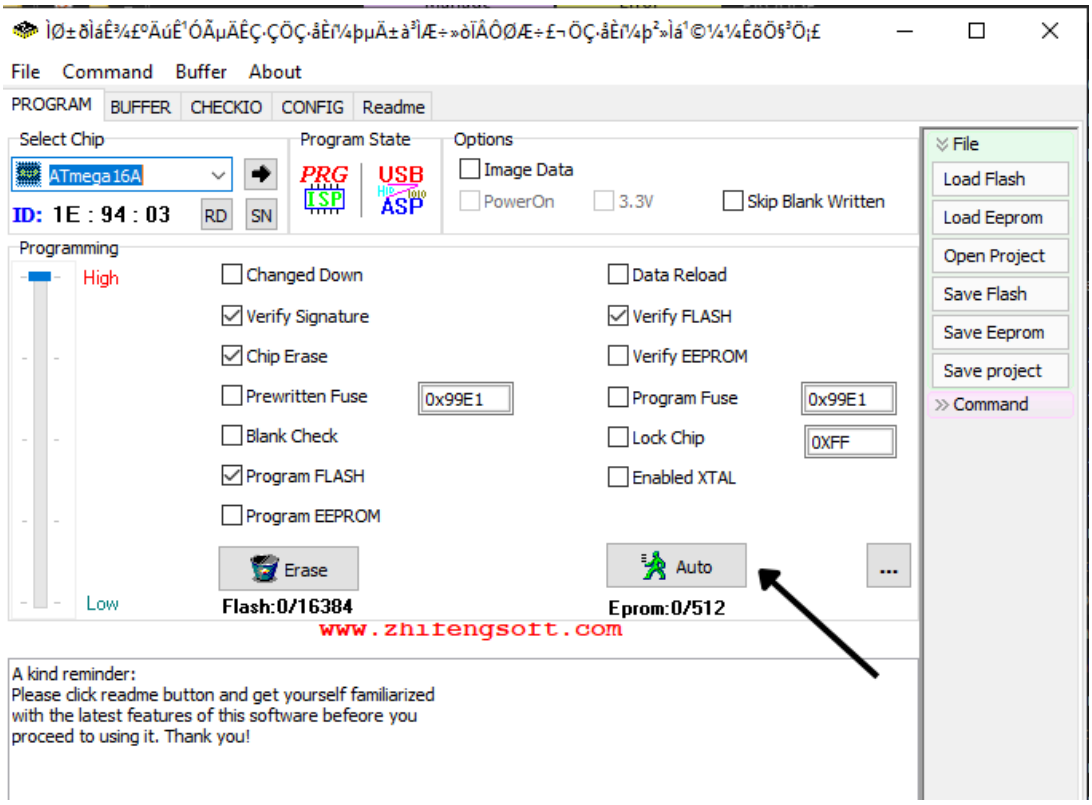
Step 3:

Load the program by selecting **Load Flash**, then browse and select your **.hex file**, and finally click **Open**.



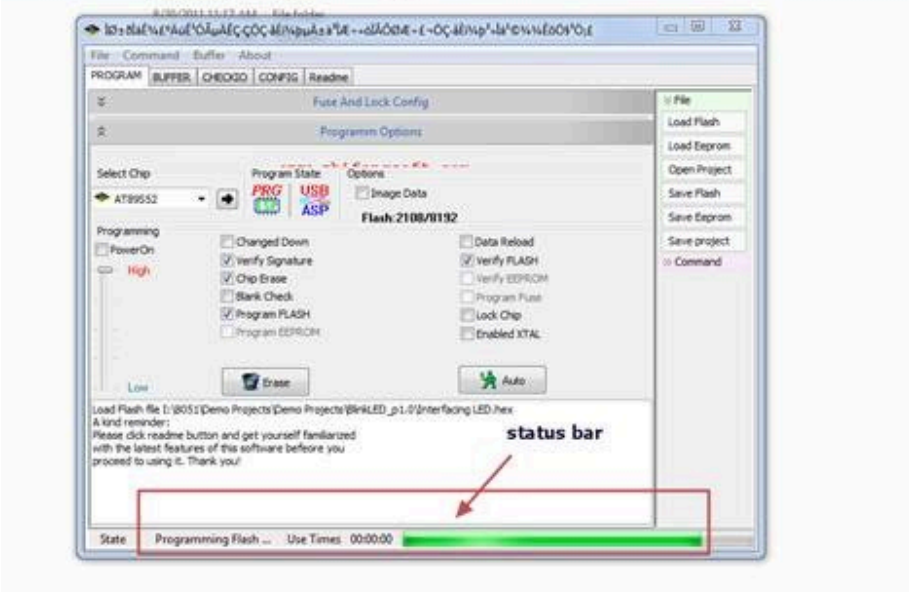


Then click on **Auto**



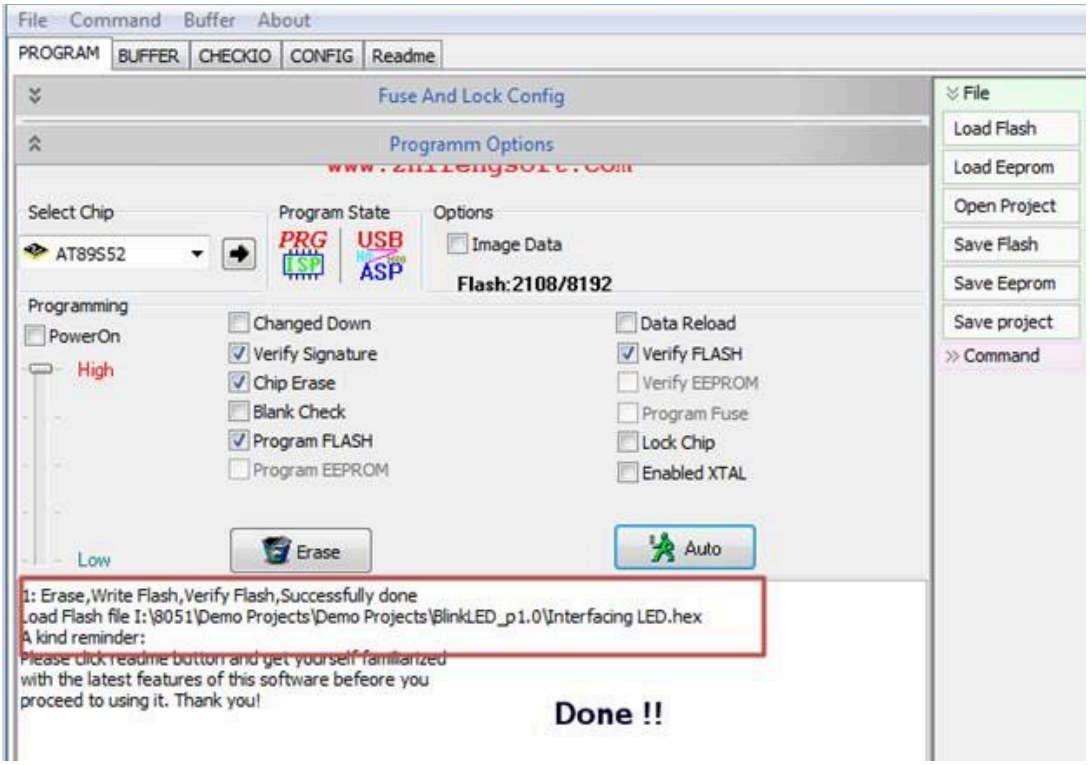
Step 4:-

See the output on kit



Step 5:-

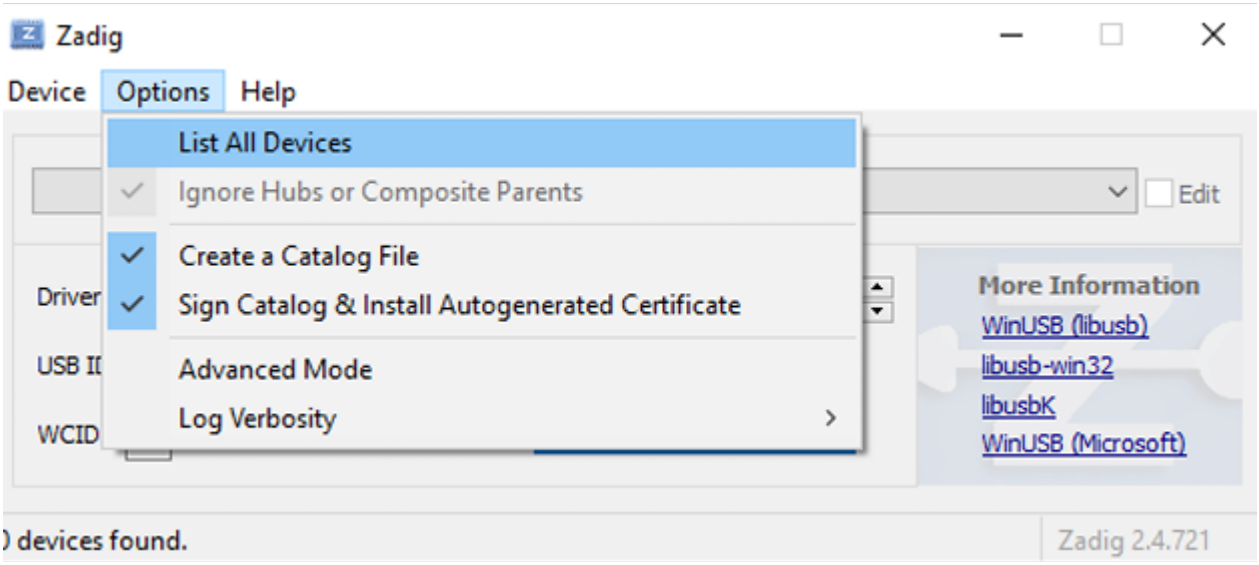
It shows the message after the file uploaded sucessfully



