VV COLLEGE OF ENGINEERING
TISAIYANVILAI

LAB MANUAL

STUDENTNAME : .....

REGISTERNUMBER : .....

SUBJECTCODE : CS3691

SUBJECT NAME : EMBEDDED SYSTEMS AND IOT

DEGREE /BRANCH: BE / CSE

YEAR / SEM : III / 06

ACADEMIC YEAR : 2023 – 2024 (EVEN)

V V COLLEGE OF ENGINEERING
(Approved By AICTE, New Delhi and Affiliated To Anna University Chennai)
V V Nagar, Arasoor, Tisaiyanvilai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

College Vision and Mission Statement

Vision

"Emerge as a premier technical institution of global standards, producing enterprising, knowledgeable engineers and entrepreneurs."

Mission

- Impart quality and contemporary technical education for rural students.
- Have the state of the art infrastructure and equipment for quality learning.
- Enable knowledge with ethics, values and social responsibilities.
- Inculcate innovation and creativity among students for contribution to society.

Vision and Mission of the Department of Computer Science and Engineering

Vision

"Produce competent and intellectual computer science graduates by empowering them to compete globally towards professional excellence".

Mission

- Provide resources, environment and continuing learning processes for better exposure in latest and contemporary technologies in Computer Science and Engineering.
- Encourage creativity and innovation and the development of self-employment through knowledge and skills, for contribution to society
- Provide quality education in Computer Science and Engineering by creating a platform to enable coding, problem solving, design, development, testing and implementation of solutions for the benefit of society.

I. PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

Graduates of Computer Science Engineering can

- Apply their technical competence in computer science to solve real world problems, with
  technical and people leadership.
- Conduct cutting edge research and develop solutions on problems of social relevance.
- Work in a business environment, exhibiting team skills, work ethics, adaptability and lifelong learning.

II. PROGRAM OUTCOMES (POs)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

III. PROGRAM SPECIFIC OUTCOMES (PSOs)

The Students will be able to

- Exhibit design and programming skills to build and automate business solutions using cutting edge technologies.
- Strong theoretical foundation leading to excellence and excitement towards research, to provide elegant solutions to complex problems.
- Ability to work effectively with various engineering fields as a team to design, build and develop system applications.

V V COLLEGE OF ENGINEERING

(Approved By AICTE, New Delhi and Affiliated To Anna University Chennai)

V V Nagar, Arasoor, Tisaiyanvilai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABORATORY

LIST OF EXPERIMENTS-R2021

PRACTICAL SUBJECT NAME	**EMBEDDED SYSTEMS AND IOT**
PRACTICAL SUBJECT CODE	**CS3691**
SEMESTER/ YEAR	**VI / III**
TOTAL HOURS	30
STAFF IN-CHARGE	Dr.R.JENSI
LAB INSTRUCTOR	Mr.Yugaraj
REGULATION	2021

CO1	Explain the architecture of embedded processors.
CO2	Write embedded C programs.
CO3	Explain the features of Arduino
CO4	Compare the communication models in IOT
CO5	Design IoT applications using Arduino/Raspberry Pi /open platform.
CO6	Design simple embedded applications.

List of Exercises

S.No	Exercise	CO Mapping
1.	Write 8051 Assembly Language experiments using simulator.	CO6
2.	Test data transfer between registers and memory	CO1
3.	Perform ALU operations	CO1
4.	Write Basic and arithmetic Programs Using Embedded C	CO2
5.	Introduction to Arduino platform and programming	CO3
6.	Explore different communication methods with IoT devices (Zigbee, GSM, Bluetooth)	CO4
7.	Introduction to Raspberry PI platform and python programming	CO5
8.	Interfacing sensors with Raspberry PI	CO5
9.	Communicate between Arduino and Raspberry PI using any wireless medium	CO5
10.	Setup a cloud platform to log the data	CO5
11.	Log Data using Raspberry PI and upload to the cloud platform	CO5
12.	Design an IOT based system	CO5

CO-PO Mapping

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	P012
CO1	3	3	3	3	-	-	-	-	1	2	3	3
CO2	2	1	3	2	2	-	-	-	1	2	2	3
CO3	3	1	3	3	1	-	-	-	1	2	1	1
CO4	3	2	3	2	1	-	-	-	1	2	2	3
CO5	2	3	3	2	2	-	-	-	1	3	3	2
CO6	2	3	3	2	2	-	-	-	1	3	3	2

1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High)

CO - PSO Mapping

CO	PSO1	PSO2	PSO3
CO1	2	1	3
CO2	3	1	3
CO3	1	3	3
CO4	2	2	1
CO5	3	1	3
CO6	3	1	3

1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High)

V V COLLEGE OF ENGINEERING

(Approved By AICTE, New Delhi and Affiliated To Anna University Chennai)

V V Nagar, Arasoor, Tisaiyanvilai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RUBRICS FOR ASSESSING LABORATORY

SI. No.	Criteria	Total Marks	Excellent (25)	Good (20)	Average (10)	Poor (5)
			91% - 100%	71% - 90%	50% - 70%	<50%
1	Preparation	25	Gives clear idea about the aim and having good capability of executing experiments.	Capability of executing experiments but no proper clarification about the objective.	Gives clear idea about the target and has less capability of executing experiments.	Gives indistinct idea about the target and has less capability of executing experiments & who feel difficult to follow the objectives.
2	Viva	25	Have executed the experiments in an efficient way & make credible and unbiased judgments regarding the experiments.	Executed the experiments with less efficient & has partial judgments regarding the experiments.	Executed the experiments with less efficiency and has no judgments regarding experiments .	Incomplete experiments & lack of judgments regarding experiments.

3	Performance	25	Followed all the instructions given in the procedure and submitted the manual on time.	Followed all the instructions given in the procedure with some assisting.	Followed some of the instructions given in the procedure & late in submission of manual.	Unable to follow the instructions given in the procedure & late in submission of manual.

V V COLLEGE OF ENGINEERING

(Approved By AICTE, New Delhi and Affiliated To Anna University Chennai)

V V Nagar, Arasoor, Tisaiyanvilai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science and Engineering		
Preparation	25	
Viva	25	
Performance	25	
Total	75	
Lab Incharge	Date	

INDEX

S.No	DATE	NAME OF THE EXPERIMENT	SIGN
1		Assembly Language Experiments using Simulator	
2		Data Transfer from register to memory	
		Data Transfer from memory to register	
		Data Transfer from memory to memory	
3		Arithmetic operations addition and subtraction	
		Arithmetic operations multiplication and division	
		Logical operations	
		find the largest number in an array	
4		convert 11111111 to decimal and display the digits on Po,P1,P2 using the Keil C51 compiler.	
		Embedded C program to perform addition and subtraction	
5		Introduction to Arduino platform and programming – LED Blink	
6		communication methods with IoT devices GSM	
		communication methods with IoT devices Bluetooth	
		communication methods with IoT devices Zigbee	
7		Introduction to Raspberry PI platform and python programming – LED Blink	
8		Interfacing Temperature sensor with Raspberry PI	
		Interfacing IR sensor with Raspberry PI	
9		Setup a cloud platform to log the data	
10		Communicate between Arduino and Raspberry PI using any wireless medium	
11		Log Data using Raspberry PI and upload to the cloud platform	
12		Design an IoT System	

Exercise 1: Assembly Language Experiments using Simulator

1.1
Aim: To write an assembly language program for LED Blink Using Keil simulator.

Steps:

1. **Create a New Project:** Open Keil IDE and create a new project. Go to "Project" in the menu bar, then select "New µVision Project...". Choose a location and give your project a name. Click "Save".
2. **Select Target Device:** In the dialog box that appears, select the appropriate target device for your simulation. This depends on the microcontroller model you're simulating. For example, if you're simulating an 8051 microcontroller, select the corresponding device (Honeywell - HT83C51) from the list.
3. **Create an Assembly Source File:** Right-click on the "Source Group" folder in the Project window and select "Add New Item to Group...". Choose "Assembly Source File" and give it a name, like "blink.asm".
4. **Write Assembly Code:** Open the assembly source file you created and write your LED blink assembly code.
5. Algorithm for the LED Blink is given below:

    1. Set up initial values:
        - Set register R4 to 05H.
    2. Start of the loop:
        - Set Port 1 (P1) to all high (11111111b).
        - Call the subroutine DELAY.
        - Set Port 1 (P1) to all low (00000000b).
        - Jump back to the label HERE.
    3. Subroutine DELAY:
        - Set register R0 to 100.
        - Start nested loop:
            - Set register R1 to 200.
            - Start inner loop:
                - Set register R2 to the address of the label AGAIN.
                - Decrement R2 and jump to AGAIN until R2 becomes zero.
            - Decrement R4 and jump back to BACK1 until R4 becomes zero.
        - Decrement R0 and jump back to BACK2 until R0 becomes zero.
        - Return from the subroutine.
    4. End of the code.

6. **Build the Project:** Click on the "Build" button in the toolbar or go to "Project" > "Build Target" to build the project. This assembles the source code into machine code.

7. **Simulate the Program:** After building the project successfully, you can simulate the program by clicking on the "Debug" button in the toolbar or going to "Debug" > "Start/Stop Debug Session". This will start the debugger and open the simulator window.

8. **Run the Simulation:** In the simulator window, you can start the simulation by clicking on the "Run" button or pressing F5. This will execute your assembly code, and you should see the LED blink simulation.

Program

```
ORG 00H
MOV R4,#05H
HERE: MOV P1,#11111111b
ACALL  DELAY
MOV P1,#00000000B
SJMP HERE
DELAY: MOV R0,#100
BACK2: MOV R1,#200
BACK1: MOV R2,AGAIN
AGAIN: DJNZ R2,AGAIN
DJNZ R4,BACK1
DJNZ R0,BACK2
RET
END
```

Result: Thus the assembly language program for LED blink is executed and the output is verified.

Exercise 2 Test data transfer between registers and memory

2.1

Aim:

To write an assembly language program for Data Transfer from register to memory

Algorithm:

1. Load immediate value:
    - Load the immediate value 12h into register A.
2. Store value:
    - Move the contents of register A into memory location 30h.
3. End of the code.

Program

```
Mov A,#12h
Mov 30h,A
End
```

2.2
Aim:
To write an assembly language program for Data Transfer from memory to register
Algorithm:

1. Move immediate value to memory:
    - Move the immediate value 12h to memory location 30h.
2. Move memory to register:
    - Move the contents of memory location 30h to register R1.
3. End of the code.

Program

```
Mov 30h,#12h
Mov R1,30h
End
```

2.3.
Aim:
To write an assembly language program for Data Transfer from memory to memory

Algorithm

1. Move immediate value to memory:
    - Move the immediate value 17D to memory location 40h.
2. Move memory to memory:
    - Move the contents of memory location 40h to memory location 50h.
3. End of the code.

Program:

```
MOV 40H,#17D
MOV 50H,40H
END
```

Result: Thus the assembly language programs for data transfer are executed and the output is verified.

Exercise 3: Perform ALU operations.

3.1

Aim:
To write an assembly language program to perform arithmetic operations addition and subtraction (30+10)-(20+2).

Algorithm:

1.Load immediate value to accumulator:
- Load the immediate value 2D (decimal 45 in base 10) into the accumulator.

2. Add immediate value to accumulator:
   - Add the immediate value 20D (decimal 32 in base 10) to the accumulator.
3. Move accumulator to register:
   - Move the contents of the accumulator to register R1.
4. Load immediate value to accumulator:
   - Load the immediate value 30D (decimal 48 in base 10) into the accumulator.
5. Add immediate value to accumulator:
   - Add the immediate value 10D (decimal 16 in base 10) to the accumulator.
6. Subtract with borrow from register:
   - Subtract the contents of register R1 from the accumulator using the "subtract with borrow" (SUBB) instruction. The borrow is automatically set based on the result of the previous operation.
7. Move accumulator to register:
   - Move the contents of the accumulator to register R2.
8. End of the code

Program:

```
MOV A,#2D
ADD A,#20D
MOV R1,A
MOV A,#30D
ADD A,#10D
SUBB A,R1
MOV R2,A
END
```

3.2
Aim:
To write an assembly language program to perform arithmetic operations multiplication and division (20*4)/2.

Algorithm:

1.Load immediate value to accumulator:
    - Load the immediate value 20D (decimal 32 in base 10) into the accumulator.
2. Load immediate value to register:
    - Load the immediate value 4D (decimal 77 in base 10) into register B.
3. Multiply accumulator and register:
    - Multiply the contents of the accumulator and register B, storing the result in register A.
4. Move accumulator to register:
    - Move the contents of the accumulator to register R1.
5. Move register to register:
    - Move the contents of register B to register R2.
6. Load immediate value to register:
    - Load the immediate value 2D (decimal 45 in base 10) into register B.
7. Divide accumulator by register:
    - Divide the contents of the accumulator by register B, storing the quotient in register A and the remainder in register B.
8. Move accumulator to register:
    - Move the contents of the accumulator to register R3.
9. Move register to register:
    - Move the contents of register B to register R4.
10. End of the code.

Program:

```
MOV A,#20D
MOV B,#4D
MUL AB
MOV R1,A
MOV R2,B
MOV B,#2D
DIV AB
MOV R3,A
MOV R4,B
END
```

3.3

Aim

To write an assembly language program to perform Logical operations.

Algorithm:

1.Load immediate value to accumulator:
    - Load the immediate value 45H (hexadecimal 69 in base 10) into the accumulator.

2. Load immediate value to register:
    - Load the immediate value 67H (hexadecimal 103 in base 10) into register R0.
3. Logical AND between accumulator and register:
    - Perform a bitwise logical AND operation between the accumulator and register R0. Store the result in the accumulator.
4. Move accumulator to memory:
    - Move the contents of the accumulator to memory location 20H.
5. Load immediate value to accumulator:
    - Load the immediate value 45H (hexadecimal 69 in base 10) into the accumulator.
6. Logical OR between accumulator and register:
    - Perform a bitwise logical OR operation between the accumulator and register R0. Store the result in the accumulator.
7. Move accumulator to memory:
    - Move the contents of the accumulator to memory location 21H.
8. Load immediate value to accumulator:
    - Load the immediate value 45H (hexadecimal 69 in base 10) into the accumulator.
9. Exclusive OR between accumulator and register:
    - Perform a bitwise exclusive OR operation between the accumulator and register R0. Store the result in the accumulator.
10. Move accumulator to memory:
    - Move the contents of the accumulator to memory location 22H.
11. Load immediate value to accumulator:
    - Load the immediate value 45H (hexadecimal 69 in base 10) into the accumulator.
12. Complement accumulator:
    - Perform a bitwise complement operation on the accumulator (NOT logic).
13. End of the code.

Program:

```
MOV A,#45H
MOV R0,#67H
ANL A,R0
MOV 20H,A
MOV A,#45H
ORL A,R0
MOV 21H,A
MOV A,#45H
XRL A,R0
MOV 22H,A
MOV A,#45H
CPL A   ; NOT LOGIC
END
```

3.4

Aim:
To write an assembly language program to find the largest number in an array.

Algorithm:

.

1. Initialize register R0 with value 10.
2. Initialize the data pointer (DPTR) with the initial address of ROM memory (600h).
3. Load the first byte from ROM memory using move code (MOVC) instruction and store it in accumulator A.
4. Copy the value from accumulator A to register B.
5. Start a loop labeled as "back":
    - Increment the data pointer.
    - Clear the accumulator.
    - Load the next byte from ROM memory using move code (MOVC) instruction and store it in accumulator A.
    - Compare the value in accumulator A with the value in register B.
    - If they are not equal, jump to the label "check".
    - If they are equal, continue with the loop.
6. Label "check":
    - Check the carry flag. If the carry flag is set (indicating A > B), jump to the label "next".
    - If the carry flag is not set (indicating A <= B), copy the value from accumulator A to register B.
7. Label "next":
    - Decrement register R0.
    - If R0 is not zero, jump back to the label "back".
8. Store the value from register B into memory location 60h.
9. Initialize the ROM memory starting at address 600h with the following values: 34h, 54h, 73h, 12h, 31h, 99h, 13h, 11h, 09h, 76h.
10. End of the code.

