



SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur – 603 203.



**DEPARTMENT OF
ELECTRONICS AND COMMUNICATION ENGINEERING**

**EC3564- EMBEDDED SYSTEMS AND IOT DESIGN
THEORY CUM LABORATORY**

LAB MANUAL

Regulation-2023

III YEAR / V SEM ECE (2026- 2027 ODD SEMESTER)

Prepared by

Dr. G. Udhayakumar, Professor/ ECE

Dr. S.R.Preethi, Associate Professor/ECE

Dr. N.Jothy , Assistant Professor(Sr.G)/ ECE

Mr.K.R.Ganesh., Assistant Professor(Sr.G) ECE

Dr. R. Dhananjeyan, Assistant Professor/ ECE

NAME : _____

REGISTER NO : _____

YEAR/SEMESTER : _____

ABOUT OBSERVATION NOTES & PREPARATION OF RECORD

- ❖ This Observation contains the basic diagrams of the circuits enlisted in the syllabus of the **EC3564 - EMBEDDED SYSTEMS AND IOT DESIGN** course, along with the design of various components of the circuit and controller.
- ❖ The experiment's aim is also given at the beginning of each experiment. Once the student can design the circuit as per the circuit diagram, they are supposed to go through the instructions carefully and do the experiments step by step.
- ❖ They should note down the readings (observations) and tabulate them as specified.
- ❖ It is also expected that the students prepare the theory relevant to the experiment referring to prescribed reference books/journals in advance, and carry out the experiment after understanding thoroughly the concept and procedure of the experiment.
- ❖ They should get their observations verified and signed by the staff within two days and prepare & submit the record of the experiment when they come to the laboratory in the subsequent week.
- ❖ The record should contain experiment No., Date, Aim, Apparatus required, Theory, Procedure, and result on one side (i.e., Right-hand side, where rulings are provided) and Circuit diagram, Design, Model Graphs, Tabulations, and Calculations on the other side (i.e., Left-hand side, where no rulings are provided)
- ❖ All the diagrams and table lines should be drawn in pencil
- ❖ The students are directed to discuss & clarify their doubts with the staff members as and when required. They are also directed to follow strictly the guidelines specified.

EC3564 - EMBEDDED SYSTEMS AND IOT DESIGN**Theory cum laboratory****LABORATORY SYLLABUS****LIST OF PRACTICAL EXPERIMENTS:**

- Programming Arithmetic and Logical Operations in 8051.
- Implementation of IoT using Python Programming.
- Study of ARM processor.
- Implementation of IoT using BLYNK - –Installation and Activation - Blinking an LED.
- Monitoring of temperature using sensor and Softcore Processors.
- Study of microcontrollers used in IoT.
- Logging of data in cloud using IoT.
- WAP for LED blink.
- Design of IoT System.

LIST OF EXPERIMENTS

S.No.	NAME OF THE EXPERIMENT	PAGE NO.	REMARKS
1	PROGRAMMING ARITHMETIC AND LOGICAL OPERATIONS IN 8051	16	
2.A	IMPLEMENTATION OF ULTRASONIC SENSOR USING ARDUINO	28	
2.B	IMPLEMENTATION OF TEMPERATURE SENSOR USING ARDUINO	31	
3.A	STUDY OF ARM PROCESSORS	35	
3.B	PYTHON PROGRAMMING IN RASPBERRY PI	47	
3.C	INTERFACING SENSORS WITH RASPBERRY PI	53	
4	IMPLEMENTATION OF IOT USING BLYNK	58	
5	MONITORING OF TEMPERATURE USING SENSOR AND SOFTCORE PROCESSOR	65	
6	STUDY OF MICROCONTROLLERS USED IN IOT	70	
7	LOGGING OF DATA IN CLOUD USING IOT	74	
8	WAP(Wireless Application Protocol) FOR LED BLINK	79	
9.A	DESIGN OF IOT SYSTEM- Automatic Door System	83	
9.B	DESIGN OF IOT SYSTEM – GAS LEAKAGE DETECTION SYSTEM	86	
Additional Experiment beyond Syllabus			
10	Edge AI Based Object Detection Using ESP32-CAM and IoT Dashboard	91	

INTRODUCTION TO KEIL μ VISION 5

INTRODUCTION TO KEIL μ VISION 5 SOFTWARE

The μ Vision5 IDE is a Windows-based software development platform that combines a robust editor, project manager, and make facility. μ Vision5 integrates all tools, including the C compiler, macro assembler, linker/locator, and HEX file generator. μ Vision4 helps expedite the development process of your embedded applications by providing the following:

- ✓ Full-featured source code editor
- ✓ Device database for configuring the development tool set
- ✓ Project manager for creating and maintaining your projects
- ✓ Integrated make facility for assembling, compiling, and linking your embedded applications
- ✓ Dialogs for all development tool settings
- ✓ True integrated source-level Debugger with high-speed CPU and peripheral simulator
- ✓ Advanced GDI interface for software debugging in the target hardware and for connection to Keil ULINK
- ✓ Flash programming utility for downloading the application program into Flash ROM
- ✓ Links to development tools manuals, device datasheets & users' guides

The Keil μ Vision5 IDE offers numerous features and advantages that help you quickly and successfully develop embedded applications. It is easy to use and guaranteed to help you achieve your design goals.

The installation steps for Keil software are given below:

1. Double click on Keil μ vision4.exe file.
2. Then click on **Next**.
3. Tick the check box towards to license agreements and click **Next**.
4. Select Destination folder and click **Next**.
5. Fill the necessary text boxes and click **Next**.
6. Finally click on **Finish**.

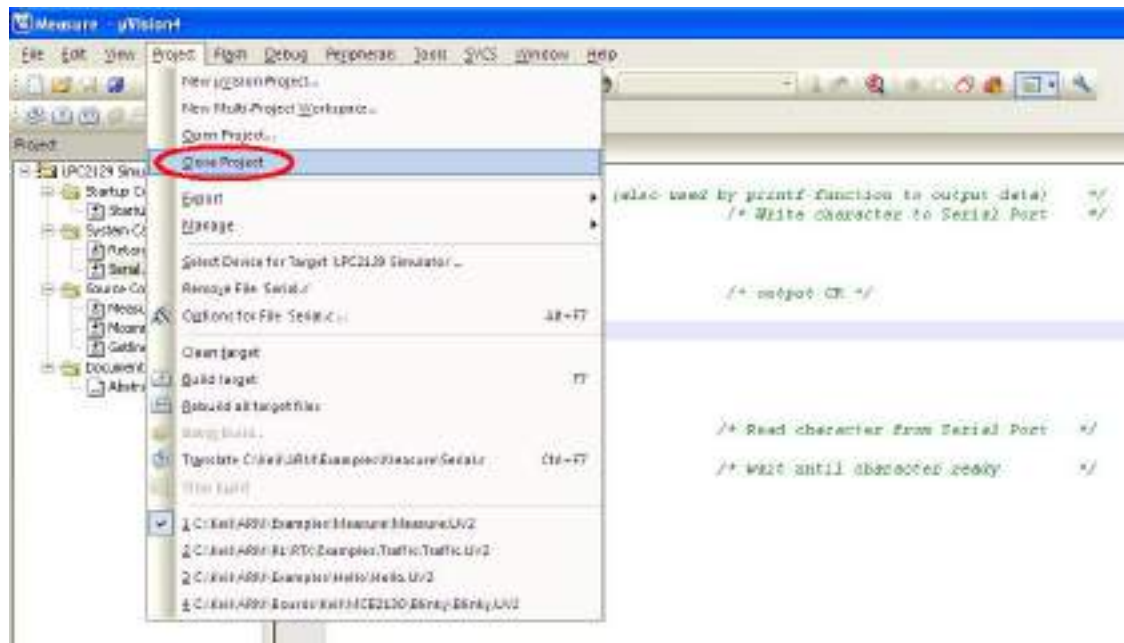
Software Flow

First open the icon keil μ vision4 and the follow the steps are given below. The menu bar provides you with menus for editor operations, project maintenance, development tool option settings, program debugging, external tool control, window selection and manipulation, and on-line help. The toolbar buttons allow you to rapidly execute μ Vision4 commands. A Status Bar provides editor and debugger information. The various toolbars and the status bar can be enabled or disabled from the View Menu commands.

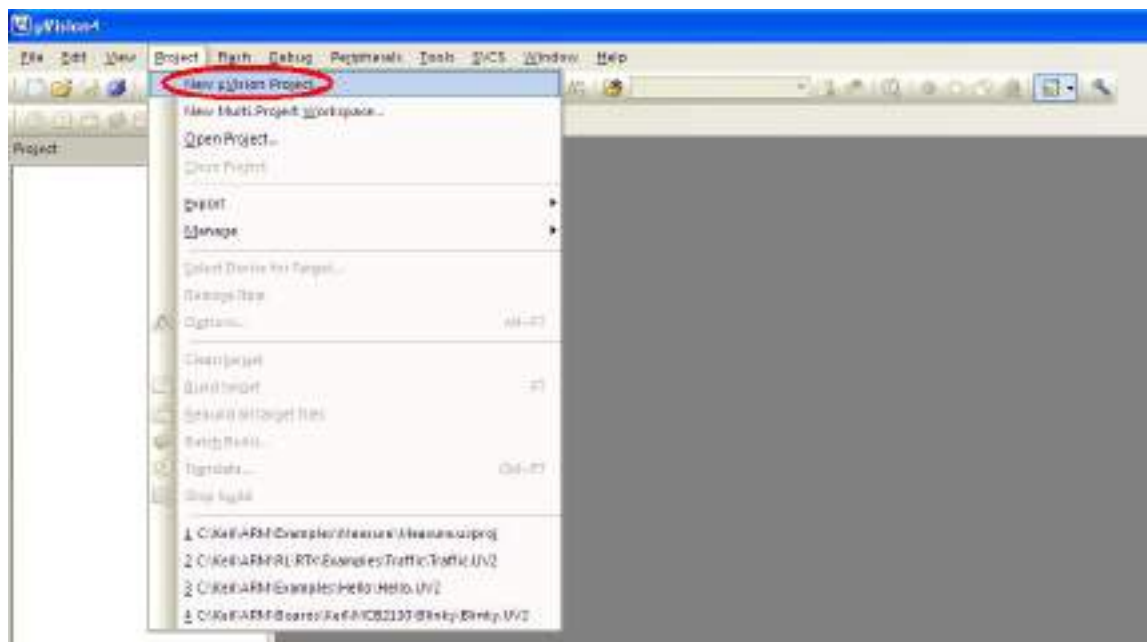
Creating a New Project

The mentioned procedures will explain the steps required to set up a simple application and to generate a HEX output.

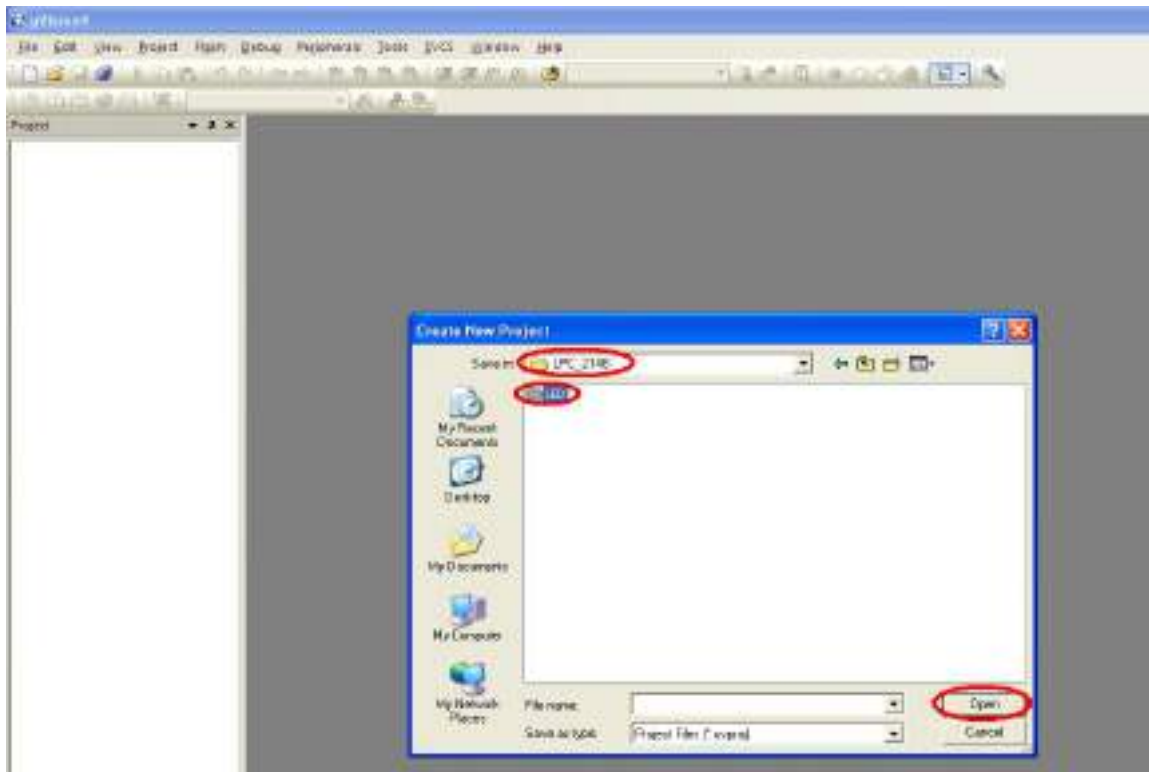
STEP 1: Go to “**Project**” and close the current project “**Close Project**”.



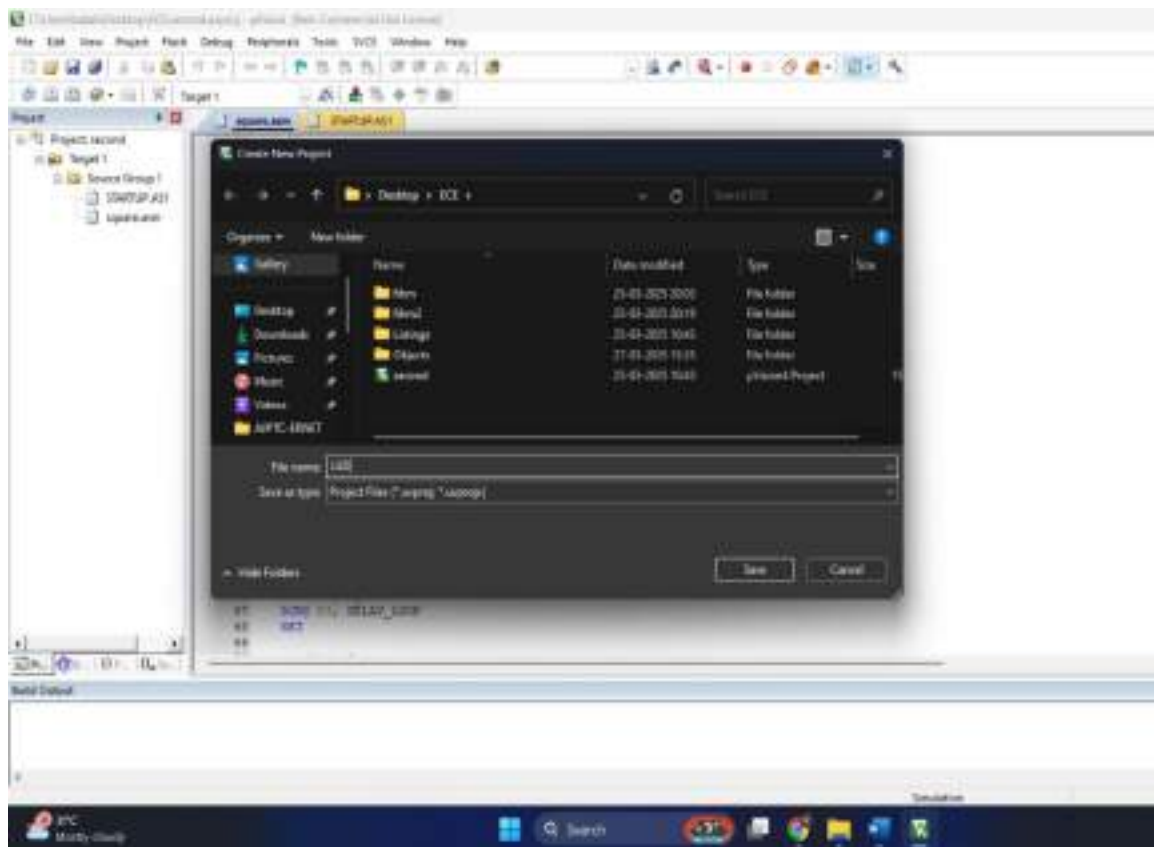
STEP 2: Go to the “**Project**” and click on “**New uVision Project**”



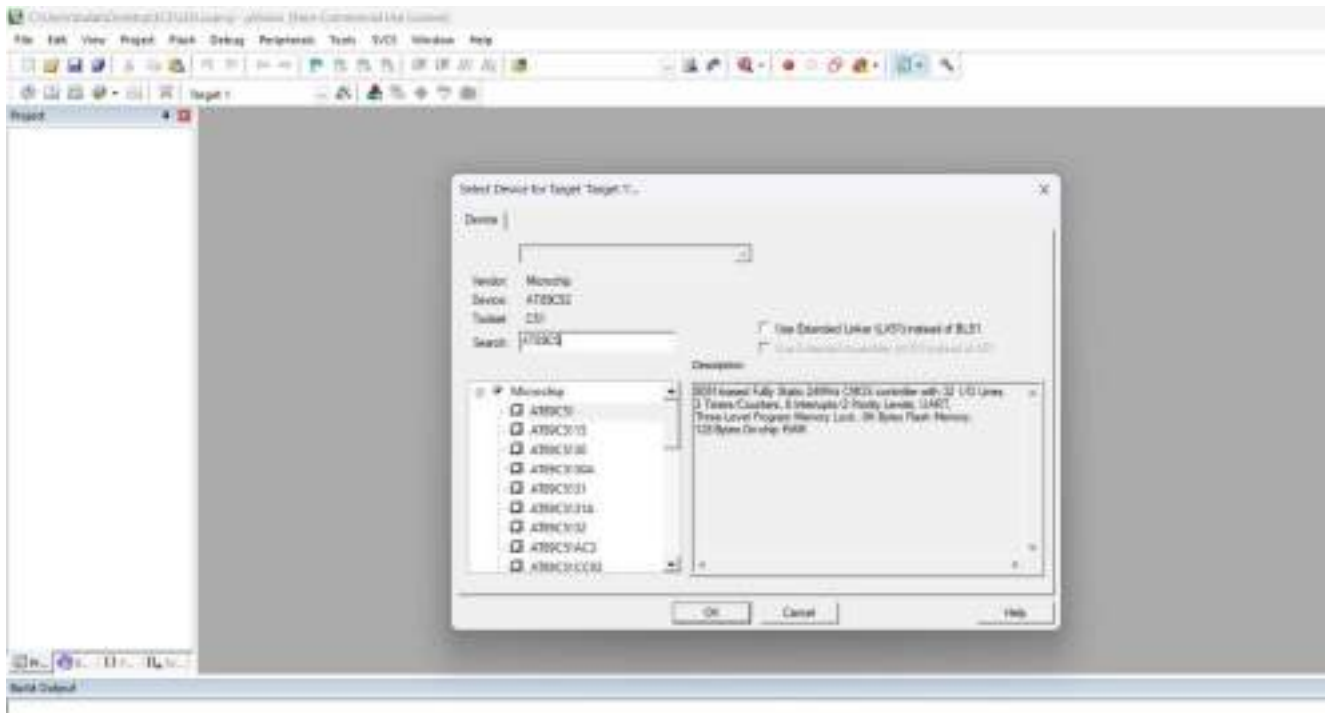
STEP 3: A small window will pop up with the name “**Create New Project**” and can be created and select destination path.



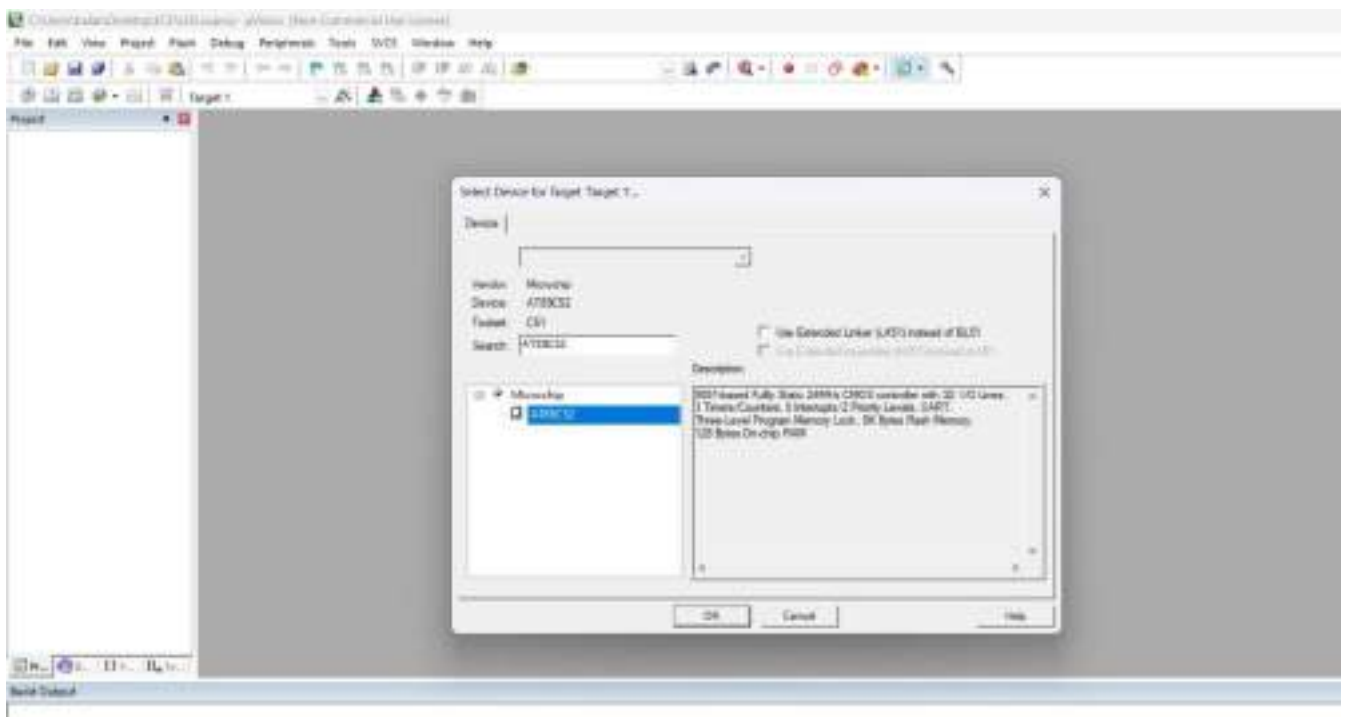
STEP 4: Create a folder and give a proper name that can be related to the Project.



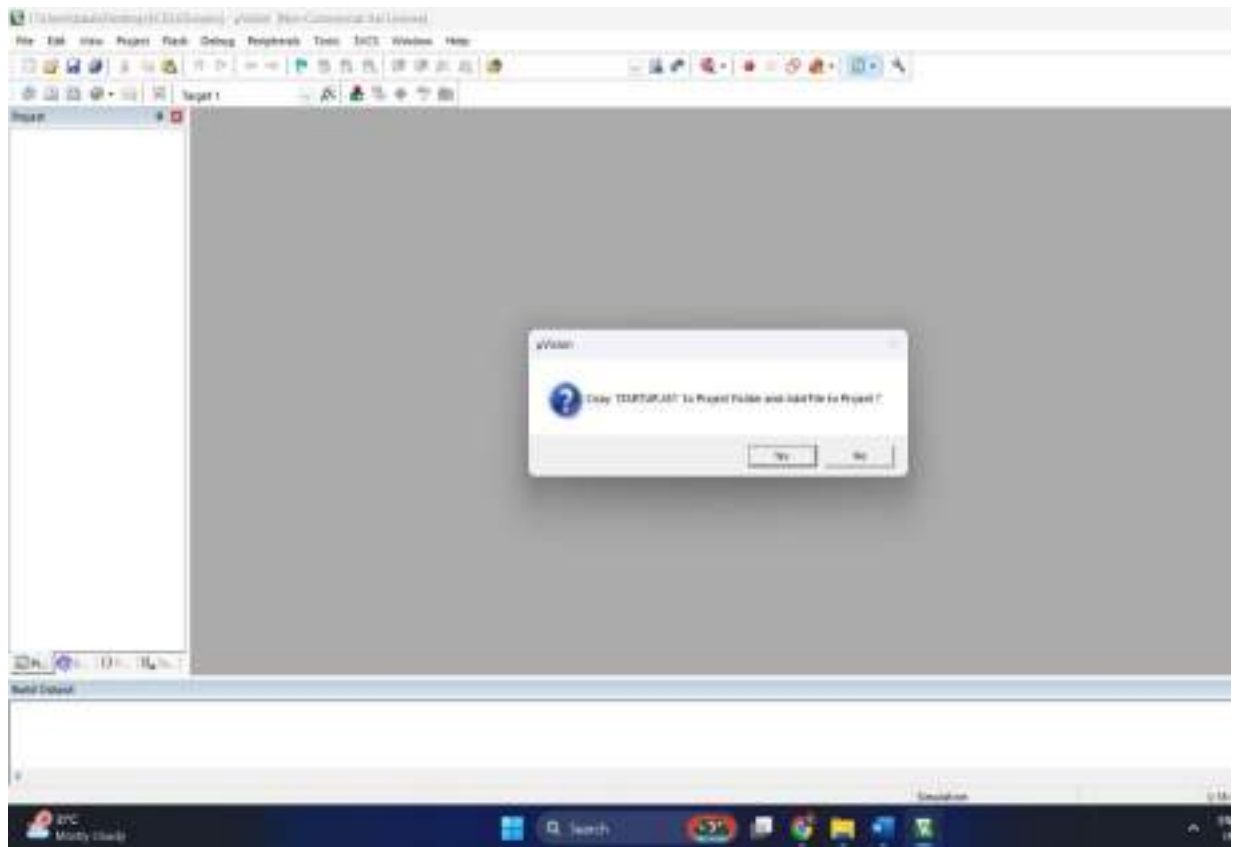
STEP 5: A small window will pop up with the name “Select Device for Target ‘Target 1’”, and select the data base Microchip.