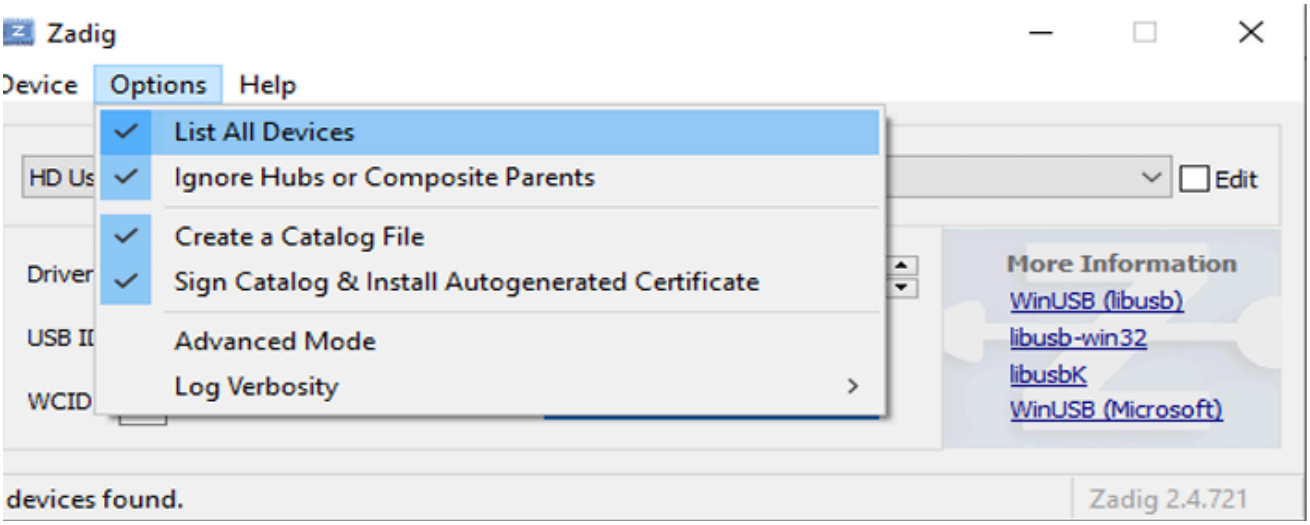
Zadig-2.7 Driver Installation

1. **Plug in your RTL device**
   - When you connect the device for the first time, Windows may either request a driver or automatically install a Microsoft driver.
   - This is fine—it will be replaced in the next steps using Zadig.
2. **Do not use the CD driver**
   - Do **not** install any software from the CD that comes with the device.
3. **Download Zadig**
   - Get the latest version from: http://zadig.akeo.ie
4. **Extract the downloaded file**
   - Zadig is usually packed in a **.7z file**.
   - Use **7-Zip** (http://www.7-zip.org) to extract it.
5. *(Alternative download: You can directly get version 2.1.1 as a zip file from this link)*
6. **Run Zadig**
   - Open **Zadig.exe**.
   - You should now see the application running with an **empty list** of devices.

- Plug in USBASP
- Install Zadig Step
- Open Option

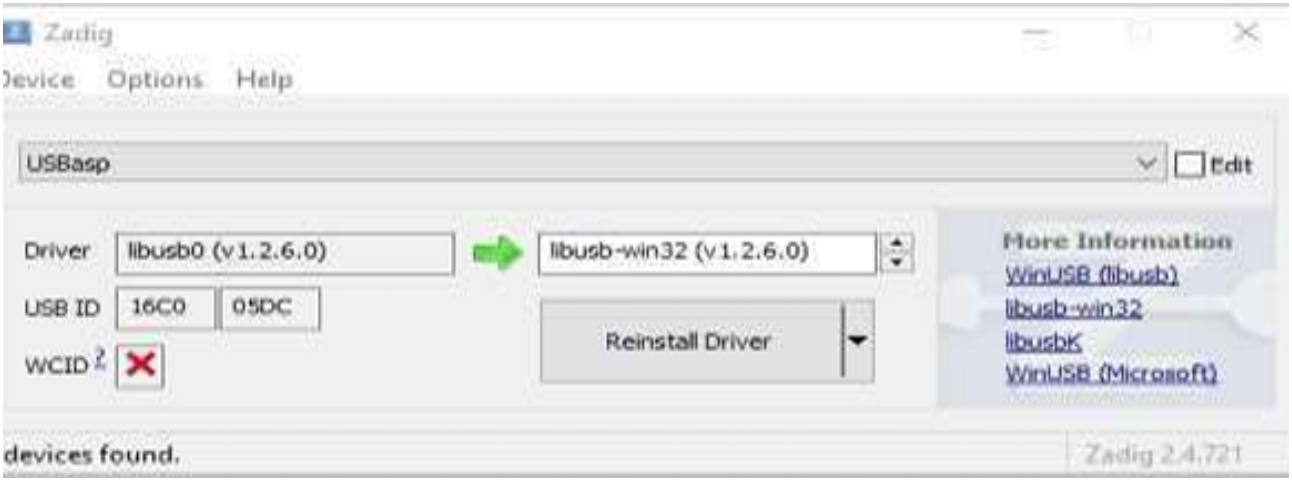


- Check List All Devices

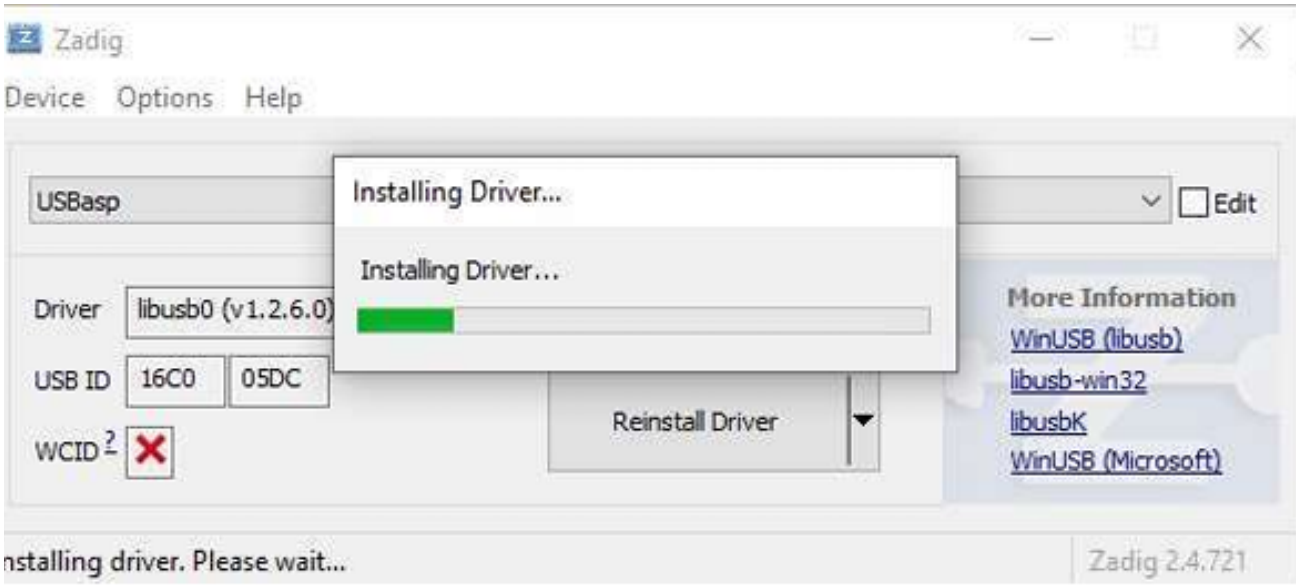


- Select USB ASP

Select Libusb- win32


Zadig
Device Options Help
USBasp
Edit
Driver
libusb0 (v1.2.6.0)
libusb-win32 (v1.2.6.0)
USB ID
16C0
05DC
WCID
Reinstall Driver
More Information
WinUSB (libusb)
libusb-win32
libusbK
WinUSB (Microsoft)
devices found.
Zadig 2.4.721

- Click Reinstall Driver



- Check Your Device Manager

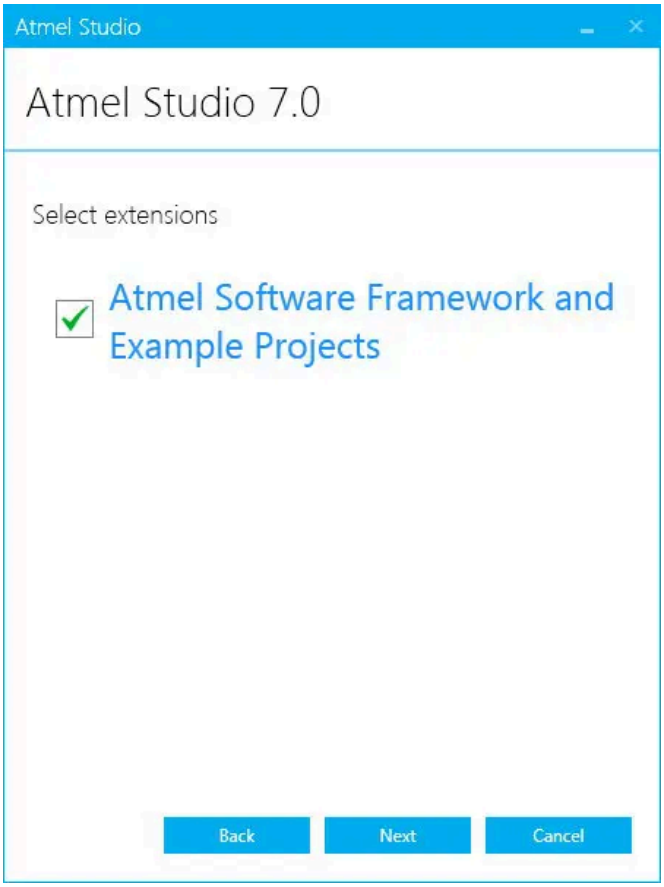
Microchip Studio 7.0 Installation

Step 1: Download the Microchip Studio installer from link & double click to run.