Program:

```
MOV R0,#10
MOV dptr,#600h    ; initial address of rom memory
movc a,@a+dptr    ; 1st number from ROM memory
mov b,a
back:  inc dptr
clr a
movc a,@a+dptr
cjne a,b,check
check:   jc next    ; if c=0 a<b or c=1, b>a
```

```
mov b,a
next: djnz r0,back
mov 60h,b   ; biggest number saved in 60h
org 600h
   db 34h,54h,73h,12h,31h,99h,13h,11h,09h,76h
      end
```

Exercise 4: Write Basic and arithmetic Programs Using Embedded C.

4.1

Aim:

To write embedded C code for the 8051 microcontroller to convert 11111111 to decimal and display the digits on Po,P1,P2 using the Keil C51 compiler.

Algorithm:

1. **Include Header File**:
    - #include <reg51.h>: This includes the header file necessary for programming the 8051 microcontroller family.
2. **Main Function**:
    - void main(): This is the main function where the program execution starts.
3. **Variable Declaration**:
    - unsigned char x, binbyte, d1, d2, d3;: These are unsigned char variables used for storing data.
4. **Assign Value to binbyte**:
    - binbyte = 0xFF;: This assigns the value 0xFF (255 in decimal) to the variable binbyte.
5. **Perform Division and Modulus Operations**:
    - x = binbyte / 10;: This performs integer division of binbyte by 10 and stores the result in x.
    - d1 = binbyte % 10;: This calculates the remainder of binbyte divided by 10 and stores it in d1.
    - d2 = x % 10;: This calculates the remainder of x divided by 10 and stores it in d2.
    - d3 = x / 10;: This performs integer division of x by 10 and stores the result in d3.
6. **Output to Ports**:
    - P0 = d1;, P1 = d2;, P2 = d3;: These statements output the values of d1, d2, and d3 to the respective ports P0, P1, and P2.
7. **End of Program**:
    - The main function does not have a return statement, and thus it implicitly returns control to the system.

Program:

```
#include <reg51.h>
```

```
void main()
{
   unsigned char x,binbyte,d1,d2,d3;
   binbyte=0xFF;
   x=binbyte/10;
   d1=binbyte%10;
   d2=x%10;
   d3=x/10;
   P0=d1;
   P1=d2;
   P2=d3;
}
```

4.2

Aim:
To write a 8051 Embedded C program to perform addition and subtraction.

Algorithm:

1. **Include Header File**:
   - Include the header file <reg51.h>, which contains definitions specific to the 8051 microcontroller family.
2. **Main Function**:
   - Define the main function (void main()) where the program execution starts.
3. **Variable Declaration**:
   - Declare three unsigned char variables: num1, num2, and result, representing the two numbers to be added and subtracted, and the result of the operations, respectively.
4. **Initialization**:
   - Initialize num1 with the value 50 and num2 with the value 20.
5. **Addition Operation**:
   - Perform addition of num1 and num2 and store the result in the variable result.
6. **Output of Addition Result**:
   - Output the result of the addition operation (result) to port P0. This assumes that P0 is connected to an output device for display.
7. **Subtraction Operation**:
   - Perform subtraction of num2 from num1 and store the result in the variable result.
8. **Output of Subtraction Result**:
   - Output the result of the subtraction operation (result) to port P1. This assumes that P1 is connected to an output device for display.
9. **Infinite Loop**:
   - Enter an infinite loop (while(1)) to prevent the program from exiting.
10. **End of the Program**:

- End of the main() function.

Program:

```
#include <reg51.h>

void main() {
   unsigned char num1 = 50;   // First number
   unsigned char num2 = 20;   // Second number
   unsigned char result;      // Variable to store the result

   // Addition
   result = num1 + num2;

   // Output result of addition
   P0 = result;  // Assuming P0 is connected to an output device for display

   // Subtraction
   result = num1 - num2;

   // Output result of subtraction
   P1 = result;  // Assuming P1 is connected to an output device for display

   while(1);  // Infinite loop to prevent the program from exiting
}
```

Exercise 5: Introduction to Arduino platform and programming

Aim:

To provide a comprehensive introduction to the Arduino platform and programming, offering a foundational understanding for beginners to embark on their journey in creating interactive electronic projects.

Steps to be followed using Proteus software:

1. **Launch Proteus:** Open Proteus and create a new project.
2. Add the ArduinoUnoTEP.IDX and ArduinoUoTEP.LIB library files to the folder C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\DATA\LIBRARY, if they do not already exist.
3. **Add Components:** In the Proteus workspace, locate the Components tab and add the necessary components:
    - Arduino board: Search for Arduino Uno or the specific model you're using.

- LED: Search for an LED component.
- Resistor: Add a current-limiting resistor for the LED (usually around 220 ohms).

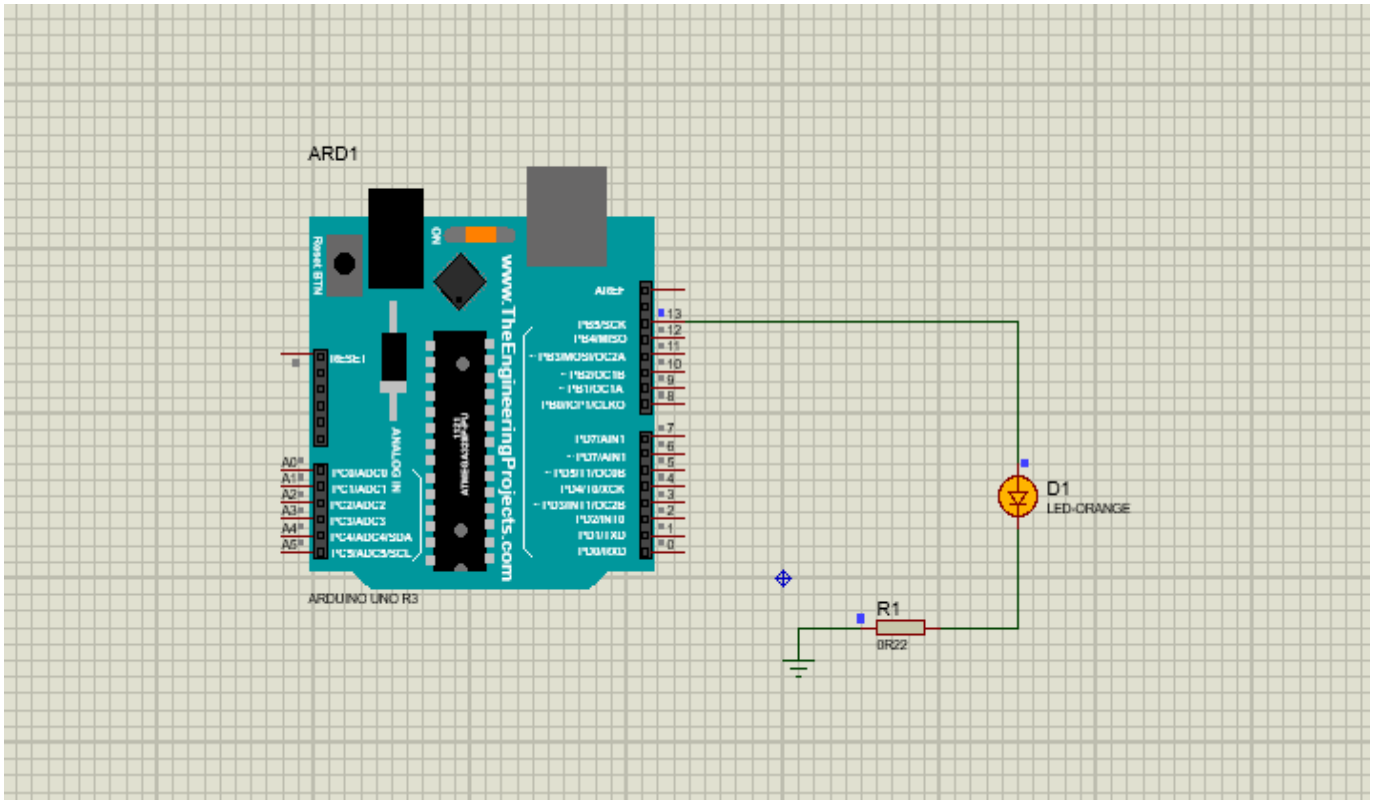
4. **Wiring the Circuit:** Connect the components by placing wires between them. Ensure the connections are correct:
    - Connect one leg of the LED to a digital pin on the Arduino (e.g., pin 13).
    - Connect the other leg of the LED to the resistor.
    - Connect the other end of the resistor to the ground (GND) pin on the Arduino.
5. **Writing the Arduino Sketch:**
    - Open the Arduino IDE or any text editor to write the sketch.
    - Write a simple sketch to blink the LED on and off.
    - Save the sketch with a suitable name
6. **Simulating in Proteus**:
    - In Proteus, click on the Arduino component and then click on the "Edit Properties" button.
    - In the Properties window, browse and select the .hex of Arduino sketch (.ino file) you saved earlier.
    - Click OK to close the Properties window.
    - Now, you can simulate the circuit by clicking on the Play button or pressing F5.
    - You should see the LED blinking according to the sketch.

Arduino Sketch:

```
// Define the LED pin
int ledPin = 13;

// Setup function, runs once when the sketch starts
void setup() {
  // Initialize the digital pin as an output
  pinMode(ledPin, OUTPUT);
}

// Loop function, runs repeatedly
void loop() {
  // Turn the LED on (HIGH)
  digitalWrite(ledPin, HIGH);
  // Wait for 0.1 second
  delay(100);
  // Turn the LED off (LOW)
  digitalWrite(ledPin, LOW);
```

```
  // Wait for another 0.1 second
  delay(100);
 }
```

OUTPUT:



Steps to be followed using hardware:

1. Gather Components:
    - Arduino UNO board
    - LED
    - Resistor(usually around 220 ohms)
    - Jumper wires
    - Breadboard
    - USB Cable
    2. Setup Circuit on Breadboard:
        - Place the Arduino board on the breadboard.
        - Insert the LED into the breadboard, ensuring the longer leg (anode) is connected to digital pin 13 on the Arduino.
        - Connect the shorter leg (cathode) of the LED to one leg of the resistor.
        - Connect the other leg of the resistor to the ground (GND) pin on the Arduino.
3. Upload Arduino Sketch:
    - Connect the Arduino board to your computer using a USB cable.

- Open the Arduino IDE on your computer.
- Write or copy the above Arduino sketch
- Verify the sketch for any errors and then upload it to the Arduino board.

4. Test LED Blinking:
    - Once the sketch is uploaded successfully, the LED should start blinking on and off at one-second intervals.

5. Troubleshooting:

    - If the LED does not blink, double-check the connections on the breadboard.
    - Ensure the correct pin is specified in the sketch for the LED.
    - Verify that the Arduino board is properly powered and connected to the computer.

Result:

Thus the study of Arduino platform and programming for the LED blinking project have been successfully completed.

Exercise 6: Explore different communication methods with IoT devices (Zigbee, GSM, Bluetooth)

6.1

Aim:

To explore and understand communication methods used with IoT devices, particularly focusing on GSM (Global System for Mobile Communications), to enable efficient and reliable data transmission, control, and monitoring in IoT applications.

Steps:

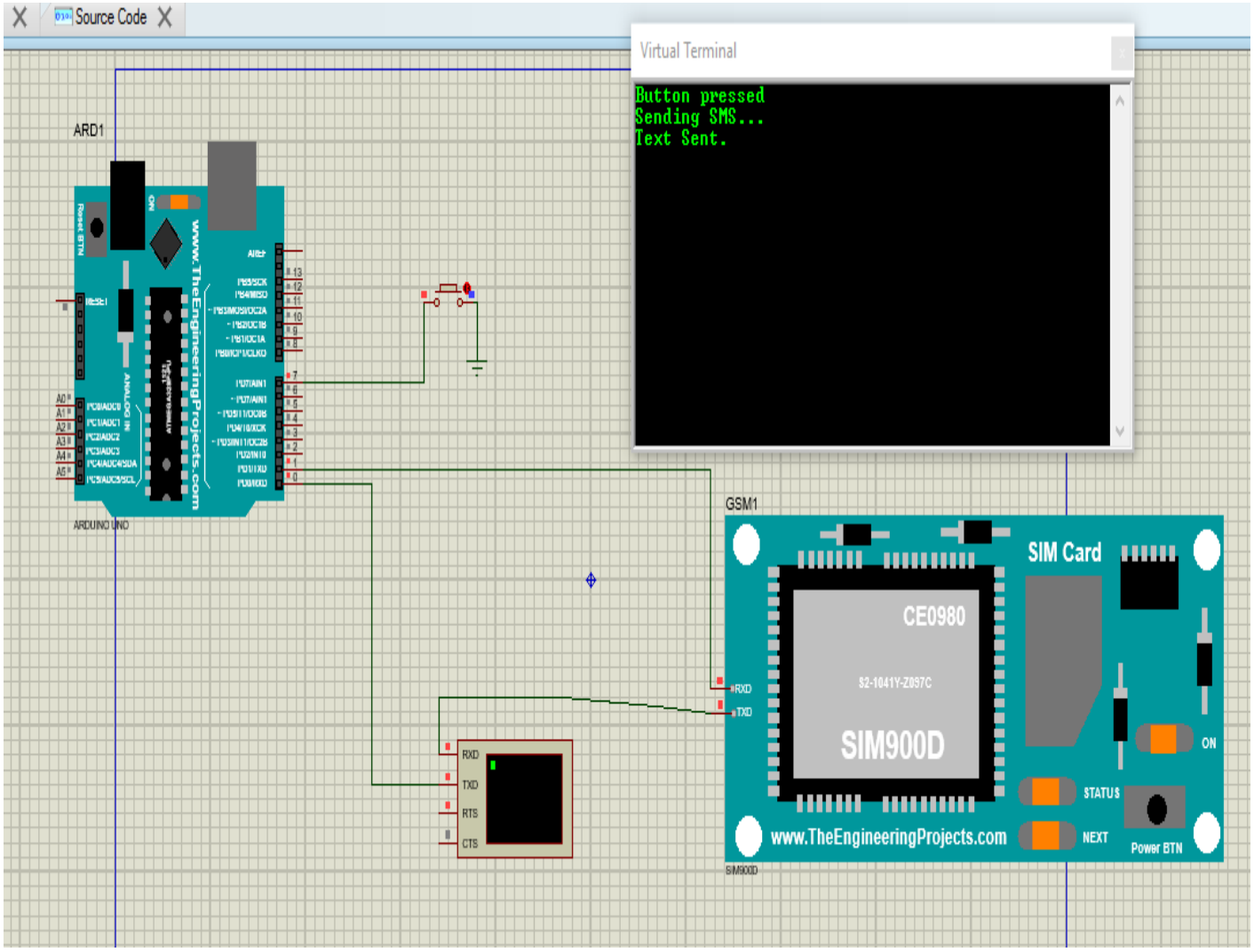
5. Include the GSM Module:
    - Add GSMLibraryTEP.IDX,GSMLibraryTEP.LIB,GSMLibraryTEP.hex library files to the folder C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\ DATA\LIBRARY, if they do not already exist. Add the GSM module component SIM900 to your Proteus project.
6. Add Arduino UNO, push button, virtual terminal to the project.
7. Connect the GSM module to the microcontroller (such as Arduino) using appropriate pins for communication (usually UART). (i.e) Connect Rx pin of GSM module to Tx pin of Arduino, Tx pin of GSM to Rx pin of Virtual terminal, Tx pin of virtual terminal to Rx pin of Arduino.
8. Connect one end of the push button to Arduino pin 7 and the other end to Ground.
9. In Proteus, click on the GSM component and then click on the "Edit Properties" button.
10. In the Properties window, browse and select the GSMLibraryTEP..hex.
11. Click OK to close the Properties window.
12. Write Arduino Sketch:
    - Open the Arduino IDE or any text editor to write the Arduino sketch for sending SMS.
    - Include the necessary libraries for GSM communication.
    - Initialize the GSM module and define necessary variables.
    - Write code to set up communication with the GSM module and send the SMS.
13. Upload Arduino Sketch: Upload the Arduino sketch to the microcontroller in Proteus.
14. Simulate Sending SMS: Run the simulation in Proteus. Open the serial monitor to interact with the Arduino. The GSM module in the simulation will simulate sending the SMS.
15. Verify: Check the simulation results to ensure that the SMS was sent successfully.