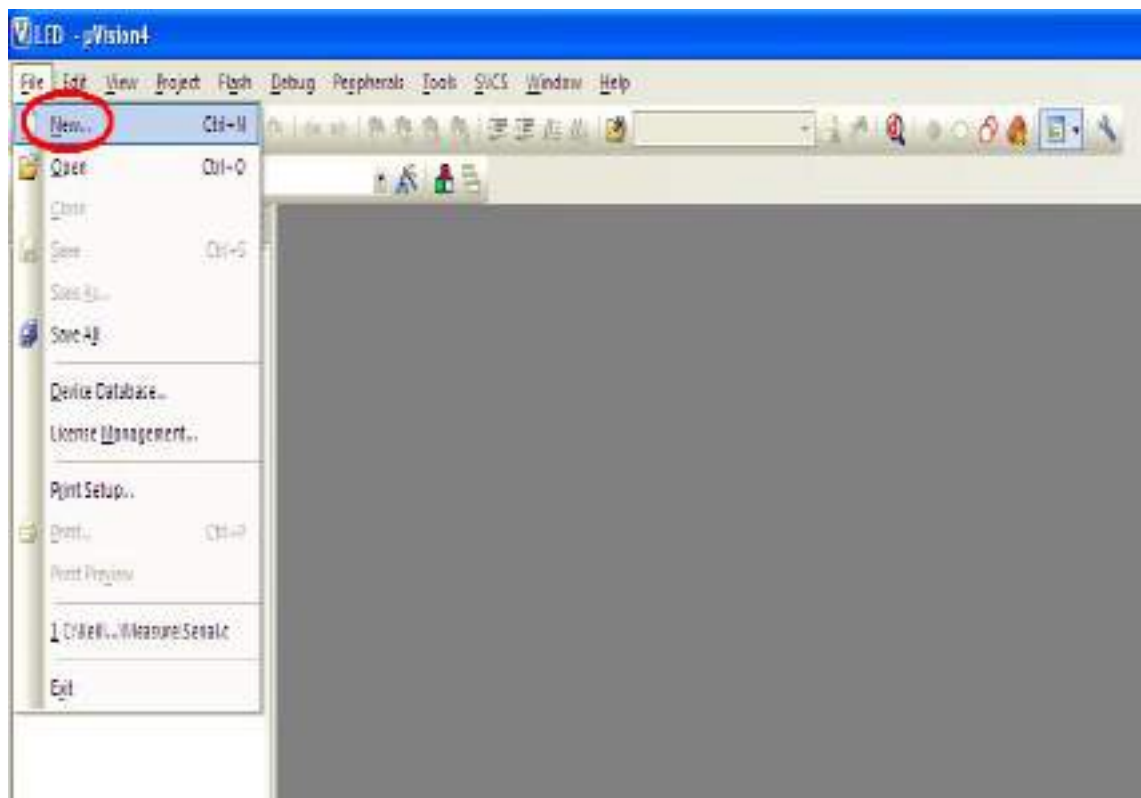
STEP 6: Within the Microchip, select AT895C2.



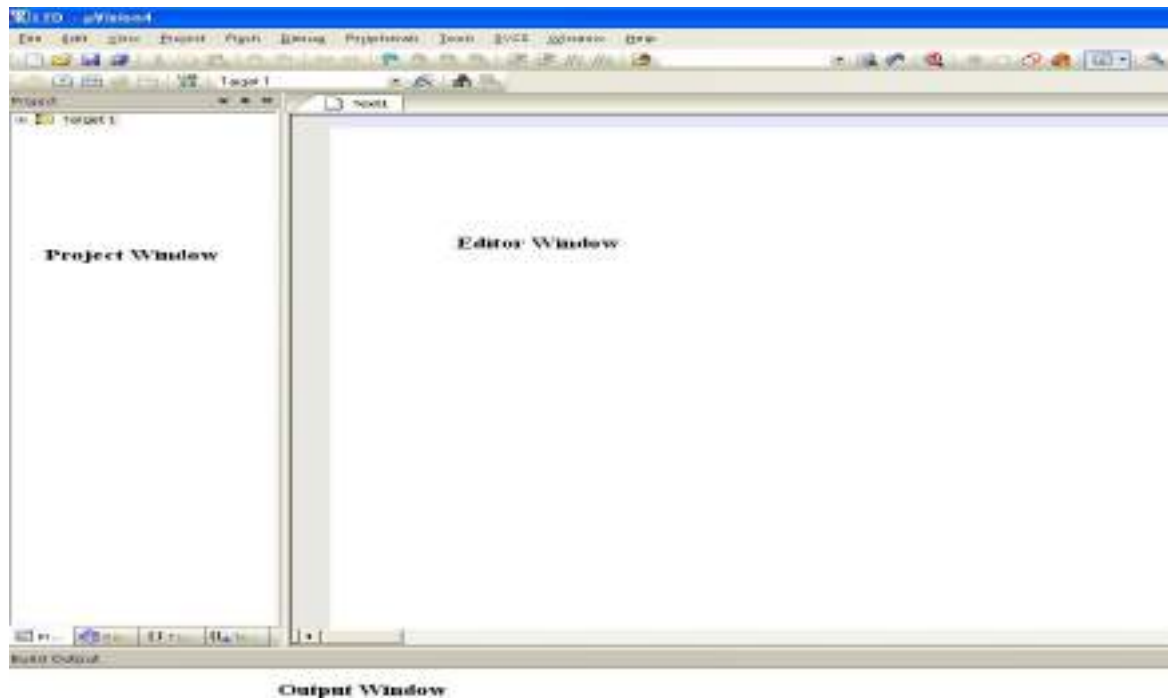
STEP 7: Add Startup file to the project by clicking “Yes”.



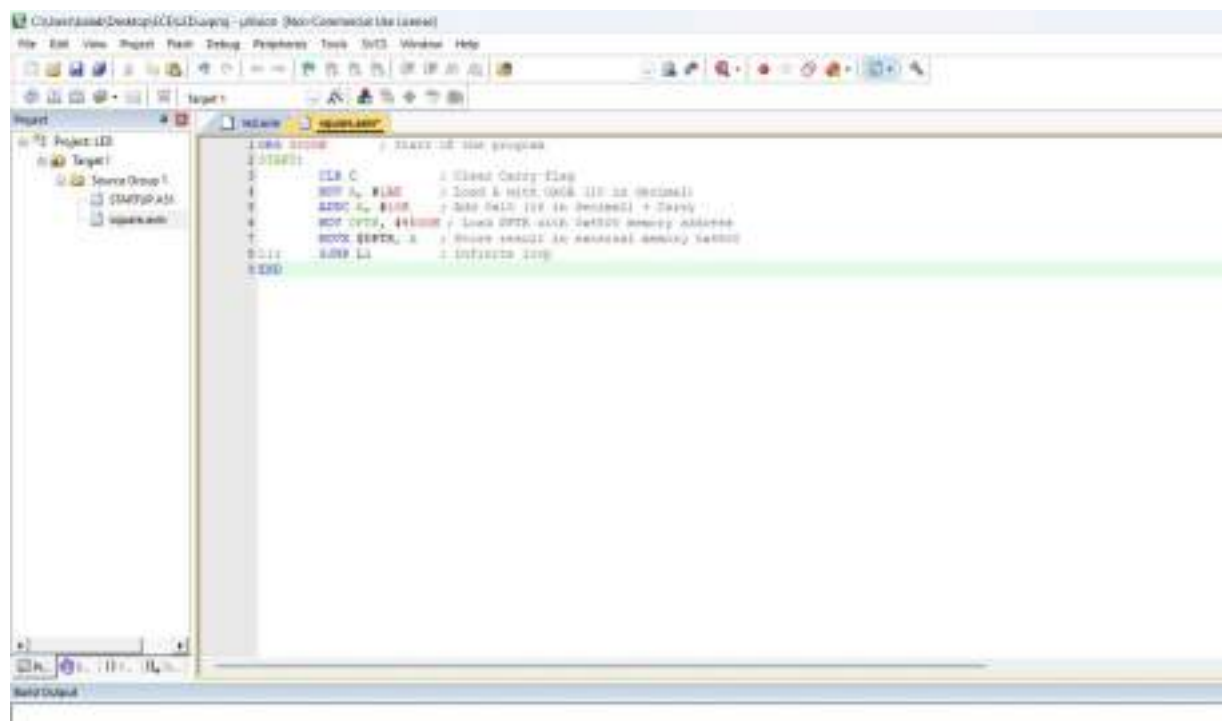
STEP 8: Next go to “File” and click “New”.



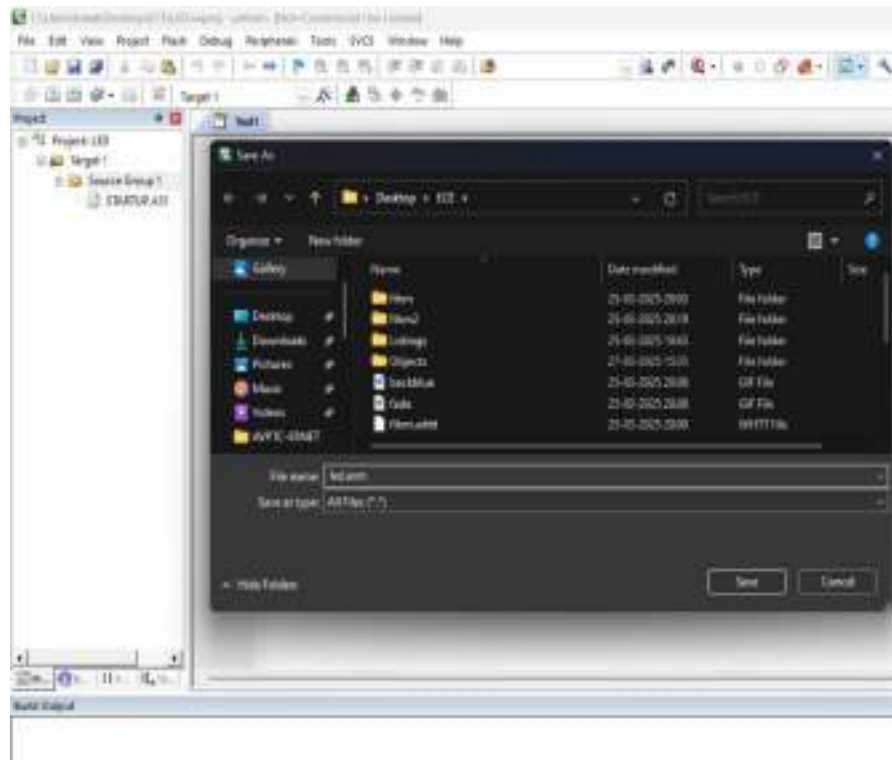
STEP 9: There are the three main windows are available in the keil IDE. First one is Project Workspace, the second one is Editor Window and third one is Output Window.



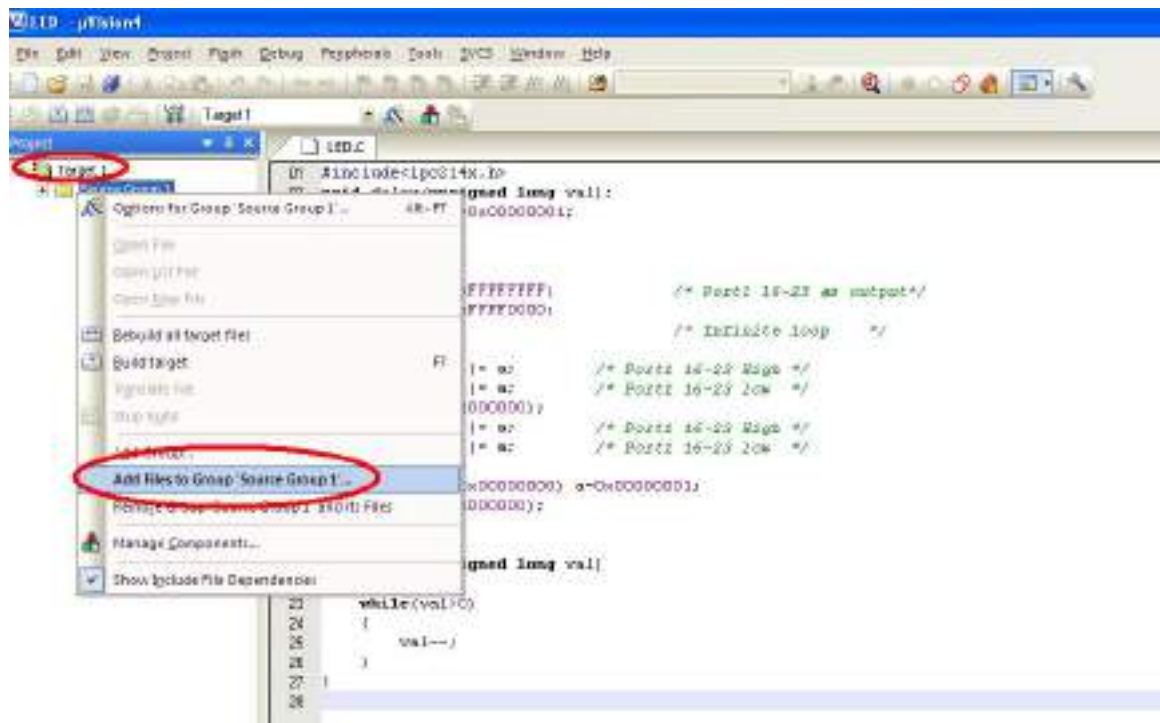
STEP 10: Can start to write *.asm code on the editor window.



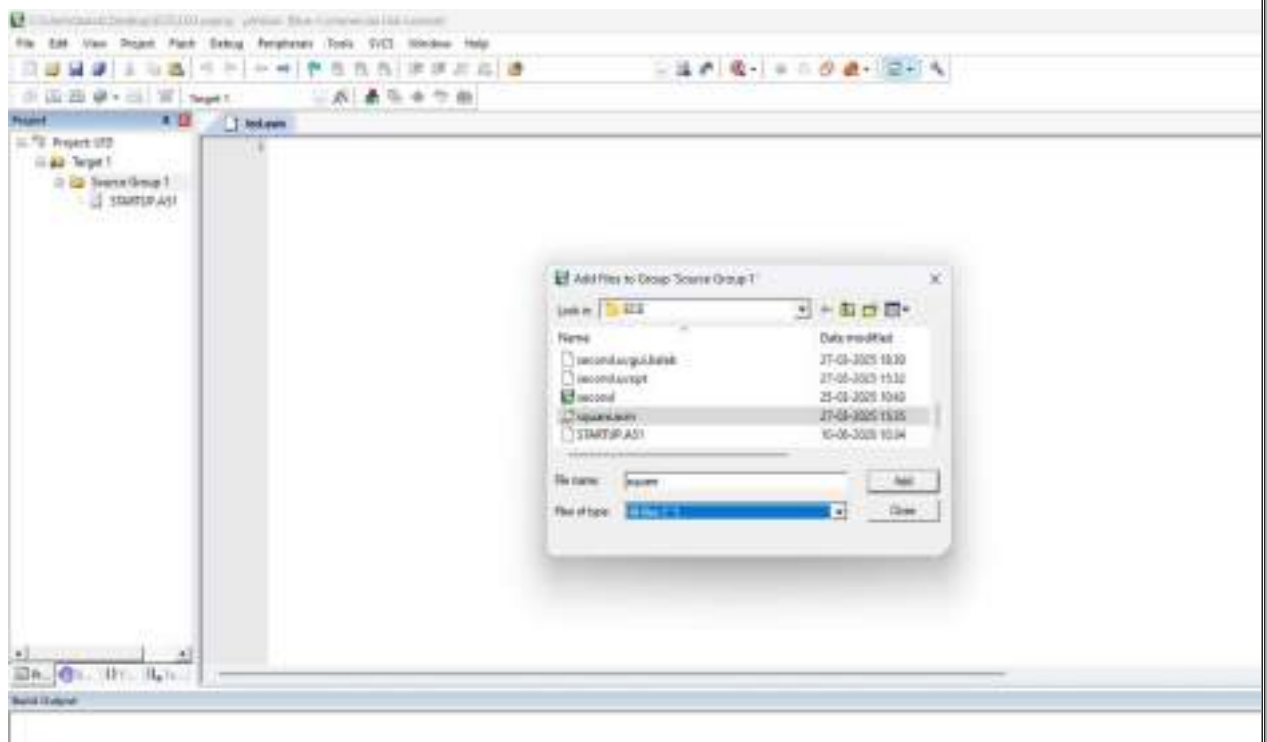
STEP 11: Can save the file, if the program is in “C” save as save as “filename.ASM”.



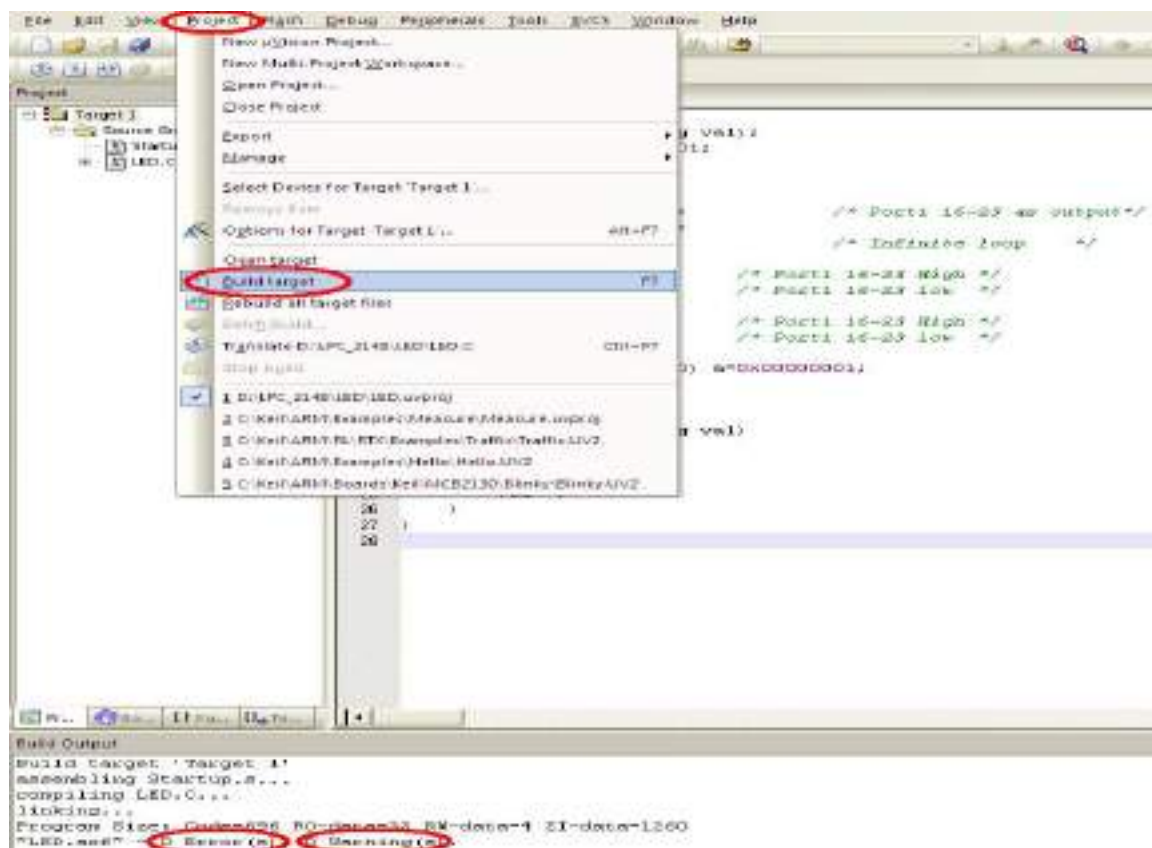
STEP 12: Add this source file to Group1, Go to the “Project Workspace” drag the +mark “Target 1” in that right click on “Source Group1” and click on “Add Files to Group “Source Group1””.



STEP 13: A small window will pop up with the name “Add Files to Group Source Group1”, by default, the Files of type will be in **All Files (*.asm)**. If the program is asm select **ASM Source file (*.s,*.src,*.a*)**.



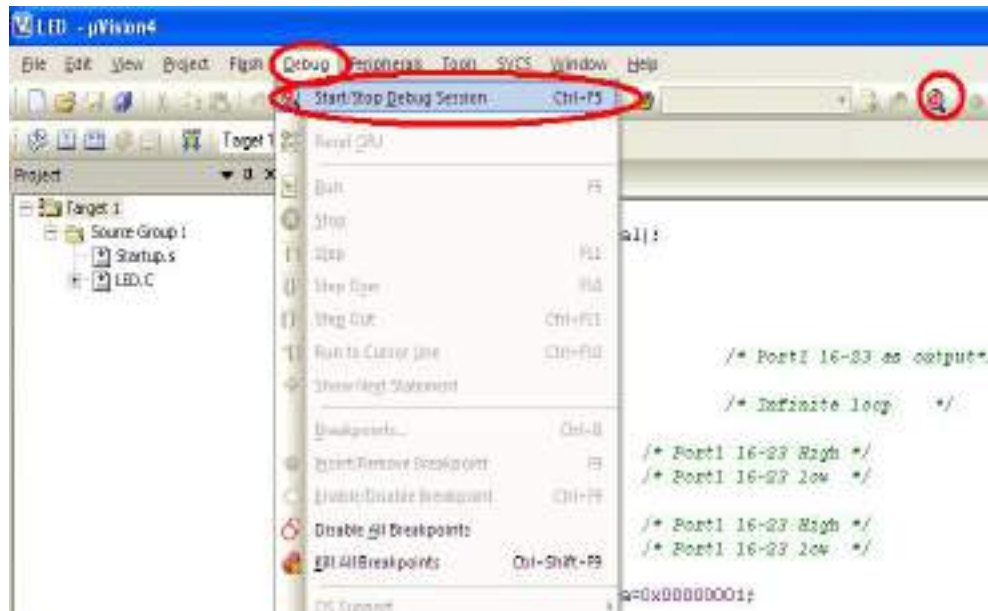
STEP 14: Then go to “Project” click on “Build Target” or F7. Output window will display related error and warning messages.



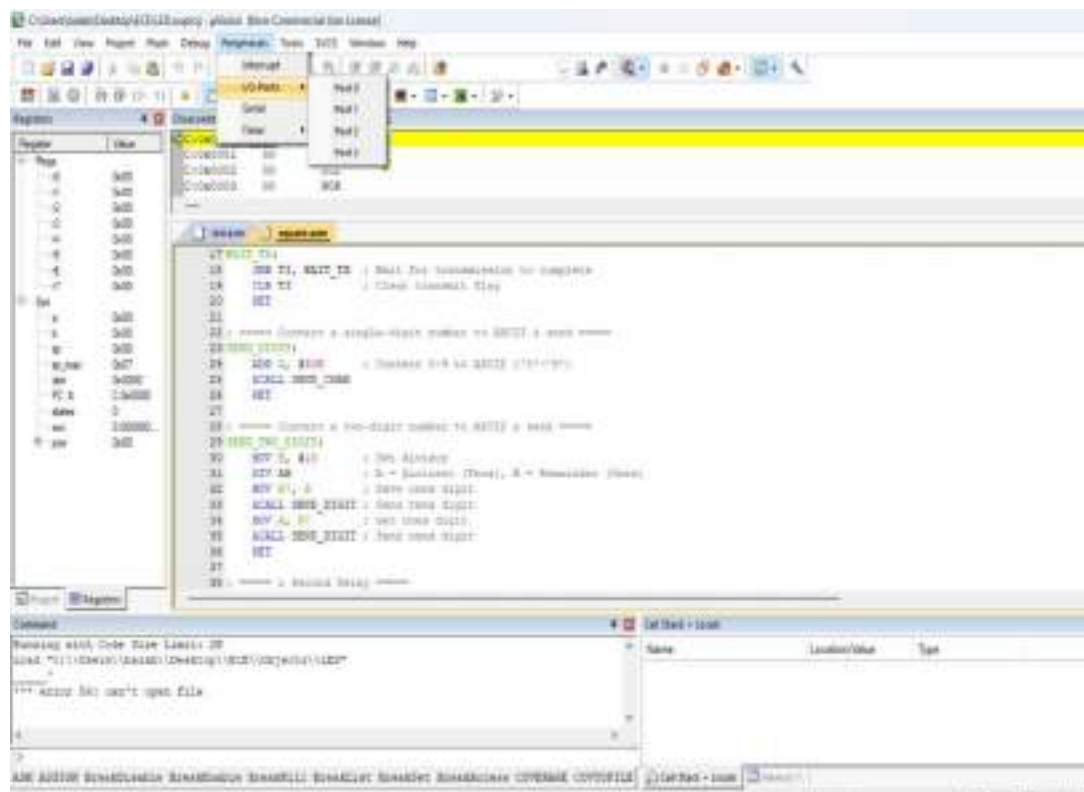
Simulation

Part:

STEP 15: Go to “Project” menu, click on “Rebuild all target Files” and start **Debug**. From **View** menu can get **Memory Window** and from **Peripherals** can get I/O ports, Serial etc. For access internal memory type **i:0x_memory** location example: **i:0x30** and for external and program memory **x:0x_memory** location, **c:0x_memory** location respectively. From **Register** window we can edit and access the values also.



STEP 16: If Output has to be seen on Port select Peripherals → IO Ports → Select Port



EXPERIMENTS USING 8051

EXP NO: 1	PROGRAMMING ARITHMETIC AND LOGICAL OPERATIONS IN 8051
DATE:	

AIM:

To write 8051 Assembly Language Program to demonstrate arithmetic and logic operations using Keil simulator and execute it.

SOFTWARE REQUIRED:

S.No	Software Requirements	Quantity
1	Keil μ vision5 IDE	1

INTRODUCTION TO 8051 SIMULATORS:

A simulator is software that will execute the program and show the results exactly to the program running on the hardware, if the programmer finds any errors in the program while simulating the program in the simulator, he can change the program and re-simulate the code and get the expected result, before going to the hardware testing. The programmer can confidently dump the program in the hardware when he simulates his program in the simulator and gets the expected results.

8051 controller is a most popular 8-bit controller which is used in a large number of embedded applications and many programmers write programs according to their application. So testing their programs in the software simulators is a way. Simulators will help the programmer to understand the errors easily and the time taken for the testing is also decreased.

These simulators are very useful for students because they do need not to build the complete hardware for testing their program and validate their program very easily in an interactive way.

List of 8051 Simulators:

The list of simulators is given below with their features:

1. MCU 8051: MCU 8051 is an 8051 simulator that is very simple to use and has an interactive IDE (Integrated Development Environment). It is developed by Martin Osmera and most important of all is that it is completely free. There are many features for this IDE they are

- ✓ It supports both C and assembly language for compilation and simulation

- ✓ It has an in-built source code editor, graphical notepad, ASCII charts, Assembly symbol viewer, etc. It also supports several 8051 ICs like at89c51, A89S52, 8051, 8052, etc.
- ✓ It will support certain electronic simulations like LED, 7segment display, LCD etc. which will help in giving the output when you interface these things to the hardware directly.
- ✓ It has tools like hex decimal editors, base converters, special calculator, file converters, inbuilt hardware programmers, etc.
- ✓ It has syntax validation, pop base auto-completion etc.

You can download this tool from <https://sourceforge.net/projects/mcu8051ide/files/>.

2. EDSIM 51: This is a virtual 8051 interfaced with virtual peripherals like 7 segment display, motor, keypad, UART etc. This simulator is exclusively for students developed by James Rogers,.

The features of this simulator are

- ✓ Have virtual peripherals like ADC, DAC with scope to display, comparator etc.
- ✓ Supports only assembly language
- ✓ IDE is completely written in JAVA and supports all the OS.
- ✓ Completely free and with user guide, examples, etc.

You can download this simulator from the <https://www.edsim51.com/index.html>.

3. 8051 IDE: This simulation software is exclusively for the Windows operating system (98/xp).

The features of this simulator are

- ✓ Text editor, assembler, and software simulate in one single program.
- ✓ Has facilities like Breakpoint setter, execute to break point, predefined simulator watch window, etc.
- ✓ It is available in both free version and paid version.

You can download this tool from <https://www.acebus.com/win8051.htm>

4. KEIL μ Vision: KEIL is the most popular software simulator. It has many features like interactive IDE and supports both C and assembly languages for compilation and simulation.

You can download and get more information from <https://www.keil.com/c51/>.

INSTALLATION OF KEIL SOFTWARE

Set up Keil IDE for Programming

Keil μ Vision IDE is a popular way to program MCUs containing the 8051 architectures. It supports over 50 microcontrollers and has good debugging tools including logic analyzers and watch windows.