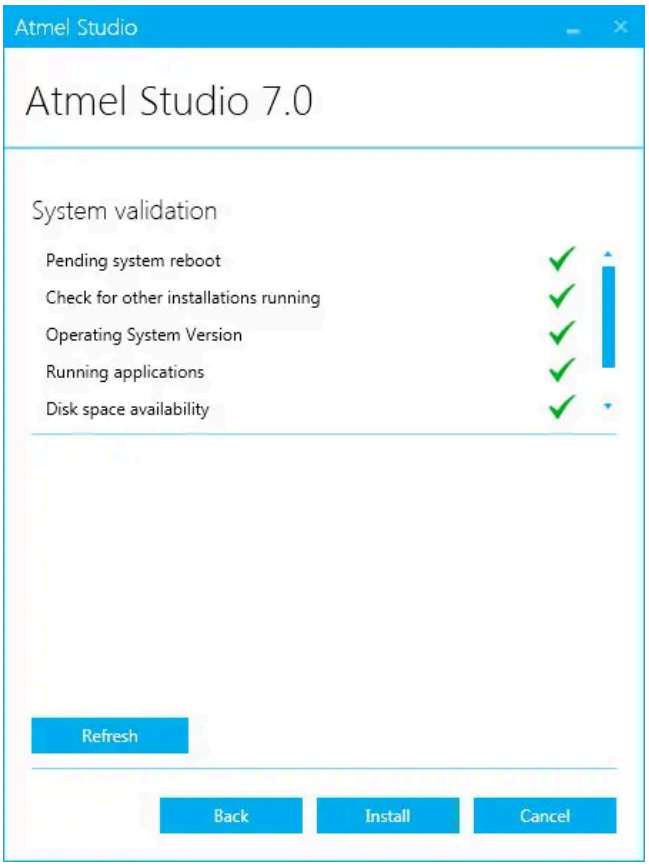
Step 2: This will launch the “License Terms” and prompt you to accept the “license terms and conditions”.

Step 3: Select the necessary architectures you would like to work with.

Step 4: Next, depending on the version, it may ask if you wish to install the Atmel Software Framework (ASF).



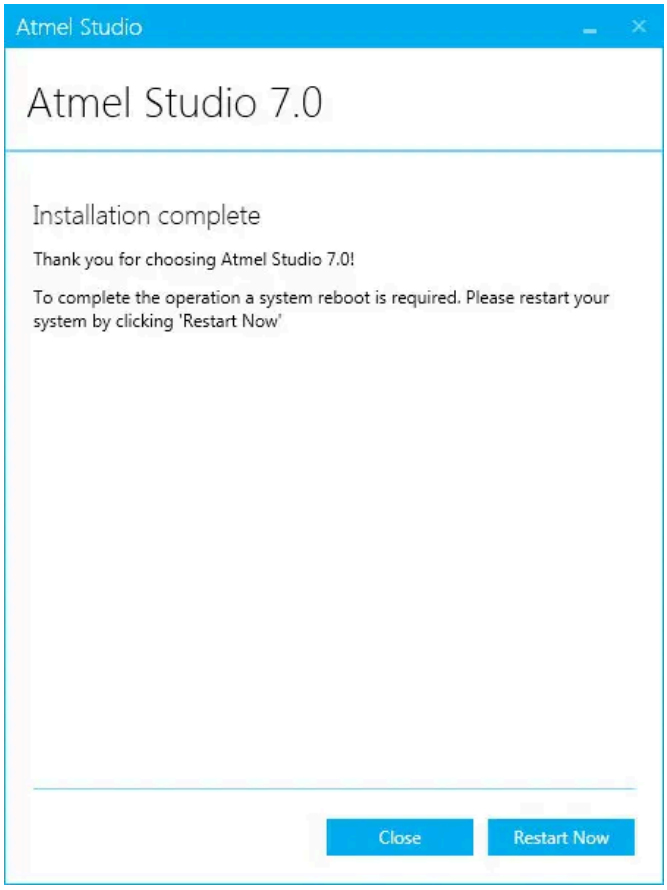
Step 5: Atmel Studio Installer will validate your system before starting the installation.



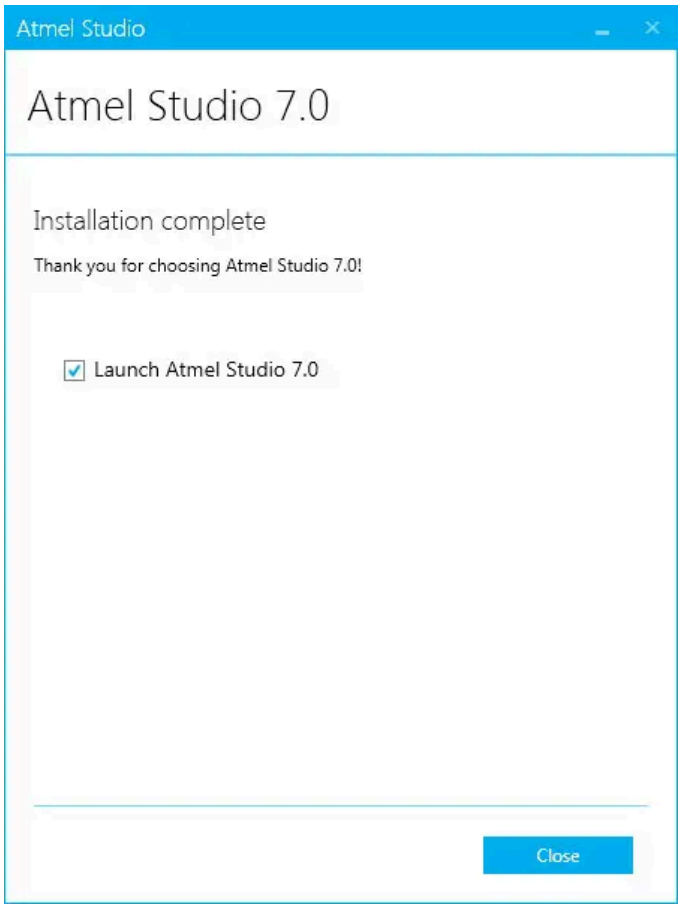
Step 6: The installation will take some time according to your system performance.

Step:7 Installation Complete

You are recommended to restart your System.



Step: 8 After restarting the system, the installation should finish with this window. You can choose to launch Atmel Studio.