Arduino Sketch:

```
#include<SoftwareSerial.h>
SoftwareSerial sim8001(0,1);
#define button1 7
bool button_state;
```

```
void setup()
{
pinMode(button1,INPUT_PULLUP);
sim8001.begin(9600);
Serial.begin(9600);
delay(100);
}
void loop()
{
button_state=digitalRead(button1);
if(button_state==LOW)
{
Serial.println("Button pressed");
delay(200);

SendSMS();
}
if(sim8001.available())
Serial.write(sim8001.read());
}

void SendSMS()
{
Serial.println("Sending SMS...");
sim8001.print("AT+CMGF=1\r");
delay(100);
sim8001.print("AT+CMGS=\"+9865533668\"\r");
delay(100);
sim8001.print("SIM8001 is working");
delay(100);
sim8001.pr1awA2A34`int((char)26);
delay(100);
sim8001.println();
Serial.println("Text Sent.");
delay(400);
}
```

OUTPUT



6.2

Aim:

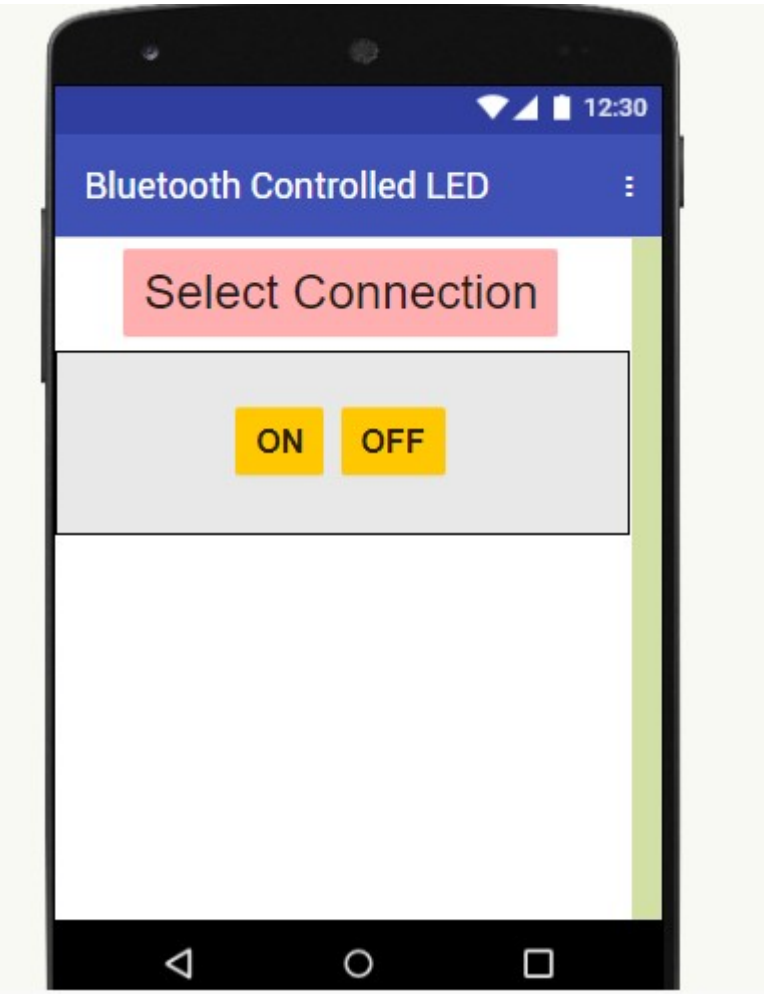
To explore and understand communication methods used with IoT devices, particularly focusing on Bluetooth, to enable efficient and reliable data transmission, control, and monitoring in IoT applications.

Steps:

1. Include the Bluetooth Module:
    - Add BluetoothTEP.IDX,BluetoothTEP.LIB library files to the folder C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\DATA\LIBRARY, if they do not already exist. Add the Bluetooth module component to your Proteus project.
2. Add Arduino UNO, LED, resistor to the project.

3. Connect the Bluetooth module to the microcontroller (such as Arduino) using appropriate pins for communication (usually UART). (i.e) Connect Rx pin of Bluetooth module to Tx pin of Arduino, Tx pin of Bluetooth to Rx pin of Arduino.
4. Connect one leg of the LED to a digital pin on the Arduino (e.g., pin 13). Connect the other leg of the LED to the resistor. Connect the other end of the resistor to the ground (GND) pin on the Arduino.
5. Write Arduino Sketch:
    - Open the Arduino IDE or any text editor to write the Arduino sketch for LED Blinking.
    - Include the necessary libraries for Bluetooth communication.
    - Enable bluetooth on the computer. Configure bluetooth port is the same as Bluetooth module port.
    - Write code to set up communication with the Bluetooth module and port setting.

6. Design an App in MIT App Inventor as follows:



7. Download and install the App on the mobile phone.

8.Upload Arduino Sketch: Upload the Arduino sketch to the microcontroller in Proteus.

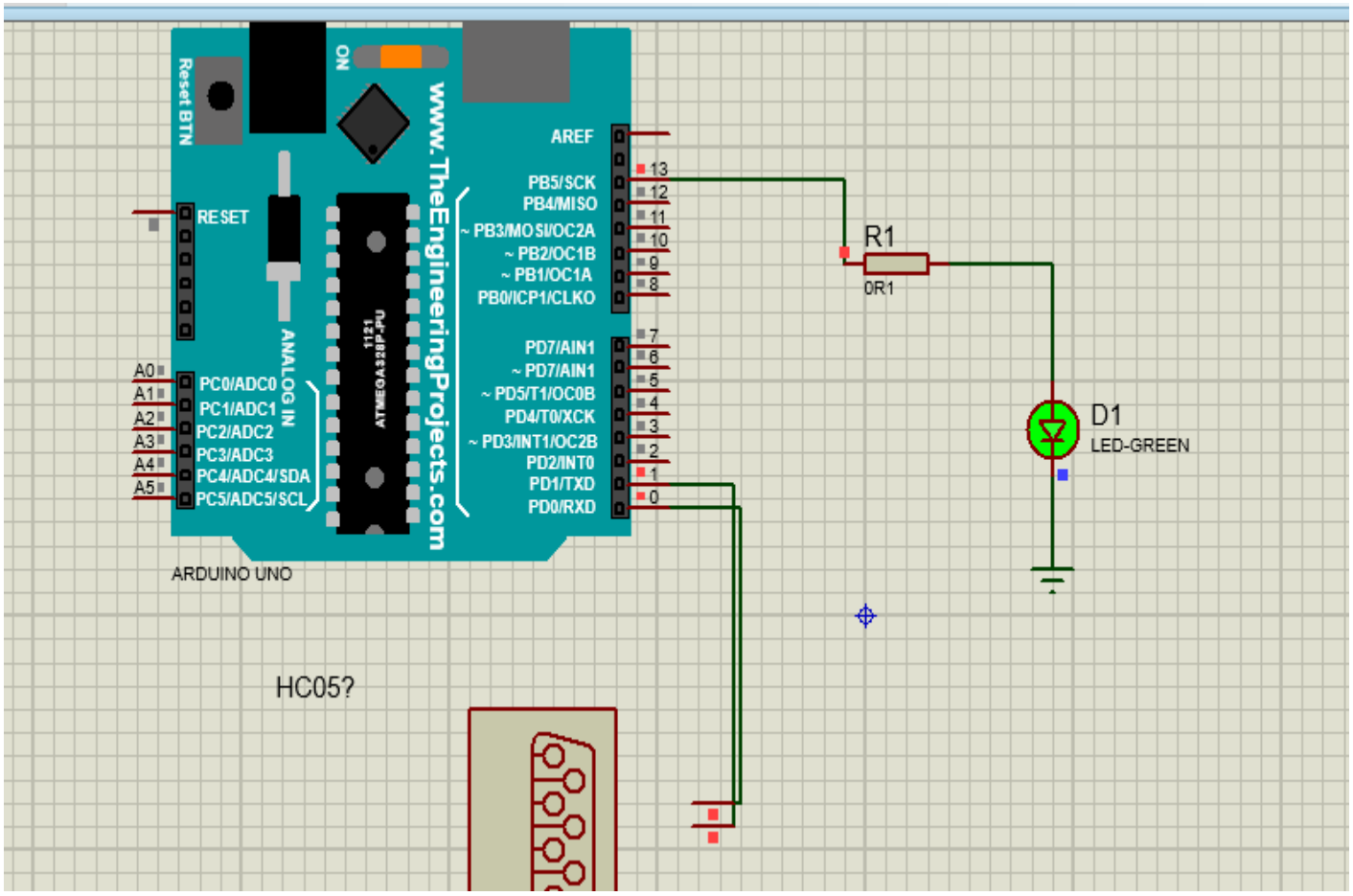
9. Run the simulation in Proteus. Open the Mobile App and establish the bluetooth connection and then click ON/OFF button to turn on or off the LED.

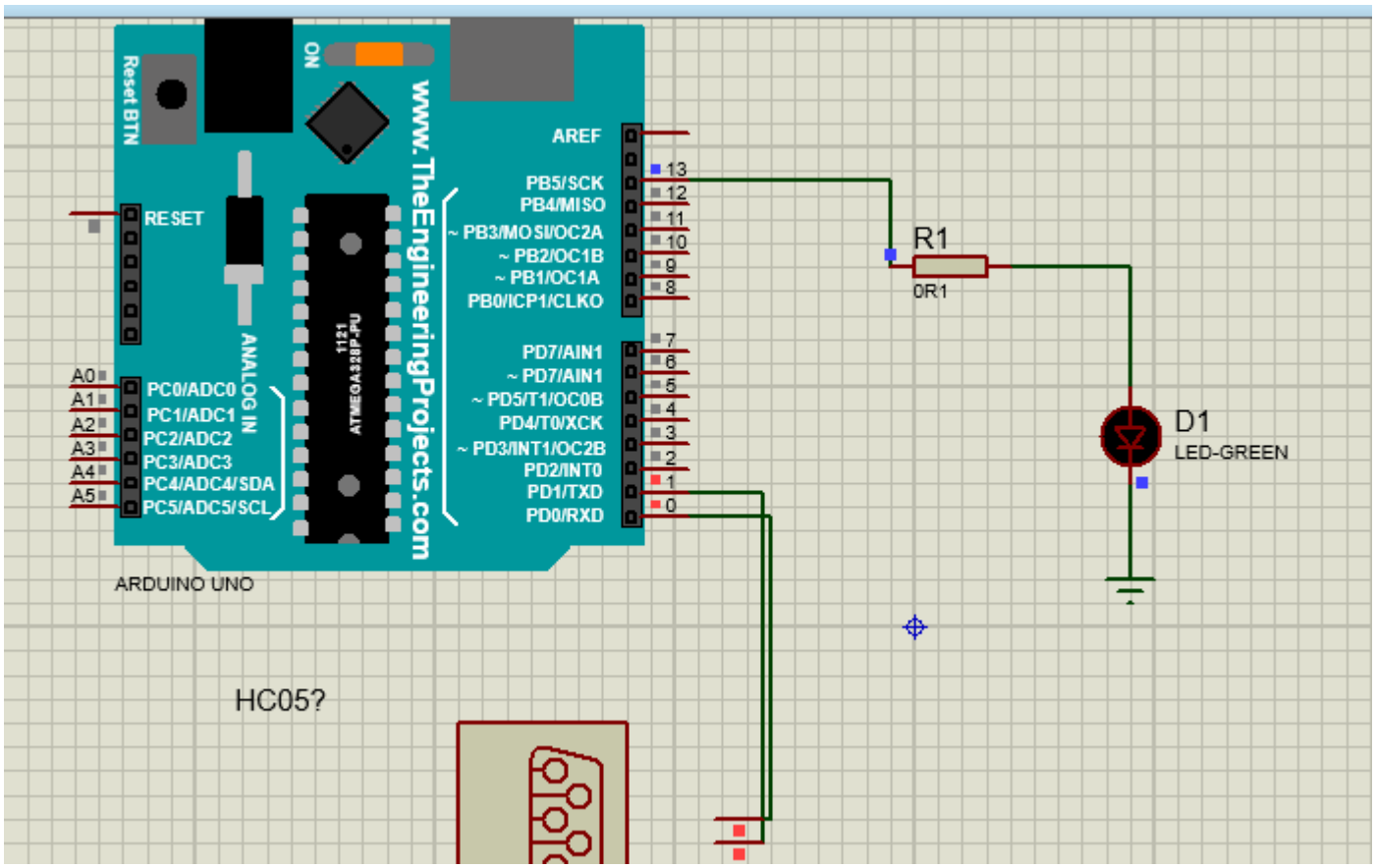
10. Verify: Check the simulation results to ensure that the LED is turning on/off successfully.

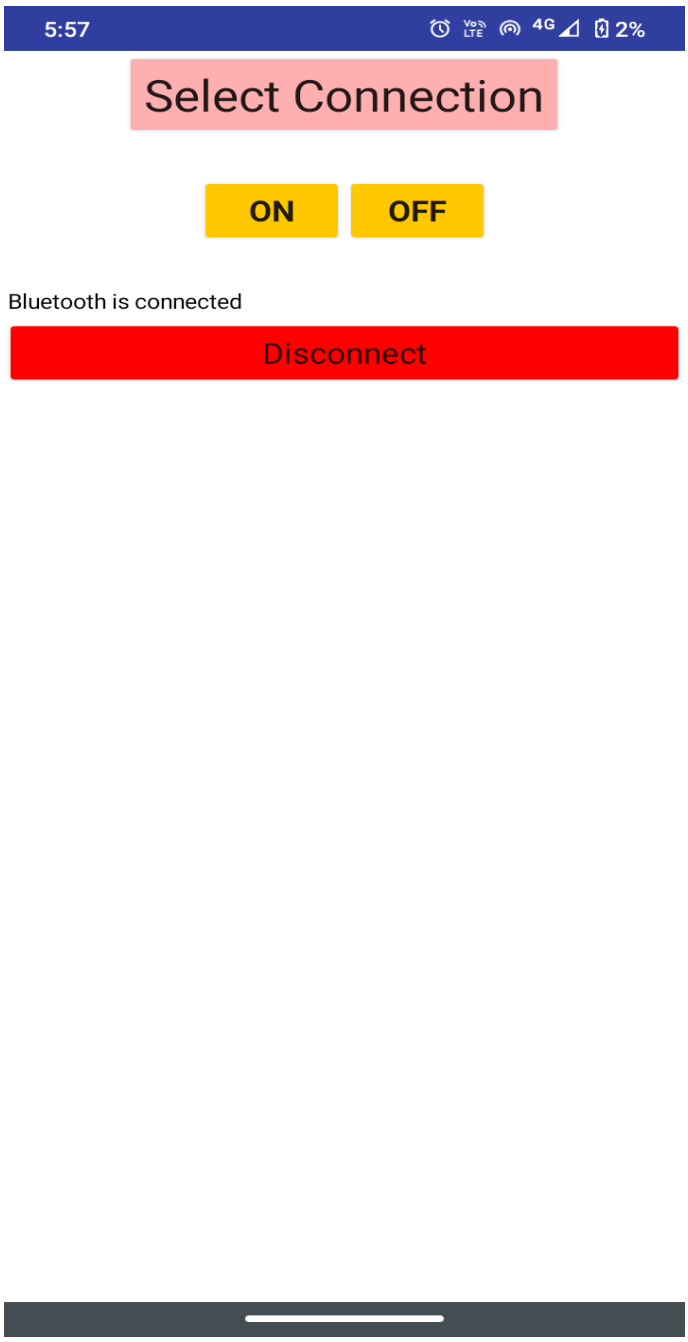
Arduino Sketch:

```
char inputByte;
void setup() {
 Serial.begin(9600);
 pinMode(13,OUTPUT);

}

void loop() {
while(Serial.available()>0){
  inputByte= Serial.read();
  Serial.println(inputByte);
  if (inputByte=='1'){
  digitalWrite(13,HIGH);
  }
  else if (inputByte=='0'){
  digitalWrite(13,LOW);
  }
  }
}
```

OUTPUT:


Reset BTN
ON
www.TheEngineeringProjects.com
AREF
RESET
PB5/SCK
PB4/MISO
~ PB3/MOSI/OC2A
~ PB2/OC1B
~ PB1/OC1A
PB0/ICP1/CLKO
ANALOG IN
A0
A1
A2
A3
A4
A5
PC0/ADC0
PC1/ADC1
PC2/ADC2
PC3/ADC3
PC4/ADC4/SDA
PC5/ADC5/SCL
PD7/AIN1
~ PD7/AIN1
~ PD5/T1/OC0B
PD4/T0/XCK
~ PD3/INT1/OC2B
PD2/INT0
PD1/TXD
PD0/RXD
ARDUINO UNO
R1
0R1
D1
LED-GREEN
HC05?