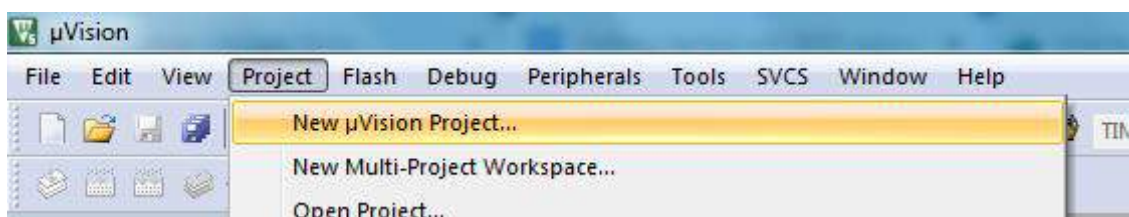
In this article, we will use the AT89C51ED2 microcontroller, which has:

- 64 KB FLASH ROM
- On-chip EEPROM
- 256 Bytes RAM
- In-System programming for uploading the program
- 3 Timer/counters
- SPI, UART, PWM



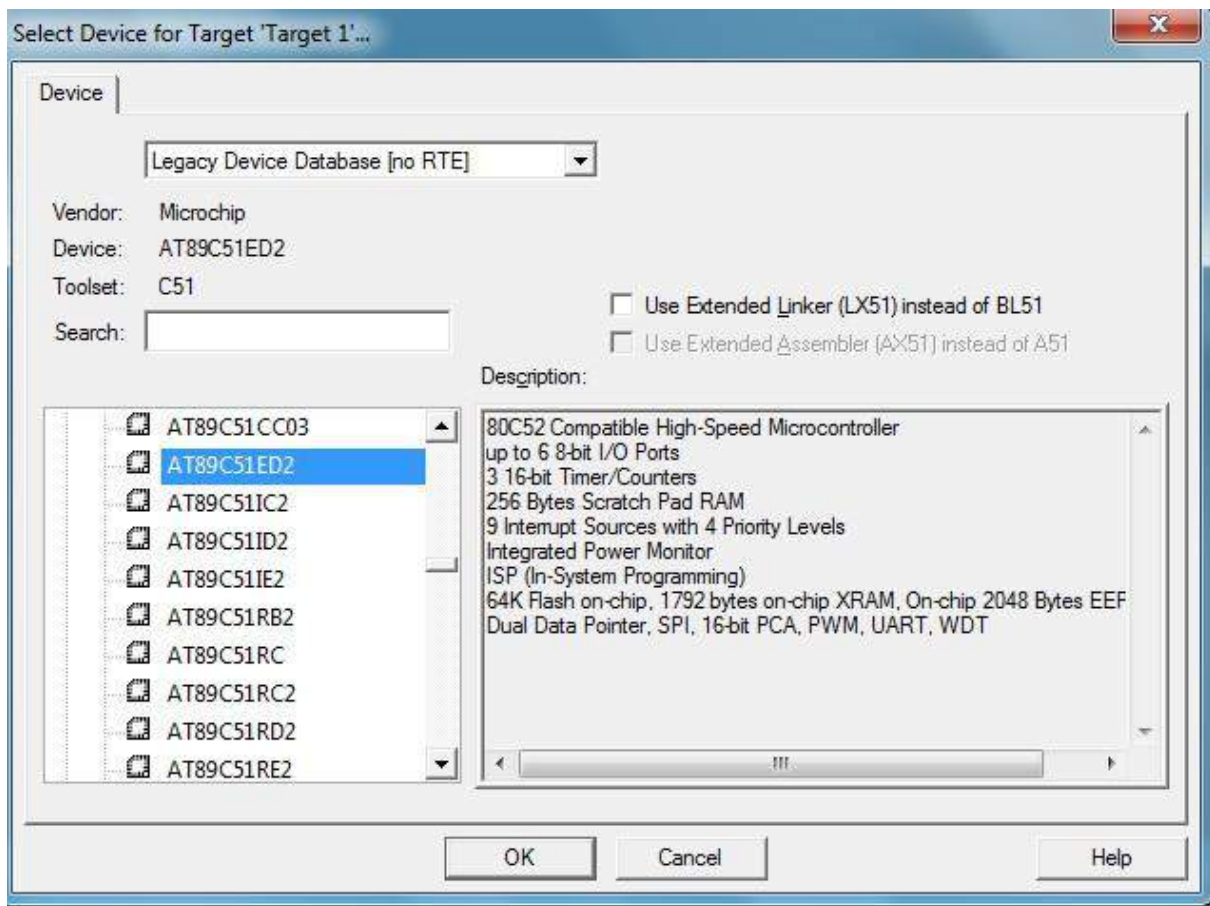
The Keil μ Vision icon.

To start writing a new program, you need to create a new project. Navigate to **project** —> **New μ Vision project**. Then save the new project in a folder.

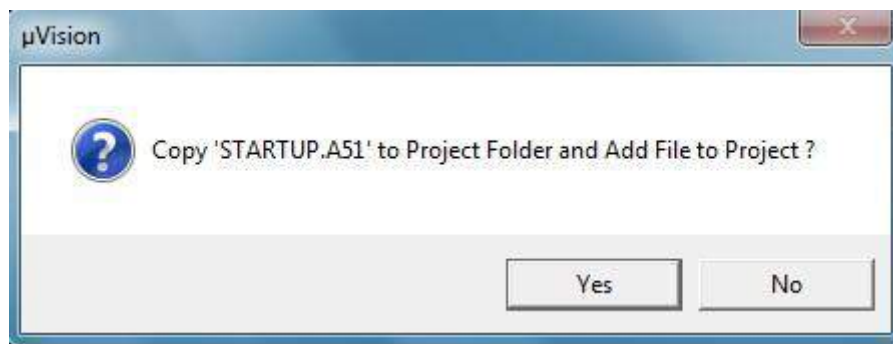


After saving the file, a new window will pop up asking you to select your microcontroller.

As discussed, we are using AT89C51/AT89C51ED2/AT89C52, so select this controller under the Microchip section (as Atmel is now a part of Microchip).

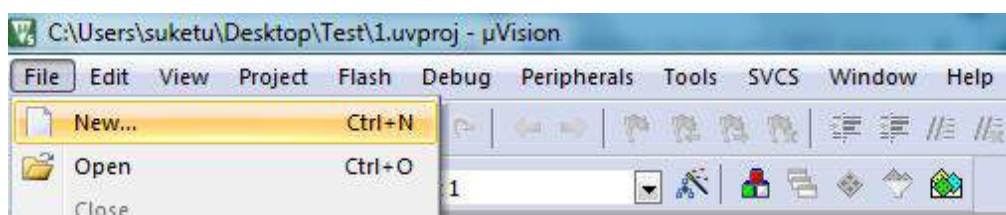


Select 'Yes' in the next pop-up, as we do not need this file in our project.

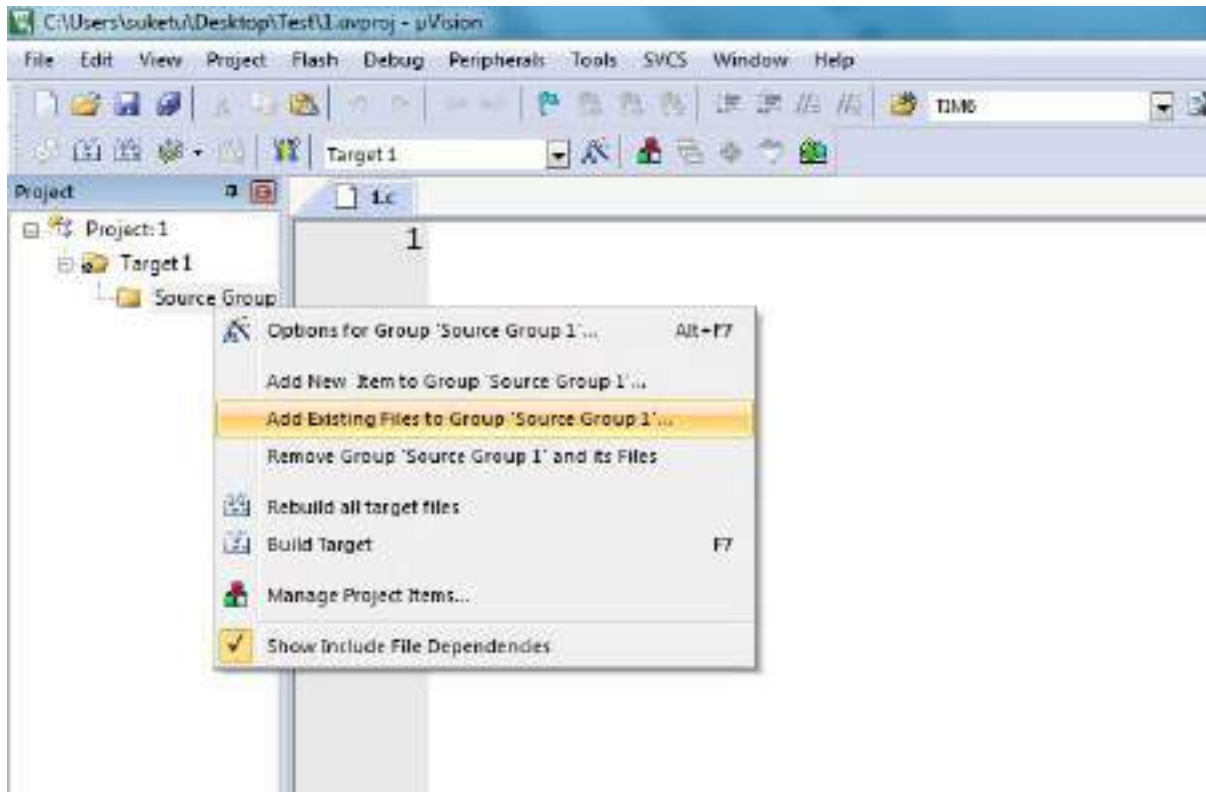


Our project workspace is now ready!

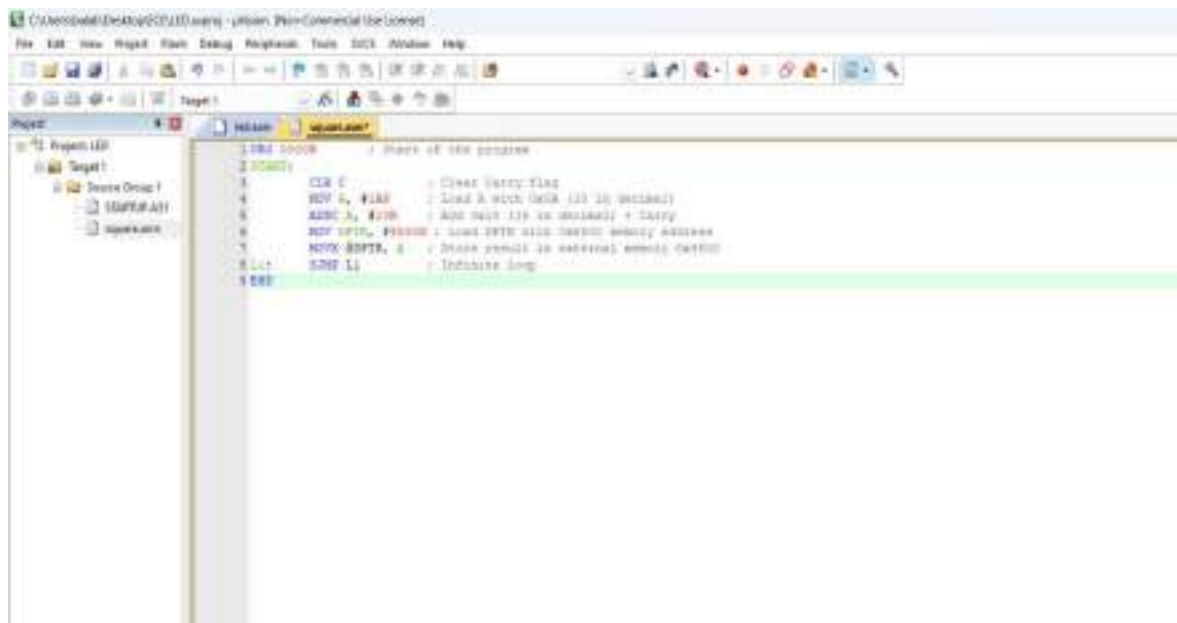
From here, we need to create a file where we can write our C code. Navigate to **File** → **New**. Once the file is created, save it with .c extension in the same project folder.



Next, we have to add that .c or .asm file to our project workspace. Select **Add Existing** Files and then select the created .c or .asm file to get it added.



The workspace and project file are ready.



PROCEDURE

1. Create a new project, go to “Project” and close the current project “Close Project”.
2. Next, Go to the Project New μ Vision Project and Create a New Project, Select the Device for the Target.
3. Select the device AT89C51ED2 or. AT89C51 or AT89C52
4. Add Startup file Next go to “File” and click “New”.
5. Write a program on the editor window and save it with .asm extension.
6. Add this source file to Group and click on “Build Target” or F7.
7. Go to debugging mode to see the result of simulation by clicking Run or step run.8.

PROGRAM:**8-BIT ADDITION**

```
ORG 0000H    ; Start of the program
START:
    CLR C      ; Clear Carry flag
    MOV A, #1AH ; Load A with 0x0A (10 in decimal)
    ADDC A, #10H ; Add 0x10 (16 in decimal) + Carry
    MOV DPTR, #4500H ; Load DPTR with 0x4500 memory address
    MOVX @DPTR, A ; Store result in external memory 0x4500
L1:    SJMP L1    ; Infinite loop
END
```

OUTPUT :**PROGRAM****8-BIT SUBTRACTION**

```
ORG 0000H    ; Start of the program
START:    CLR C      ; Clear Carry flag (used in SUBB)
    MOV A, #0AH    ; Load A with 0x0A (decimal 10)
    SUBB A, #05H    ; Subtract 0x05 (decimal 5) from A (A = A - 5 - CY)
    MOV DPTR, #4500H ; Load DPTR with address 0x4500H
    MOVX @DPTR, A ; Store result of subtraction at external memory (XDATA) 0x4500
L1:    SJMP L1    ; Infinite loop to hold execution
END
```

OUTPUT :

PROGRAM**8-BIT MULTIPLICATION**

ORG 0000H ; Start of the program

START:

MOV A, #05H ; Load A with 0x05 (decimal 5)

MOV B, #03H ; Load B with 0x03 (decimal 3)

MUL AB ; Multiply A $\times$ B (Result stored in A & B)

MOV DPTR, #4500H ; Load DPTR with address 0x4500H

MOVX @DPTR, A ; Store the lower byte of the result (A) at 0x4500H

INC DPTR ; Increment DPTR to 0x4501H

MOV A, B ; Move higher byte of result (B) to A

MOVX @DPTR, A ; Store the higher byte at 0x4501H

L1: SJMP L1 ; Infinite loop to hold execution

END

OUTPUT :

PROGRAM**8-BIT DIVISION**

ORG 0000H ; Start of the program

START:

MOV A, #H ; Load A with 0x15 (decimal 21)

MOV B, #03H0E ; Load B with 0x03 (decimal 3)

DIV AB ; Divide A by B ($A \div B$)

MOV DPTR, #4500H ; Load DPTR with address 0x4500H

MOVX @DPTR, A ; Store the quotient in external memory (XDATA 0x4500)

INC DPTR ; Increment DPTR to 0x4501H

MOV A, B ; Move the remainder to A

MOVX @DPTR, A ; Store the remainder in external memory (XDATA 0x4501)

L1: SJMP L1 ; Infinite loop to hold execution

END

OUTPUT :

PROGRAM**LOGICAL OPERATION**

```
ORG 0000H      ; Start of program
START:
    MOV A, #0F0H ; Load A with 1111 0000 (F0H)
    MOV R1, #0AAH ; Load R1 with 1010 1010 (AAH)

    ; Perform AND operation
    ANL A, R1    ; A = A AND R1 (1010 0000)
    MOV DPTR, #4500H
    MOVX @DPTR, A ; Store AND result in memory

    ; Perform OR operation
    ORL A, R1    ; A = A OR R1 (1010 1010)
    INC DPTR
    MOVX @DPTR, A ; Store OR result in memory

    ; Perform XOR operation
    XRL A, R1    ; A = A XOR R1 (0000 0000)
    INC DPTR
    MOVX @DPTR, A ; Store XOR result in memory

    ; Perform NOT (Complement) operation
    MOV A, #55H  ; Load A with 0101 0101 (55H)
    CPL A        ; Complement A (1010 1010)
    INC DPTR
    MOVX @DPTR, A ; Store NOT result in memory

L1: SJMP L1      ; Infinite loop
END
```

OUTPUT :

Embedded C Program to perform basic arithmetic operations like addition, subtraction, multiplication and division

```
#include<reg51.h>
void main(void)
{
    unsigned char x,y,z, a,b,c, d,e,f, p,q,r; //define variables
    //addition
    x=0x12; //first 8-bit number
    y=0x34; //second 8-bit number
    P0=0x00; //declare port 0 as output port
    z=x+y; // perform addition
    P0=z; //display result on port 0
    //subtraction
    a=0x12; //first 8-bit number
    b=0x34; //second 8-bit number
    P1=0x00; //declare port 1 as output port
    c=b-a; // perform subtraction
    P1=c; //display result on port 1
    //multiplication
    d=0x12; //first 8-bit number
    e=0x34; //second 8-bit number
    P2=0x00; //declare port 2 as output port
    f=e*d; // perform multiplication
    P2=f; //display result on port 2
    //division
    p=0x12; //first 8-bit number
    q=0x34; //second 8-bit number
    P3=0x00; //declare port 3 as output port
    r=q/p; // perform division
    P3=r; //display result on port 3
    while(1);
}
```

Embedded C Program to perform basic Logic operations like OR, AND, EXOR, NOT

```
#include<reg51.h>
void main(void)
{
    unsigned char x,y,z, a,b,c, d,e,f, p,q,r; //define variables
    // AND
    x=0x12; //first 8-bit number
    y=0x34; //second 8-bit number
    P0=0x00; //declare port 0 as output port
    z=x&y; // perform AND
    P0=z; //display result on port 0
    //OR
    a=0x12; //first 8-bit number
    b=0x45; //second 8-bit number
    P1=0x00; //declare port 1 as output port
```

```
=a|b; // perform OR
P1=c; //display result on port 1
//EXOR
d=0x12; //first 8-bit number
e=0x34; //second 8-bit number
P2=0x00; //declare port 2 as output port
f=e^d; // perform EXOR
P2=f; //display result on port 2
//NOT
p=0x12; // 8-bit number
P3=0x00; //declare port 3 as output port
r=~p; // perform Complement
P3=r; //display result on port 3
while(1);
}
```

Viva-Voce Questions

1. What are the arithmetic instructions available in the 8051 microcontroller?
2. What is the difference between logical AND, OR, and XOR operations?
3. Which register is primarily used for arithmetic operations in the 8051?
4. What is the function of the Carry flag during arithmetic operations?
5. Differentiate between the ADD and ADDC instructions.

RESULT:

Thus 8051 Assembly Language Program to demonstrate arithmetic and logic operations using Keil simulator is written and executed.

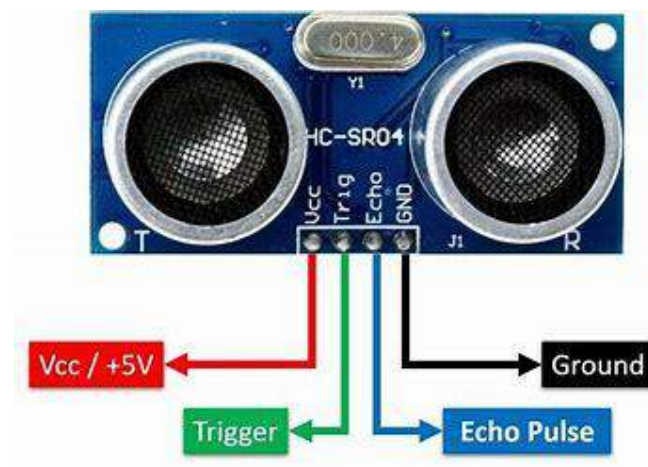
EXP NO: 2.A	IMPLEMENTATION OF ULTRASONIC SENSOR USING ARDUINO
DATE:	

AIM

To write and execute program to implement Ultrasonic Sensor using Arduino.

SOFTWARE /HARDWARE REQUIRED

S. No	Requirements	Quantity
1	Arduino IDE	
2	Arduino Uno Board	1
3	Ultra Sonic Sensor	1
4	Bread Board	1
5	Connecting Wires	10
6	LCD Display	1

ULTRASONIC SENSOR

The **HC-SR04** is an ultrasonic distance measuring sensor commonly used with Arduino and other microcontrollers. It uses **ultrasonic sound waves** (inaudible to humans) to measure the distance of an object.

PIN CONFIGURATION

- **VCC (Red)** → Connect to +5V
- **GND (Black)** → Connect to Ground
- **Trig (Green)** → Trigger input pin (from Arduino)
- **Echo (Blue)** → Echo output pin (to Arduino)

Working Principle

1. Triggering

- The Arduino sends a **10 μs HIGH pulse** to the **TRIG** pin.
- This makes the sensor emit **ultrasonic waves at 40 kHz** from the **Transmitter (T)**.

2. Reflection

- The sound waves travel through the air and, if they hit an object, they bounce back.

3. Echo

- The sensor receives the reflected waves at the **Receiver (R)**.
- It then outputs a HIGH pulse on the **Echo pin**.
- The **duration** of this pulse is proportional to the time taken by the sound to travel to the object and back.

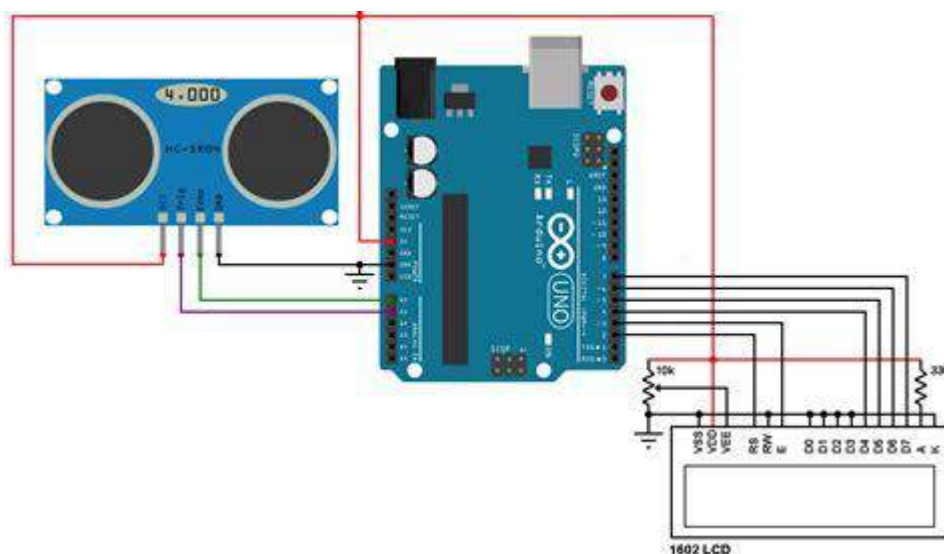
4. Distance Calculation

- Since **speed of sound = 343 m/s (≈0.034 cm/μs)**:
- $Distance = \frac{Time \times 0.034}{2}$
- The division by 2 is because the wave travels to the object and back.

Example Application

- Obstacle detection in **robots**
- **Parking sensors** in cars
- **Water level measurement**
- **Security systems** (detecting intruders)

CIRCUIT DIAGRAM



PROGRAM

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 4, 5, 6, 7);
const int trigPin = 3;
const int echoPin = 2;
long duration;
int distanceCm ;
void setup() {
  lcd.begin(16, 2);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}
void loop()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distanceCm = duration * 0.034 / 2;
  Serial.print("Distance = ");
  Serial.println(distanceCm );
  lcd.setCursor(0, 0);
  lcd.print("Distance: ");
  lcd.print(distanceCm);
  lcd.print(" cm");
  delay(10);
}
```

VIVA-VOCE Questions

1. What is the working principle of an ultrasonic sensor?
2. What is the purpose of the TRIG and ECHO pins?
3. How is distance calculated using the ultrasonic sensor?
4. What is the operating frequency of the HC-SR04 ultrasonic sensor?
5. Mention two applications of ultrasonic sensors.

RESULT

Thus, the Ultrasonic Sensor was successfully interfaced with Arduino, and the measured Distance values were displayed on both the Serial Monitor and the LCD.

EXP NO: 2.B	IMPLEMENTATION OF TEMPERATURE SENSOR USING ARDUINO
DATE:	

AIM

To write and execute program to implement Temperature Sensor using Arduino.

SOFTWARE /HARDWARE REQUIRED

S. No	Requirements	Quantity
1	Arduino IDE	1
2	Arduino Uno Board	1
3	LM 35 -Temperature Sensor	1
4	Bread Board	1
5	Connecting Wires	10
6	LCD Display	

TEMPERATURE SENSOR – LM 35

The **LM35** is a **precision temperature sensor** IC whose output voltage is **linearly proportional to the Celsius (°C) temperature**.

It is one of the most widely used sensors in electronics and Arduino projects because it is easy to use, accurate, and cheap.

- The LM35 generates a **voltage output proportional to the temperature**.
- The **scale factor** is:

10 mV=1°C

- Example:

➤ At **25°C**, output = 250 mV (0.25 V)

➤ At **100°C**, output = 1000 mV (1.0 V)

So, you can directly read the temperature by converting voltage.

If using Arduino (10-bit ADC, 0–1023 counts, 5V reference):

$$V_{out} = \frac{ADC\ Value * 5.0}{1023}$$

$$T^{\circ}(C) = V_{out} * 100$$

FEATURES

- Linear output in °C (not °F)
- Calibrated directly in Celsius
- Operating range: **-55°C to +150°C**
- Accuracy: $\pm 0.5^{\circ}\text{C}$ (at 25°C)
- Very low self-heating (does not affect readings)

APPLICATIONS

- Digital thermometers
- Air conditioners, refrigerators (temperature monitoring)
- Weather monitoring stations
- Industrial controllers
- Battery temperature measurement

```
#include <LiquidCrystal.h>
// Initialize the LCD (RS, E, D4, D5, D6, D7)
LiquidCrystal lcd(12, 11, 4, 5, 6, 7);
const int sensor = A0; // LM35 temperature sensor on A0
float tempc;           // temperature in °C
float vout;            // sensor voltage
void setup() {
  pinMode(sensor, INPUT);
  Serial.begin(9600); // Start serial communication
  lcd.begin(16, 2);   // Initialize LCD as 16x2
  delay(500);
  lcd.print("Temp Monitor");
  delay(1000);
  lcd.clear(); }
void loop()
{  int reading = analogRead(sensor); // Read ADC value (0–1023)
   // Convert ADC value to voltage (assuming 5V reference)
   vout = (reading * 5.0) / 1023.0; // LM35: 10mV per °C → 1°C = 0.01V
   tempc = vout * 100.0;
   // ===== Serial Monitor Output =====
   Serial.print("Temperature = ");
   Serial.print(tempc);
   Serial.println(" °C");
   // ===== LCD Output =====
```