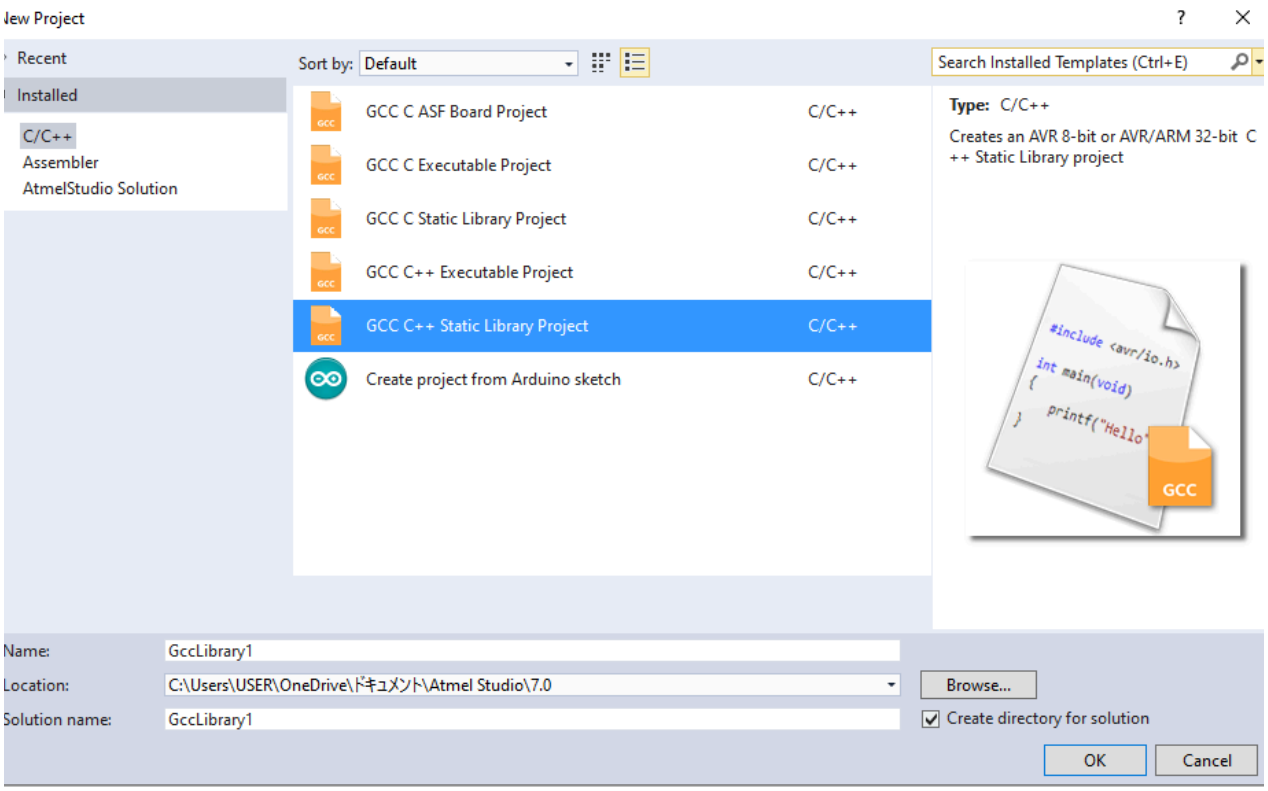


Step: 9 Click On the following Icon and these type of window will Open

Step 10: File → New → Project

Step 11**:** Select **GCC C Executable Project (C/C++)**

Step 12: Click **OK** to create the project



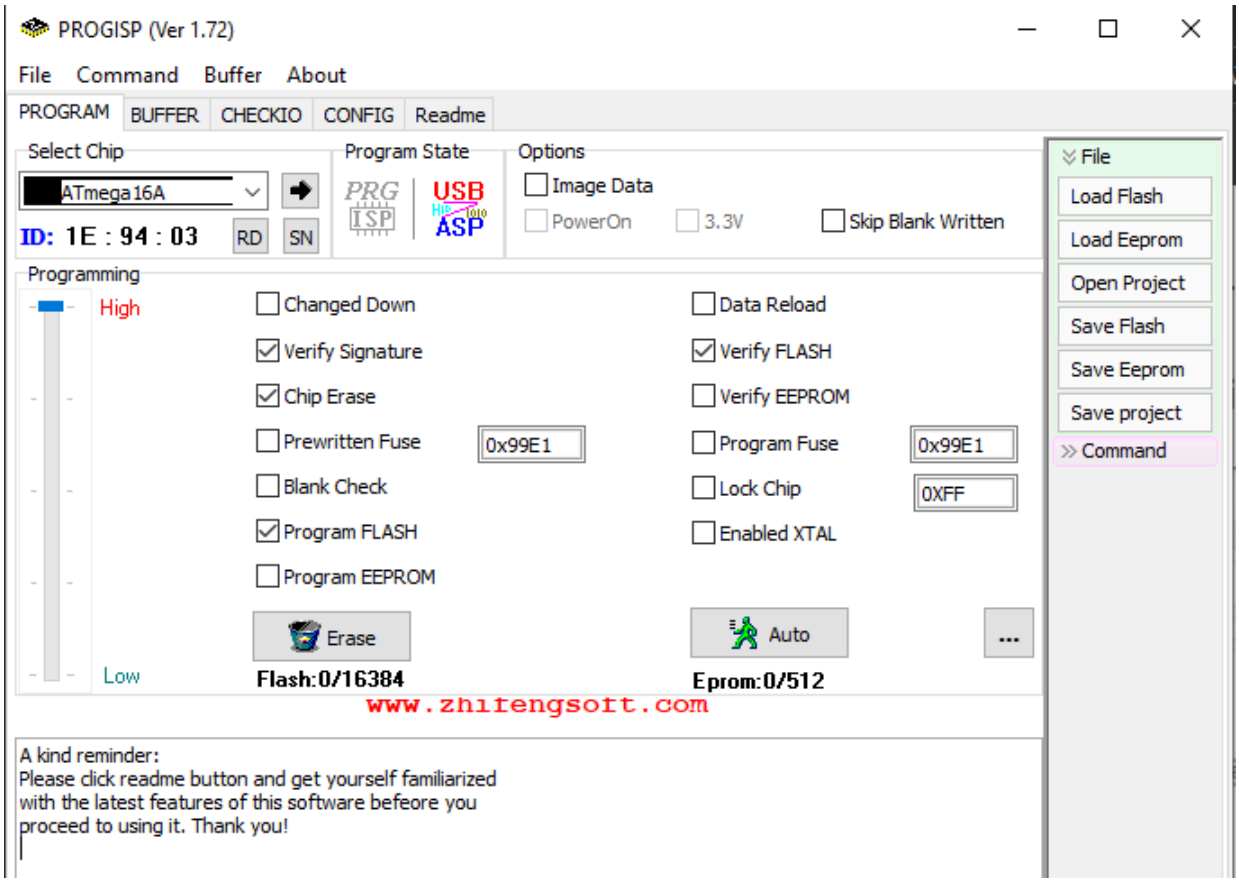
SSG
Embedded solutions

Step: 13 Select **ATmega32A** → Click **OK**

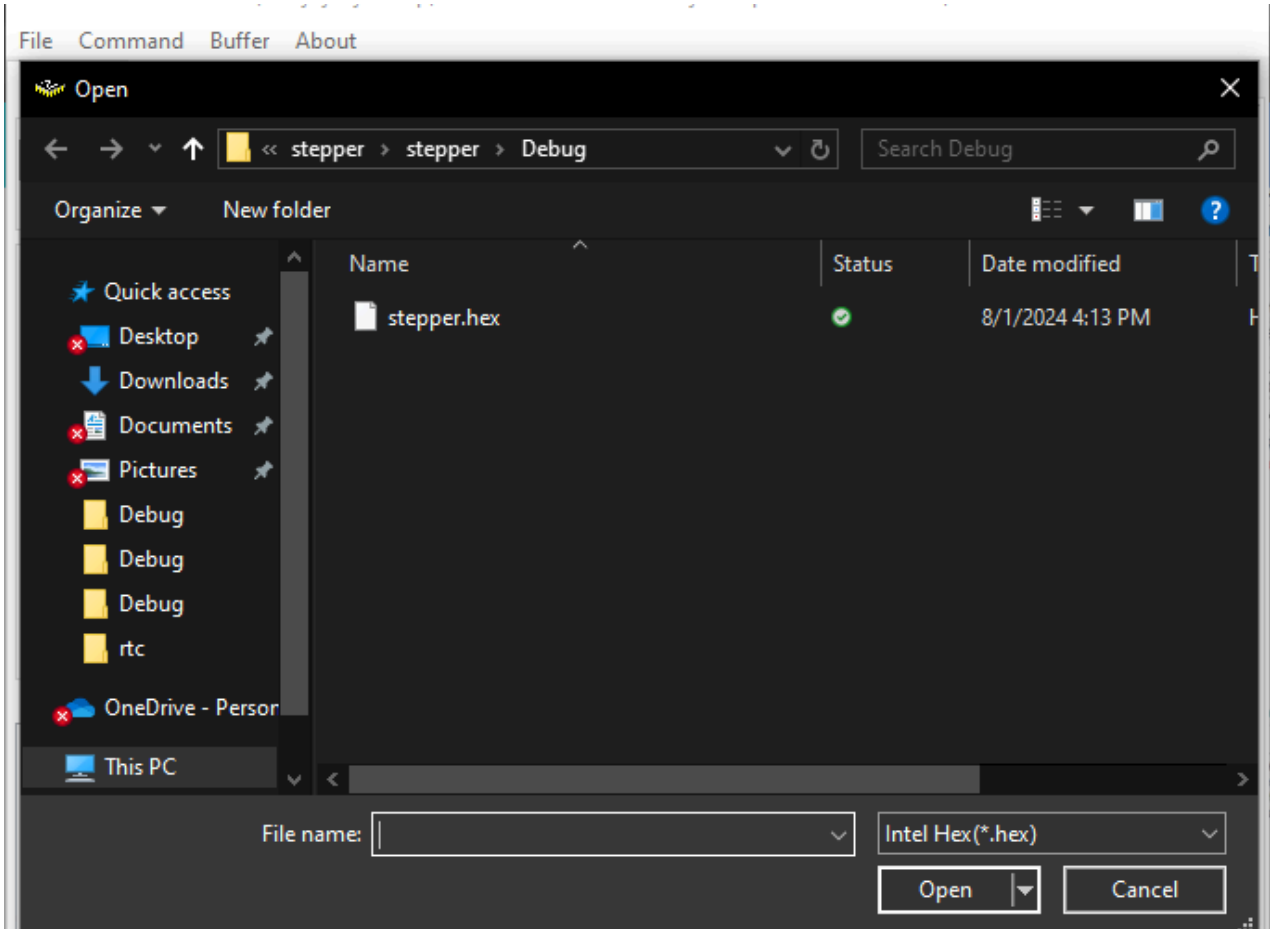
Step: 14 Write the program

Step: 15 Build solution → **Build Succeeded** → **.hex file created**

Step: 16 Open **ProgISP (577KB application)** → ProgISP window opens.