Reset BTN
ON
www.TheEngineeringProjects.com
AREF
RESET
PB5/SCK
PB4/MISO
~ PB3/MOSI/OC2A
~ PB2/OC1B
~ PB1/OC1A
PB0/ICP1/CLKO
ANALOG IN
A0
A1
A2
A3
A4
A5
PC0/ADC0
PC1/ADC1
PC2/ADC2
PC3/ADC3
PC4/ADC4/SDA
PC5/ADC5/SCL
PD7/AIN1
~ PD7/AIN1
~ PD5/T1/OC0B
PD4/T0/XCK
~ PD3/INT1/OC2B
PD2/INT0
PD1/TXD
PD0/RXD
ARDUINO UNO
R1
0R1
D1
LED-GREEN
HC05?



6.3

Aim:

To explore and understand communication methods used with IoT devices, particularly focusing on Zigbee, to enable efficient and reliable data transmission, control, and monitoring in IoT applications.

Steps:

1. Open Proteus: Launch the Proteus software on your computer.
2. Create New Project: Start a new project by clicking on "File" > "New Project".

3. Add Components:
    - Search for "Arduino" in the component library and add it to your workspace.
    - Add XbeeTEP.IDX,XbeeTEP.LIB library files to the folder C:\Program Files (x86)\ Labcenter Electronics\Proteus 8 Professional\DATA\LIBRARY, if they do not already exist.
    - Search for an XBee module component and add it to your workspace.
4. Wire Connections:
    - Connect the necessary pins on the Arduino board to the corresponding pins on the XBee module.
    - Typically, you'll need to connect the RX pin of the XBee module to a digital pin (e.g., pin 2) on the Arduino for receiving data, and the TX pin of the XBee module to another digital pin (e.g., pin 3) on the Arduino for transmitting data.

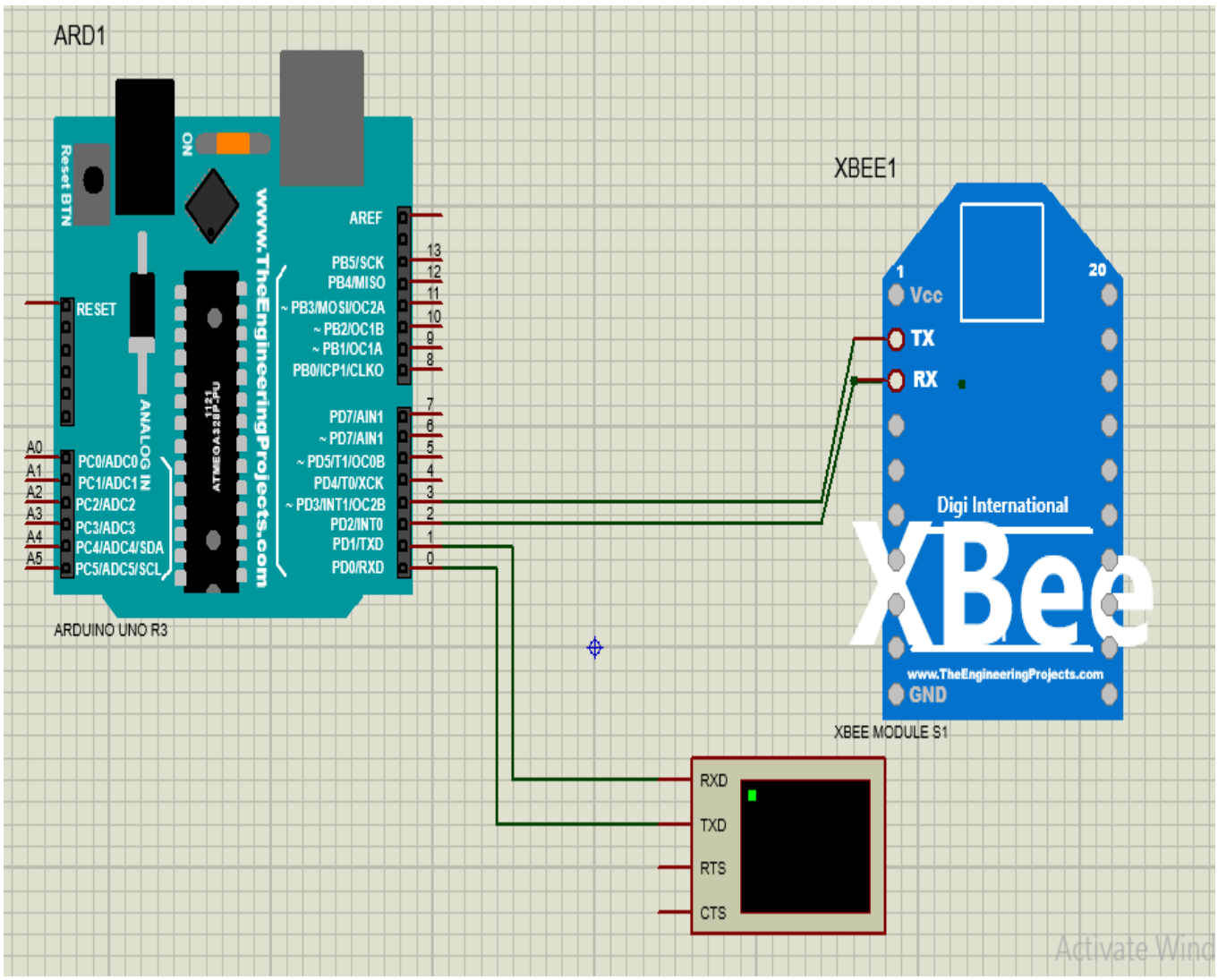
5. Virtual Terminal:
    - Use the Virtual Terminal component in Proteus to monitor serial communication. To add a Virtual Terminal:
    - Search for "Virtual Terminal" in the component library and add it to your workspace.
    - Connect the RX pin of the Virtual Terminal to the TX pin of the Arduino (and vice versa).
    - When you run the simulation, you should see the serial output messages displayed in the Virtual Terminal window.
    5. Write Arduino Sketch:
    - Open the Arduino IDE or any text editor and write the Arduino sketch to communicate with the XBee module.
    - Use the Serial library to communicate with the XBee module via the Arduino's serial ports.
6. Save Arduino Sketch: Save the Arduino sketch with a suitable name and with the ".ino" extension.
7. Add Arduino Sketch to Arduino Board:
    - In Proteus, double-click on the Arduino component to open its properties.
    - Navigate to the "Program File" field and browse to select the Arduino sketch you've saved.
    - Click "OK" to apply the changes.
8. Run Simulation: Run the simulation in Proteus by clicking on the "Play" button or pressing F5.
    - The Arduino sketch should execute, and you should be able to send and receive data between the Arduino and the XBee module.

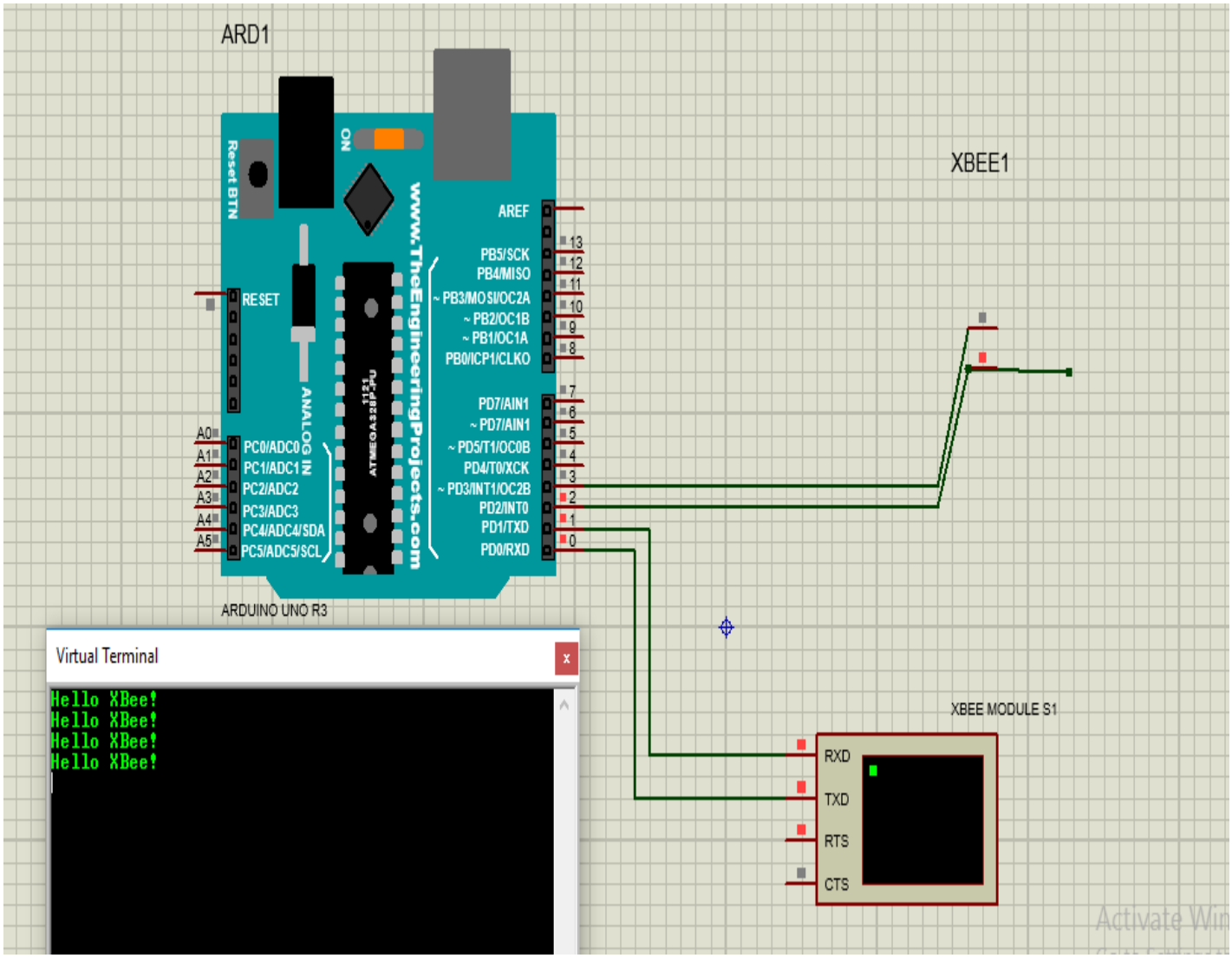
9. Verify Operation: Verify that the system behaves as expected in the simulation, transmitting and receiving data between the Arduino and the XBee module.

Program

```
void setup() {
  Serial.begin(9600); // Set the baud rate to 9600
}

void loop() {
  // Send data to XBee module
  Serial.println("Hello XBee!");

  // Receive data from XBee module
  if (Serial.available() > 0) {
    String receivedData = Serial.readString();
    Serial.println("Received data: " + receivedData);
  }

  delay(1000); // Delay for 1 second
}
```

OUTPUT:


ARD1
Reset BTN
ON
RESET
ANALOG IN
www.TheEngineeringProjects.com
AREF
PB5/SCK
PB4/MISO
~ PB3/MOSI/OC2A
~ PB2/OC1B
~ PB1/OC1A
PB0/ICP1/CLKO
PD7/AIN1
~ PD7/AIN1
~ PD5/T1/OC0B
PD4/T0/XCK
~ PD3/INT1/OC2B
PD2/INT0
PD1/TXD
PD0/RXD
A0
A1
A2
A3
A4
A5
PC0/ADC0
PC1/ADC1
PC2/ADC2
PC3/ADC3
PC4/ADC4/SDA
PC5/ADC5/SCL
ARDUINO UNO R3
XBEE1
1
20
Vcc
TX
RX
Digi International
XBee
www.TheEngineeringProjects.com
GND
XBEE MODULE S1
RXD
TXD
RTS
CTS



Result:

Thus the communication methods (GSM,Bluetooth,Zigbee) with IoT devices have been successfully executed and the output is verified.

Exercise 7: Introduction to Raspberry PI platform and python programming

Aim:

To provide an introductory overview of the Raspberry Pi platform and Python programming, equipping beginners with foundational knowledge to explore hardware interfacing and develop basic projects on the Raspberry Pi using Python programming language.

Steps:

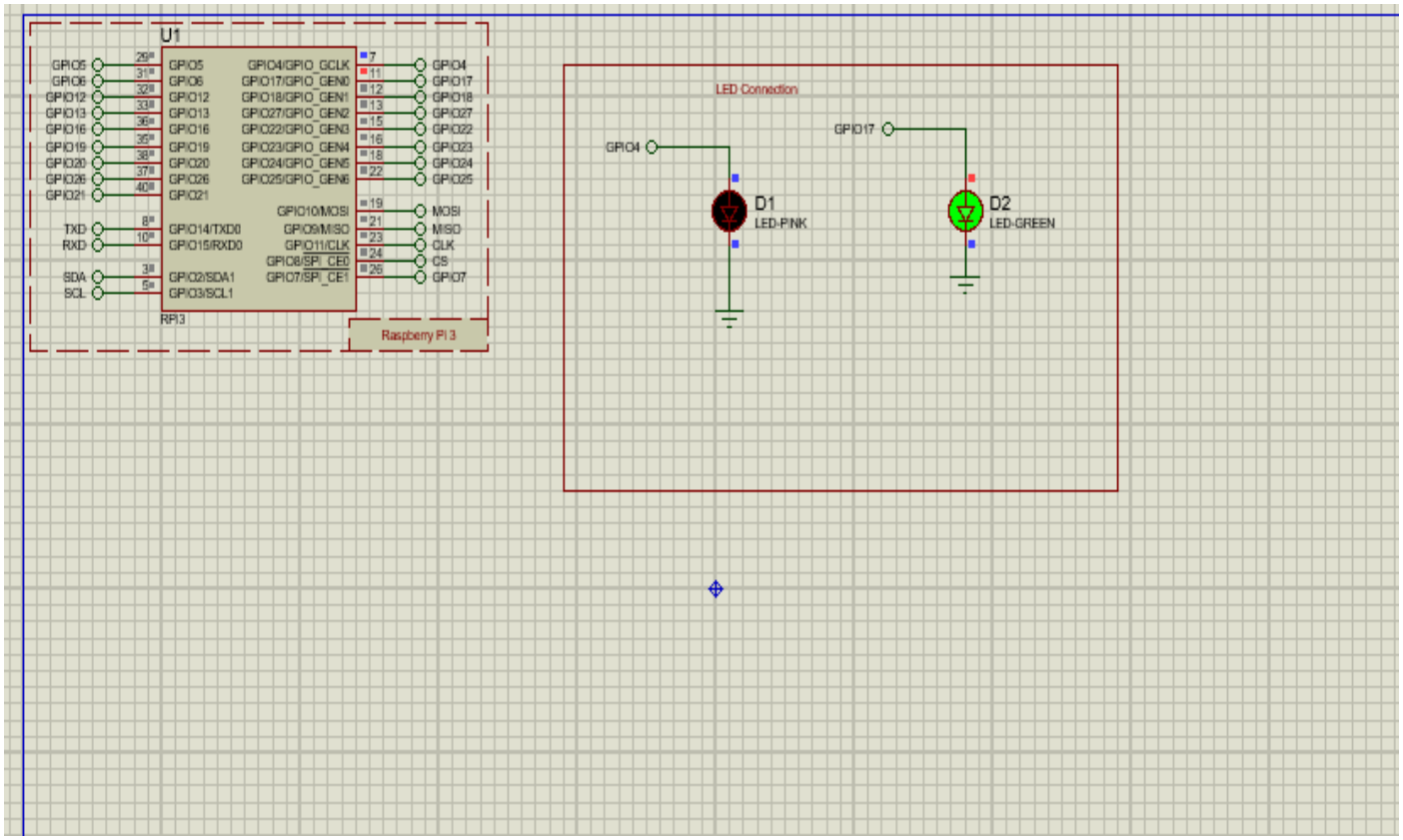
1. Open Proteus: Launch Proteus software on your computer.
2. Create New Project: Start a new project by clicking on "File" > "New Project" and choose create firmware project as Raspberry Pi..
3. Add Components:
    - Search for an LED component and add 2 LEDs to your workspace.
4. Wire Connections:
    - Connect the anode (longer leg) of the one LED to one of the GPIO pins on the Raspberry Pi (e.g., GPIO 7).
    - Connect the cathode (shorter leg) of the LED to the ground (GND) pin on the Raspberry Pi.
    - Connect the anode (longer leg) of the one LED to one of the GPIO pins on the Raspberry Pi (e.g., GPIO 11).
    - Connect the cathode (shorter leg) of the LED to the ground (GND) pin on the Raspberry Pi.
5. Write Python Script:
    - Open a text editor or Python IDE and write a Python script to control the LED.
6. Save Python Script: Save the Python script with a suitable name and with the ".py" extension.
7. Add Python Script to Raspberry Pi:
    - In Proteus, double-click on the Raspberry Pi component to open its properties.
    - Navigate to the "Program File" field and browse to select the Python script you've saved.
    - Click "OK" to apply the changes.
8. Run Simulation: Run the simulation in Proteus by clicking on the "Play" button or pressing F5.
    - The LED should start blinking according to the Python script.
9. Verify Operation: Verify that the LED is blinking as expected. You can adjust the blink rate by modifying the time.sleep() durations in the Python script.