```
lcd.setCursor(0, 0);  
lcd.print("Temp degree C : ");  
lcd.print(tempc);  
delay (1000); } // Update every 1 second
```

VIVA-VOCE Questions

1. Which temperature sensor was used in this experiment?
2. How does the LM35 sensor convert temperature into voltage?
3. What is the operating voltage of the LM35?
4. Why is analog-to-digital conversion required?
5. Name two applications of temperature sensors.

RESULT

Thus, the LM35 temperature sensor was successfully interfaced with Arduino, and the measured temperature values were displayed on both the Serial Monitor and the LCD.

EXPERIMENTS USING ARM

EXP NO: 3.A	STUDY OF ARM PROCESSORS
DATE:	

Aim :

To study the ARM processor architecture, hardware features and installing OS in Raspberri Pi.

Hardware / Software Requirement:**Raspberry Pi****Overview of ARM Processors in IoT Applications:**

ARM processors are the dominant architecture in the Internet of Things (IoT) landscape due to their energy efficiency, scalability, robust ecosystem, and cost-effectiveness. These processors power a wide range of IoT devices, from simple sensors to complex edge computing nodes, enabling real-time data processing, secure operations, and low-power consumption.

Key Features of ARM Processors for IoT**1. Energy Efficiency**

ARM processors are based on the RISC (Reduced Instruction Set Computing) principle, which uses a simplified instruction set. This leads to lower power consumption and less heat generation, making ARM ideal for battery-powered and energy-constrained IoT devices.

2. Scalability and Flexibility

The ARM architecture is highly scalable, supporting a wide range of devices from low-end microcontrollers (e.g., Cortex-M series) to high-performance application processors (e.g., Cortex-A series).

Modular design allows customization for specific IoT use cases, such as wearables, smart home devices, and industrial sensors.

3. Real-Time Processing

Many IoT applications require real-time responsiveness. ARM's Cortex-M series is optimized for real-time tasks such as sensor data collection and control systems, ensuring timely and predictable performance.

4. Security

ARM processors integrate security features like TrustZone, which provides hardware-enforced isolation for secure execution of sensitive code and data. This is crucial as IoT devices often handle personal or critical operational data.

5. Robust Software Ecosystem

ARM offers extensive development tools, compilers, and libraries tailored for IoT, simplifying software development, debugging, and optimization.

The widespread adoption of ARM ensures strong community and vendor support, reducing development time and cost.

6. Connectivity

ARM-based chips often include built-in support for common IoT communication protocols such as Wi-Fi, Bluetooth, Zigbee, and cellular, enabling seamless device integration and networking.

Main ARM Processor Families Used in IoT

Series	Typical Use Cases	Key Features
Cortex-M	Sensors, wearables, smart meters	Ultra-low power, real-time processing, cost-effective
Cortex-A	Edge gateways, smart cameras	High performance, supports OS like Linux, multimedia
Cortex-R	Industrial control, automotive	Real-time, high reliability, error management

- Cortex-M Series: Widely used in microcontrollers for low-power, real-time IoT applications. Examples include Cortex-M0, M3, and M4, which balance performance and energy efficiency.
- Cortex-A Series: Found in more powerful IoT edge devices and gateways, supporting advanced operating systems and applications that require higher computational power.
- Cortex-R Series: Used in applications demanding high reliability and real-time performance, such as industrial automation and automotive systems.

Architectural Components and Innovations

- Pipeline Architecture: Enables simultaneous processing of instruction stages, increasing throughput while maintaining energy efficiency.
- Instruction Sets: Evolve across generations (e.g., ARMv6, ARMv7, ARMv8, ARMv9), with enhancements for performance, security, and AI capabilities.
- DSP and ML Extensions: Features like Helium technology and DSP extensions in Cortex-M processors boost signal processing and machine learning performance at the edge.
- Custom Instructions: Allow for application-specific optimizations without fragmenting the

software ecosystem.

Advantages of ARM in IoT

Raspberry Pi 2 (2015) single-board computer, featuring ARMv7 architecture

Raspberry Pi 2 (2015) single-board computer, featuring ARMv7 architecture

Cost-Effectiveness: ARM's licensing model and widespread adoption drive down chip costs, making large-scale IoT deployment feasible.



Raspberry Pi 2 (2015) single-board computer, featuring ARMv7 architecture

Development Platforms: Popular boards like Raspberry Pi (Cortex-A) and Arduino (Cortex-M) use ARM processors, accelerating prototyping and development.

Example Applications

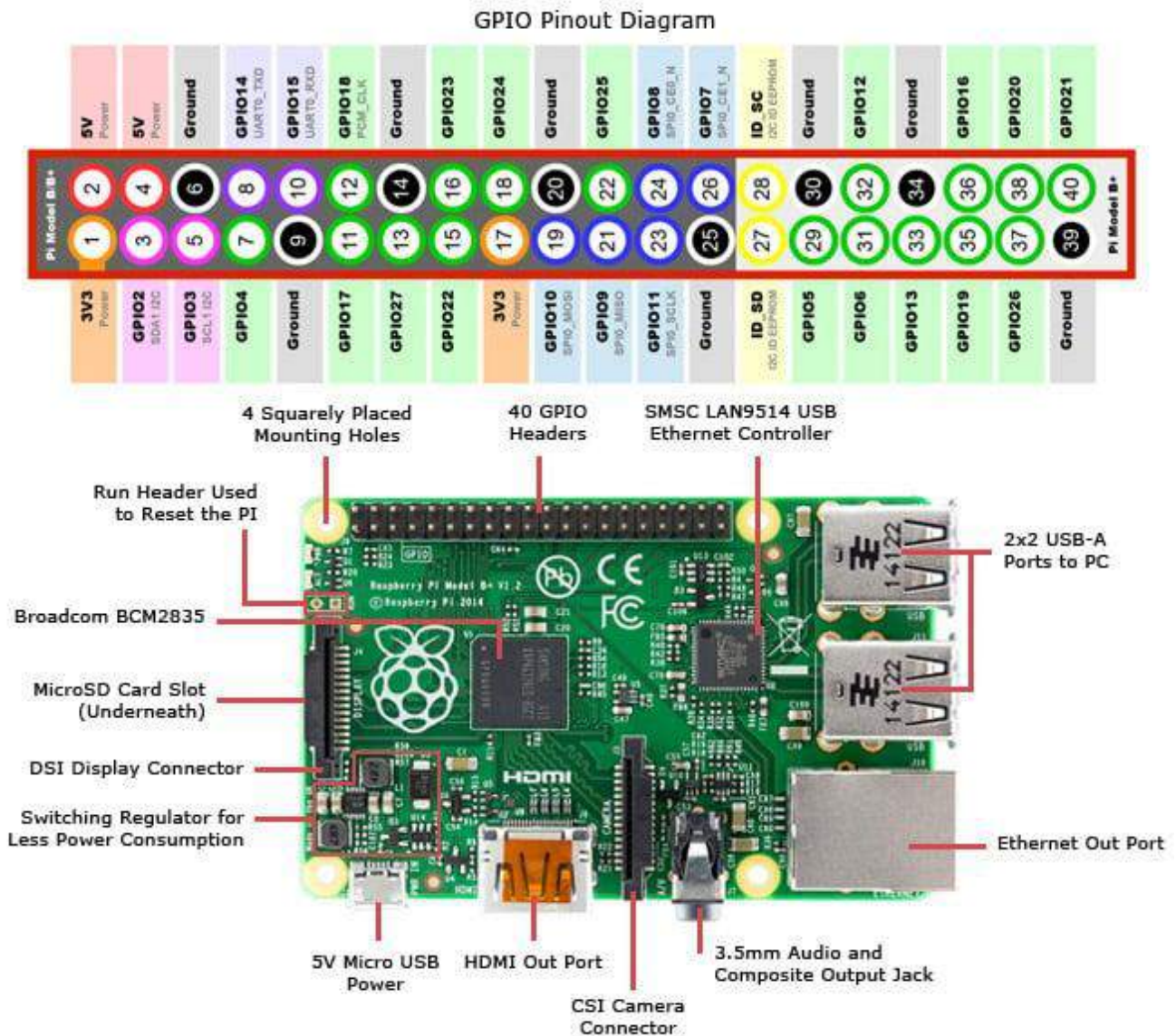
Smart Home Devices: Thermostats, security cameras, and smart lighting often use ARM Cortex-M or Cortex-A processors for efficient local processing and connectivity.

Wearables: Fitness trackers and smartwatches leverage ARM's low-power operation for extended battery life.

Industrial IoT: ARM Cortex-R and Cortex-M processors power sensors, controllers, and gateways in

manufacturing and logistics, providing real-time data and reliability.

HARWARE DETAILS :



Components of the Raspberry Pi :

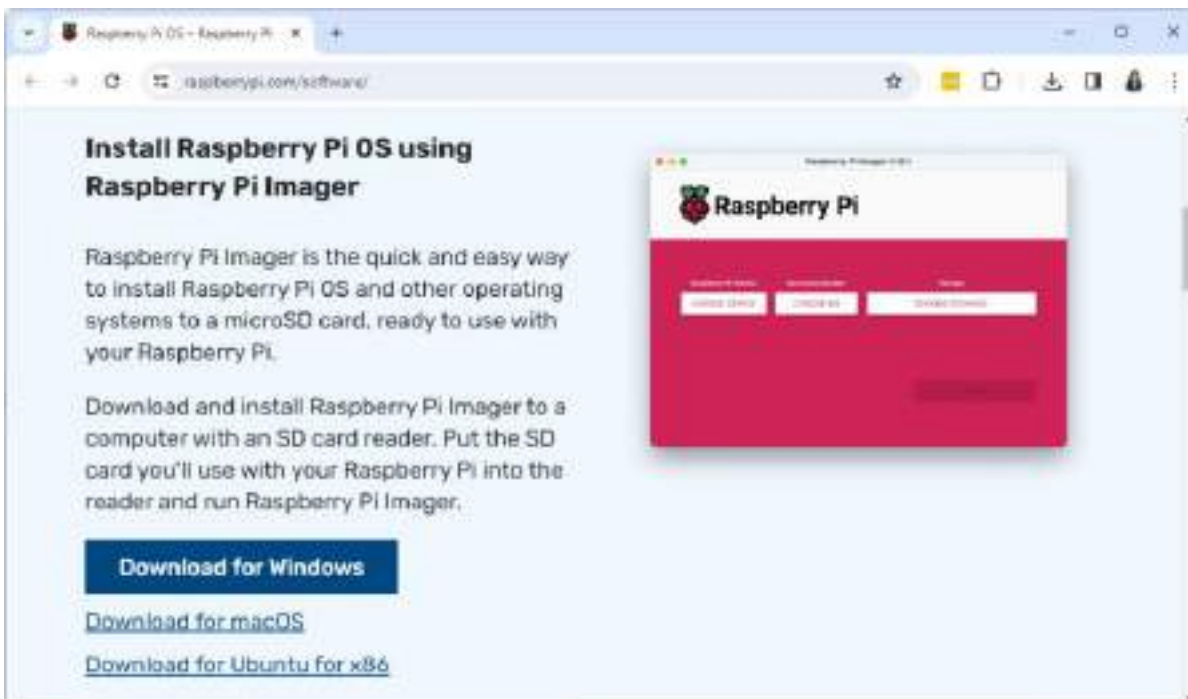
- **USB ports:** to connect a mouse, a keyboard, or other peripherals. It comes with two USB 3.0 and two USB 2.0 ports;
- **Ethernet port:** to connect to the internet using an Ethernet cable;
- **Audio jack:** to connect an audio device;

- **CSI connector:** to connect a camera with a CSI ribbon;
- **HDMI connector:** to connect a monitor or TV;
- **Processor:** is the brain of the Raspberry Pi;
- **MicroSD card slot:** to insert a microSD card to store your files and your operating system;
- **MicroUSB power input:** to power up your Pi;
- **DSI connector:** to connect DSI-compatible displays;
- **Antenna:** picks up wireless LAN and Bluetooth signals;