Step: Go to **Load Program → Load Flash → Select your .hex file → Open**



Document → Atmel Studio 7 → 7.0 → GCC Application1 → GCC Application → main

Examples:-

1 LED

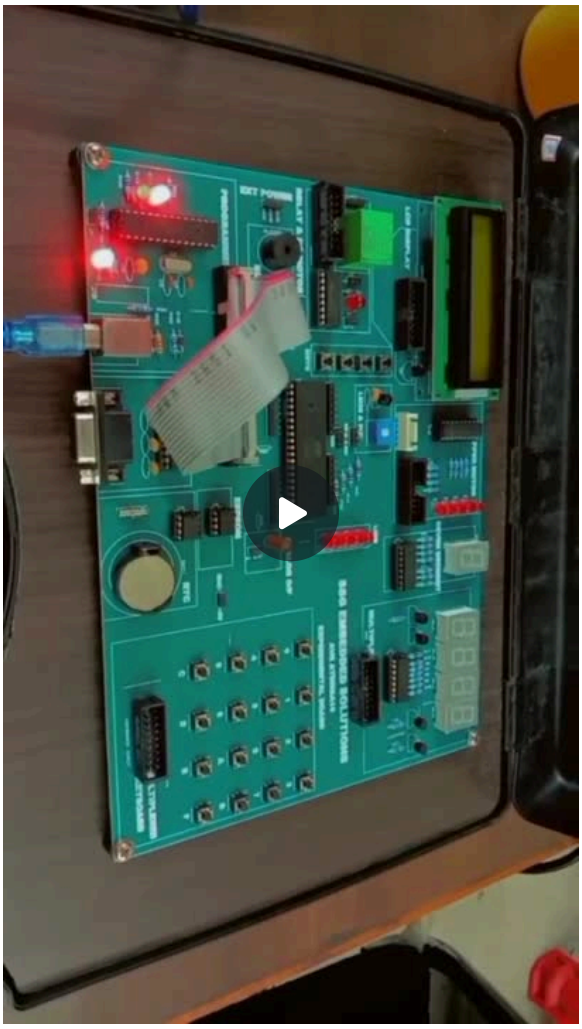
A Light Emitting Diode (LED) is one of the most widely used semiconductor devices that emits either visible light or invisible infrared light when it is forward biased. Remote controls, for example, generate invisible infrared light.

The LED converts electrical energy into light energy through a process called electroluminescence. When a voltage is applied across the LED, electrons and holes recombine in the semiconductor material, releasing energy in the form of photons (light).

LED BLINKING CODE

```
/*
* ATmega32_LED_Blinking.c


*/


#define F_CPU 1000000UL /* Define CPU frequency here 8MHz */
```

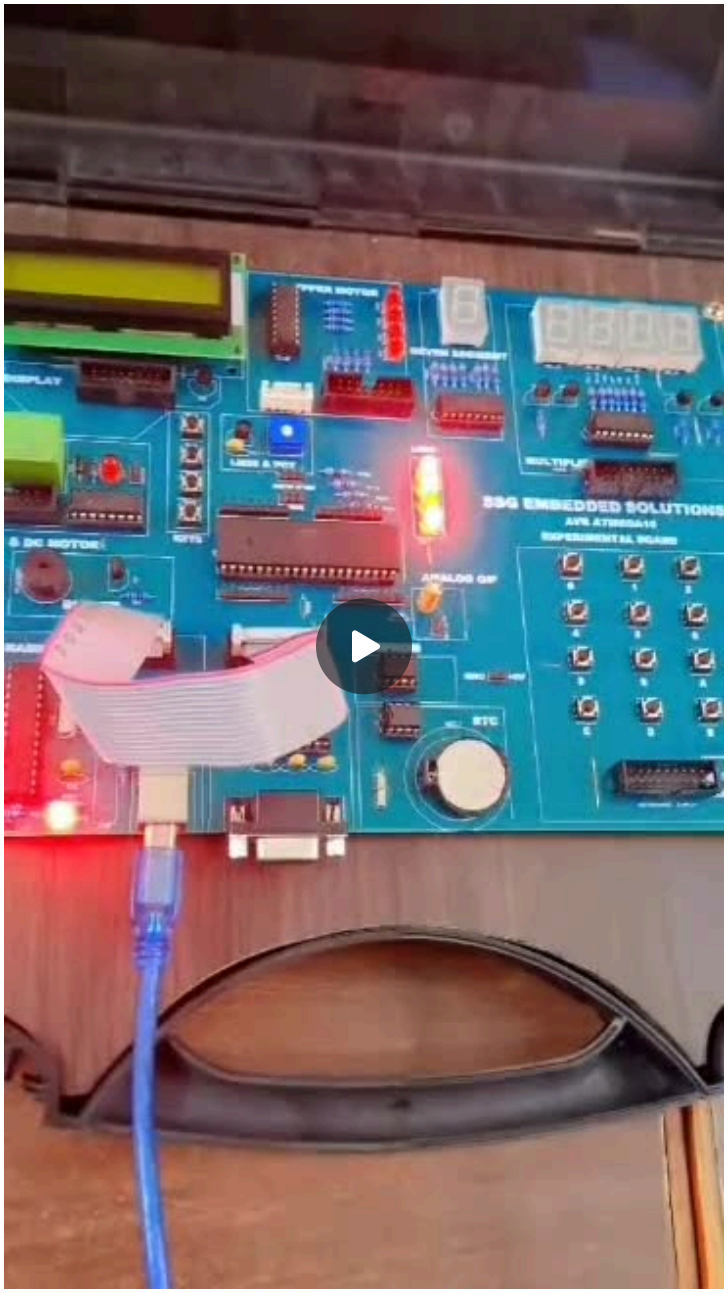
```
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{

DDRC = 0xF0; /* Make all pins of PORTD as output pins */
while (1)
{
/* Blink PORTD infinitely */
PORTC = 0x00;
_delay_ms(500); /* Delay of 500 milli second */
PORTC = 0xF0;
_delay_ms(500);
}
}
```



2 LED CHASER

```
/*
* ATmega32_LED_Blinking.c

*/

#define F_CPU 1000000UL /* Define CPU frequency here 8MHz */
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{

DDRC = 0b11110000;
_delay_ms(500);; /* Make all pins of PORTB as output pins */
while (1)
{
/* Blink PORTB infinitely */
PORTC = 0b10000000;// pin 0 of port c set HIGH
_delay_ms(100); /* Delay of 100 milli second */
PORTC = 0b01000000;// pin 1 of port c set HIGH
_delay_ms(100); /* Delay of 100 milli second */
_delay_ms(100); /* Delay of 100 milli second */
PORTC = 0b00100000;// pin 2 of port c set HIGH
_delay_ms(100); /* Delay of 100 milli second */
PORTC = 0b00010000;// pin 3 of port c set HIGH
```

```
_delay_ms(100); /* Delay of 100 milli second */
}
}
```



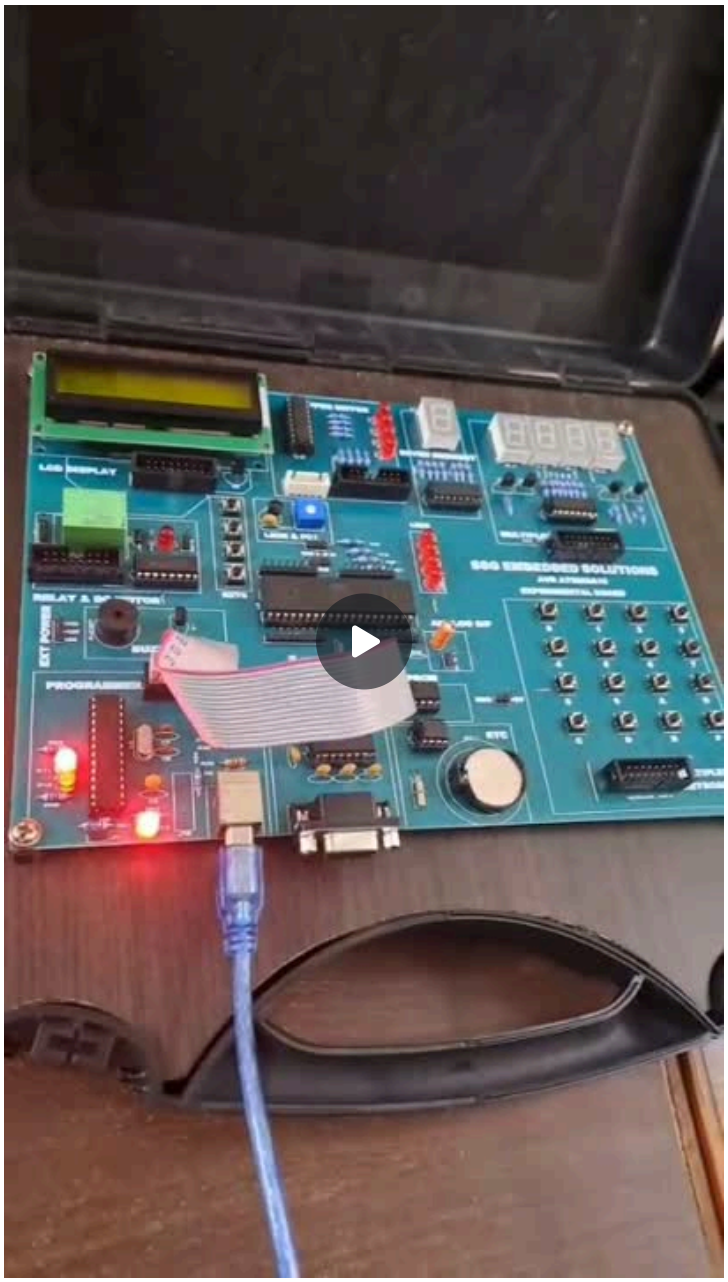
3 BUZZER

A buzzer is an electronic device that generates sound by converting electrical energy into sound energy. It typically consists of a piezoelectric crystal, which expands and contracts when an alternating current is applied to it, creating sound waves.