Program:

```
import time

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)
```

```
LED_Red = 7

LED_Yellow= 11

GPIO.setup(LED_Red, GPIO.OUT)

GPIO.setup(LED_Yellow, GPIO.OUT)


 while 1:

   GPIO.output(LED_Red, True)

   time.sleep(0.2)

   GPIO.output(LED_Yellow, True)

   time.sleep(.1)

   GPIO.output(LED_Red, False)

   time.sleep(.1)

   GPIO.output(LED_Yellow, False)

   time.sleep(.1)
```

OUTPUT:



Result:

Thus the LED Blinking using Raspberry Pi and python has been successfully completed and the output is verified.

Exercise 8: Interfacing sensors with Raspberry PI

8.1

Aim: To design and implement a temperature monitoring system using a temperature sensor interfaced with Raspberry Pi, aiming to collect real-time temperature data, display it on the Raspberry Pi.

Steps:

1. Open Proteus: Launch the Proteus software on your computer.
2. Create New Project: Start a new project by clicking on "File" > "New Project".
3. Add Components:
    - Search for "Raspberry Pi" in the component library and add it to your workspace.
    - Search for a temperature sensor component (such as LM35) and add it to your workspace as well.
    - Search for an ADC module component (e.g., MCP3208) and add it to your workspace.
    - Search for an LCD display component (e.g., 16x2 LCD) and add it to your workspace.
4. Wire Connections:
    - Connect the output pin of the temperature sensor to the input pin of the ADC module.
    - Connect the output pins of the ADC module to the Raspberry Pi's GPIO pins (e.g., SPI pins for MCP3208).
    - Connect the LCD display to the Raspberry Pi's GPIO pins for data and control signals.
5. Write Python Script:
    - Open a text editor or Python IDE and write a Python script to read the temperature sensor data.
    - Use the appropriate Python libraries to interface with the temperature sensor. For example, if you're using the LM35 sensor, you can use the RPi.GPIO library for GPIO access and time library for timing, and read analog values directly.
6. Save Python Script: Save the Python script with a suitable name and with the ".py" extension.
7. Add Python Script to Raspberry Pi:
    - In Proteus, double-click on the Raspberry Pi component to open its properties.

- Navigate to the "Program File" field and browse to select the Python script you've saved.
- Click "OK" to apply the changes.

8. Run Simulation: Run the simulation in Proteus by clicking on the "Play" button or pressing F5.
    - The Python script should execute and read the temperature sensor data.
9. Verify Operation: Verify that the temperature readings are displayed correctly in the simulation.

Program:

```
#!/usr/bin/python

import spidev

import time

import os

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)

# Open SPI bus

spi = spidev.SpiDev()

spi.open(0,0)

# Define GPIO to LCD mapping

LCD_RS = 15

LCD_E  = 16

LCD_D4 = 7
```

```
LCD_D5 = 11

LCD_D6 = 12

LCD_D7 = 13

# Define sensor channels

temp_channel  = 0

'''

define pin for lcd

'''

# Timing constants

E_PULSE = 0.0005

E_DELAY = 0.0005

delay = 1

GPIO.setup(LCD_E, GPIO.OUT)  # E

GPIO.setup(LCD_RS, GPIO.OUT) # RS

GPIO.setup(LCD_D4, GPIO.OUT) # DB4

GPIO.setup(LCD_D5, GPIO.OUT) # DB5

GPIO.setup(LCD_D6, GPIO.OUT) # DB6

GPIO.setup(LCD_D7, GPIO.OUT) # DB7

# Define some device constants

LCD_WIDTH = 16    # Maximum characters per line

LCD_CHR = True

LCD_CMD = False
```

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line

LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line

'''

Function Name :lcd_init()

Function Description : this function is used to initialized lcd by sending the different commands

'''

def lcd_init():

  # Initialise display

  lcd_byte(0x33,LCD_CMD) # 110011 Initialise

  lcd_byte(0x32,LCD_CMD) # 110010 Initialise

  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction

  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off

  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size

  lcd_byte(0x01,LCD_CMD) # 000001 Clear display

  time.sleep(E_DELAY)

'''

Function Name :lcd_byte(bits ,mode)

Fuction Name :the main purpose of this function to convert the byte data into bit and send to lcd port

'''

def lcd_byte(bits, mode):

  # Send byte to data pins

  # bits = data

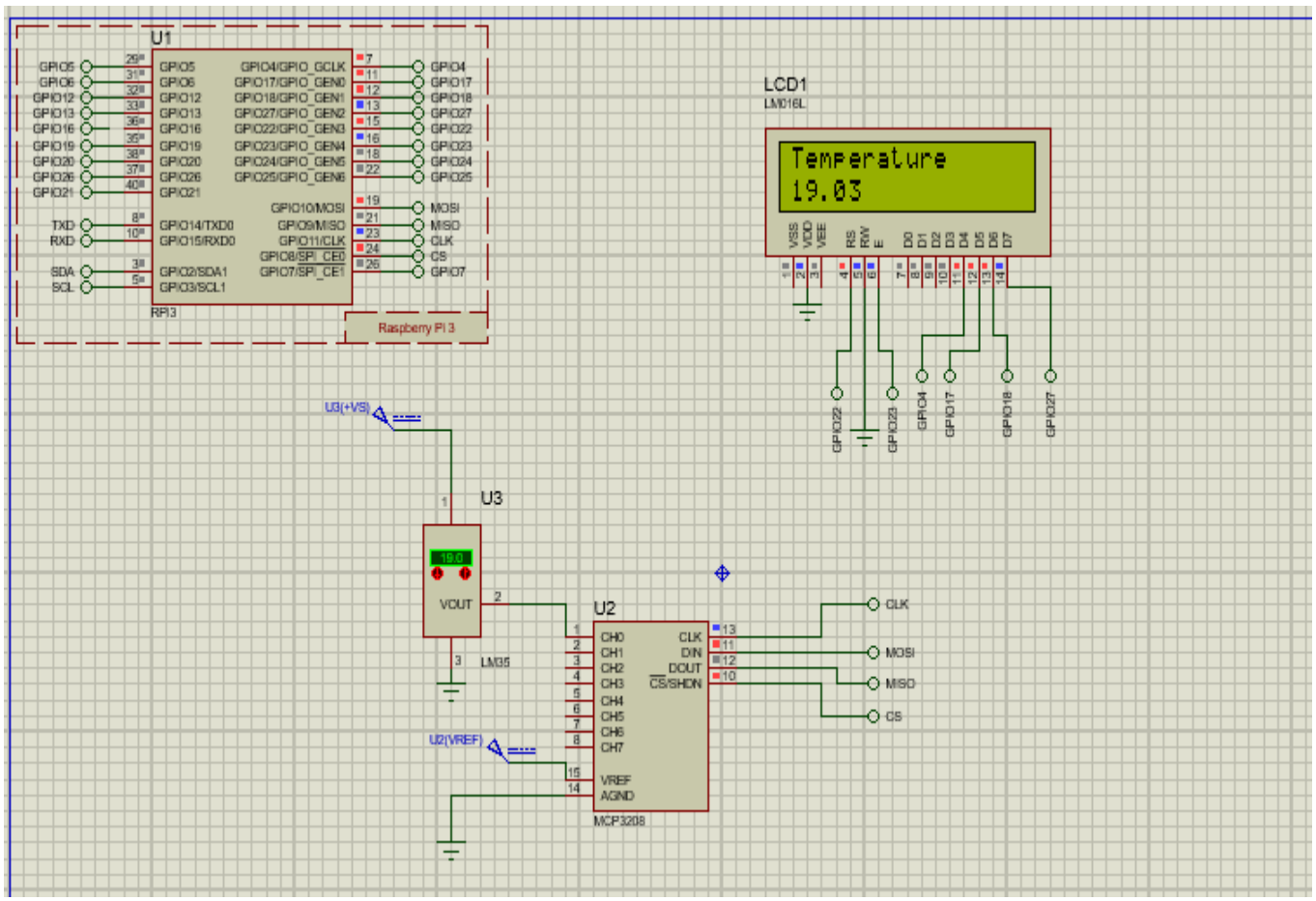
```
# mode = True  for character
#        False for command
 GPIO.output(LCD_RS, mode) # RS
 # High bits
GPIO.output(LCD_D4, False)
GPIO.output(LCD_D5, False)
GPIO.output(LCD_D6, False)
GPIO.output(LCD_D7, False)
if bits&0x10==0x10:
  GPIO.output(LCD_D4, True)
if bits&0x20==0x20:
  GPIO.output(LCD_D5, True)
if bits&0x40==0x40:
  GPIO.output(LCD_D6, True)
if bits&0x80==0x80:
  GPIO.output(LCD_D7, True)
 # Toggle 'Enable' pin
lcd_toggle_enable()
 # Low bits
GPIO.output(LCD_D4, False)
GPIO.output(LCD_D5, False)
GPIO.output(LCD_D6, False)
```

```
  GPIO.output(LCD_D7, False)
  if bits&0x01==0x01:
    GPIO.output(LCD_D4, True)
  if bits&0x02==0x02:
    GPIO.output(LCD_D5, True)
  if bits&0x04==0x04:
    GPIO.output(LCD_D6, True)
  if bits&0x08==0x08:
    GPIO.output(LCD_D7, True)

  # Toggle 'Enable' pin
  lcd_toggle_enable()
'''
Function Name : lcd_toggle_enable()
Function Description:basically this is used to toggle Enable pin
'''
def lcd_toggle_enable():
  # Toggle enable
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
```

```
  time.sleep(E_DELAY)
'''
Function Name :lcd_string(message,line)
Function  Description :print the data on lcd
'''
def lcd_string(message,line):
  # Send string to display
  message = message.ljust(LCD_WIDTH," ")
 lcd_byte(line, LCD_CMD)
 for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)
# Function to read SPI data from MCP3008 chip
# Channel must be an integer 0-7
def ReadChannel(channel):
  adc = spi.xfer2([1,(8+channel)<<4,0])
  data = ((adc[1]&3) << 8) + adc[2]
  return data
# Function to calculate temperature from
# TMP36 data, rounded to specified
# number of decimal places.
def ConvertTemp(data,places):
   # ADC Value
```

```
  temp = ((data * 330)/float(1023))
  temp = round(temp,places)
  return temp

 # Define delay between readings
delay = 5
lcd_init()
lcd_string("welcome ",LCD_LINE_1)
time.sleep(.2)
while 1:
  temp_level = ReadChannel(temp_channel)
  temp       = ConvertTemp(temp_level,2)
   # Print out results
  lcd_string("Temperature  ",LCD_LINE_1)
  lcd_string(str(temp),LCD_LINE_2)
  time.sleep(.1)
```

OUTPUT:



8.2

Aim:

To construct a motion detection system utilizing an infrared (IR) sensor interfaced with Raspberry Pi, aiming to detect the presence of nearby objects or individuals, for enhanced automation or security purposes.

Steps:

1. Open Proteus: Launch the Proteus software on your computer.
2. Create New Project: Start a new project by clicking on "File" > "New Project".
3. Add Components:
    - Search for "Raspberry Pi" in the component library and add it to your workspace.
    - Add InfraredSensorsTEP.IDX,InfraredSensorsTEP.LIB, InfraredSensorsTEP.hex library files to the folder C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\DATA\LIBRARY ,if they do not already exist.

- Search for an IR sensor component and add it to your workspace.
- Search for an LCD display component (e.g., 16x2 LCD) and add it to your workspace.

4. Wire Connections:
    - Connect the output pin of the IR sensor to one of the GPIO pins on the Raspberry Pi (e.g., GPIO 24).
    - Connect the IR sensor's VCC pin to the 3.3V pin on the Raspberry Pi.
    - Connect the IR sensor's GND pin to the ground (GND) pin on the Raspberry Pi.
    - Connect the LCD display to the Raspberry Pi's GPIO pins for data and control signals.
5. Write Python Script:
    - Open a text editor or Python IDE and write a Python script to detect IR sensor input and display it on the LCD display.
    - Use the appropriate Python libraries to interface with the IR sensor and the LCD display. For example, you can use the RPi.GPIO library for GPIO access.
6. Save Python Script: Save the Python script with a suitable name and with the ".py" extension.
7. Add Python Script to Raspberry Pi:
    - In Proteus, double-click on the Raspberry Pi component to open its properties.
    - Navigate to the "Program File" field and browse to select the Python script you've saved.
    - Click "OK" to apply the changes.
8. Run Simulation: Run the simulation in Proteus by clicking on the "Play" button or pressing F5.
    - The Python script should execute, and the LCD display should show whether an object is detected by the IR sensor.
9. Verify Operation: Verify that the system behaves as expected in the simulation, detecting object presence and displaying the appropriate message on the LCD display.