- **GPIOs (general purpose input output pins):** connect devices to interact with the outside world like sensors and outputs like LEDs and motors.

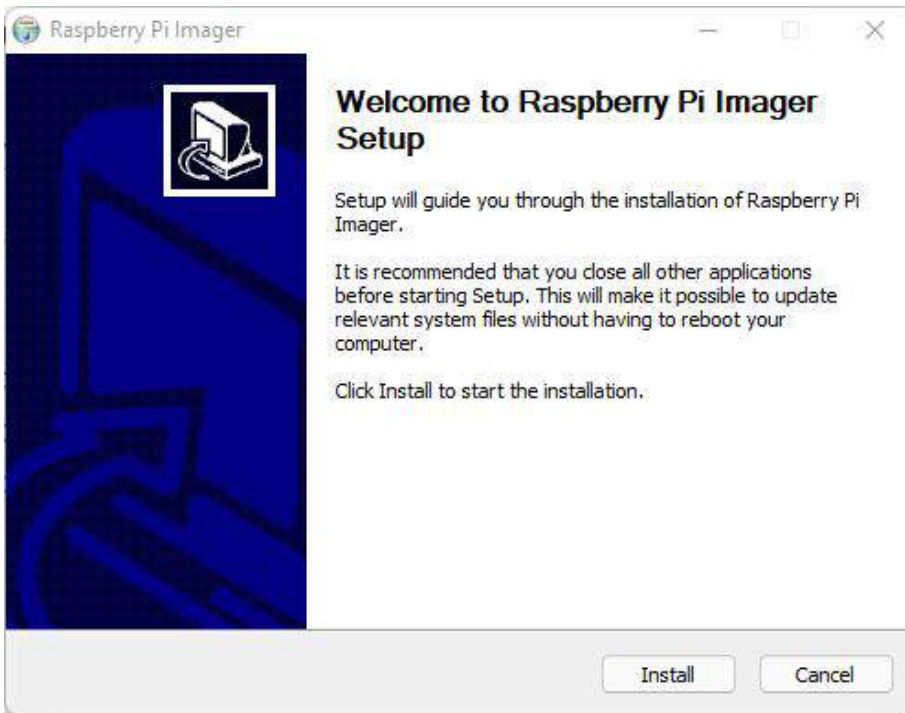
Installing the Operating System

There are several operating systems suitable for the Pi. The official distribution for the Raspberry Pi is Raspberry Pi OS and that's the one we recommend you install.

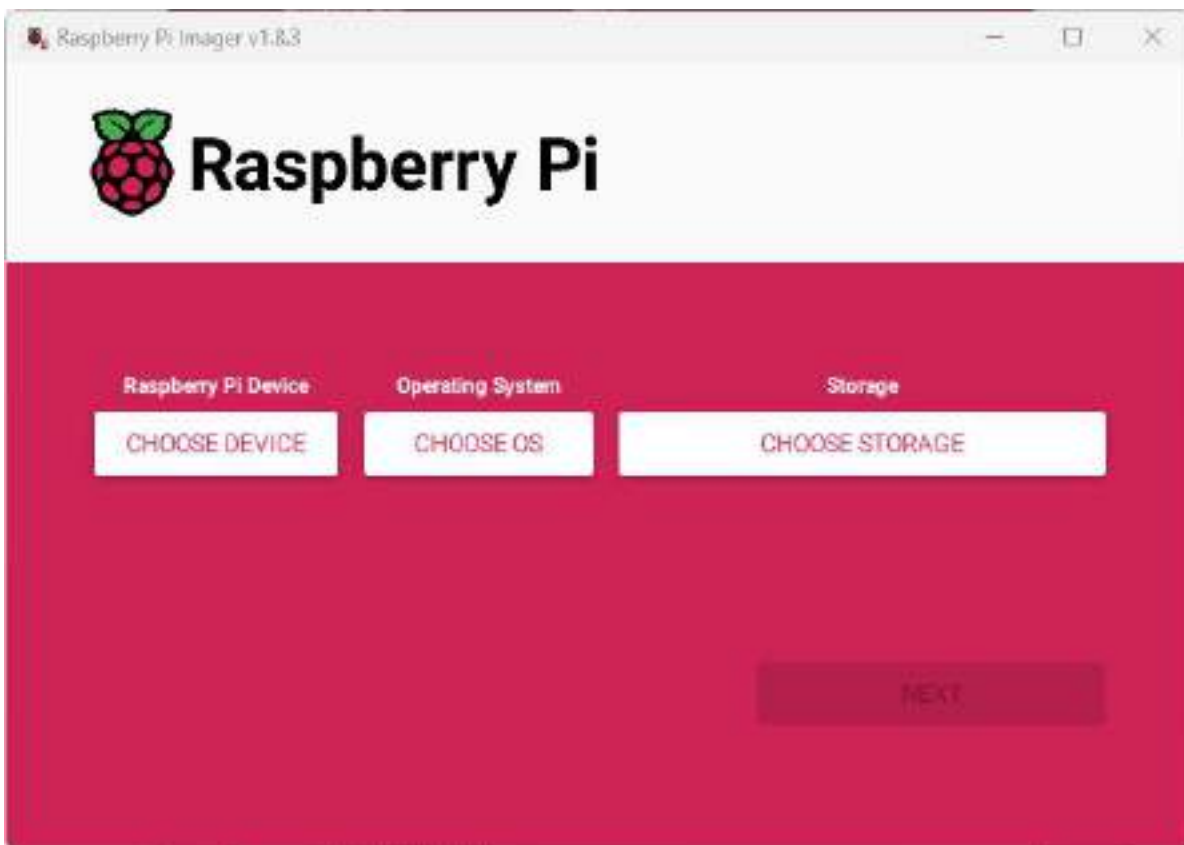
- 1) Start by connecting the microSD card to your computer.
- 2) Go to the [Raspberry Pi Software page](https://www.raspberrypi.com/software/).
- 3) Select and download the Raspberry Pi Imager (a tool to flash the OS on the microSD card) for your computer's operating system.



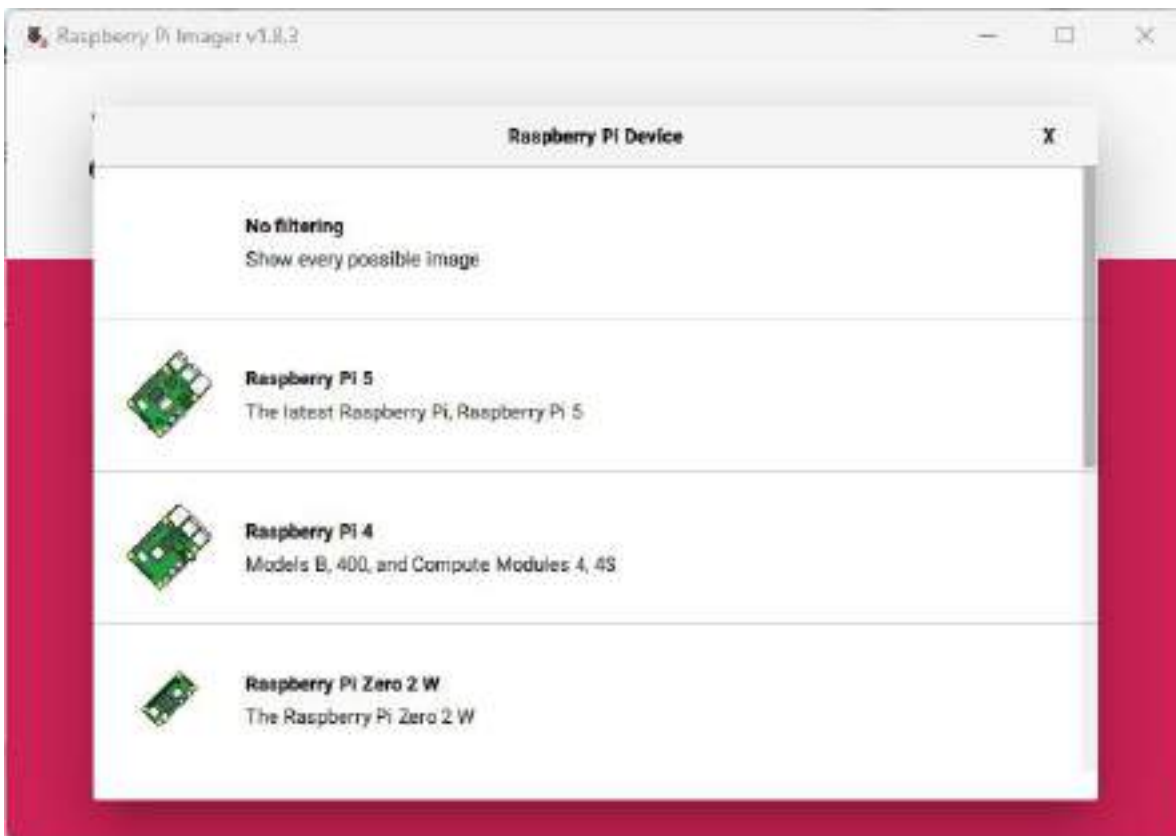
- 4) Click on the downloaded file to install the Raspberry Pi Imager.



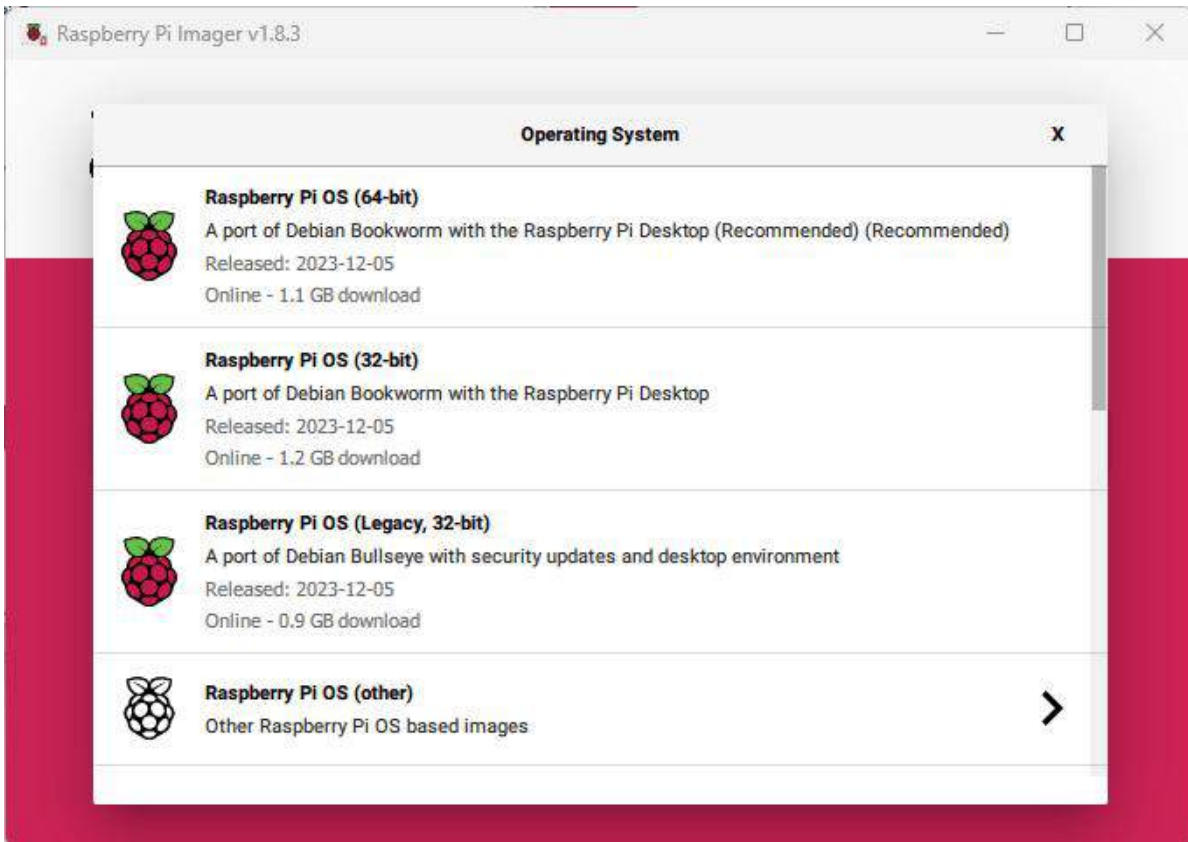
5) When the installation is complete, the Raspberry Pi Imager will open.



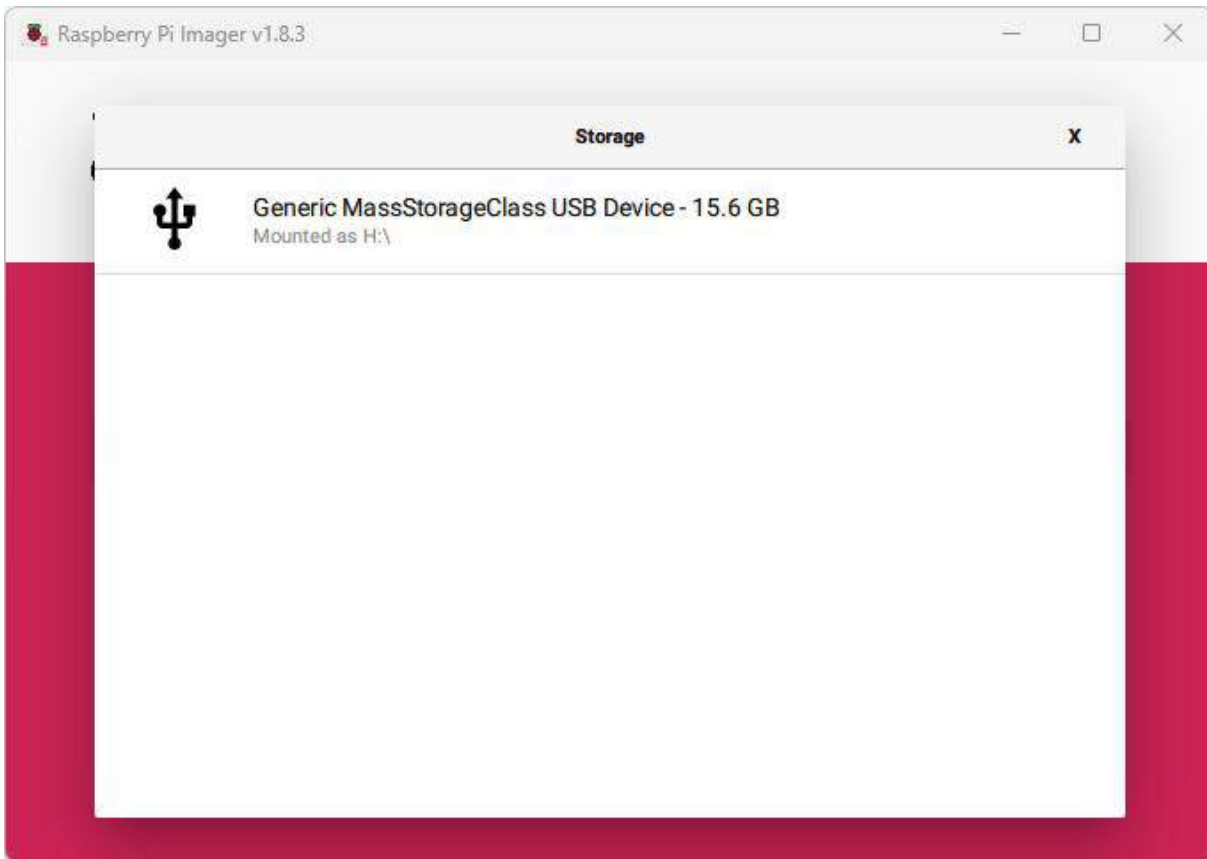
6) Click on Choose Device and select the Raspberry Pi board you're using.



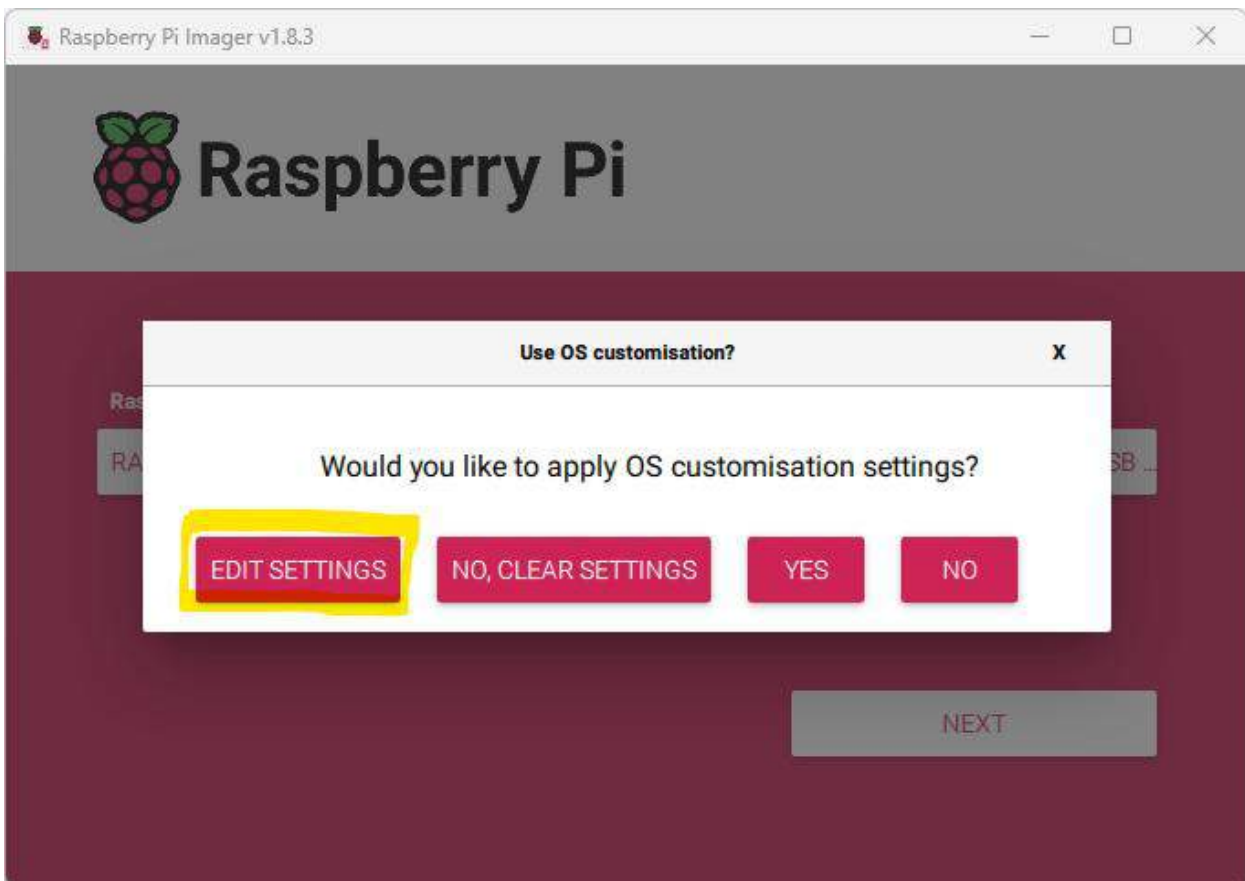
7) Click on Choose OS to select the Operating System. Select the Raspberry Pi OS (32-bit) or (64-bit).



8) Choose storage. You must choose the microSD card where you want to install the OS.



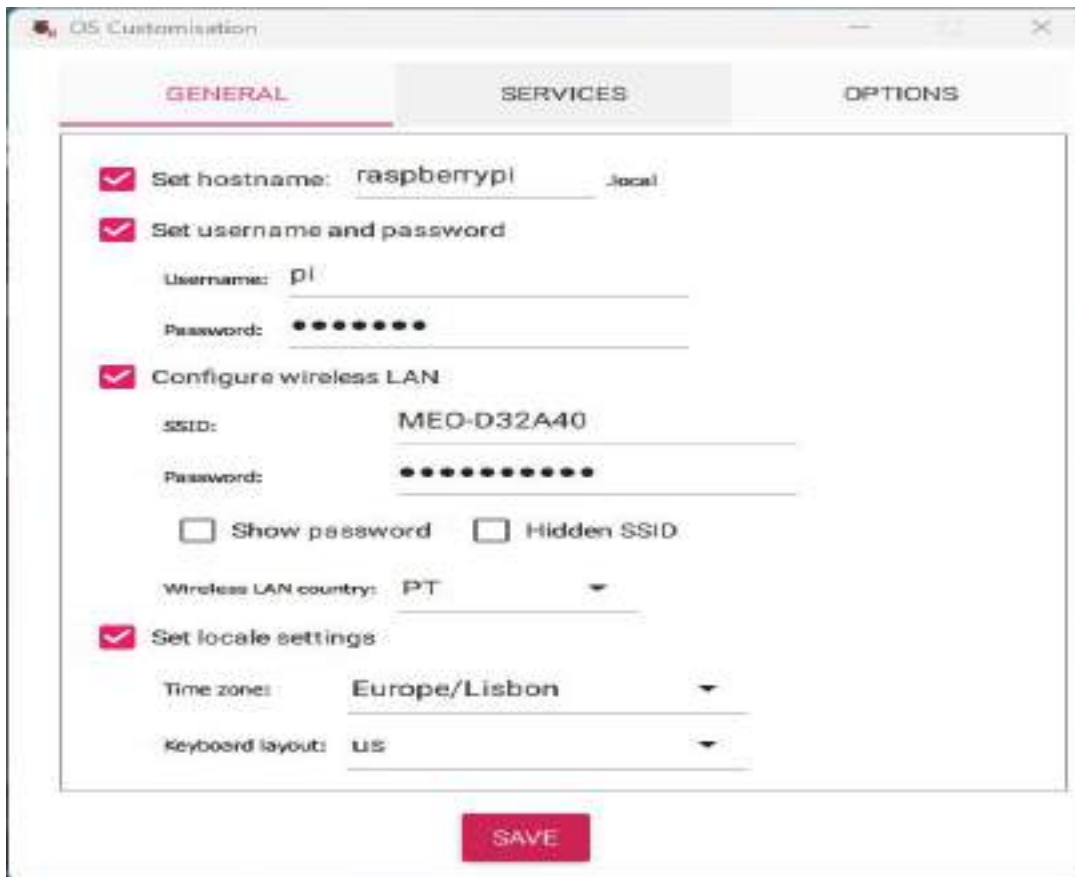
9) Click Next. Next, you'll be asked if you would like to apply customisation settings. Click on Edit Settings to set up the Wi-Fi credentials, and enable SSH.



10) Under the GENERAL tab, you can set an hostname (the default will be *raspberrypi*), user, and

password, and set Wi-Fi with your local network credentials, so that you can connect to your Raspberry Pi using Wi-Fi later on. You can also select your timezone.

Don't forget to set your Wireless LAN country!

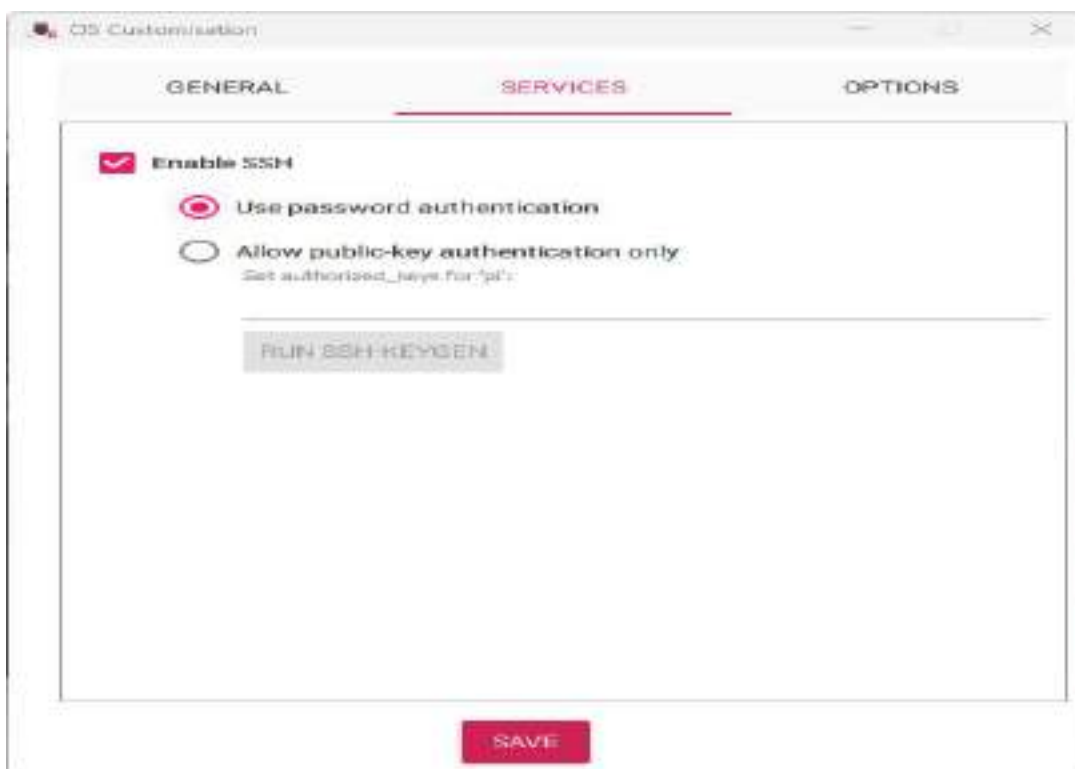


The screenshot shows the 'OS Customisation' window with the 'GENERAL' tab selected. The window has three tabs: 'GENERAL', 'SERVICES', and 'OPTIONS'. Under the 'GENERAL' tab, there are several settings:

- ☒ Set hostname: `raspberrypi` (dropdown menu shows `.local`)
- ☒ Set username and password:
 - Username: `pi`
 - Password: `••••••••`
- ☒ Configure wireless LAN:
 - SSID: `ME0-D32A40`
 - Password: `••••••••••`
 - ☐ Show password ☐ Hidden SSID
 - Wireless LAN country: `PT` (dropdown menu)
- ☒ Set locale settings:
 - Time zone: `Europe/Lisbon` (dropdown menu)
 - Keyboard layout: `US` (dropdown menu)

A red 'SAVE' button is located at the bottom center of the window.

11) Then, click on the SERVICES tab and enable SSH with password authentication.

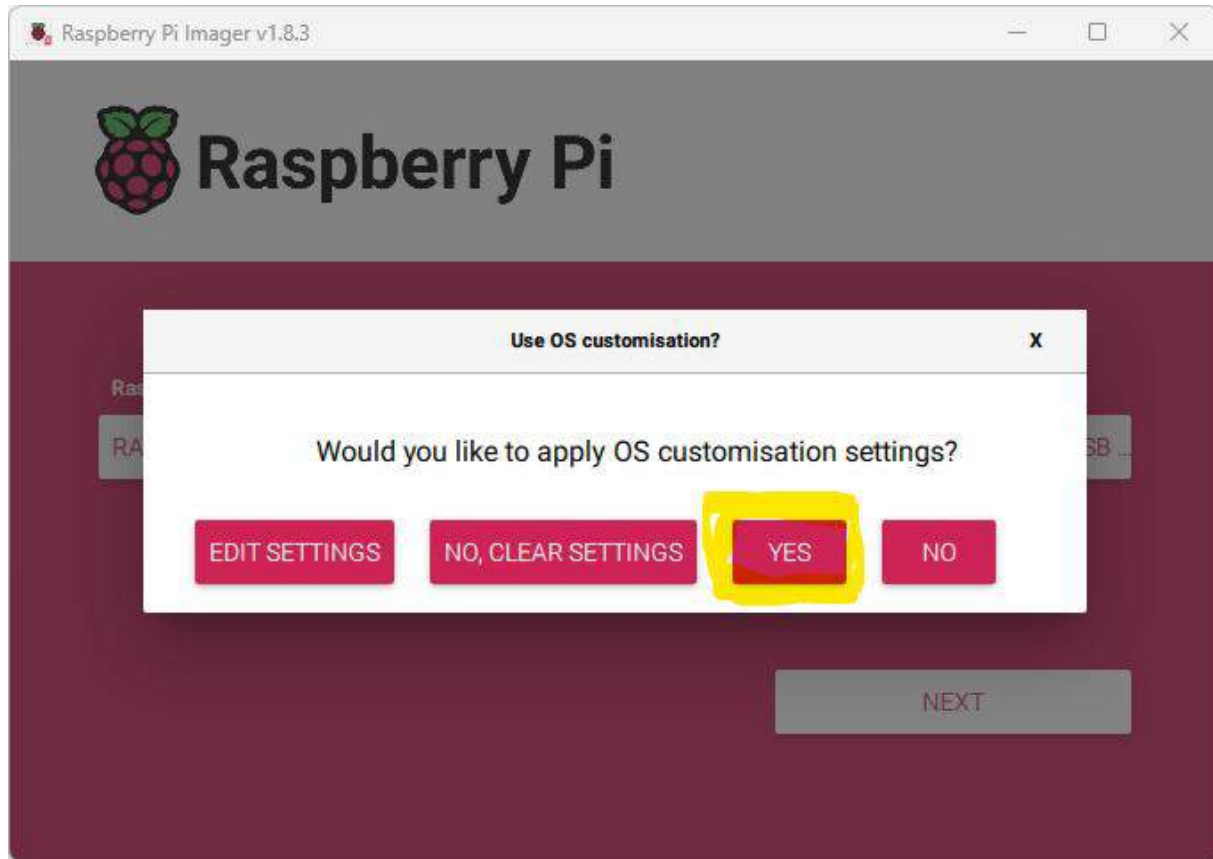


The screenshot shows the 'OS Customisation' window with the 'SERVICES' tab selected. The window has three tabs: 'GENERAL', 'SERVICES', and 'OPTIONS'. Under the 'SERVICES' tab, there are several settings:

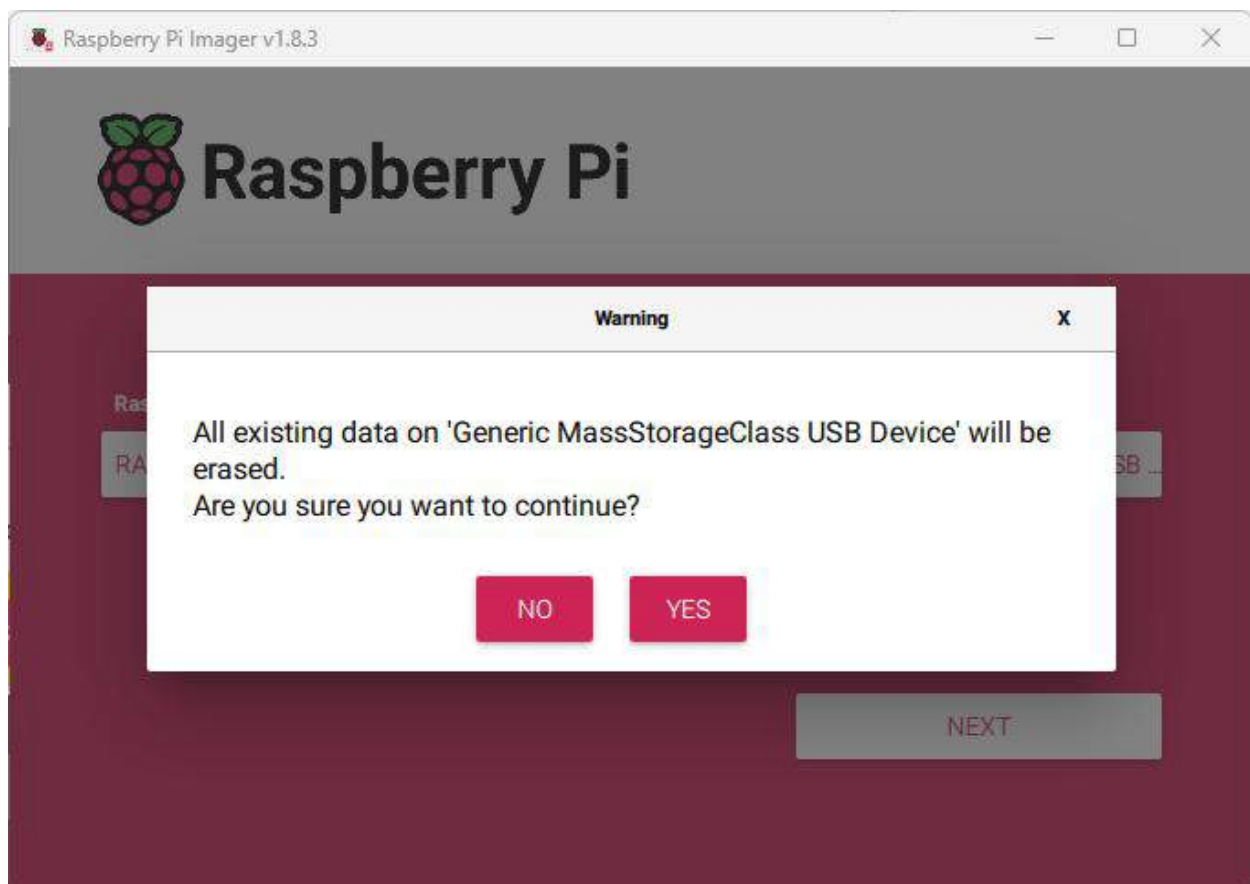
- ☒ Enable SSH:
 - ☒ Use password authentication
 - ☐ Allow public-key authentication only:
 - Set `authorized_keys` for `'pi'`:

A grey button labeled 'RUN SSH-KEYGEN' is located below the SSH settings. A red 'SAVE' button is located at the bottom center of the window.

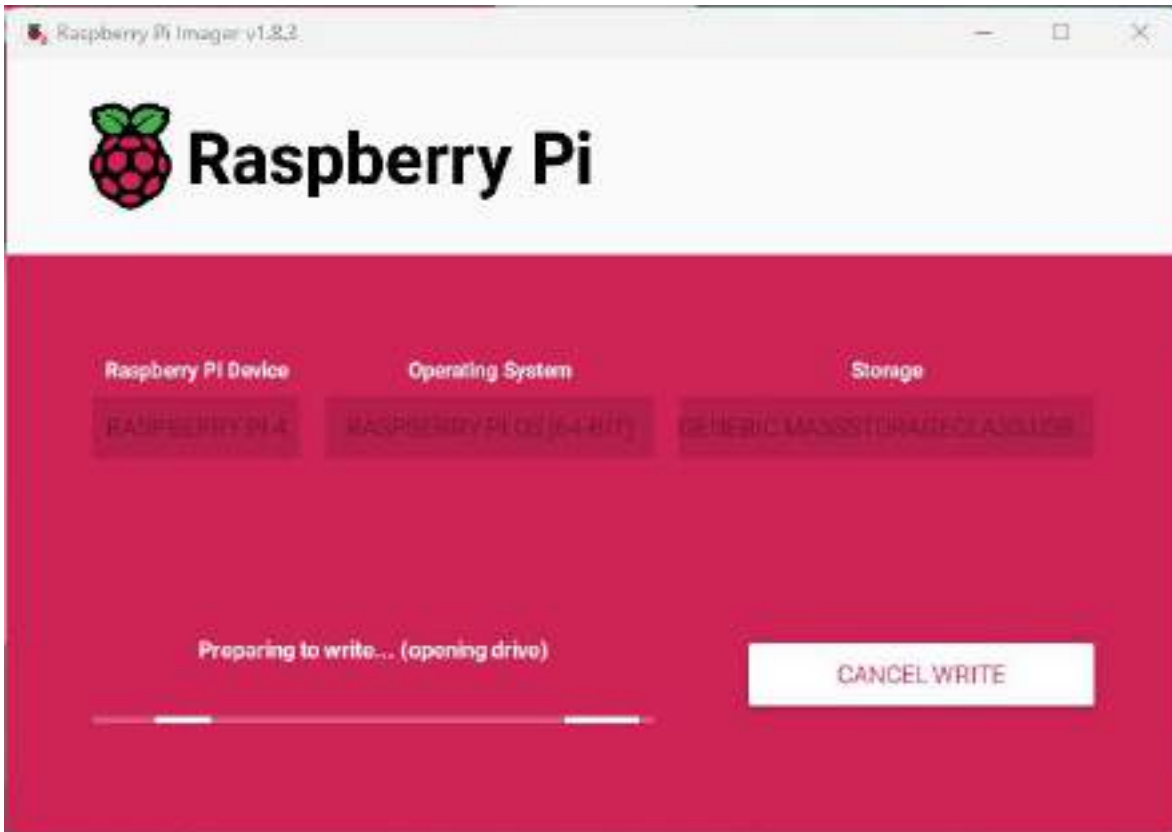
12) Click Save. You'll be asked if you want to apply the OS customisation settings. Click YES.



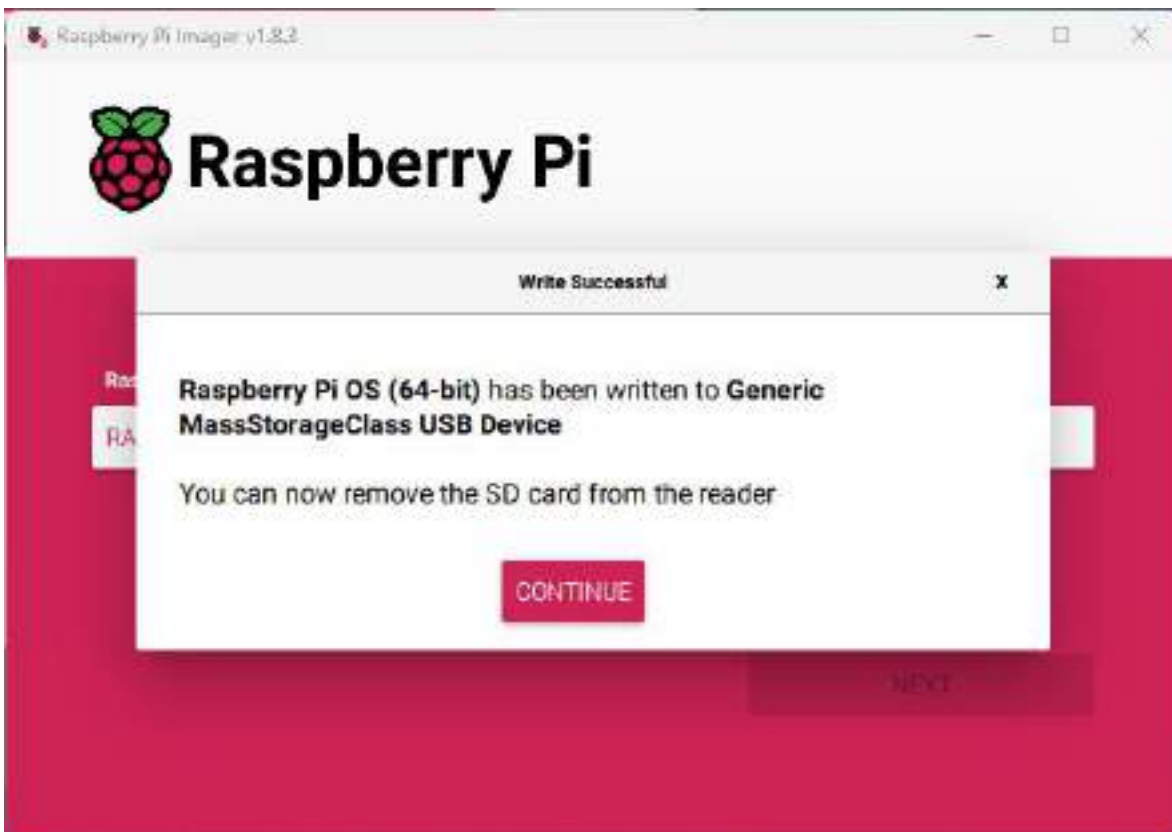
13) Finally, you'll be asked if you want to continue. Click YES to start burning the Raspberry Pi OS on the microSD card.



14) Wait a few minutes while it installs the Operating System.



15) When the installation is complete click on Continue. It will eject the microSD card safely.



15) Now, remove the card from your computer and insert it into your Raspberry Pi. Then, apply power to the Raspberry Pi to start it.

Viva - Voce Questions

1. What does ARM stand for?
2. Mention two advantages of ARM processors.
3. What is the difference between ARM and 8051 architectures?
4. What are the applications of ARM processors?
5. What is meant by Reduced Instruction Set Computing (RISC)?

RESULT :

The features of ARM processors and specifically Raspberry Pi controller is studied. The installation as PC is done. Raspberry Pi can be used for IoT applications.

EXP NO: 3.B	PYTHON PROGRAMMING IN RASPBERRY PI
DATE:	

Aim :

To develop Raspberry Pi platform and Python programming to implement LED blinking applications .

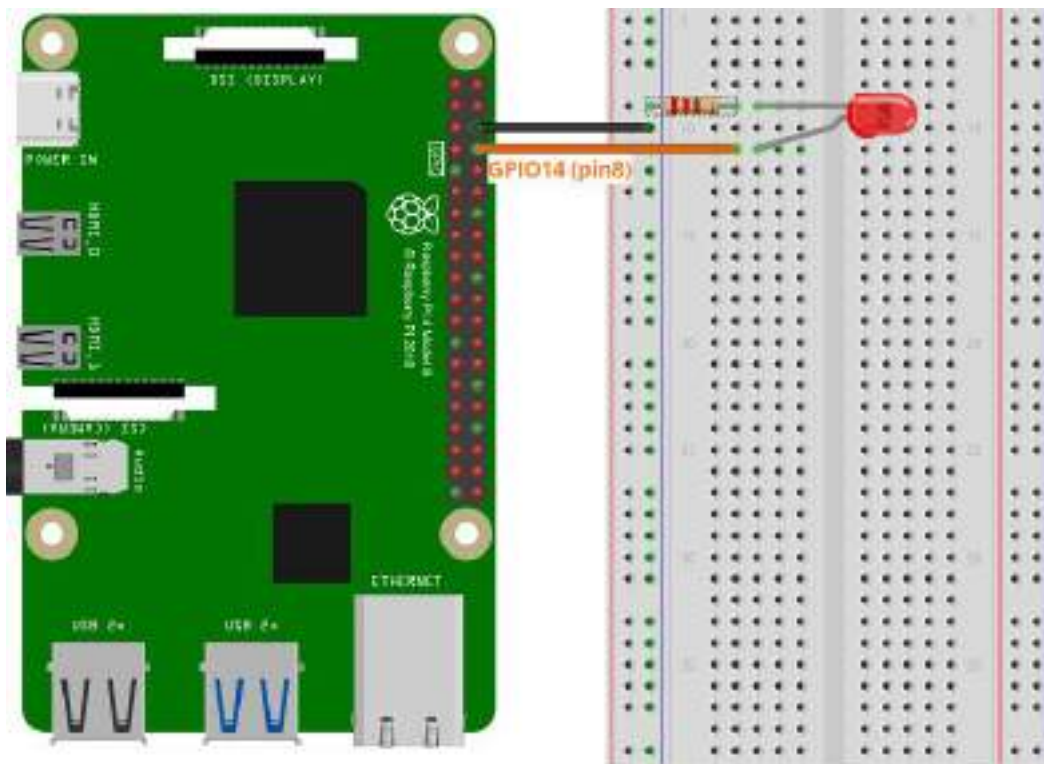
Hardware / Software Requirement:

Raspberry Pi board -1
 Breadboard -1
 LED -1
 Resistors -220 ohm -1
 Connecting wires.

Getting familiar with GPIOs:

	3V3	1	2	5V	
I2C SDA	GPIO2	3	4	5V	
I2C SCL	GPIO3	5	6	GND	
	GPIO4	7	8	GPIO14	UART TX
	GND	9	10	GPIO15	UART RX
	GPIO17	11	12	GPIO18	PCM CLK
	GPIO27	13	14	GND	PWM0
	GPIO22	15	16	GPIO23	
	3V3	17	18	GPIO24	
SPI MOSI	GPIO10	19	20	GND	
SPI MISO	GPIO9	21	22	GPIO25	
SPI SCLK	GPIO11	23	24	GPIO8	SPI CE0
	GND	25	26	GPIO7	SPI CE1
I2C EEPROM	GPIO0	27	28	GPIO1	I2C EEPROM
	GPIO5	29	30	GND	
	GPIO6	31	32	GPIO12	PWM0
PWM1	GPIO13	33	34	GND	
PWM1	PCMF5	GPIO19	35	GPIO16	
	GPIO26	37	38	GPIO20	PCM DIN
	GND	39	40	GPIO21	PCM DOUT

Connection diagram:

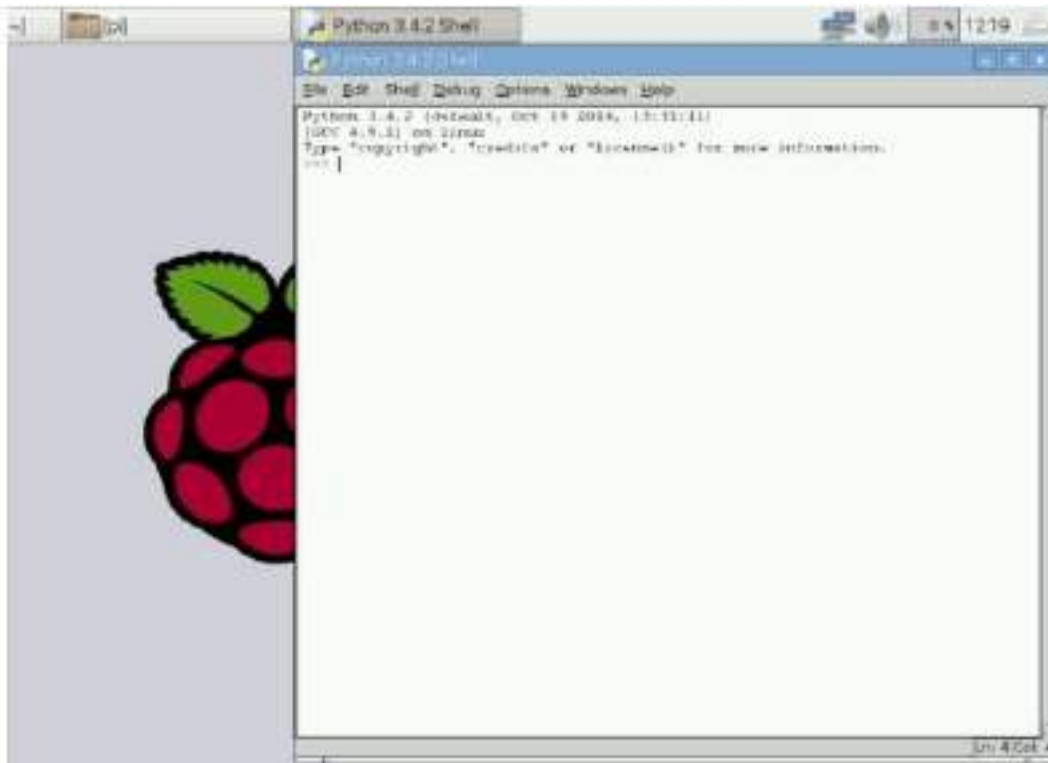


Procedure:

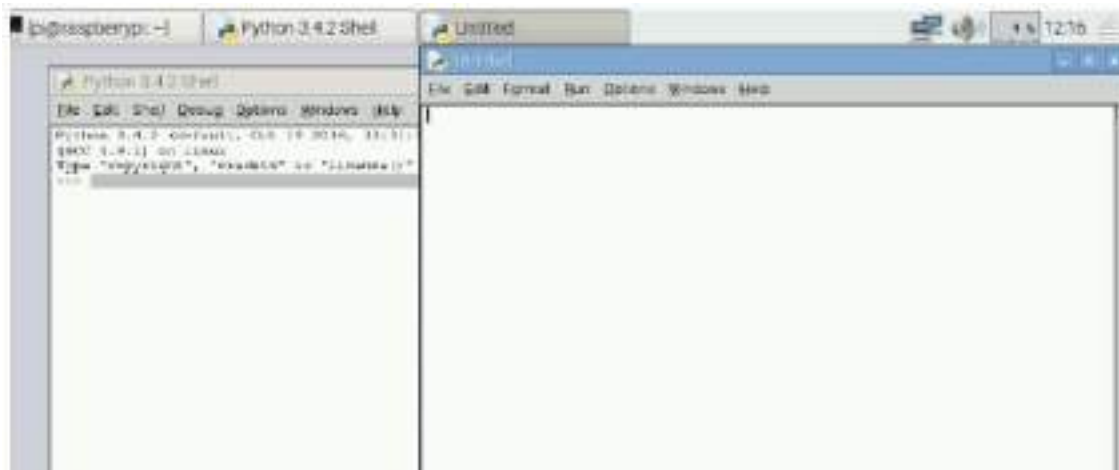
Step 1: On the desktop, go the Start Menu and choose for the PYTHON 3, as shown in figure below



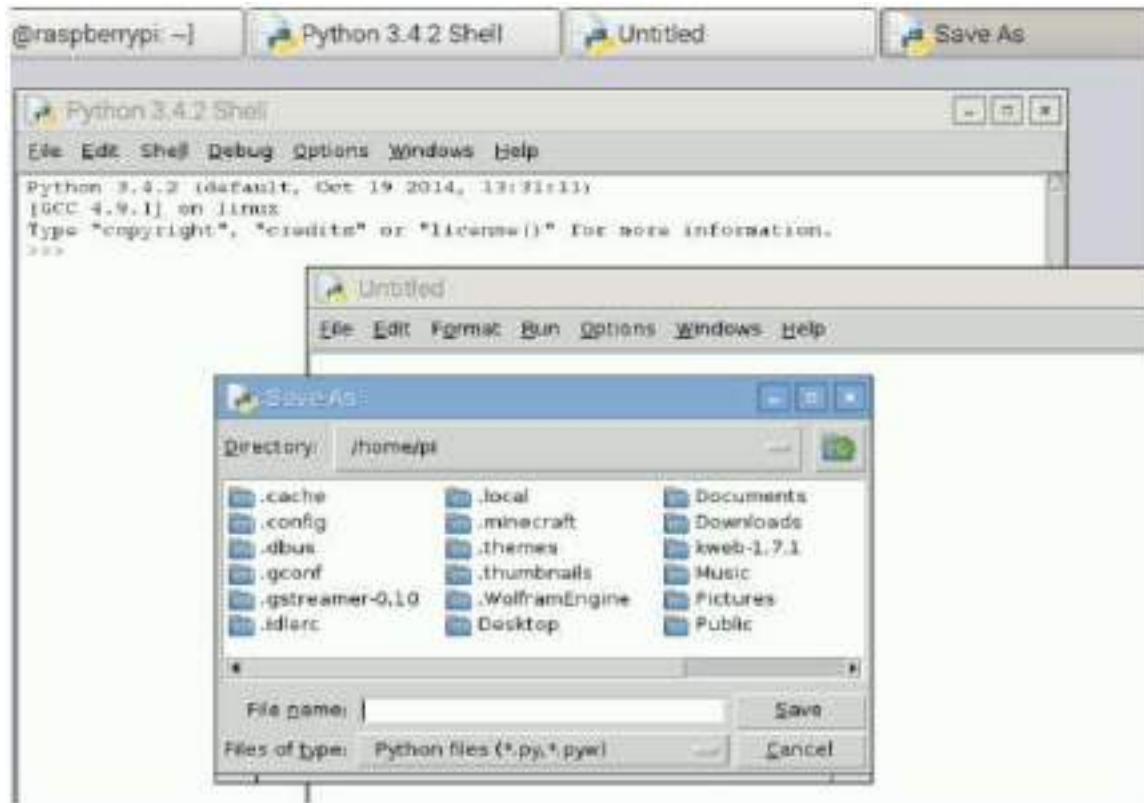
Step 2: After that, PYTHON will run and you will see a window as shown in below figure.



Step 3: After that, click on *New File* in *File Menu*, you will see a new window Open



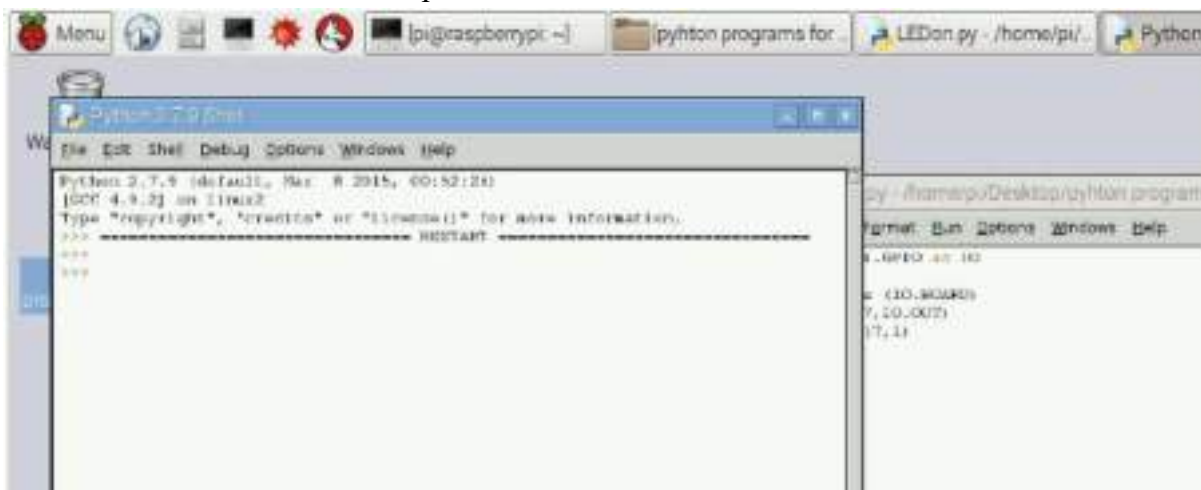
Step 4: Save this file as blinky on the desktop.



Step 5: Connect 5v of Raspberry Pi board with LED Anode through GPIO ports

Step 6: Connect the cathode of LED to ground of Raspberry Pi board.

Step 7: After that write the program for blinky as given below and execute the program by clicking on “RUN” on “DEBUG” option.



Step 6: Verify the LED blinking status with the respective time delay given by the program

To install the RPi.GPIO library for a Raspberry Pi 4, which allows Python to control the GPIO pins, the following steps are typically used:

1. Update package lists:

Code