Buzzers are commonly used in a wide range of applications such as alarms, timers, and warning systems. They can also be used in electronic devices such as mobile phones, computers, and other electronic devices to generate different sounds and tones.

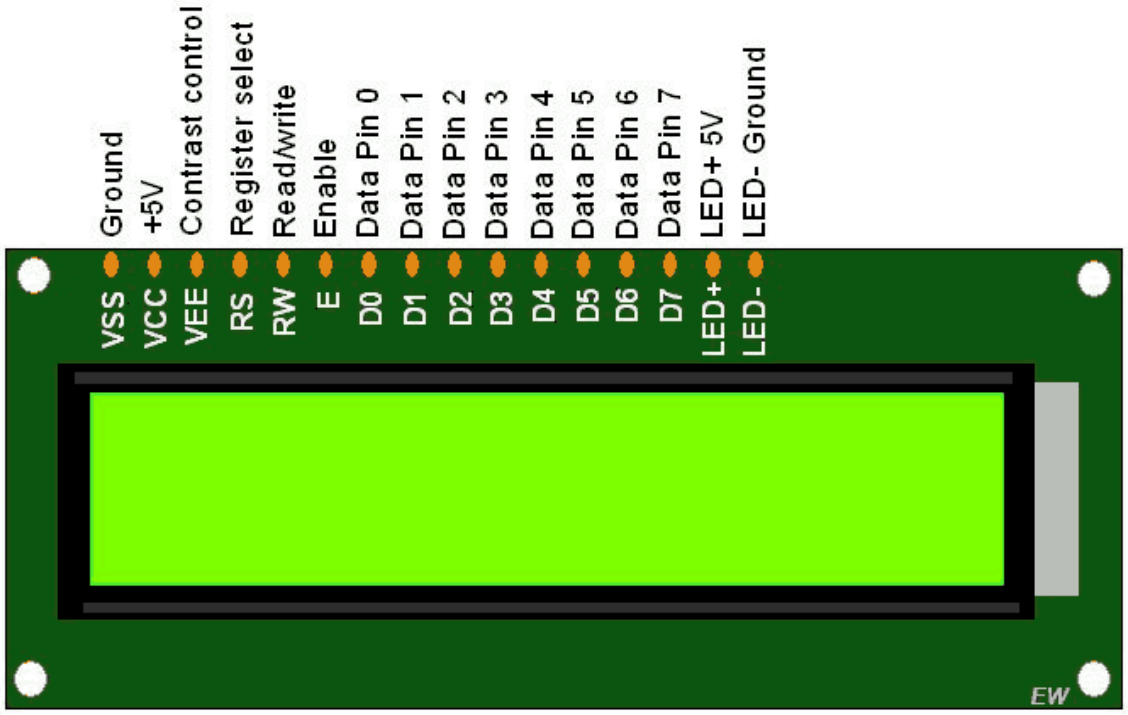
```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
int main()
{
PORTD=0x00;
DDRD=0xfd;
while(1)
{
```

```
PORTD=0x00;
_delay_ms(1000);
PORTD=0xfd;
_delay_ms(1000);
}
}
```



LCD (Liquid Crystal Display)

An LCD is a widely used electronic display module found in devices such as mobile phones, calculators, computers, and TVs. It is often preferred over multi-segment LEDs and seven-segment displays because it can show custom characters, symbols, and animations with ease. LCDs are also inexpensive, programmable, and versatile.



Features of LCD

- Operating voltage: 4.7V – 5.3V
- Two display rows, each supporting 16 characters
- Low current consumption (~1mA without backlight)
- Each character is displayed in a 5×8 pixel matrix
- Supports alphabets, numbers, and symbols
- Can operate in 4-bit or 8-bit mode

* LCD16x2 4-bit Interface with ATmega16

* Demonstration Code

* This program initializes a 16x2 LCD in 4-bit mode

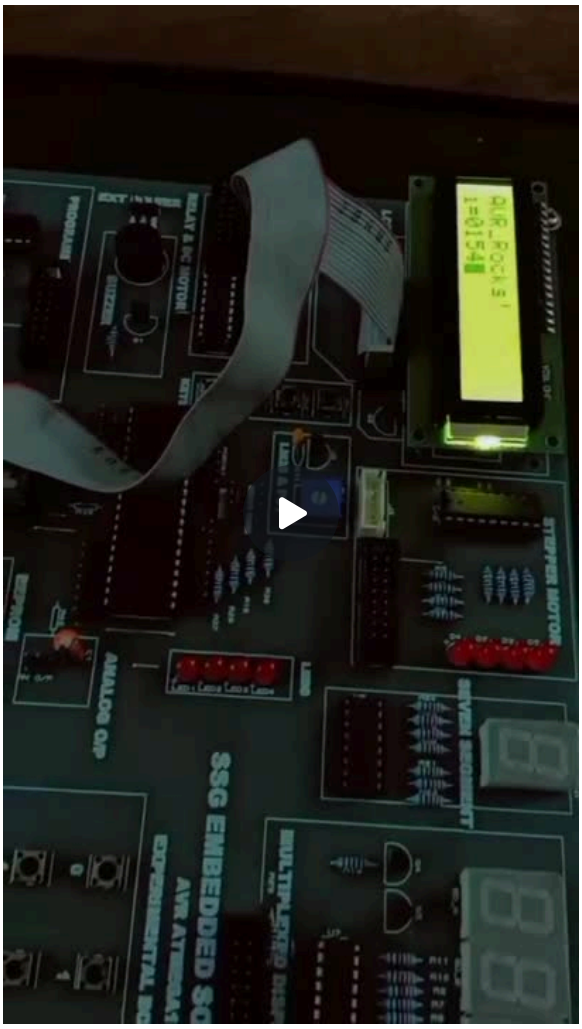
* and displays two strings on it.

```
#define F_CPU 1000000UL     // Define CPU Frequency (1 MHz here)

#include <avr/io.h>        // AVR standard I/O library
#include <util/delay.h>     // Standard delay library

// LCD Control Pin Configuration
#define LCD_Dir DDRB       // Define LCD port direction (Data Direction Register)
#define LCD_Port PORTB      // Define LCD port (Output Register)
#define RS PB0             // Register Select pin
#define EN PB2             // Enable pin
#define BL PB3             // Backlight pin
void LCD_Command(unsigned char cmnd);
void LCD_Char(unsigned char data);
void LCD_Init(void);
void LCD_String(char *str);
void LCD_String_xy(char row, char pos, char *str);
void LCD_Clear(void);
// Send Command to LCD
void LCD_Command(unsigned char cmnd)
{
   // Send higher nibble
   LCD_Port = (LCD_Port & 0x0F) | (cmnd & 0xF0);
   LCD_Port &= ~(1 << RS);      // RS = 0 → Command mode
   LCD_Port |= (1 << EN);       // Enable pulse
   _delay_us(1);
   LCD_Port &= ~(1 << EN);

   _delay_us(200);

   // Send lower nibble
```

```
    LCD_Port = (LCD_Port & 0x0F) | (cmnd << 4);
    LCD_Port |= (1 << EN);
    _delay_us(1);
    LCD_Port &= ~(1 << EN);

    _delay_ms(2);
}

// Send Character to LCD
void LCD_Char(unsigned char data)
{
    // Send higher nibble
    LCD_Port = (LCD_Port & 0x0F) | (data & 0xF0);
    LCD_Port |= (1 << RS);       // RS = 1 → Data mode
    LCD_Port |= (1 << EN);
    _delay_us(1);
    LCD_Port &= ~(1 << EN);

    _delay_us(200);

    // Send lower nibble
    LCD_Port = (LCD_Port & 0x0F) | (data << 4);
    LCD_Port |= (1 << EN);
    _delay_us(1);
    LCD_Port &= ~(1 << EN);

    _delay_ms(2);
}

// Initialize LCD in 4-bit mode
```