PROGRAM:

```
#!/usr/bin/python

import time

import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)

'''

define pin for lcd

'''

# Timing constants

E_PULSE = 0.0005

E_DELAY = 0.0005

delay = 1

buzzer=37

GPIO.setup(buzzer, GPIO.OUT)


# Define GPIO to LCD mapping

LCD_RS = 7

LCD_E  = 11

LCD_D4 = 12

LCD_D5 = 13

LCD_D6 = 15

LCD_D7 = 16
```

```
IR_Sensor = 18

GPIO.setup(LCD_E, GPIO.OUT)  # E

GPIO.setup(LCD_RS, GPIO.OUT) # RS

GPIO.setup(LCD_D4, GPIO.OUT) # DB4

GPIO.setup(LCD_D5, GPIO.OUT) # DB5

GPIO.setup(LCD_D6, GPIO.OUT) # DB6

GPIO.setup(LCD_D7, GPIO.OUT) # DB7

GPIO.setup(IR_Sensor, GPIO.IN) # DB7

# Define some device constants

LCD_WIDTH = 16    # Maximum characters per line

LCD_CHR = True

LCD_CMD = False

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line

LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line

'''

Function Name :lcd_init()

Function Description : this function is used to initialized lcd by sending the different commands

'''

def lcd_init():

  # Initialise display

  lcd_byte(0x33,LCD_CMD) # 110011 Initialise

  lcd_byte(0x32,LCD_CMD) # 110010 Initialise
```

```
  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction

  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off

  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size

  lcd_byte(0x01,LCD_CMD) # 000001 Clear display

  time.sleep(E_DELAY)
'''
```

Function Name :lcd_byte(bits ,mode)

Fuction Name :the main purpose of this function to convert the byte data into bit and send to lcd port

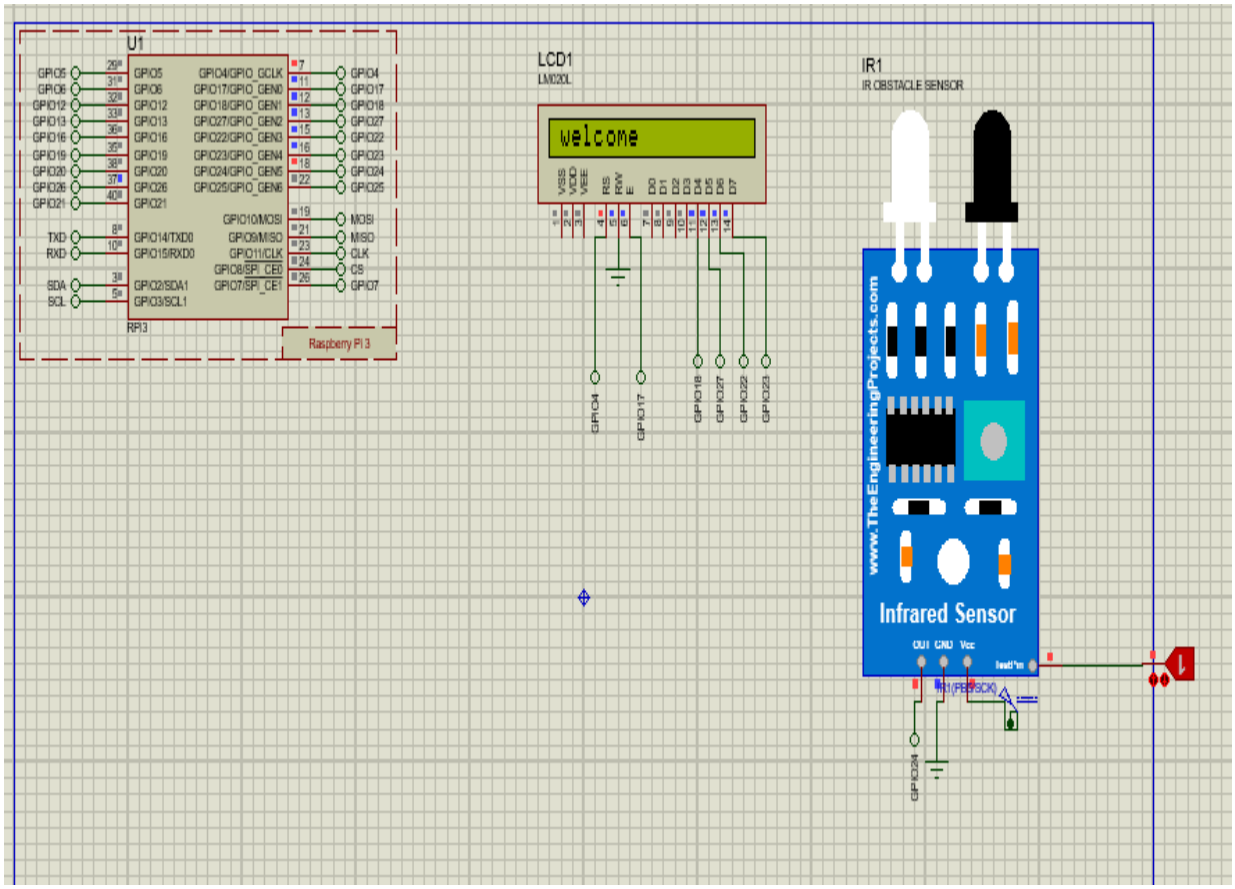
```
'''
def lcd_byte(bits, mode):

  # Send byte to data pins

  # bits = data

  # mode = True  for character

  #        False for command


  GPIO.output(LCD_RS, mode) # RS


  # High bits

  GPIO.output(LCD_D4, False)

  GPIO.output(LCD_D5, False)

  GPIO.output(LCD_D6, False)

  GPIO.output(LCD_D7, False)
```

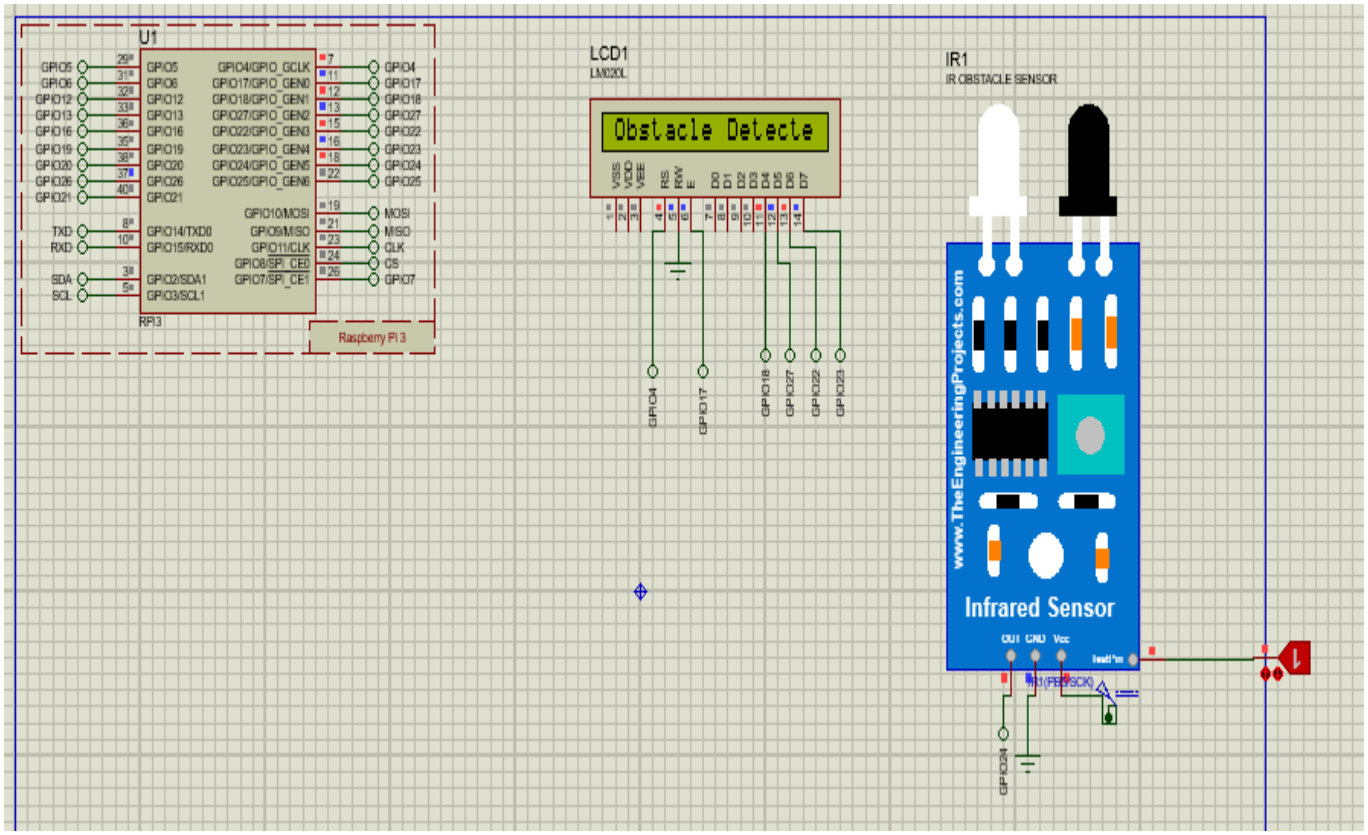
```
if bits&0x10==0x10:
  GPIO.output(LCD_D4, True)
if bits&0x20==0x20:
  GPIO.output(LCD_D5, True)
if bits&0x40==0x40:
  GPIO.output(LCD_D6, True)
if bits&0x80==0x80:
  GPIO.output(LCD_D7, True)

# Toggle 'Enable' pin
lcd_toggle_enable()

# Low bits
GPIO.output(LCD_D4, False)
GPIO.output(LCD_D5, False)
GPIO.output(LCD_D6, False)
GPIO.output(LCD_D7, False)
if bits&0x01==0x01:
  GPIO.output(LCD_D4, True)
if bits&0x02==0x02:
  GPIO.output(LCD_D5, True)
if bits&0x04==0x04:
```

```
    GPIO.output(LCD_D6, True)
  if bits&0x08==0x08:
    GPIO.output(LCD_D7, True)


  # Toggle 'Enable' pin
  lcd_toggle_enable()
'''
Function Name : lcd_toggle_enable()
Function Description:basically this is used to toggle Enable pin
'''
def lcd_toggle_enable():
  # Toggle enable
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)
'''
Function Name :lcd_string(message,line)
Function  Description :print the data on lcd
'''
def lcd_string(message,line):
```

```
  # Send string to display
   message = message.ljust(LCD_WIDTH," ")
   lcd_byte(line, LCD_CMD)

  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)

lcd_init()
lcd_string("welcome ",LCD_LINE_1)
time.sleep(2)
# Define delay between readings
delay = 5
while 1:
  # Print out results
  if GPIO.input(IR_Sensor):
   lcd_string("Obstacle Detected  ",LCD_LINE_1)
   time.sleep(1)
  else:
    lcd_string("Obstacle Removed  ",LCD_LINE_1)
    time.sleep(1)
```

OUTPUT:


U1
RPI3
Raspberry PI 3
LCD1
LM020L
welcome
IR1
IR OBSTACLE SENSOR
www.TheEngineeringProjects.com
Infrared Sensor
OUT GND Vcc


U1
RPI3
Raspberry PI 3
LCD1
LM020L
Obstacle Detecte
IR1
IR OBSTACLE SENSOR
www.TheEngineeringProjects.com
Infrared Sensor
OUT GND Vcc

Result:

Thus the interfacing sensors with Raspberry PI are executed successfully and the output is verified.

Exercise 9: Setup a cloud platform to log the data

Aim

To set up a ThingSpeak cloud to log data from IoT devices and sensors, analyze it, and visualize it in real-time.

Steps:

1.Sign Up /Login to ThinkSpeak

Go to the ThingSpeak website (https://thingspeak.com/) and sign up for a new account or log in if you already have one.

2.Create a New Channel

- After logging in, click on the "Channels" tab at the top of the page.
- Click on the “New Channel” button.
- Enter the necessary information such as the name, description, and field labels for your channel.
- Configure additional settings like the number of fields, update rate, and privacy settings according to your requirements.
- Click on the "Save Channel" button to create the channel

3. Get Write API Key

- Once the channel is created, click on the "API Keys" tab.
- You will find the "Write API Key" listed there. Copy this key as you will need it to send data to your channel.

4. Configure Data Source (Device)

- Configure your IoT device or sensor to send data to ThingSpeak.
- Use the Write API Key obtained in the previous step to authenticate data transmission
    - Depending on your device and communication protocol (HTTP, MQTT, etc.), configure it to send data to ThingSpeak using the appropriate method.

5.Send Data to ThingSpeak

- Start sending data from your device to ThingSpeak using the Write API Key.
- You can send data using HTTP requests, MQTT, or other supported protocols.

- Make sure to send the data in the format expected by ThingSpeak, typically as HTTP GET or POST requests with parameters for each field.

6. View and Analyze Data:

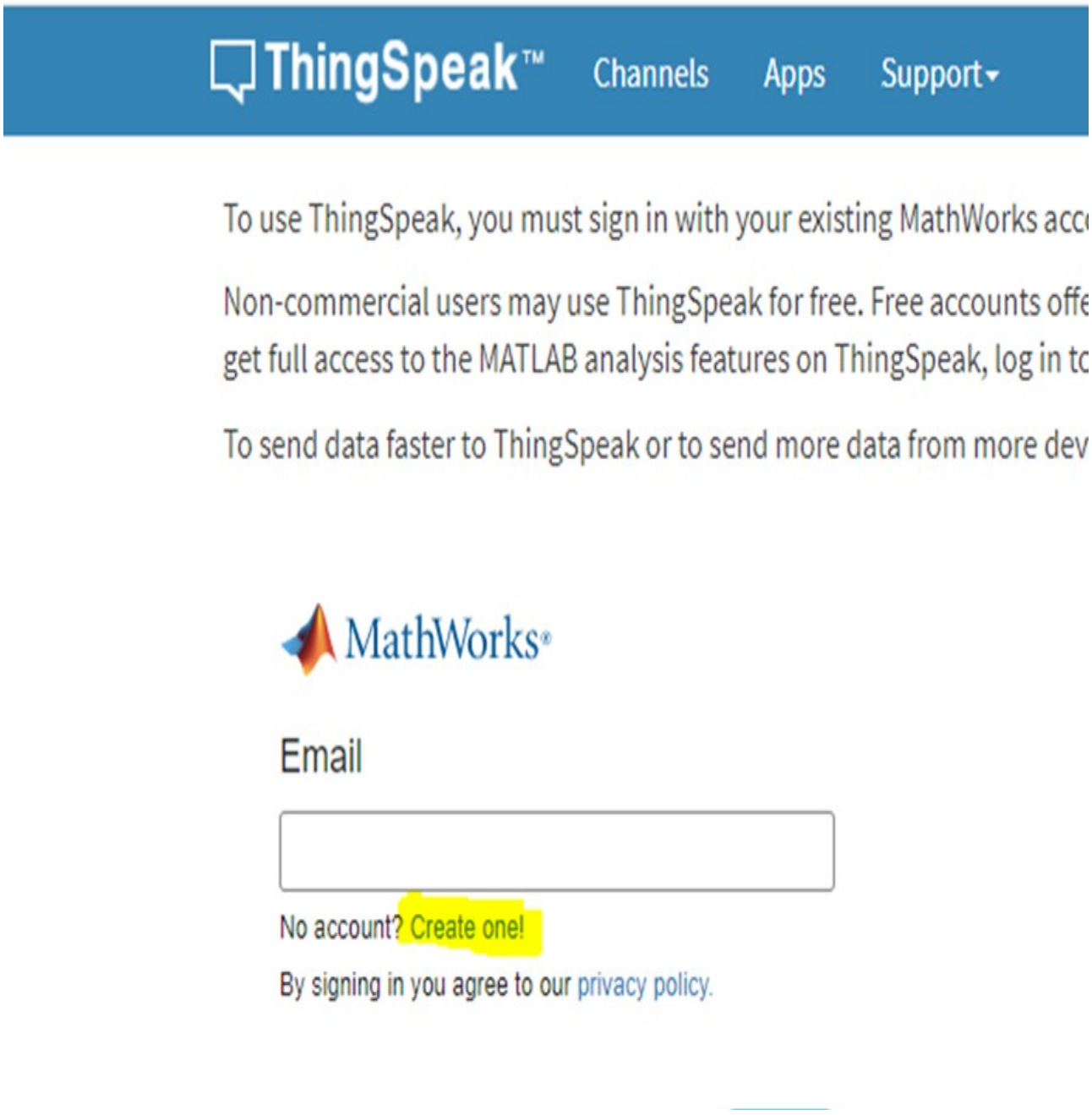
- Once data is being sent to your ThingSpeak channel, you can view and analyze it using the ThingSpeak web interface.
- Go to your channel's page and click on the "Charts" or "Visualizations" tab to see your data plotted on graphs.