```
sudo apt update
```

2. Install python3-dev (if not already installed):

This package provides the necessary development files for Python, which RPi.GPIO might require for compilation.

Code

```
sudo apt install python3-dev
```

3. Install RPi.GPIO:

The recommended way is to use pip, the Python package installer. Ensure you are using pip3 for Python 3.

Code

```
sudo pip3 install RPi.GPIO
```

4. Add your user to the gpio group (if not already done):

This grants your user account the necessary permissions to access the GPIO pins without requiring sudo for every script execution. Replace your_username with your actual username.

Code

```
sudo usermod -a -G gpio your_username
```

5. Reboot your Raspberry Pi:

This ensures that the changes to user groups and any newly installed packages are fully applied.

Code

```
sudo reboot
```

alternate method for GPIO library

install the Raspberry Pi GPIO Python module. This is a library that allows us to access the GPIO port directly from Python.

To install the Python library open a terminal and execute the following

```
$ sudo apt-get install python-rpi.gpio python3-rpi.gpio
```

Program:

```
import Rpi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
GPIO.setup(8,GPIO.OUT)
While True:
    GPIO.output(8,True)
    time.sleep(1)
    GPIO.output(40,False)
    time.sleep(1)
```

Alternate code:

```
import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library
from time import sleep # Import the sleep function from the time module
GPIO.setwarnings(False) # Ignore warning for now
GPIO.setmode(GPIO.BOARD) # Use physical pin numbering
GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW) # Set pin 8 as output pin and set initial value to low (off)
while True: # Run forever
    GPIO.output(8, GPIO.HIGH) # Turn on
    sleep(1) # Sleep for 1 second
    GPIO.output(8, GPIO.LOW) # Turn off
    sleep(1) # Sleep for 1 second
```

OUTPUT:

LED ON

LED OFF

Viva - Voce Questions

1. Why is Python widely used in Raspberry Pi?
2. How do you execute a Python program in Raspberry Pi?
3. What is GPIO in Raspberry Pi?
4. What is the role of the Python GPIO library?
5. Mention two applications of Raspberry Pi.

Result:

Thus, the development of Raspberry Pi platform and Python programming to implement LED blinking applications was executed successfully and the output was verified.

EXP NO: 3.C

DATE:

INTERFACING SENSORS WITH RASPBERRY PI

Aim :

To interface sensors with Raspberry Pi platform and Python programming to illustrate the operation of sensors.

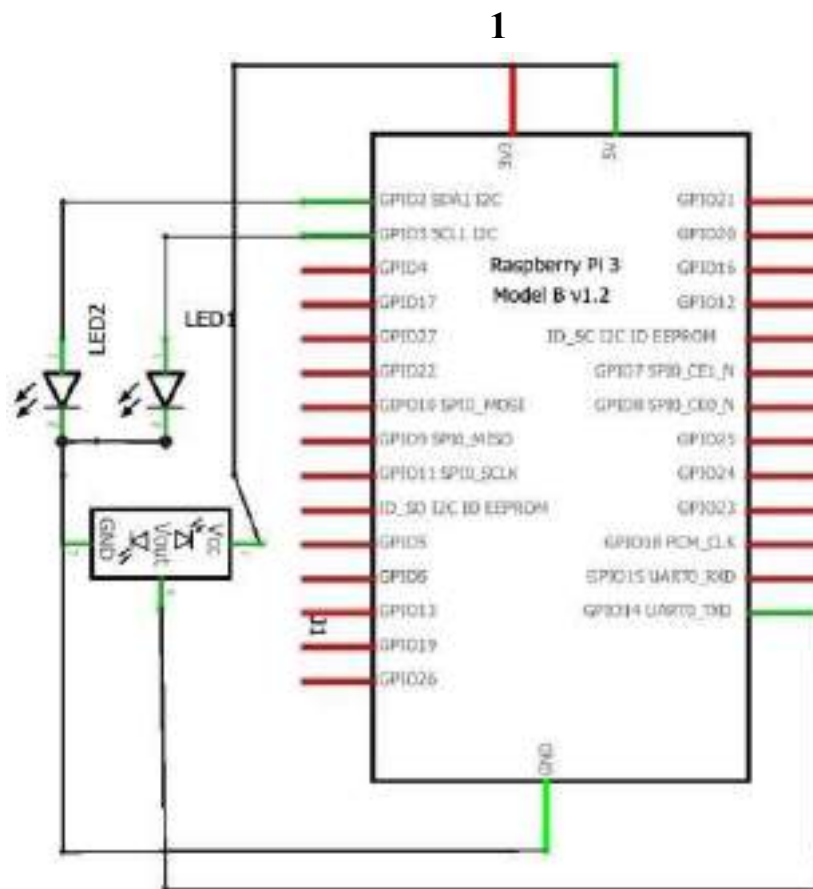
Hardware / Software Requirement:

Raspberry Pi board	-1
Breadboard	-1
LED Different colours	-2
IR Sensor or LDR Sensor	-1
Connecting wires.	

Getting familiar with GPIOs:



	3V3	1	2	5V	
I2C SDA	GPIO2	3	4	5V	
I2C SCL	GPIO3	5	6	GND	
	GPIO4	7	8	GPIO14	UART TX
	GND	9	10	GPIO15	UART RX
	GPIO17	11	12	GPIO18	PCM CLK
	GPIO27	13	14	GND	PWM0
	GPIO22	15	16	GPIO23	
	3V3	17	18	GPIO24	
		19	20	GND	
SPI MOSI	GPIO10	21	22	GPIO25	
SPI MISO	GPIO9	23	24	GPIO8	SPI CE0
SPI SCLK	GPIO11	25	26	GPIO7	SPI CE1
	GND	27	28	GPIO1	DC ID EEPROM
I2C ID EEPROM	GPIO0	29	30	GND	
	GPIO5	31	32	GPIO12	PWM0
PWM1	GPIO13	33	34	GND	
PWM1	GPIO19	35	36	GPIO16	
PCM FS	GPIO26	37	38	GPIO20	PCM DIN
	GND	39	40	GPIO21	PCM DOUT



Theory: IR Sensor Module

IR (Infrared) Sensor IR (Infrared) Sensor works by emitting infrared signal/radiation and receiving of the signal when the signal bounces back from any obstacle. In other words, the IR Sensor works by continuously sending signal (in a direction) and continuously receive signal, if comes back by bouncing on any obstacle in the way.

Components:

IR Sensor Emitter: This component continuously emits the infrared signal Receiver: It waits for the signal which is bounced back by obstacle

Indicator: On board LED to signal if obstacle is deducted by the sensor Output:

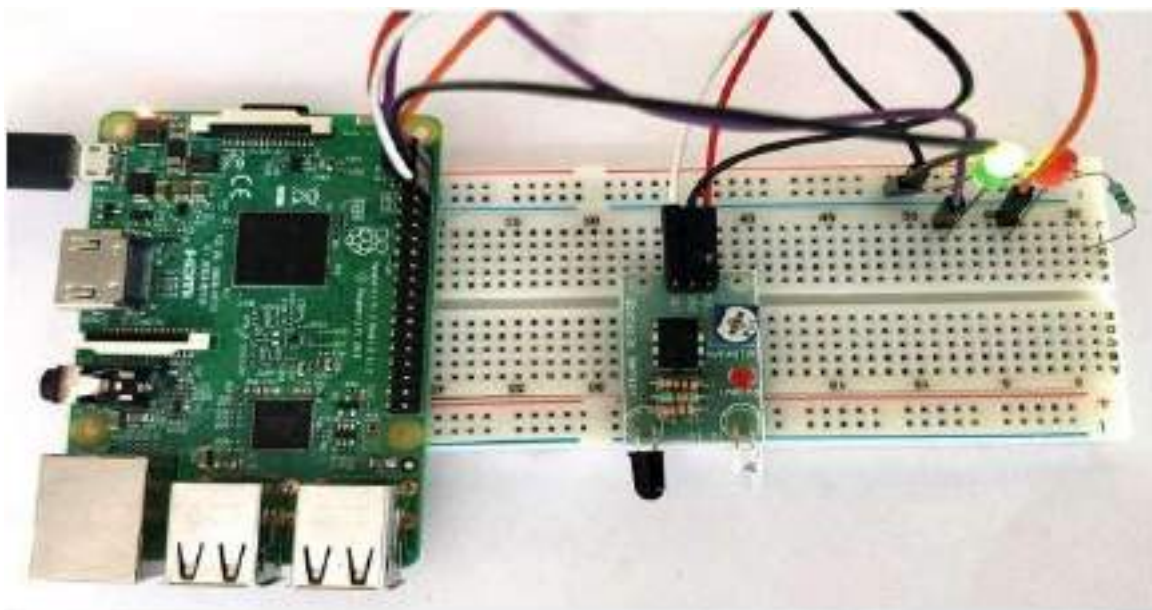
Could be used as Input for further processing of the signal Ground: Ground/Negative point of the circuit

Voltage: Input 3.3V In this tutorial we will learn how we can Interface an IR sensor with Raspberry pi.



These sensors are most commonly use in small robots like line follower robot, Edge avoiding robot etc.. Simply putting, it can detect the presence of objects before it and also differentiate between white and black colour. So lets learn how to interface this sensor with Raspberry Pi. In this experiment, when there is no object in front of IR sensor then the Red LED remains turned on and soon as we put something in front of IR sensor then red LED turns off and Green LED turn on. This circuit can also serve as Security Alarm Circuit.

IR Sensor Module: IR sensors (Infrared sensor) are modules which detect the presence of objects before them. If the object is present it give 3.3V as output and if it is not present it gives 0 volt. This is made possible by using a pair of IR pair (transmitter and receiver), the transmitter (IR LED) will emit an IR ray which will get reflected if there is a object present before it. This IR ray will be received back by the receiver (Photodiode) and the output will be made high after amplified using an op-amp link LM358 The IR Sensor used in this assignment is like all IR sensor it has three pins which are 5V, Gnd and Out respectively. The module is powered by the 5V pin from Raspberry Pi and the out pin is connected to GPIO14 of Raspberry Pi. The potentiometer on top of the module can be used to adjust the range of



the IR sensor.

Circuit Diagram and Explanation:

The circuit diagram for connecting Raspberry Pi with IR sensor is shown below.

As you can see the circuit diagram is very simple. We have directly powered the IR module from the 5V and Ground Pin of Raspberry Pi. The output pin of the IR module is connected to the GPIO14. We have also used two LED (Green and Red) to indicate the status of the object. These two LEDs are connected to GPIO3 and GPIO2 respectively. Since the GPIO pins of Raspberry Pi are 3.3V, a current limiting resistor is not mandatory. However if desired a resistor of value 470 ohms can be added between the ground pin of LEDs and Raspberry Pi. The whole circuit is powered by a 5V mobile charger through the micro USB port of the Raspberry pi.

PROGRAM:

```
import RPi.GPIO as IO
```

```
import time IO.setwarnings(False)
```

```
IO.setmode(IO.BCM)
```

```
IO.setup(2,IO.OUT) #GPIO 2 -> Red LED as output
```

```
IO.setup(3,IO.OUT) #GPIO 3 -> Green LED as output
```

```
IO.setup(14,IO.IN) #GPIO 14 -> IR sensor as input
```

```
while 1:
```

```
    if(IO.input(14)==True): #object is far away
```

```
        IO.output(2,True) #Red led ON
```

```
    IO.output(3,False) # Green led OFF
```

```
    if(IO.input(14)==False): #object is near
```

```
        IO.output(3,True) #Green led ON
```

```
    IO.output(2,False) # Red led OFF
```

Procedure:

- Step 1: Connections are made as per the circuit diagram.
- Step 2: On the desktop, go the Start Menu and choose for the PYTHON 3
- Step 3: After that, PYTHON will run and you will see a programming window Step 4: Click on *New File* in *File Menu*, you will see a new window
- Step 5: After that write the program for IR sensor module and execute the program by clicking on “RUN” on “DEBUG” option.
- Step 6: Verify the LED’s blinking status with respect to the program.

VIVA -VOCE QUESTIONS

1. Why are sensors interfaced with Raspberry Pi?
2. What is the difference between digital and analog sensors?
3. Why is an ADC required for analog sensors?
4. Which communication protocols are commonly used for sensor interfacing?
5. Mention any two sensors commonly used with Raspberry Pi.

Result:

Thus, the interfacing sensor with Raspberry Pi platform was executed and the output was verified successfully.

EXP NO: 4	IMPLEMENTATION OF IOT USING BLYNK
DATE:	

AIM:

To implement IoT using Blynk and demonstrate blinking of LED.

HARDWARE & SOFTWARE TOOLS REQUIRED:

S.No.	Software Requirements	Quantity
1	Blynk APP	1
2	ESP8266 NODE MCU	1

ABOUT BLYNK:

Blynk is an IoT platform with customizable mobile apps, private cloud, rules engine, and device management analytics dashboard, designed for easy and customizable Internet of Things applications. Designing dashboard on Blynk App for IoT projects is really easy, you just have to organize buttons, sliders, graphs, and alternative widgets onto the screen. We can also edit the widgets as per our requirements.

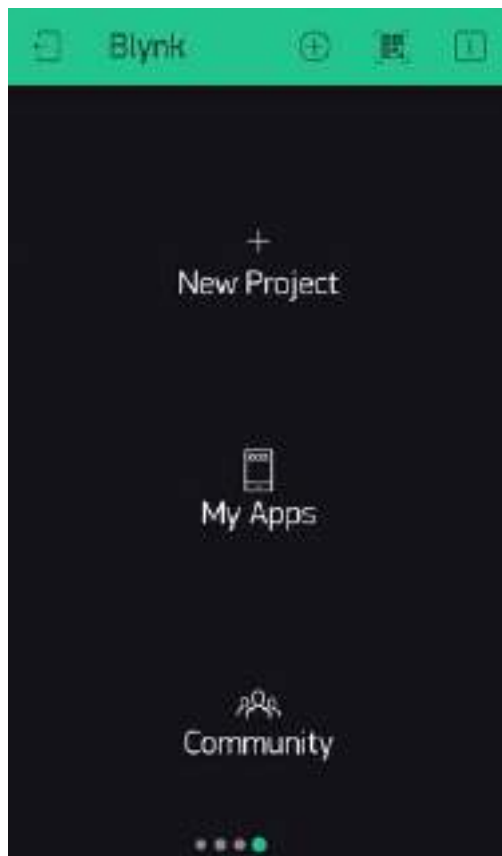
With the help of Blynk, the software side gets easier than the hardware. Blynk is perfect for interfacing with simple projects like monitoring the temperature of your room or turning lights on and off remotely. Here, in this project we are controlling a LED using Blynk App and Esp8266. Earlier we have controlled the LED using ESP32. So, not only with ESP8266, it's easy to interface Blynk app with Raspberry Pi, Arduino and other microcontrollers.

PROCEDURE:**Blynk App Dashboard Setup for Controlling LED**

To control LED with Blynk app, download and install the Blynk app from Google or Apple app store.

Then, if you are new to Blynk app, create a new account by using your Email and password.

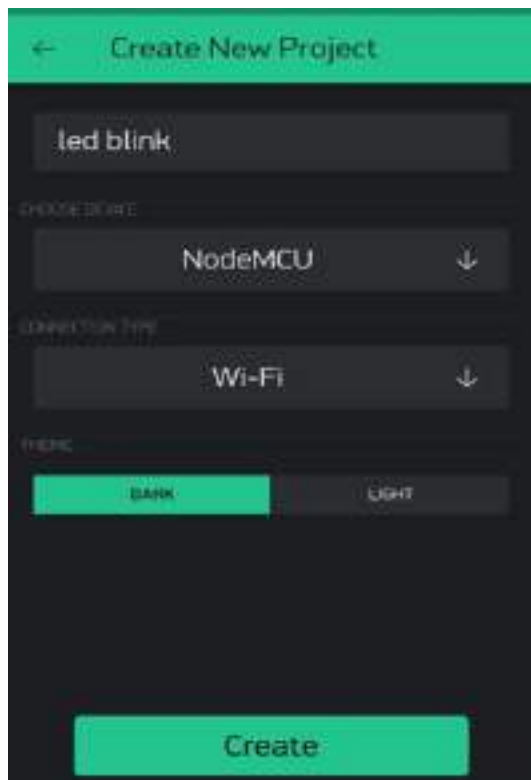
After sign up click on 'New Project' to start your project.



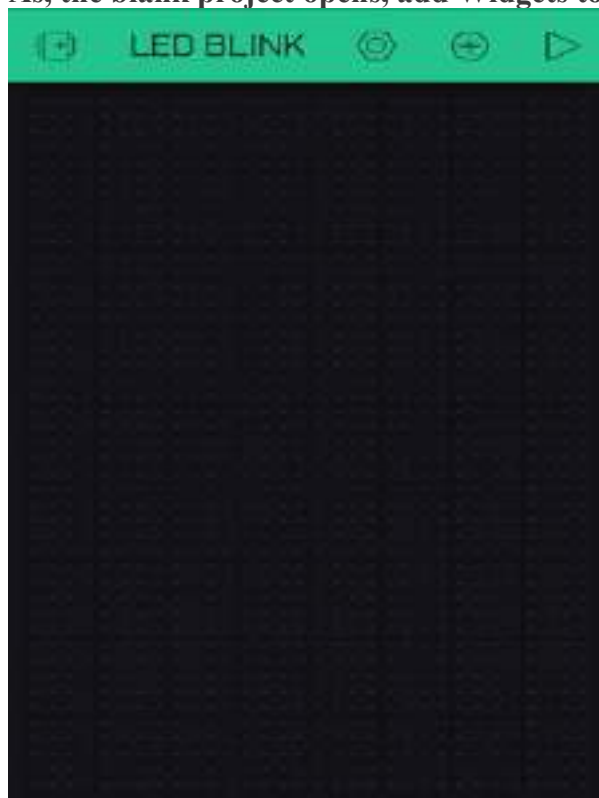
Now provide your project a name and as blynk app works with different hardware models. Thus opt for your device from *choices*.

Here, we are using ESP8266 node MCU, so I am proceeding with node MCU. While selecting your board opt for your association sort whether or not it's Wi-Fi or LAN or USB association.

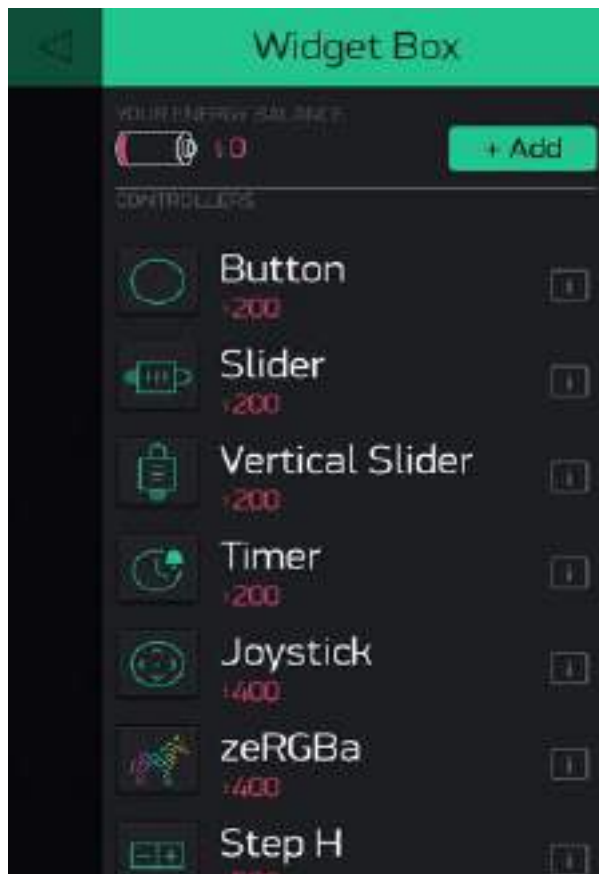
After these steps click on 'Create' button to form your project.



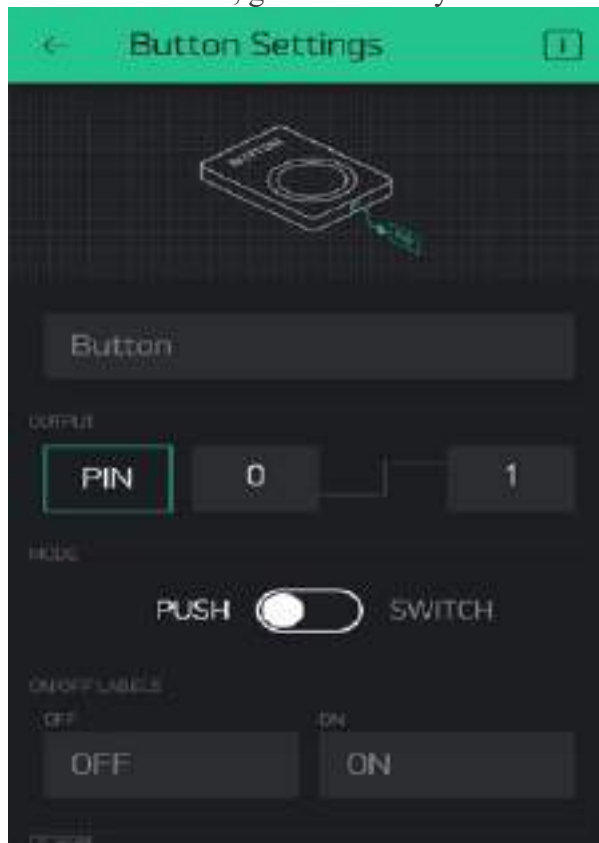
As, the blank project opens, add Widgets to it by clicking on Add button (Plus sign button)



Now after this click on 'Button' to add a button in your project.

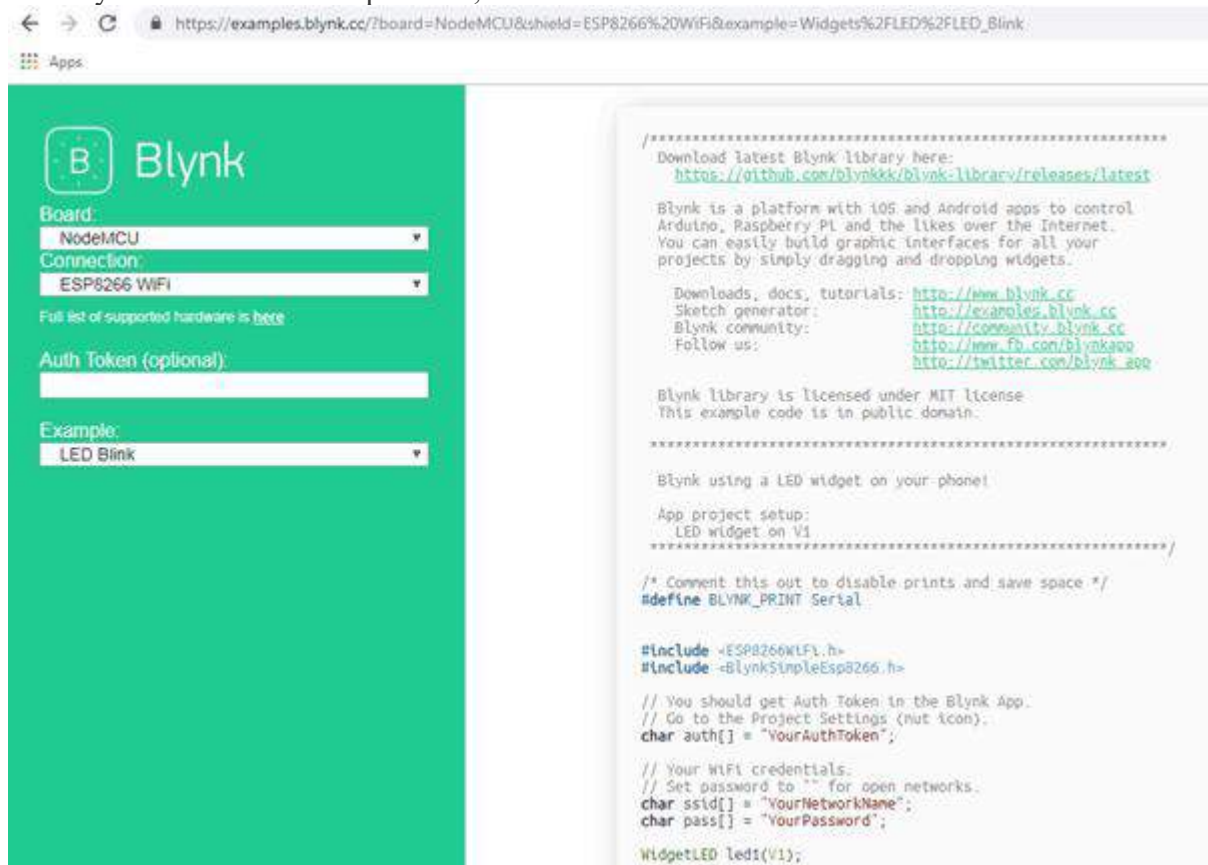


Now in button settings provide a name to your button. After assign the pin number to the 'OUTPUT'. Also, give names to your On/Off labels.



NODE MCU Setup for Blynk App

In your browser search Blynk code generator then open Blynk example browser, Select your board and example code, as shown below:



The screenshot shows the Blynk code generator interface. On the left, a green sidebar contains the Blynk logo, a 'Board' dropdown set to 'NodeMCU', a 'Connection' dropdown set to 'ESP8266 WiFi', an 'Auth Token (optional)' input field, and an 'Example' dropdown set to 'LED Blink'. On the right, the generated Arduino code is displayed in a light gray box. The code includes comments for downloading the Blynk library, a list of links for downloads, docs, tutorials, and community resources, and the actual Arduino sketch for controlling an LED via Blynk. The sketch includes necessary headers, defines the Blynk pin, and sets up the LED widget.

```

Download latest Blynk library here:
https://github.com/blynkkk/blynk-library/releases/latest

Blynk is a platform with iOS and Android apps to control
Arduino, Raspberry PI and the likes over the Internet.
You can easily build graphic interfaces for all your
projects by simply dragging and dropping widgets.

Downloads, docs, tutorials: http://www.blynk.cc
Sketch generator: http://examples.blynk.cc
Blynk community: http://community.blynk.cc
Follow us: http://www.fb.com/blynkapp
http://twitter.com/blynk\_app

Blynk library is licensed under MIT license
This example code is in public domain.

Blynk using a LED widget on your phone!

App project setup:
LED widget on V1

/* Comment this out to disable prints and save space */
#define BLYNK_PRINT Serial

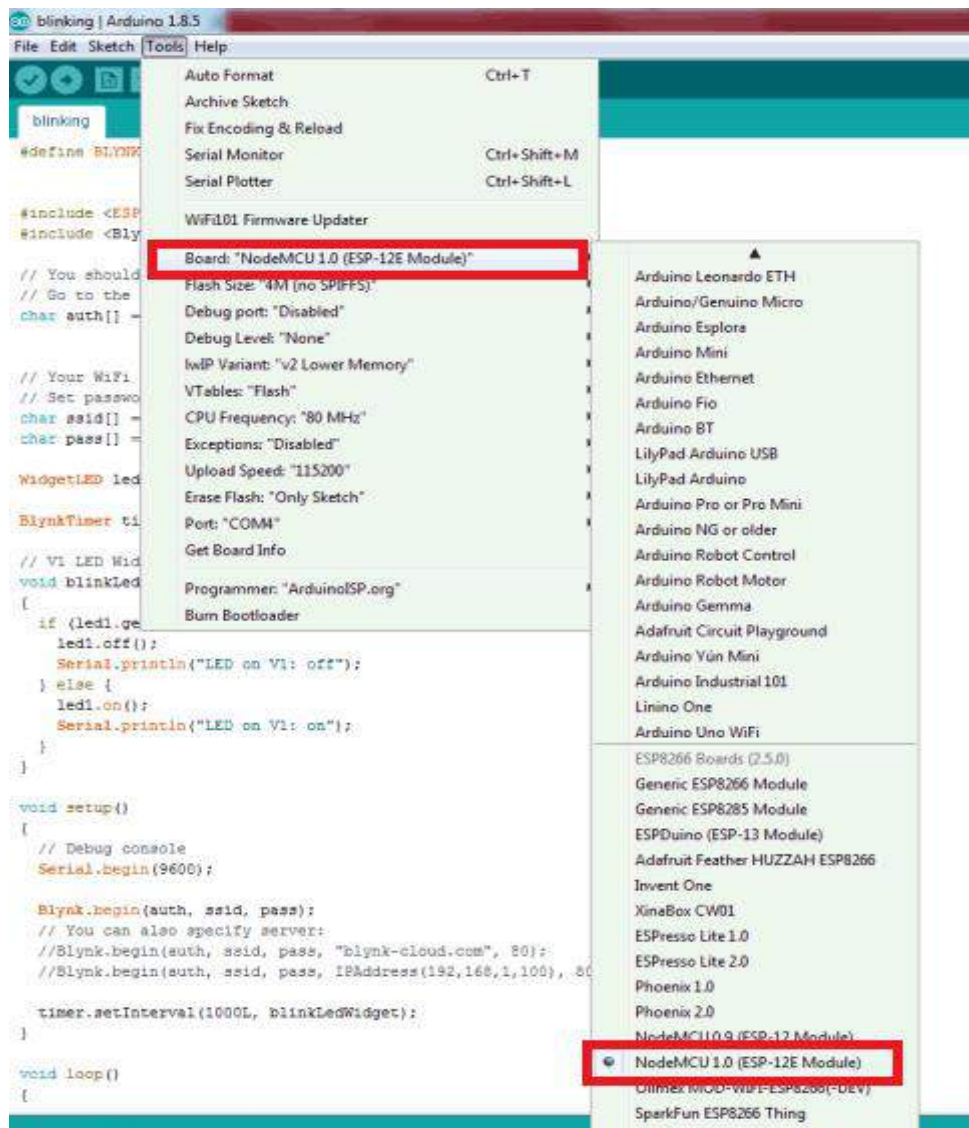
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>

// You should get Auth Token in the Blynk App.
// Go to the Project Settings (Nut icon).
char auth[] = "YourAuthToken";

// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "YourNetworkName";
char pass[] = "YourPassword";

WidgetLED led1(V1);
  
```

Open Arduino IDE and select Tools option as shown below:



Note: You should have ESP8266 library installed in Arduino IDE.

Then program in Arduino IDE.

PROGRAM

```
#define BLYNK_PRINT Serial
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
// You should get Auth Token in the Blynk App.
// Go to the Project Settings (nut icon).
char auth[ ] = "YourAuthToken";
// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[ ] = "YourNetworkName";
char pass[ ] = "YourPassword";
WidgetLED led1(V1);
BlynkTimer timer;
// V1 LED Widget is blinking
void blinkLedWidget() // function for switching off and on LED
```

```
{
  if (led1.getValue()) {
    led1.off();
    Serial.println("LED on V1: off");
  } else {
    led1.on();
    Serial.println("LED on V1: on");
  }
}
void setup()
{
  Serial.begin(9600);
  Blynk.begin(auth, ssid, pass);
  timer.setInterval(1000L, blinkLedWidget);
}
//In the loop function include Blynk.run() command.
void loop()
{
  Blynk.run();
  timer.run();
}
```

VIVA-VOCE QUESTIONS

1. What is Blynk?
2. Which communication protocol is commonly used by Blynk?
3. What is an authentication token in Blynk?
4. Name two widgets available in the Blynk application.
5. Mention two applications of Blynk.

RESULT:

Thus, IoT using ESP8266 and Blynk App dashboard designed successfully and controlled the LED through Blynk app .

EXP NO: 5	Monitoring of Temperature Using Sensor and Softcore Processor
DATE:	

AIM:

To interface a temperature sensor (LM35 / DS18B20 / PT100) with a softcore processor implemented on an FPGA development board, acquire temperature data through the onboard ADC, process it digitally, and display the measured temperature in degrees Celsius on a seven-segment display or serial terminal.

Theory**Temperature Sensors**

Temperature sensors convert thermal energy into an electrical signal. Common types used in embedded systems are:

LM35 — Analog output, 10 mV/°C sensitivity, range –5 °C to +150 °C. No external calibration required.

DS18B20 — Digital (1-Wire protocol), 9–12-bit resolution, range –55 °C to +125 °C. Each device has a unique 64-bit ROM ID.

PT100 / PT1000 — RTD (Resistance Temperature Detector). Resistance increases linearly with temperature. Requires a Wheatstone bridge and signal conditioning circuit.

For this experiment the LM35 is preferred because its output voltage ($V_{out} = 10 \text{ mV} \times T^{\circ}\text{C}$) can be fed directly to the FPGA evaluation board's ADC channel without additional amplification.

Analog-to-Digital Converter (ADC)

The ADC samples the analog voltage from the sensor and produces a binary value proportional to the input. For an n-bit ADC with reference voltage V_{ref}

$$\text{ADC Output (counts)} = (V_{out} / V_{ref} \times (2^n - 1))$$

Rearranging to find temperature:

$$T (^{\circ}\text{C}) = [\text{ADC Count} \times V_{ref} / (2^n - 1)] \div 0.01$$

$$\text{For a 12-bit ADC and } V_{ref} = 3.3 \text{ V: } T (^{\circ}\text{C}) = \text{ADC Count} \times 3.3 / 4095 / 0.01$$

Softcore Processors on FPGA

A softcore processor is a processor implemented in configurable FPGA logic (LUTs, FFs, DSPs) rather than dedicated silicon. Popular softcore processors include:

MicroBlaze (Xilinx / AMD) — 32-bit RISC, configurable pipeline, AXI4 bus interface.

Nios II (Intel / Altera) — 32-bit RISC, Avalon bus, configurable caches and multiply/divide.

PicoRV32 (open-source) — compact RISC-V (RV32IMC) core, suitable for resource-constrained

designs.

The processor communicates with peripherals (ADC, UART, GPIO, timers) through a bus fabric (AXI / Avalon). The hardware description is generated using a graphical IP integrator tool (Vivado Block Design / Platform Designer), then synthesised and programmed onto the FPGA. Software is developed in C using the provided BSP (Board Support Package) and cross-compiled with the vendor toolchain or GCC.

Apparatus / Software Required

S.No.	Component / Equipment	Quantity	Specification / Part No.
1	FPGA Development Board	1	Xilinx Nexys A7 / Basys 3 / DE10-Lite
2	Temperature Sensor	1	LM35DZ (Analog, TO-92 package)
3	Breadboard	1	830-point solderless
4	Jumper Wires	10	Male-to-female, 20 cm
5	USB-UART Cable / On-board UART	1	CP2102 / FT232 (if external)
6	Regulated Power Supply / USB	1	5V / 3.3V via FPGA board
7	PC with Vivado / Quartus IDE	1	Version 2022.x or later
8	Serial Terminal (PuTTY / Tera Term)	1	115200 baud, 8N1
9	Multimeter	1	Digital, for verification
10	Reference thermometer	1	Calibrated digital thermometer

Circuit / Pin Connections

Connect the LM35 sensor to the FPGA board as follows. Refer to the board's schematic for ADC input pins (XADC on Xilinx; external SPI ADC on Intel boards).

LM35 Pin	LM35 Signal	FPGA Board Connection	Remarks
Pin 1 (Left)	VCC	3.3 V rail on board	Do NOT use 5 V — damages XADC
Pin 2 (Middle)	V _{out} (Analog Out)	XADC_VP / ADC_IN0	Via 100 Ω series resistor
Pin 3 (Right)	GND	GND rail on board	Common ground

Hardware Setup Procedure

FPGA IP/Hardware Design

- Open Vivado / Quartus Prime and create a new project targeting your board.
- Launch the IP Integrator (Vivado Block Design) or Platform Designer (Quartus).
- Add the softcore processor IP (MicroBlaze or Nios II) to the design.
- Add and configure the following IPs / modules:
- AXI UARTLITE (or UART 16550) — set baud rate to 115200.
- XADC Wizard (Xilinx) or ADC controller peripheral — enable single-channel unipolar mode on the VP/VN or dedicated Aux channel connected to LM35.
- AXI GPIO — for optional LED status indicators.
- AXI Timer — for sampling interval control.
- Run Connection Automation / Auto-Connect to wire resets and clocks.
- Validate the design, generate HDL wrapper, and run Synthesis, Implementation, and Bitstream Generation.
- Program the FPGA with the generated bitstream (.bit file).

Software / BSP Setup

- Launch Vitis / Eclipse and create a new application project linked to the hardware XSA file.
- Generate the Board Support Package (BSP) — this creates drivers for XADC, UART, and other peripherals.
- Create a new C application; copy the provided source code (Section 7) into main.c.
- Build the project; verify there are no compilation errors.
- Run / Debug the application on the connected FPGA target.

Program / Source Code

The following C program reads the XADC raw count, converts it to temperature, and sends the result over UART every one second.

```

/* Temperature Monitoring – MicroBlaze / Nios II
   File: main.c
   Description: Read LM35 via XADC, convert to Celsius, transmit over UART
*/

#include "xparameters.h"

#include "xsysmon.h"
/* XADC driver */

#include "xuartlite.h"
/* UART driver */
#include "xil_printf.h"
#include "sleep.h"

/* ADC reference voltage and resolution */
#define VREF_MV    3300.0f /* 3.3 V in millivolts */

```

```

#define ADC_BITS    12
#define ADC_MAX     ((1 << ADC_BITS) - 1) /* 4095 */
#define LM35_MV_PER_C 10.0f /* 10 mV per degree Celsius */

XSysMon XAdcInst;
XUartLite UartInst;

int main(void)
{
    u32 raw_adc;
    float v_mv, temp_c;
    char buf[64];

    /* Initialize XADC */
    XSysMon_Config *cfg = XSysMon_LookupConfig(XPAR_SYSMON_0_DEVICE_ID);
    XSysMon_CfgInitialize(&XAdcInst, cfg, cfg->BaseAddress);

    /* Initialize UART */
    XUartLite_Initialize(&UartInst, XPAR_UARTLITE_0_DEVICE_ID);

    xil_printf("\r\n=== Temperature Monitor Started ===\r\n");

    while (1)
    {
        /* Read raw ADC value from XADC dedicated channel */
        raw_adc = XSysMon_GetAdcData(&XAdcInst, XSM_CH_VPVN);
        raw_adc = raw_adc >> 4; /* Right-shift: XADC stores in upper 12 bits */

        /* Convert to millivolts */
        v_mv = ((float)raw_adc / ADC_MAX) * VREF_MV;

        /* Convert to Celsius (LM35: 10 mV/degC) */
        temp_c = v_mv / LM35_MV_PER_C;

        /* Send result via UART */
        xil_printf("ADC=%4lu Voltage=%5.2f mV Temp=%5.2f C\r\n",
            (unsigned long)raw_adc, v_mv, temp_c);

        sleep(1); /* 1-second sampling interval */
    }
    return 0;
}

```

Experimental Procedure

1. Assemble the circuit as per the connection table. Double-check polarity of the LM35.
2. Connect the FPGA board to the PC via USB programming cable and USB-UART cable.
3. Program the FPGA bitstream using Vivado Hardware Manager.
4. Open a serial terminal (PuTTY) at 115200 baud, 8 data bits, no parity, 1 stop bit.
5. Download and run the embedded C application from Vitis IDE.

6. Observe temperature readings on the terminal. Let the system stabilise for 30 seconds.
7. Using a calibrated thermometer, measure the ambient temperature near the sensor.
8. Apply gentle heat (warm hand, hair dryer at safe distance) and record the response.
9. Record ADC count, computed voltage, computed temperature, and reference temperature in the observation table.
10. Remove heat source and observe recovery behaviour.
11. Repeat for at least 8 data points spanning a temperature range of approximately 10–15 °C.

Observation Table

Vref = 3.3 V | ADC Resolution = 12 bits | LM35 Sensitivity = 10 mV/°C

S.No.	Time (s)	ADC Output (counts)	Temperature (°C)	Observations
1				
2				
3				
4				
5				
6				
7				
8				

VIVA VOCE QUESTIONS

1. What is a softcore processor?
2. Which temperature sensor is used in this experiment?
3. What is the role of FPGA in implementing a softcore processor?
4. Why is temperature monitoring important in embedded systems?
5. Mention two advantages of softcore processors.

Result

The temperature monitoring system using the LM35 sensor and softcore processor was successfully implemented on the FPGA. The system continuously reads the analog temperature data, converts it through the onboard ADC, processes the digital output using the embedded software running on the softcore processor, and transmits the result over UART.

EXP NO: 6	STUDY OF MICROCONTROLLERS USED IN IOT
DATE:	

AIM:

To study the features, role and selection of microcontrollers in IoT systems and study working of sensor interfacing .

Apparatus / Software Required

- Arduino Uno or Mega board
- ESP8266 or ESP32 Wi-Fi microcontroller board
- STM32 Nucleo or Discovery board
- Raspberry Pi (for comparison)
- USB cables, breadboard, jumper wires, LEDs, sensors (e.g., DHT11/22)
- Computer with Arduino IDE, STM32CubeIDE, Thonny (for Raspberry Pi Pico), or other relevant IDEs

Theory**Microcontroller**

A microcontroller is a compact integrated circuit designed to govern a specific operation in an embedded system. It includes a processor core, memory, and programmable input/output peripherals.

Microcontrollers for IoT

Microcontrollers are ideal for IoT because they are small, low-power, and can interface with sensors, actuators, and communication modules to collect, process, and transmit data.

Common Microcontrollers in IoT:

- **Arduino Series:** Widely used for prototyping and education due to ease of use and community support.
- **ESP8266/ESP32:** Popular for built-in Wi-Fi/Bluetooth and low power consumption, ideal for wireless IoT devices.
- **STM32 Series:** High performance, low power, and rich connectivity options, used in industrial and commercial IoT.
- **Raspberry Pi:** A single-board computer with microcontroller capabilities, suitable for complex IoT systems and edge computing.
- **ATtiny:** Affordable and compact, good for simple IoT tasks.
- **Texas Instruments MSP430/Tiva C:** Known for low power and high integration, used in sensing and control.

- Infineon XMC/PSoC: Designed for IoT with a focus on energy efficiency and connectivity.

Procedure

Literature and Datasheet Study

1. List specifications (CPU speed, memory, power consumption, communication interfaces) for Arduino, ESP8266/ESP32, STM32, and Raspberry Pi using datasheets and online resources.
2. Compare features such as wireless connectivity, analog/digital I/O, and suitability for different IoT applications.

Hands-On Exploration

1. Setup the Development Environment:
 - Install Arduino IDE for Arduino/ESP boards.
 - Install STM32CubeIDE for STM32 boards.
 - Install Thonny or Raspberry Pi OS for Raspberry Pi.
2. Connect a simple sensor (e.g., DHT11/22 temperature and humidity sensor) to each board.
3. Write and upload a basic program to read sensor data and display it via serial monitor or onboard LEDs.
 - For ESP8266/ESP32, optionally send data to a cloud service or local web server.
4. Observe and record the ease of programming, speed of response, and power consumption (if possible).

Discussion and Comparison

1. Discuss the strengths and weaknesses of each microcontroller in terms of:
 - Ease of use
 - Connectivity (Wi-Fi/Bluetooth)
 - Power efficiency
 - Community and library support
 - Suitability for various IoT applications (smart home, health, industrial, etc.)
2. Summarize findings in a comparison table.

Observation Table

Board/MCU	CPU Speed	Memory	Connectivity	Power Use	Ease of Use	Typical IoT Use
Arduino Uno	16 MHz	2 KB	None	Low	Very easy	Prototyping, education
ESP8266	80 MHz	80 KB	Wi-Fi	Low	Easy	Smart home, wireless node
ESP32	240 MHz	520 KB	Wi-Fi/BT	Low	Easy	Wearables, automation
STM32F103	72 MHz	20 KB	UART/SPI/I2C	Low	Moderate	Industrial, medical
Raspberry Pi	700 MHz+	512 MB	Ethernet/WiFi	High	Moderate	Edge computing, gateway

Program to interface IR Sensor:

```

const int ProxSensor=A0;
int inputVal = 0;
void setup()
{
  pinMode(7,OUTPUT);
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(ProxSensor,INPUT);//Pin A0 is connected to the output of proximity sensor
  Serial.begin(9600);
}

```