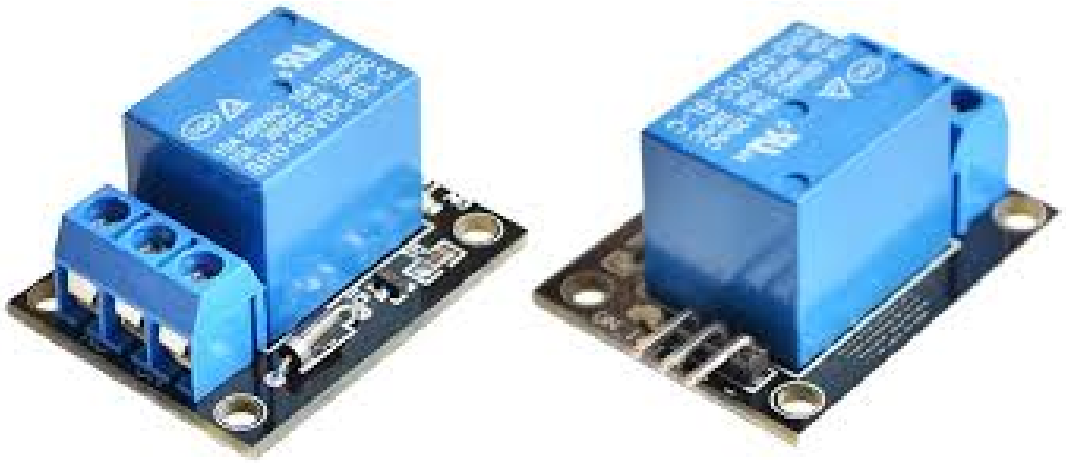
```
void LCD_Init(void)
{
   LCD_Dir = 0xFF;      // Set LCD port as output
   _delay_ms(20);       // LCD power ON delay

   LCD_Port |= (1 << BL);  // Turn ON backlight

   LCD_Command(0x33);    // Initialize LCD for 4-bit mode
   LCD_Command(0x32);
   LCD_Command(0x28);    // 2 line, 5x7 matrix, 4-bit mode
   LCD_Command(0x0C);    // Display ON, Cursor OFF
   LCD_Command(0x06);    // Auto-increment cursor (right shift)
   LCD_Command(0x01);    // Clear display
   _delay_ms(2);
   LCD_Command(0x80);    // Set cursor to 1st row, 0th column
}

// Send a String to LCD
void LCD_String(char *str)
{
   int i;
   for(i = 0; str[i] != 0; i++)
   {
      LCD_Char(str[i]);
   }
}

// Send a String to LCD with XY Position
void LCD_String_xy(char row, char pos, char *str)
{
```

```
    if (row == 0 && pos < 16)
        LCD_Command((pos & 0x0F) | 0x80); // Row 1
    else if (row == 1 && pos < 16)
        LCD_Command((pos & 0x0F) | 0xC0); // Row 2

    LCD_String(str); // Print string
}

// Clear LCD Display
void LCD_Clear()
{
    LCD_Command(0x01);   // Clear display
    _delay_ms(2);
    LCD_Command(0x80);   // Move cursor to 1st row, 0th column
}
int main(void)
{
    LCD_Init();                 // Initialize LCD

    LCD_String("SSG");          // Display on 1st row
    LCD_Command(0xC0);          // Move cursor to 2nd row
    LCD_String("Hello World");    // Display on 2nd row

    while(1); // Infinite loop
}
```


RELAY & DC MOTOR
BUZZER
STEPPER MOTOR
ANALOG I/P
SSG EMBEDDED SO

4 RELAY

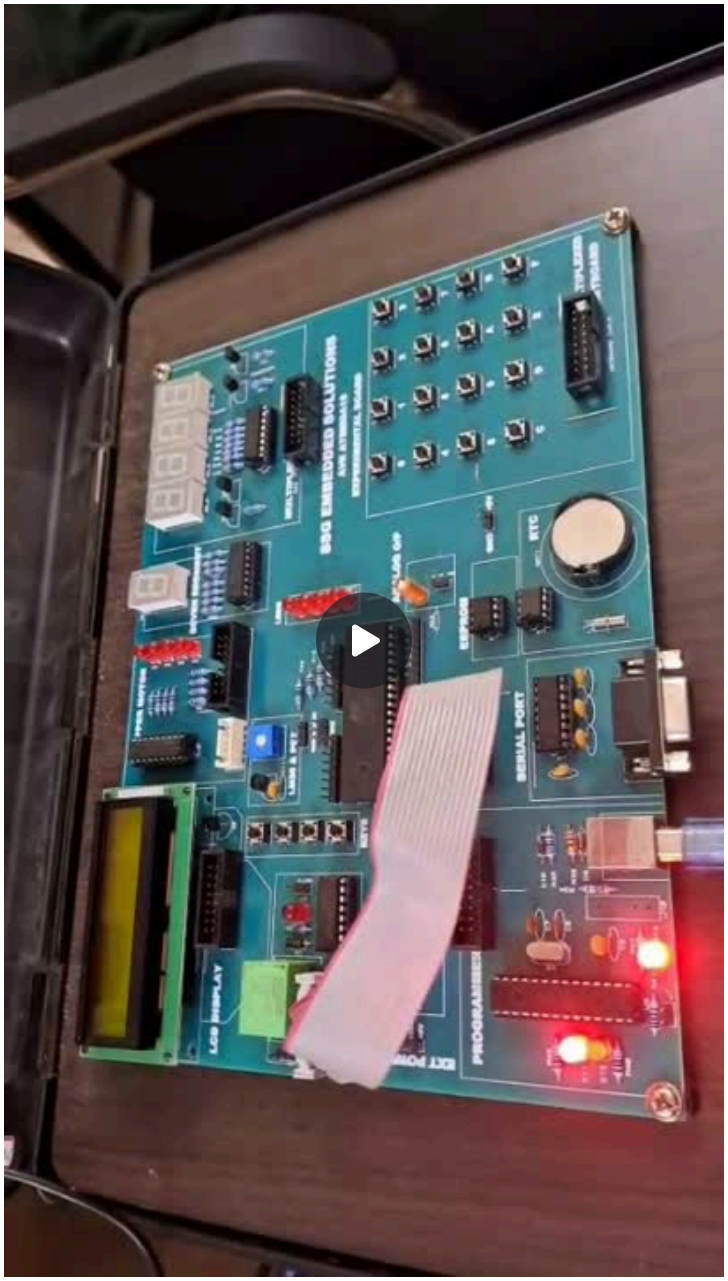
- A relay is an electromechanical or electronic switch used to open and close circuits. It works by energizing a coil that changes the state of its contacts.
- NO (Normally Open): Contact remains open when the relay is not energized. When the coil is energized, the contact closes, allowing current to flow.
- NC (Normally Closed): Contact remains closed when the relay is not energized. When the coil is energized, the contact opens stopping the current flow



Relays are widely used in control panels, automation systems, and electronic circuits. They allow a low-voltage signal to control higher voltages and currents, making them useful for switching heavy loads safely.

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>
int main()
{
PORTB=0x0f;
```

```
DDRB=0x0b;
while(1)
{
PORTB=0x00;
_delay_ms(1000);
PORTB=0x0b;
_delay_ms(1000);
}
}
```



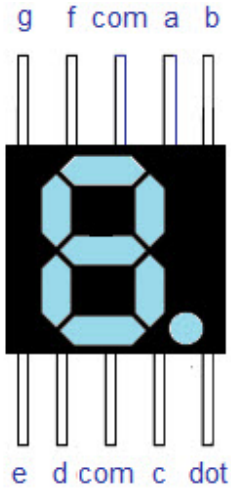
5 SEVEN SEGMENT DISPLAY

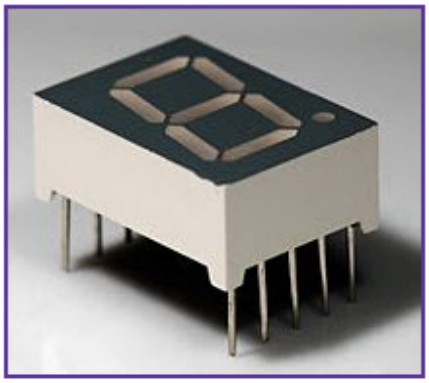
A Seven Segment Display (SSD) is a simple electronic display device used to show numerical digits (0–9) and a few characters like A, B, C, E, F, H.

It is made up of 8 LEDs — seven LEDs form the segments labeled a–g, and one LED is used for the decimal point (dp). By turning ON or OFF different segments, digits and characters can be displayed.

Each SSD has 10 pins:

8 pins for the segments (a–g + dp) and 2 pins as common terminals (COM). The COM pins are internally shorted, so you only need to use one of them.



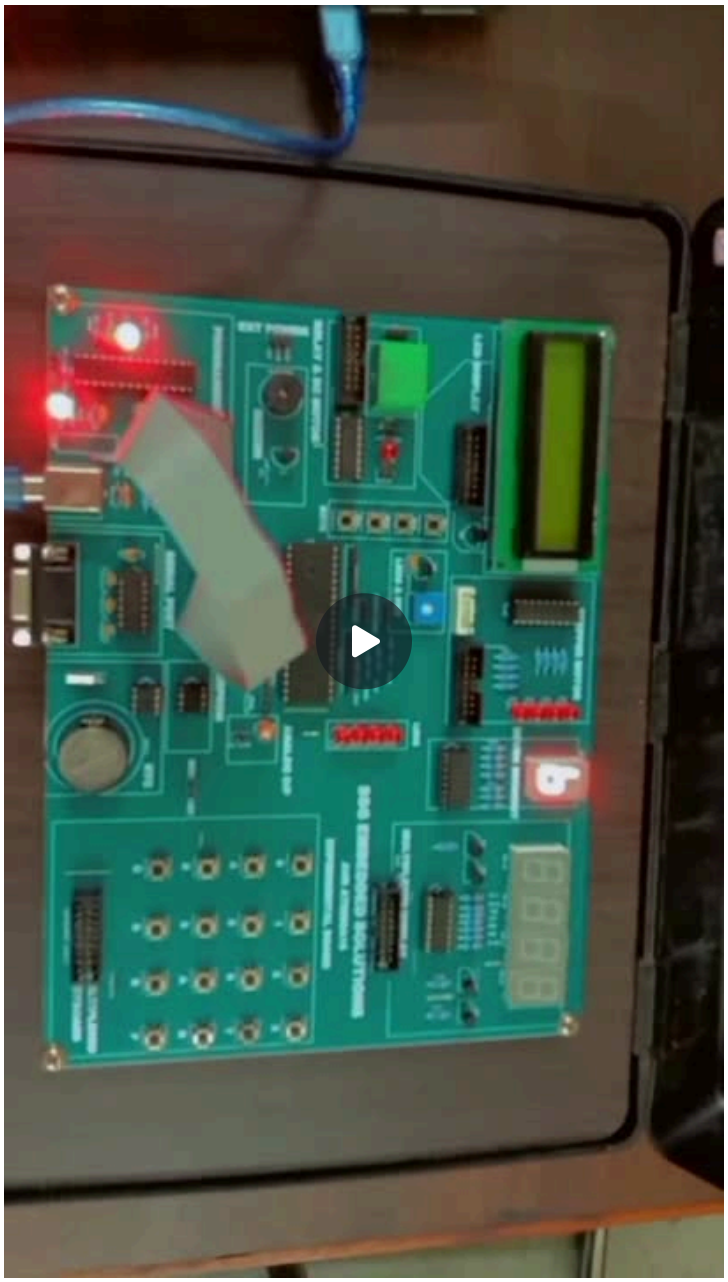


Types of Seven Segment Displays

- Common Anode (CA): All anodes are connected together at the COM pin. Segments light up when their cathode is connected to GND.
- Common Cathode (CC): All cathodes are connected together at the COM pin. Segments light up when their anode is connected to VCC.

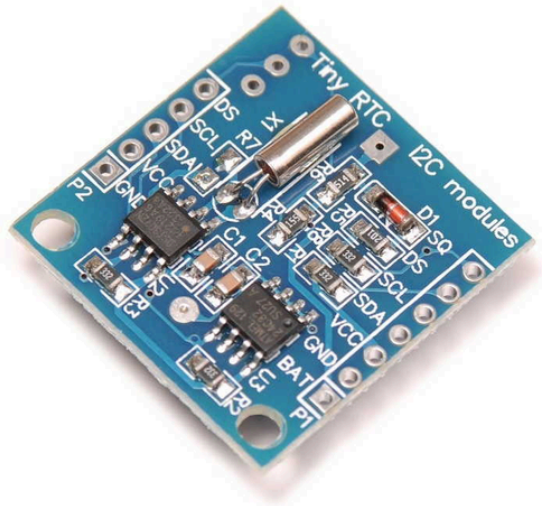
```
#define F_CPU 1000000UL

#include <avr/io.h>

#include <util/delay.h>

#define LED_direction DDRC

#define LED_PORT PORTC

int main(void)

{
```

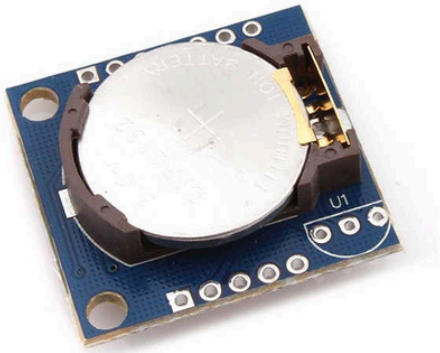
```
LED_direction |= 0x0f;
LED_PORT = 0x0f;
}
/* define LED Direction */
/* define LED PORT */
/* define LED port direction is output */
char array[]={0,1,2,3,4,5,6,7,8,9};
/* write BCD value for CA display from 0 to 9 */
while(1)
{
for(int i=0;i<10;i++)
{
LED_PORT = array[i];/* write data on to the LED port */
_delay_ms(1000); /* wait for 1 second */
}
}
```



5 RTC (REAL TIME CLOCK)

- Real Time Clock (RTC) is used to track the current time and date. It is generally used in computers, laptops, mobiles, embedded system applications devices, etc.
- In many embedded systems, we need to put time stamps while logging data i.e. sensor values, GPS coordinates, etc. For getting timestamps, we need to use RTC (Real Time Clock).
- Some microcontrollers like LPC2148, LPC1768, etc., have on-chip RTC. But in other microcontrollers like PIC, and ATmega16/32, do not have on-chip RTC. So, we should use an external RTC chip.





Specification of DS1307

- I2C Interface: Standard I2C interface, with 7-bit addressing
- SRAM Memory: 56 bytes of battery-backed SRAM

```
/*
* avr_lcd_1.c
*
* Created: 10-Oct-23 9:25:00 AM
* Author : user
*/
```

```
#include <avr/io.h>
#include <util/delay.h>
#include <lcd.h>
#include "clock.h"
void Wait()
{
uint8_t i;
for(i=0;i<20;i++)
_delay_loop_2(0);
}
void write_new_time(void)
{
LCDClear();
LCDWriteString("press1toWrtNew");
LCDWriteStringXY(0,1,"newtime, 2toExit");
while(1)
{
int i;
i = PINA;
i = i & 0x0f;
if (i == 0x07)
{
LCDClear();
LCDWriteString("writing_newTime");
PORTC=0xef;
uint8_t hours = 4, minutes = 52, seconds = 0, meridian = 1;//
change these values if you want to reset the time
SetHour(hours);
```

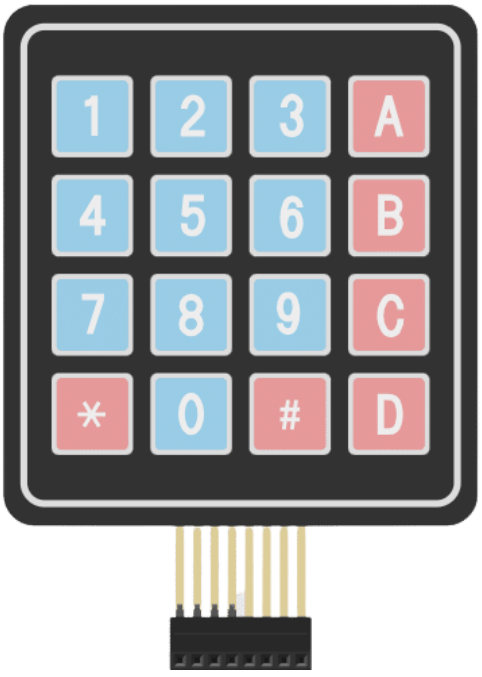
```
SetMinute(minutes);
SetSecond(seconds); SetAmPm(meridian);
_delay_ms(3000);
PORTC=0xff;
return;
}
if (i == 0x0b)
{
PORTC=0xdf;
LCDClear();
LCDWriteString("SwitchingToRead");
LCDWriteStringXY(0,1,"mode ... ");
_delay_ms(3000);
PORTC=0xff;
return ;
//goto back2main;
}
_delay_ms(100);
}
}
int main()
{
DDRA=0x00;
PORTA =0xff;
DDRC=0xff;
PORTC=0xff;
//Wait Util Other device startup
_delay_ms(500);
```

```
//Initialize the LCD Module
LCDInit(LS_NONE);
DDRB|=(1<<PB3);
PORTB|=(1<<PB3);
LCDWriteString("LCD_initialized!");
_delay_ms(3000);
//ClockInit();
//LCDWriteStringXY(0,1,"clock_init");
//_delay_ms(1000);
//Initialize the Clock Module
if(ClockInit()==0)
{
//If we fail to initialize then warn user
LCDClear();
LCDWriteString("Error !"); LCDWriteStringXY(0,1,"DS1307 Not Found");
while(1); //Halt
}
write_new_time();
char Time[12]; //hh:mm:ss AM/PM
//Now Read and display time
while(1)
{
GetTimeString(Time);
LCDClear();
LCDWriteString("AVR_Rocks!!!");
LCDWriteStringXY(3,1,Time);
LCDGotoXY(17,1);
_delay_ms(500);
```

```
}
}
```

6 Keypad

Normally, we use a single I/O pin of a microcontroller to read a digital signal, such as the input from a switch. However, in applications that require multiple keys (like 9, 12, or 16 keys), connecting each key individually would consume a large number of I/O pins. For example, connecting 16 keys directly would occupy 16 I/O pins of the microcontroller. These I/O pins are not only used for reading digital inputs but are also required for other important peripheral functions such as ADC, I²C, and SPI communication. If too many pins are dedicated to switches, we lose the flexibility of using them for these peripheral connections.



// This code demonstrates the 4x4 keypad array interfaced with ATmega

at PORTB,

// Since, the output response of the key_presses are being displayed on the

hardwired 7 segment display which is using BCD decoder,

// it can only display the output in proper form upto 0-9 digits, the rest of

the key_responses are displayed in terms of symbols as follows:

/*

DCBA BCD_output_responses on 7 seg display

0000 0

0001 1

0010 2

0011 3

0100 4

0101 5

0110 6

0111 7

1000 8

1001 9

_

1010 |_ (A)

_

1011 _| (B)

1100 |_| (C)

_

1101 |_

_ (D)

1110 |_ (E)

|_

1111 (F)

*/

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <util/delay.h>

uint8_t GetKeyPressed()
{
uint8_t r,c;
PORTB|= 0XF0;
for(c=0;c<4;c++)
{
DDRB&=~(0XFF);
DDRB|=(0X01 << c);
for(r=0;r<4;r++)
{
if(!(PINB & (0X10 << r)))
{
return (r+(c*4));
}
}
}
return 0XFF;//Indicate No key pressed
}
int main(void)
{
```

```
DDRC|=((1<<PC0)|(1<<PC1)|(1<<PC2)|(1<<PC3));
PORTC&=~((1<<PC0)|(1<<PC1)|(1<<PC2)|(1<<PC3));
char array_1[]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
uint8_t key;
while(1)
{
key=GetKeyPressed(); //Get the keycode of pressed key
PORTC=array_1[key];
}
}
```