```
void loop()
{
  if(digitalRead(ProxSensor)==HIGH)
  {
    //Check the sensor output
    digitalWrite(7,HIGH); // set the LED on
  }
  else
  {
    digitalWrite(7,LOW); // set the LED off
  }
  inputVal = analogRead(ProxSensor);
  Serial.println(inputVal);
  delay(1000);
}
```

VIVA -VOCE QUESTIONS

1. Name any four microcontrollers commonly used in IoT applications.
2. Why is ESP32 preferred for IoT applications?
3. What are the differences between Arduino Uno and ESP32?
4. What communication interfaces are available in IoT microcontrollers?
5. Mention two features of NodeMCU.

Result

Studied and compared the features of various microcontrollers used in IoT and sensor interfacing are implemented.

EXP NO:7**DATE:****LOGGING OF DATA IN CLOUD USING IOT****AIM:**

To interface a temperature sensor with a Node MCU and log the data in Thingspeak Cloud.

Apparatus/Software Required**Hardware**

NodeMCU (ESP8266-based)

LM35 Temperature Sensor

Breadboard and jumper wires

Micro-USB cable

Software

Arduino IDE with ESP8266 board support

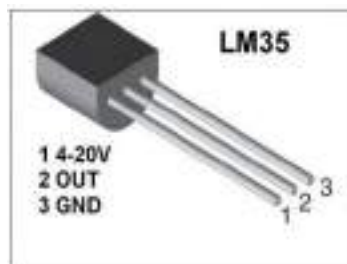
ThingSpeak account (free)

Libraries: ESP8266WiFi.h, ThingSpeak.h

Theory**Temperature Sensor:**

Converts temperature into an analog (voltage/resistance) or digital signal. Common types include:

LM35: Analog voltage proportional to temperature



Pin detail of LM 35

ThingSpeak API

ThingSpeak is a very good platform for IoT based projects. By using channels and web pages provided by ThingSpeak we can monitor any data over the Internet from anywhere and we can also control our system over the internet. ThingSpeak 'Collects' data from sensors, 'Analyze and Visualize' data and 'Acts' by triggering a reaction.

HARDWARE/ SOFTWARE SETUP

Circuit Connections

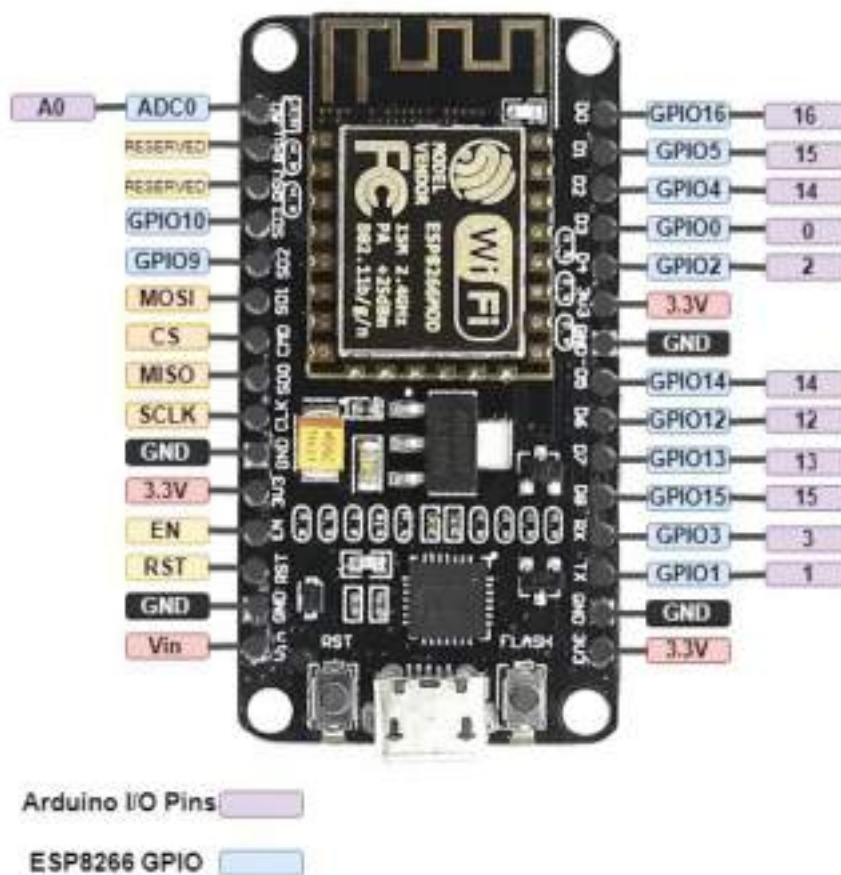
LM35 to NodeMCU

Pin 1 (VCC): Connect to NodeMCU's 3.3V pin

Pin 2 (Output): Connect to NodeMCU's A0 (Analog Input)

Pin 3 (GND): Connect to NodeMCU's GND

NODE MCU PIN DETAILS



ThingSpeak Setup

Create a Channel

Log in to ThingSpeak → Channels → My Channels → New Channel

Name the channel (e.g., "Temperature Monitor") and enable Field 1 (label: "Temperature").

Save and note the Write API Key .

Arduino IDE Configuration

Install Required Libraries

In Arduino IDE: Sketch → Include Library → Manage Libraries → Install ThingSpeak and ESP8266WiFi

Procedure

1. First of all, user needs to create an account on ThingSpeak.com, then Sign In and click on 'Get Started'.
2. Now go to the 'Channels' menu and click on the 'New Channel' option in the same page.
3. Now you will see a form for creating the channel, fill the Name and Description as per your choice. Then fill 'Temperature' in 'Field 1' field. Tick the check box 'Make Public' option below the form and finally Save the channel. Now your new channel is ready.
4. Now click on 'API keys' tab and note the Write and Read API key, here we are only using Write key. You need to copy and paste this key in the code (see below).
5. Now user need to upload the program to ESP8266 using [Arduino](#) IDE.
6. After uploading, open "PRIVATE VIEW" icon in ThingSpeak website and observe the monitored temperature value on graph as shown below.

Program :

```
#include "ThingSpeak.h"
#include <ESP8266WiFi.h>
const int LM35 = A0;
//----- Enter you Wi-Fi Details-----//
char ssid[] = "vivo 1935"; //SSID
char pass[] = "Vivo 1935"; // Password
//-----//
WiFiClient client;
unsigned long myChannelNumber = 1843008; // Channel ID here
const int FieldNumber = 1;
const char * myWriteAPIKey = "R8NVVUA948RBM0MQ"; // Your Write API Key here
void setup()
{
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  ThingSpeak.begin(client);
}
void loop()
{
```

```

if (WiFi.status() != WL_CONNECTED)
{
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);
  while (WiFi.status() != WL_CONNECTED)
  {
    WiFi.begin(ssid, pass);
    Serial.print(".");
    delay(5000);
  }
  Serial.println("\nConnected.");
}

int ADC;
float temp;
ADC = analogRead(LM35); /* Read Temperature */
temp = (ADC * 3); /* Convert adc value to equivalent voltage */
temp = (temp / 10); /* LM35 gives output of 10mv/°C */
Serial.print("Temperature = ");
Serial.print(temp);
Serial.println(" *C");
delay(1000);
ThingSpeak.writeField(myChannelNumber, FieldNumber, temp, myWriteAPIKey);
delay(5000);
}

```

SAMPLE OUTPUT IN THINKSPEAK CLOUD



Observation Table

S.No.		

VIVA -VOCE QUESTIONS

1. What is cloud computing?
2. What is the purpose of data logging?
3. Name two IoT cloud platforms.
4. Which communication protocol is commonly used for cloud communication?
5. Mention two advantages of cloud-based IoT systems.

Result

Successfully interfaced a temperature sensor with a Node MCU and Thingspeak cloud.

EXP NO:8**DATE:****Wireless Access Protocol (Wi-Fi) for LED Blink****AIM:**

To develop and implement a wireless LED blinking application using the Wi-Fi protocol on an ESP32 development board.

Theory

Wireless Access Protocols enable communication between IoT devices without physical wiring. The most commonly used wireless protocols include:

- Wi-Fi (IEEE 802.11)
- Bluetooth
- ZigBee
- LoRa
- Thread

In this experiment, the ESP32 connects to a Wi-Fi network and hosts a web server. A client device (laptop or smartphone) accesses the ESP32 using its IP address. Clicking ON or OFF on the web page sends HTTP requests to the ESP32, which changes the GPIO output state and controls the LED.

Apparatus/Software Required

Sl. No	Equipment	Quantity
1	ESP32 Development Board	1
2	USB Cable	1
3	LED	1
4	220 Ω Resistor	1
5	Computer with Arduino IDE	1
6	Wi-Fi Router / Mobile Hotspot	1

Algorithm

- Start.
- Initialize GPIO pin connected to the LED.
- Connect ESP32 to the Wi-Fi network.
- Start the web server.
- Wait for a client connection.
- If the client requests /LED=ON, turn the LED ON.
- If the client requests /LED=OFF, turn the LED OFF.
- Send the updated web page to the client.
- Repeat continuously.

Program (Arduino IDE)

```
#include <WiFi.h>
```

```
const char* ssid = "YOUR_WIFI_NAME";
```

```
const char* password = "YOUR_PASSWORD";
```

```
WiFiServer server(80);
```

```
const int ledPin = 2;
```

```
void setup()
```

```
{
```

```
  Serial.begin(115200);
```

```
  pinMode(ledPin, OUTPUT);
```

```
  digitalWrite(ledPin, LOW);
```

```
  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED)
```

```
  {
```

```
    delay(500);
```

```
    Serial.print(".");
```

```
  }
```

```
  Serial.println();
```

```
  Serial.print("IP Address: ");
```

```
  Serial.println(WiFi.localIP());
```

```
  server.begin();
```

```
}
```

```
void loop()
```

```
{
```

```
WiFiClient client = server.available();

if(client)
{
    String request = client.readStringUntil('\r');
    client.flush();

    if(request.indexOf("/LED=ON") != -1)
        digitalWrite(ledPin, HIGH);

    if(request.indexOf("/LED=OFF") != -1)
        digitalWrite(ledPin, LOW);

    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type:text/html");
    client.println();

    client.println("<h2>ESP32 LED Control</h2>");
    client.println("<a href=\"/LED=ON\"><button>LED ON</button></a>");
    client.println("<a href=\"/LED=OFF\"><button>LED OFF</button></a>");

    delay(1);
    client.stop();
}
}
```

Procedure

- Connect the LED to GPIO2 through a 220 Ω resistor.
- Open the Arduino IDE.
- Select the ESP32 development board.
- Enter the Wi-Fi SSID and password in the program.
- Compile and upload the program.
- Open the Serial Monitor and note the assigned IP address.
- Open a web browser on a device connected to the same Wi-Fi network.
- Enter the ESP32 IP address.

- Click LED ON and LED OFF to control the LED wirelessly.
- Observe the LED response.

Observation

Client Request	LED Status
/LED=ON	ON
/LED=OFF	OFF

VIVA-VOCE QUESTIONS

1. What is the purpose of the LED blink program?
2. Which GPIO pin is used to control an LED?
3. What is the function of the `delay()` function?
4. Why is a current-limiting resistor connected with the LED?
5. Mention two applications of LED interfacing.

Result

The ESP32 successfully established a Wi-Fi connection, hosted a web server, and controlled the LED wirelessly through HTTP requests.

EXP NO: 9.A**DATE:****DESIGN OF IOT SYSTEM -Automatic door system****AIM:**

To design and implement an IoT-enabled automatic door system that utilizes proximity detection to autonomously open the door.

THEORY :

Regular door opening and closing need manual physical power. It is ok when normal home doors. Where as in big size malls or industries we can't open and close manually. There are some electronic doors can be controlled by power. But those are static and can't be control from remote location. This system is designed to open the door automatically when a person comes near to the entrance of the door by sensing the distance.

DESCRIPTION:

This system includes Arduino uno receives the distance information from ultrasonic sensor and monitors continuously, when it goes below the activation distance it activates the servo motor.

Materials Required

Ultrasonic sensor (HC-SR04)

Servo motor (SG90)

Program :

```
#include <Servo.h>

const int ActivationDistance = 10

const int trigPin = 3;
const int echoPin = 2;

long duration;
int distanceCm, distanceInch;

int servoPin = 9;

Servo servo;

int angle = 0; // servo position in degrees

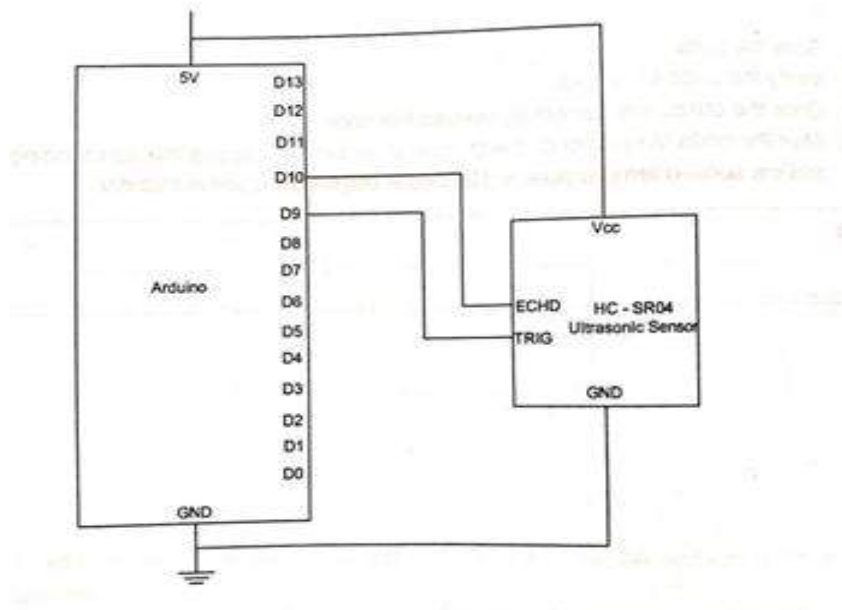
void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  servo.attach(servoPin);
}
```

```
void loop()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distanceCm = duration * 0.034 / 2;
  distanceInch = duration * 0.0133 / 2;
  delay(10);
  if (distanceCm <= ActivationDistance)
  {
    for(angle = 0; angle < 180; angle++)
    {
      servo.write(angle);
      delay(15);
    }
    // now scan back from 180 to 0 degrees
    for(angle = 180; angle > 0; angle--)
    {
      servo.write(angle);
      delay(15);
    }
  }
  delay(500);
}
```

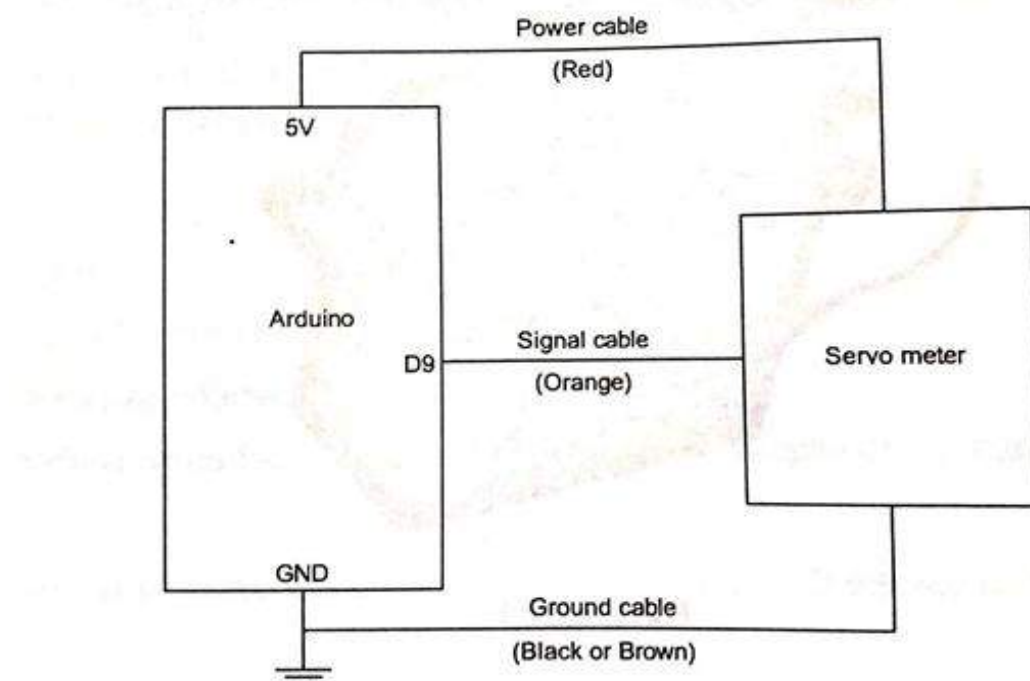
VIVA -VOCE QUESTIONS

- Which sensor is commonly used for automatic door systems?
- What actuator is used to open and close the door?
- How does an automatic door system work?
- Mention two applications of automatic door systems.
- What is the role of IoT in an automatic door system?

Ultrasonic Distance Measurement with Arduino



Circuit Diagram:



RESULT:

Thus an IoT-enabled automatic door system that utilizes proximity detection to autonomously open the door is designed and implemented.

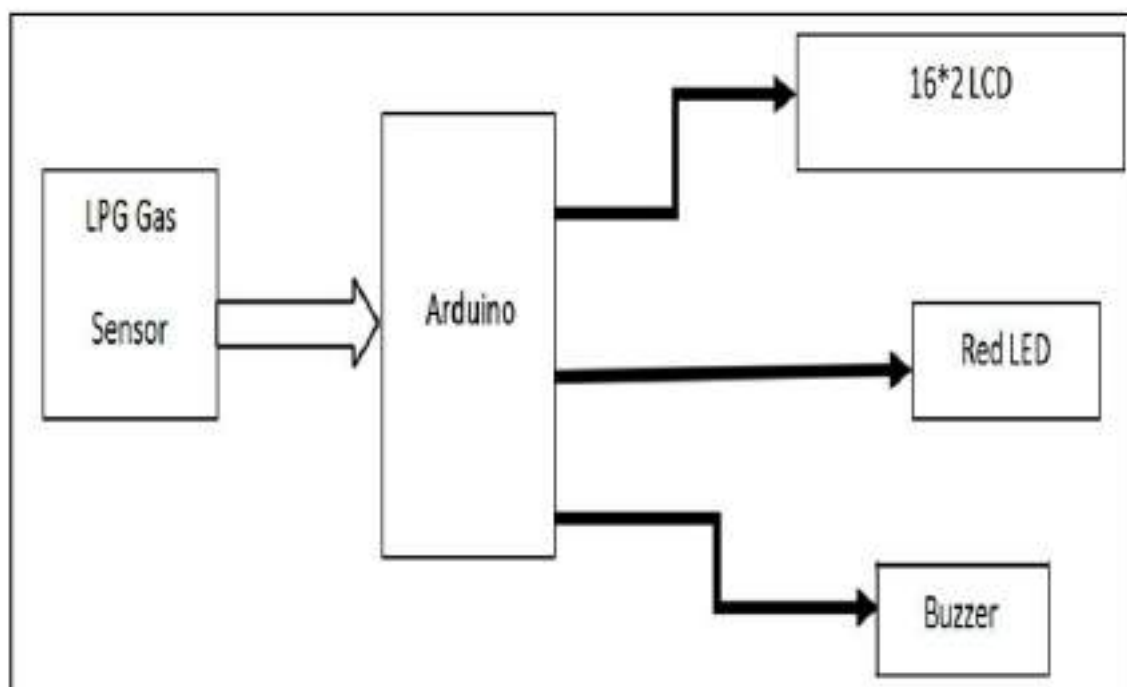
EXP NO: 9.B**DESIGN OF GAS LEAKAGE DETECTION SYSTEM****DATE:****AIM:**

To design and implement an IoT-enabled gas leakage detection system and generate alarm.

THEORY :

Gas leakage is a serious problem and nowadays it is observed in many places like residences, industries, and vehicles like Compressed Natural Gas (CNG), buses, cars, etc. It is noticed that due to gas leakage, dangerous accidents occur. The Liquefied petroleum gas (LPG), or propane, is a flammable mixture of hydrocarbon gases used as fuel in many applications like homes, hostels, industries, automobiles, and vehicles because of its desirable properties which include high calorific value, less smoke, less soot, and meager harm to the environment. Liquid petroleum gas (LPG) is highly inflammable and can burn even at some distance from the source of leakage.

A gas leakage detector becomes vital and helps to protect people from the dangers of gas leakage. There are different gas detection techniques used. This system is a low-cost advanced sensor-based gas leakage detector, alert and control system.

BLOCK DIAGRAM:

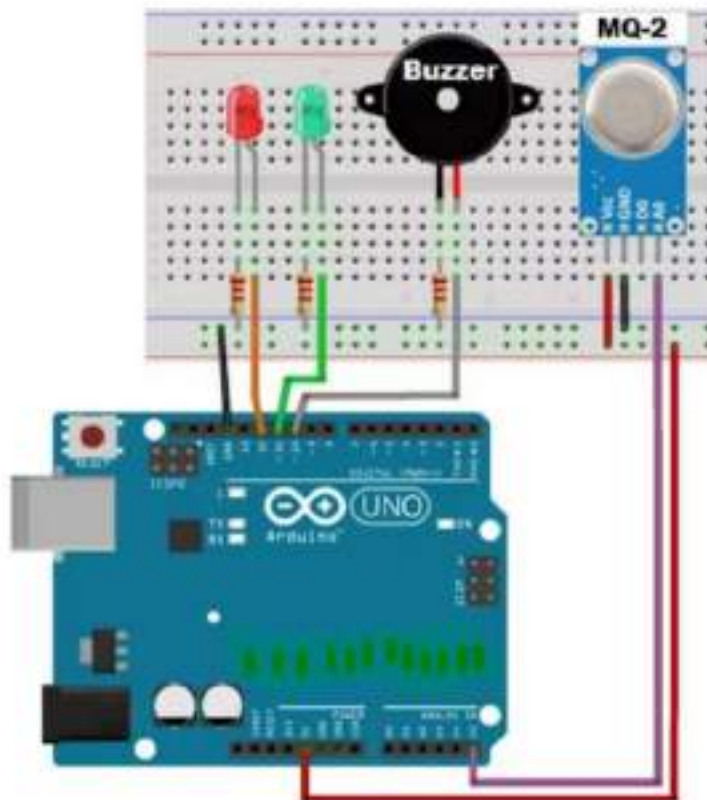
Materials Required

ARDUINO

MQ-6 GAS SENSOR

16*2 LCD

BUZZER

CIRCUIT DIAGRAM**Program :**

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(5,6,8,9,10,11);
int redled = 2;
int greenled = 3;
int buzzer = 4;
int sensor = A0;
int sensorThresh = 400;
```

```
void setup()
{
  pinMode(redled, OUTPUT);
  pinMode(greenled,OUTPUT);
  pinMode(buzzer,OUTPUT);
  pinMode(sensor,INPUT);
  Serial.begin(9600);
  lcd.begin(16,2);
}

void loop()
{
  int analogValue = analogRead(sensor);
  Serial.print(analogValue);
  if(analogValue>sensorThresh)
  {
    digitalWrite(redled,HIGH);
    digitalWrite(greenled,LOW);
    tone(buzzer,1000,10000);
    lcd.clear();
    lcd.setCursor(0,1);
    lcd.print("ALERT");
    delay(1000);
    lcd.clear();
    lcd.setCursor(0,1);
    lcd.print("EVACUATE");
    delay(1000);
  }
  else
  {
    digitalWrite(greenled,HIGH);
    digitalWrite(redled,LOW);
    noTone(buzzer);
    lcd.clear();
    lcd.setCursor(0,0);
```

```
lcd.print("SAFE");  
delay(1000);  
lcd.clear();  
lcd.setCursor(0,1);  
lcd.print("ALL CLEAR");  
delay(1000);  
}  
}
```

VIVA-VOCE QUESTIONS

1. Which gas sensor is commonly used for gas leakage detection?
2. What gases can the MQ-2 sensor detect?
3. What action is taken when gas leakage is detected?
4. Why is gas leakage detection important?
5. Mention two applications of gas detection systems.

RESULT:

Thus, an IoT-enabled gas leakage detection system that utilizes LPG gas detection to raise alarm is designed and implemented.

ADDITIONAL EXPERIMENT BEYOND SYLLABUS

EXP NO: 10	Edge AI Based Object Detection Using ESP32-CAM and IoT Dashboard
DATE:	

AIM:

To develop an Edge AI-based IoT system using ESP32-CAM for object detection and transmit the detection results to an IoT dashboard for remote monitoring.

Objectives

- To understand Edge AI concepts.
- To interface ESP32-CAM with Wi-Fi.
- To perform real-time object detection.
- To send detection status to an IoT dashboard.
- To monitor detected objects remotely.

Apparatus Required

S.No	Equipment	Specification	Quantity
1	ESP32-CAM Module	AI Thinker	1
2	USB to TTL Programmer	CP2102/FTDI	1
3	Computer	Windows/Linux	1
4	Arduino IDE	Latest Version	1
5	Wi-Fi Network	Internet Enabled	1
6	IoT Dashboard	Blynk/ThingSpeak	1

Software Required

- Arduino IDE
- ESP32 Board Package
- ESP32 Camera Library
- Wi-Fi Library
- Blynk/ThingSpeak Library

Theory

Edge AI refers to executing Artificial Intelligence algorithms directly on embedded devices without relying on cloud computing. In this experiment, ESP32-CAM captures images and performs lightweight object detection. The detected object's information is transmitted through Wi-Fi to an IoT dashboard where it can be monitored remotely.

Advantages include

- Reduced latency
- Improved privacy
- Lower bandwidth usage
- Real-time decision making

Applications include

- Smart surveillance
- Industrial monitoring
- Smart agriculture
- Home automation
- Traffic monitoring

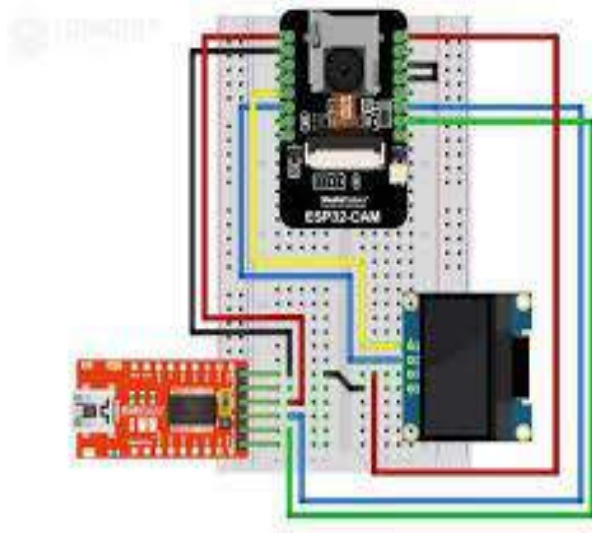
Block Diagram



Circuit Connections

ESP32-CAM Connection

5V	Power Supply
GND	Ground
U0R	USB-TTL TX
U0T	USB-TTL RX



Algorithm

1. Initialize ESP32-CAM.
2. Connect to Wi-Fi.
3. Initialize camera module.
4. Capture image continuously.
5. Perform object detection.
6. Identify detected object.
7. Send object information to IoT dashboard.
8. Display result on serial monitor.
9. Repeat continuously.

Program (Arduino IDE)

```
#include <WiFi.h>

const char* ssid = "YourSSID";
const char* password = "YourPassword";

void setup()
{
  Serial.begin(115200);

  WiFi.begin(ssid,password);

  while(WiFi.status()!=WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }

  Serial.println("WiFi Connected");

}

void loop()
```

```
{  
Serial.println("Object Detected");  
  
delay(5000);  
}
```

(This is a basic demonstration program. In a full implementation, an object detection library or ESP-WHO framework would be integrated.)

Expected Output

WiFi Connected

Object Detected

Result

Thus, the Edge AI-based IoT system was successfully implemented using ESP32-CAM, and the detected object information was monitored remotely through the IoT dashboard.

Applications

- Smart Cities
- AI Surveillance
- Automated Attendance
- Smart Retail
- Traffic Monitoring
- Smart Parking
- Wildlife Monitoring
- Security Systems

Precautions

1. Ensure stable Wi-Fi connectivity.
2. Power ESP32-CAM using a regulated 5 V supply.
3. Upload code using proper boot mode.
4. Avoid poor lighting during image capture.
5. Verify camera initialization before execution.

Viva-Voce Questions

1. What is Edge AI?
2. What are the advantages of Edge AI over Cloud AI?
3. Why is ESP32-CAM suitable for IoT applications?
4. What communication protocol is used by ESP32-CAM?
5. What is the role of Wi-Fi in this experiment?