OUTPUT:

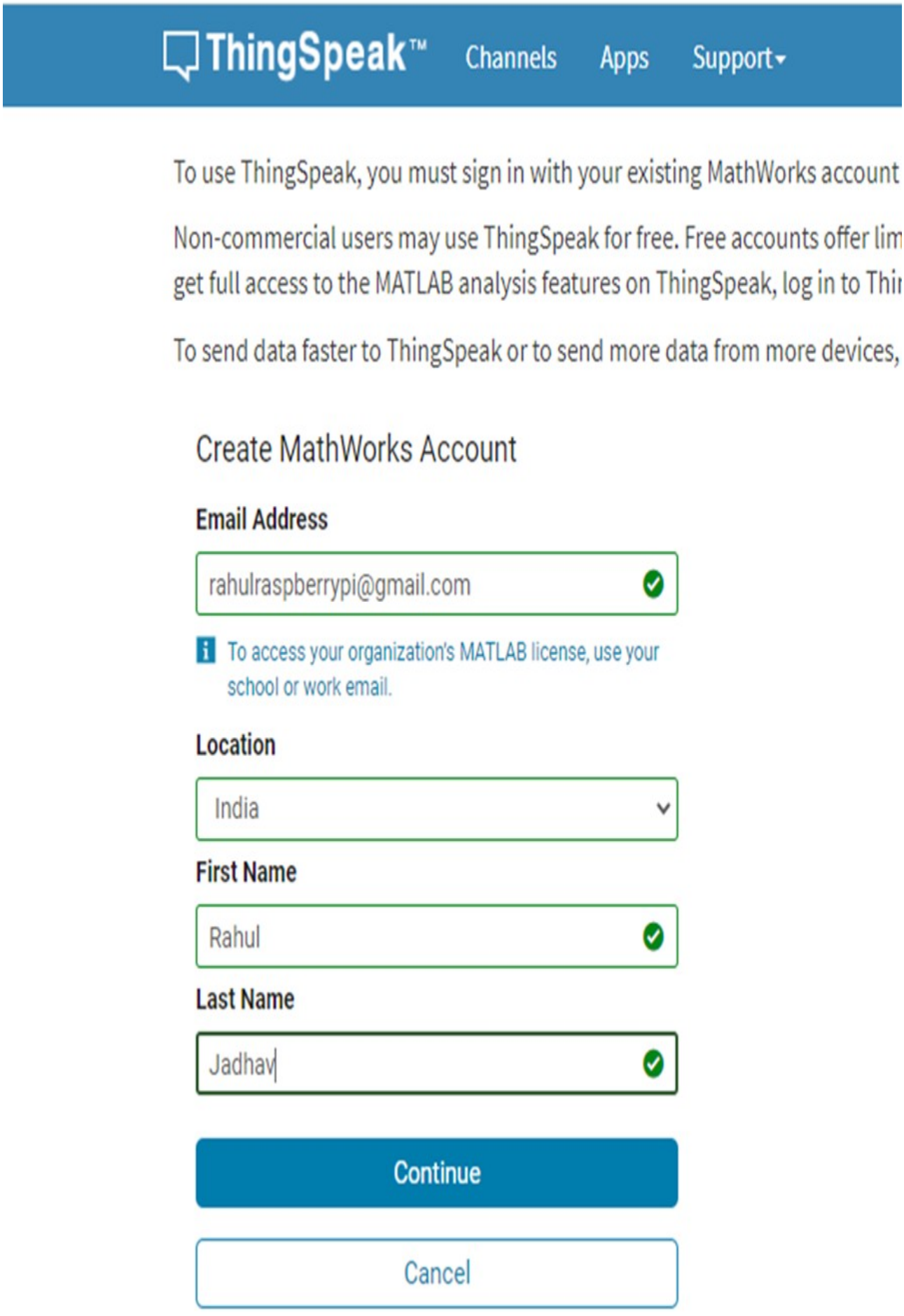
Open thingspeak website to create account ::

https://thingspeak.com/login?skipSSOCheck=true

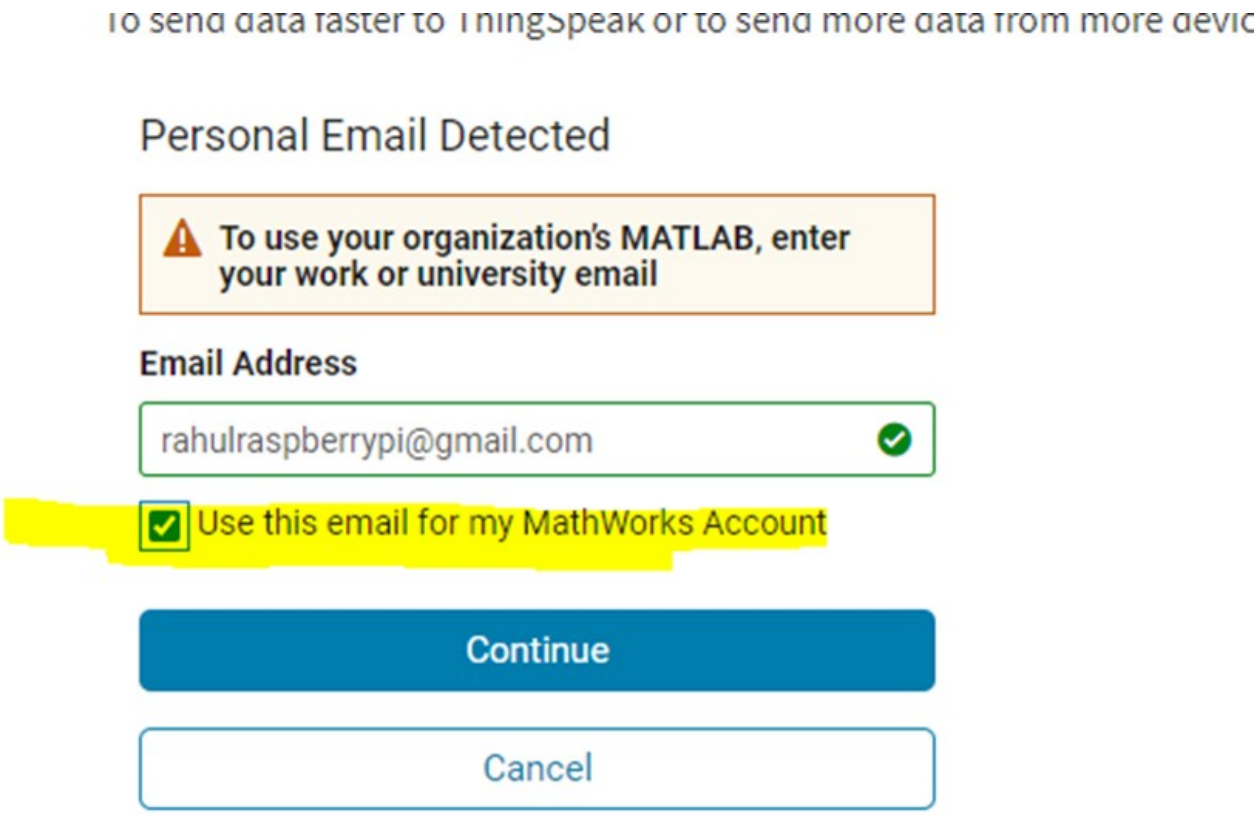
Step1:


ThingSpeak™
Channels
Apps
Support
To use ThingSpeak, you must sign in with your existing MathWorks acc
Non-commercial users may use ThingSpeak for free. Free accounts offe
get full access to the MATLAB analysis features on ThingSpeak, log in to
To send data faster to ThingSpeak or to send more data from more dev
MathWorks
Email
No account? Create one!
By signing in you agree to our privacy policy.

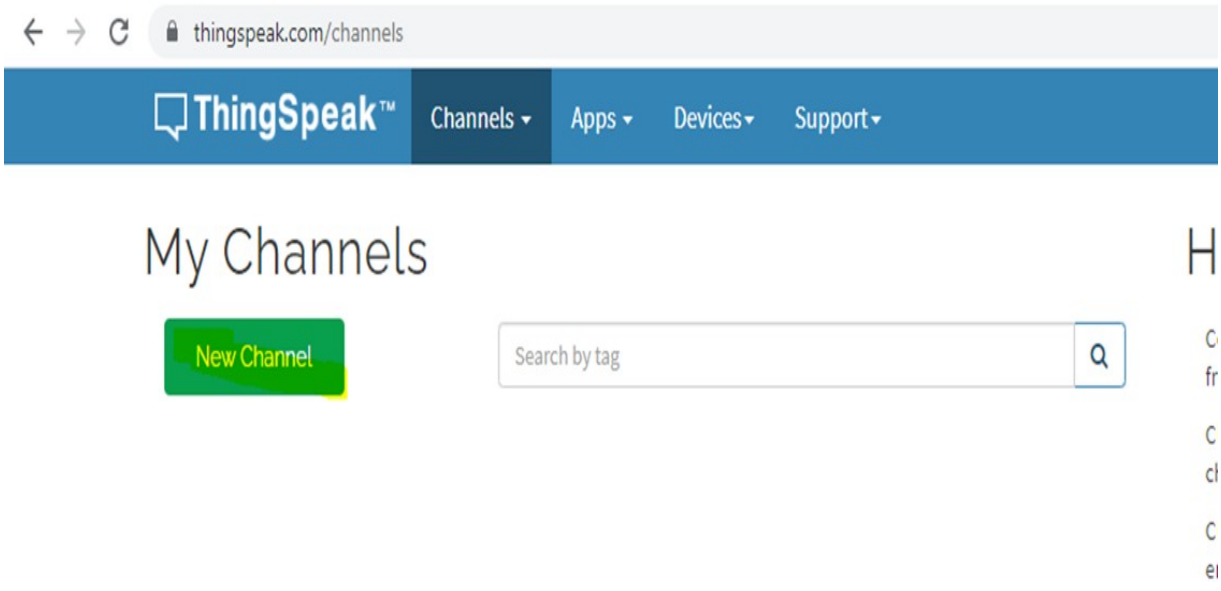
Step 2:


ThingSpeak™
Channels
Apps
Support
To use ThingSpeak, you must sign in with your existing MathWorks account
Non-commercial users may use ThingSpeak for free. Free accounts offer lim
get full access to the MATLAB analysis features on ThingSpeak, log in to Thi
To send data faster to ThingSpeak or to send more data from more devices,
Create MathWorks Account
Email Address
rahulraspberrypi@gmail.com
To access your organization's MATLAB license, use your school or work email.
Location
India
First Name
Rahul
Last Name
Jadhav
Continue
Cancel

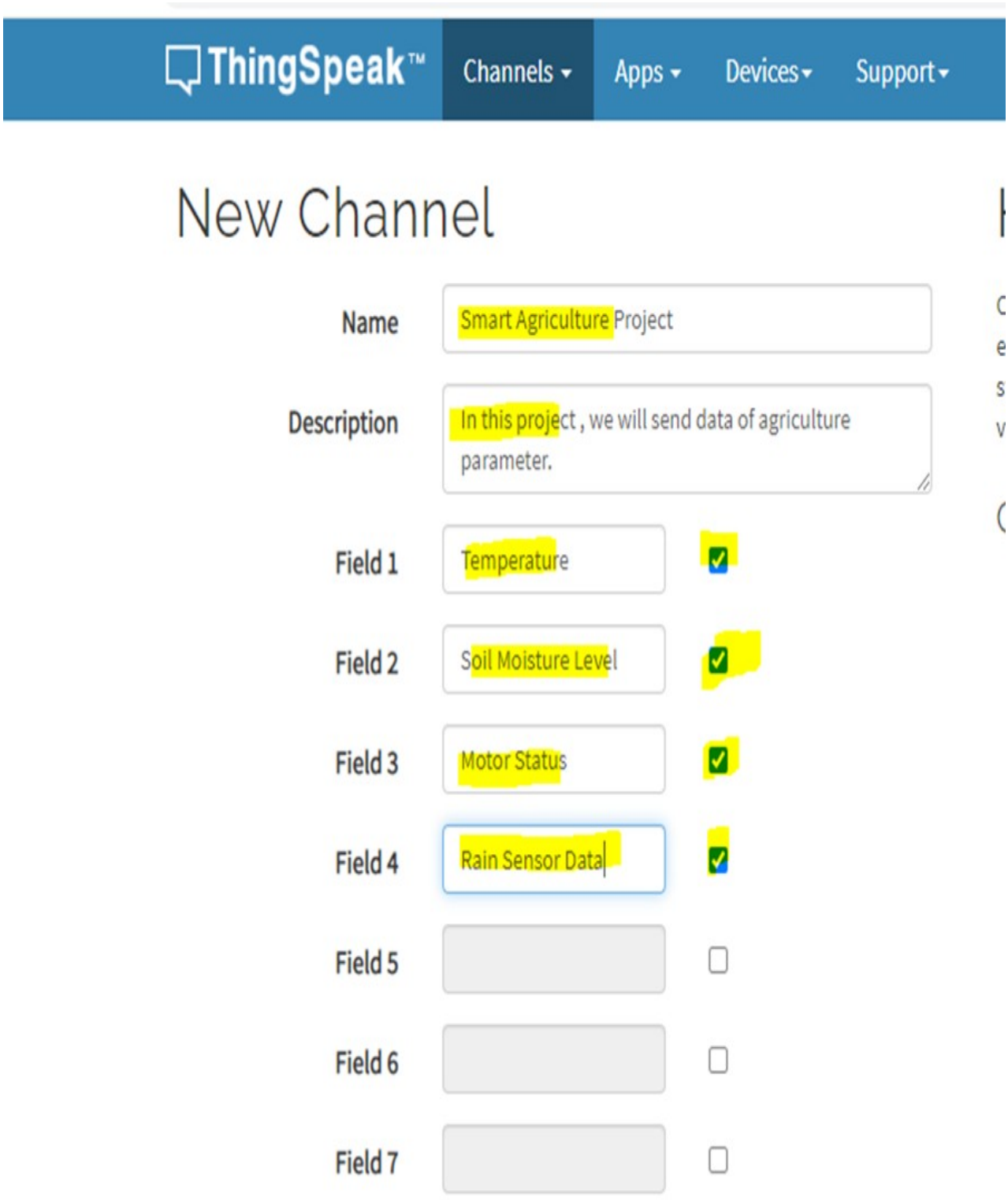
Step 3:

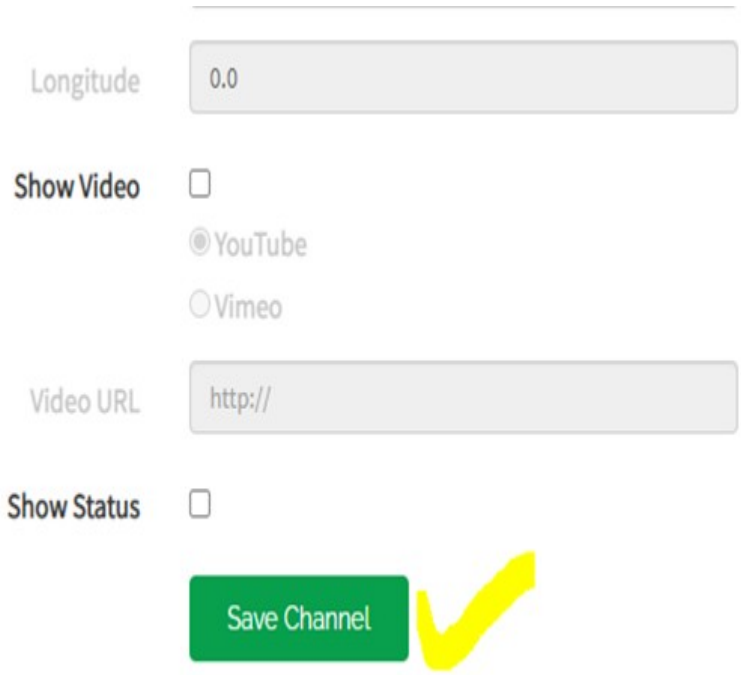

To send data faster to ThingSpeak or to send more data from more devic
Personal Email Detected
To use your organization's MATLAB, enter your work or university email
Email Address
rahulraspberrypi@gmail.com
Use this email for my MathWorks Account
Continue
Cancel

Step 4:

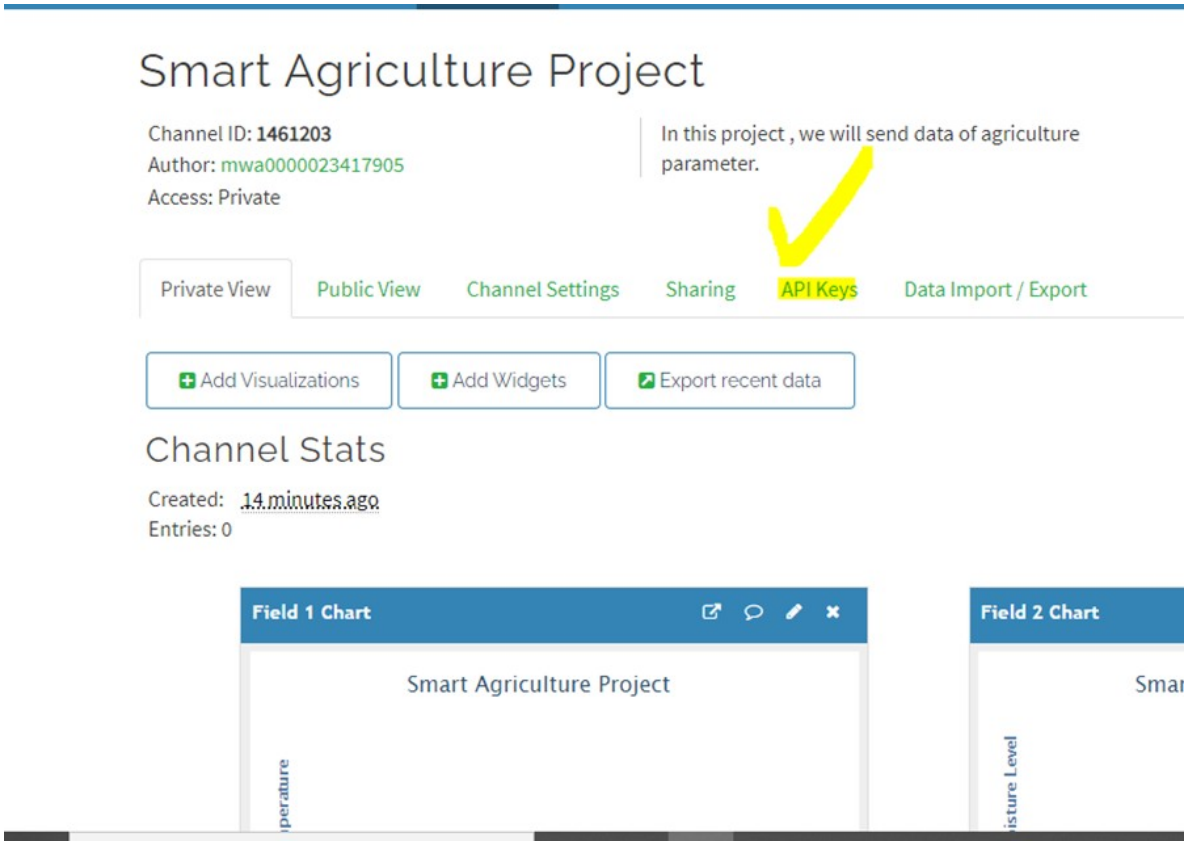

thingspeak.com/channels
ThingSpeak™
Channels
Apps
Devices
Support
My Channels
New Channel
Search by tag

Step 5:


ThingSpeak™
Channels
Apps
Devices
Support
New Channel
Name
Smart Agriculture Project
Description
In this project , we will send data of agriculture parameter.
Field 1
Temperature
Field 2
Soil Moisture Level
Field 3
Motor Status
Field 4
Rain Sensor Data
Field 5
Field 6
Field 7


Longitude
0.0
Show Video
YouTube
Vimeo
Video URL
http://
Show Status
Save Channel

Step 6:


Smart Agriculture Project
Channel ID: 1461203
Author: mwa0000023417905
Access: Private
In this project , we will send data of agriculture parameter.
Private View
Public View
Channel Settings
Sharing
API Keys
Data Import / Export
Add Visualizations
Add Widgets
Export recent data
Channel Stats
Created: 14 minutes ago
Entries: 0
Field 1 Chart
Smart Agriculture Project
Field 2 Chart

Step 8:

Copy write API Key to send data on thingspeak:

HGI92BZUPPGOMWZB

Result:

Thus the setting up of ThingSpeak cloud to log data is done.

Exercise 10: Communicate between Arduino and Raspberry PI using any wireless medium

Aim:

To establish wireless communication between Arduino and Raspberry Pi using a chosen wireless medium, with the goal of facilitating data exchange and interaction between the two platforms.

Steps:

Exercise 11: Log Data using Raspberry PI and upload to the cloud platform

Aim:

To develop a project to log data using Raspberry Pi and upload to the cloud platform.

Steps:

1. Open Proteus: Launch the Proteus software on your computer.
2. Create New Project: Start a new project by clicking on "File" > "New Project".
3. Add Components:
    - Search for "Raspberry Pi" in the component library and add it to your workspace.
4. Write Python Script:
    - Open a text editor or Python IDE and write a Python script.
    - Use the appropriate Python libraries to interface SPI.
    - Write the function to send the value to Thingspeak cloud.
5. Save Python Script: Save the Python script with a suitable name and with the ".py" extension.
6. Add Python Script to Raspberry Pi:
    - In Proteus, double-click on the Raspberry Pi component to open its properties.
    - Navigate to the "Program File" field and browse to select the Python script you've saved.
    - Click "OK" to apply the changes.
7. Run Simulation: Run the simulation in Proteus by clicking on the "Play" button or pressing F5.
    - The Python script should execute, and the value is sent to cloud platform.
8. Verify Operation: Verify that the system behaves as expected in the simulation.

Program:

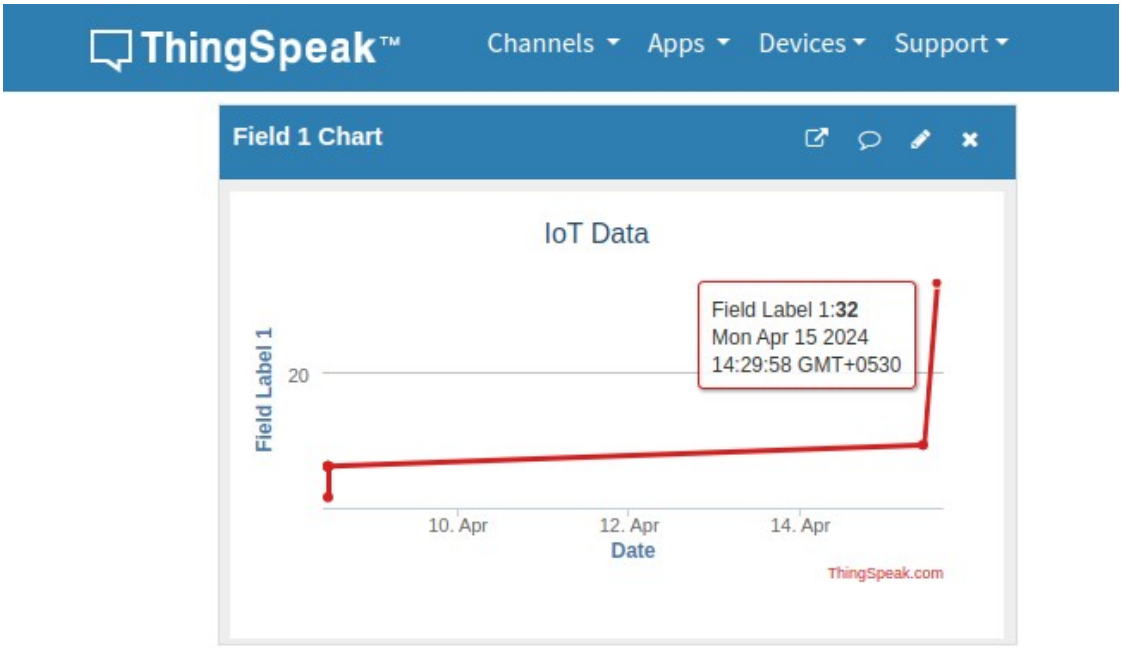
```
# Main.py file generated by New Project wizard

from goto import *
```

```
import time

import var

import pio

import resource

import spidev

import RPi.GPIO as GPIO

import urllib.request

import requests

# Peripheral Configuration Code (do not edit)

#---CONFIG_BEGIN---

import cpu

import FileStore

import VFP


def peripheral_setup () :

# Peripheral Constructors

 pio.cpu=cpu.CPU ()

 pio.storage=FileStore.FileStore ()

 pio.server=VFP.VfpServer ()

 pio.storage.begin ()

 pio.server.begin (0)

# Install interrupt handlers
```

```
def peripheral_loop () :
 pio.server.poll ()

#---CONFIG_END---

# Open SPI bus
spi = spidev.SpiDev()
spi.open(0,0)
def thingspeak_post(temp):
    URl='https://api.thingspeak.com/update?api_key='
    #Enter Your Private Key here
    KEY='24AOZ5TLM9UHE5BO'
    HEADER='&field1={}'.format(temp)
    NEW_URL=URl+KEY+HEADER
    print(NEW_URL)
    data=urllib.request.urlopen(NEW_URL)
    print(data)


# Define delay between readings
# Main function
```

```
def main () :
# Setup
 peripheral_setup()
 peripheral_loop()


# Infinite loop
 while 1 :
  temp       = 32
#Send data on thing speak server
  thingspeak_post(temp)
  pass
# Command line execution
if __name__ == '__main__' :
   main()
```

OUTPUT:



Result:

Thus the logging data and uploading to cloud using raspberry Pi has been executed sucessfully and the output is verified.

Exercise 12: Design an IoT Based System

Aim:

To develop a project using Raspberry PI, sensors and upload to the Thingspeak cloud platform

Steps:

4. Open Proteus: Launch the Proteus software on your computer.
5. Create New Project: Start a new project by clicking on "File" > "New Project".
6. Add Components:
    - Search for "Raspberry Pi" in the component library and add it to your workspace.
    - Search for sensor components and add it to your workspace.
    - Search for an LCD display component (e.g., 16x2 LCD) and add it to your workspace.
4. Wire Connections:
    - Connect the components.
5. Write Python Script:
    - Open a text editor or Python IDE and write a Python script and display it on the LCD display.
    - Use the appropriate Python libraries to interface with the sensor and the LCD display. For example, you can use the RPi.GPIO library for GPIO access.
    - Write the function to send the sensor data to Thingspeak cloud.
6. Save Python Script: Save the Python script with a suitable name and with the ".py" extension.
7. Add Python Script to Raspberry Pi:
    - In Proteus, double-click on the Raspberry Pi component to open its properties.
    - Navigate to the "Program File" field and browse to select the Python script you've saved.
    - Click "OK" to apply the changes.

8. Run Simulation: Run the simulation in Proteus by clicking on the "Play" button or pressing F5.
    - The Python script should execute, and the LCD display should show whether an object is detected by the IR sensor.
9. Verify Operation: Verify that the system behaves as expected in the simulation, detecting object presence and displaying the appropriate message on the LCD display.

PROGRAM:

```
# !/usr/bin/env python3

# Modules

from goto import *

import time

import var

import pio

import resource

import spidev

import RPi.GPIO as GPIO

import urllib.request

import requests


# Peripheral Configuration Code (do not edit)

#---CONFIG_BEGIN---

import cpu

import FileStore

import VFP
```

```
import Ports

def peripheral_setup () :

# Peripheral Constructors

 pio.cpu=cpu.CPU ()

 pio.storage=FileStore.FileStore ()

 pio.server=VFP.VfpServer ()

 pio.uart=Ports.UART ()

 pio.storage.begin ()

 pio.server.begin (0)

# Install interrupt handlers

def peripheral_loop () :

 pass

#---CONFIG_END---

# Open SPI bus

spi = spidev.SpiDev()

spi.open(0,0)

# Define GPIO to LCD mapping

LCD_RS = 4

LCD_E  = 17

LCD_D4 = 18
```

```
LCD_D5 = 27

LCD_D6 = 22

LCD_D7 = 23

Relay_pin= 24

Rain_sensor = 25

# Define sensor channels

temp_channel  = 0

Moisture_channel =1


'''

define pin for lcd

'''

# Timing constants

E_PULSE = 0.0005

E_DELAY = 0.0005

delay = 1


GPIO.setup(LCD_E, GPIO.OUT)  # E

GPIO.setup(LCD_RS, GPIO.OUT) # RS

GPIO.setup(LCD_D4, GPIO.OUT) # DB4

GPIO.setup(LCD_D5, GPIO.OUT) # DB5

GPIO.setup(LCD_D6, GPIO.OUT) # DB6

GPIO.setup(LCD_D7, GPIO.OUT) # DB7
```

```
GPIO.setup(Relay_pin, GPIO.OUT) # Motor_1

GPIO.setup(Rain_sensor, GPIO.IN)

# Define some device constants

LCD_WIDTH = 16    # Maximum characters per line

LCD_CHR = True

LCD_CMD = False

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line

LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line


'''

Function Name :lcd_init()

Function Description : this function is used to initialized lcd by sending the different commands

'''

def lcd_init():

  # Initialise display

  lcd_byte(0x33,LCD_CMD) # 110011 Initialise

  lcd_byte(0x32,LCD_CMD) # 110010 Initialise

  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction

  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off

  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size

  lcd_byte(0x01,LCD_CMD) # 000001 Clear display

  time.sleep(E_DELAY)

'''
```

Function Name :lcd_byte(bits ,mode)

Fuction Name :the main purpose of this function to convert the byte data into bit and send to lcd port

'''

```
def lcd_byte(bits, mode):
  # Send byte to data pins
  # bits = data
  # mode = True  for character
  #        False for command

  GPIO.output(LCD_RS, mode) # RS

  # High bits
  GPIO.output(LCD_D4, False)
  GPIO.output(LCD_D5, False)
  GPIO.output(LCD_D6, False)
  GPIO.output(LCD_D7, False)
  if bits&0x10==0x10:
    GPIO.output(LCD_D4, True)
  if bits&0x20==0x20:
    GPIO.output(LCD_D5, True)
  if bits&0x40==0x40:
    GPIO.output(LCD_D6, True)
```

```
if bits&0x80==0x80:
  GPIO.output(LCD_D7, True)

# Toggle 'Enable' pin
lcd_toggle_enable()

# Low bits
GPIO.output(LCD_D4, False)
GPIO.output(LCD_D5, False)
GPIO.output(LCD_D6, False)
GPIO.output(LCD_D7, False)
if bits&0x01==0x01:
  GPIO.output(LCD_D4, True)
if bits&0x02==0x02:
  GPIO.output(LCD_D5, True)
if bits&0x04==0x04:
  GPIO.output(LCD_D6, True)
if bits&0x08==0x08:
  GPIO.output(LCD_D7, True)

# Toggle 'Enable' pin
lcd_toggle_enable()
```

```
'''
Function Name : lcd_toggle_enable()
Function Description:basically this is used to toggle Enable pin
'''
def lcd_toggle_enable():
  # Toggle enable
  time.sleep(E_DELAY)
  GPIO.output(LCD_E, True)
  time.sleep(E_PULSE)
  GPIO.output(LCD_E, False)
  time.sleep(E_DELAY)
'''
Function Name :lcd_string(message,line)
Function  Description :print the data on lcd
'''
def lcd_string(message,line):
  # Send string to display

  message = message.ljust(LCD_WIDTH," ")

  lcd_byte(line, LCD_CMD)
```

```
  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)

# Function to read SPI data from MCP3008 chip
# Channel must be an integer 0-7
def ReadChannel(channel):
  adc = spi.xfer2([1,(8+channel)<<4,0])
  data = ((adc[1]&3) << 8) + adc[2]
  return data

# Function to calculate temperature from
# TMP36 data, rounded to specified
# number of decimal places.
def ConvertTemp(data,places):
  temp = ((data * 330)/float(1023))
  temp = round(temp,places)
  return temp

def thingspeak_post(temp,moisture_level,motor_status,rain_data):
    URl='https://api.thingspeak.com/update?api_key='
    #Enter Your Private Key here
    KEY='24AOZ5TLM9UHE5BO'
```

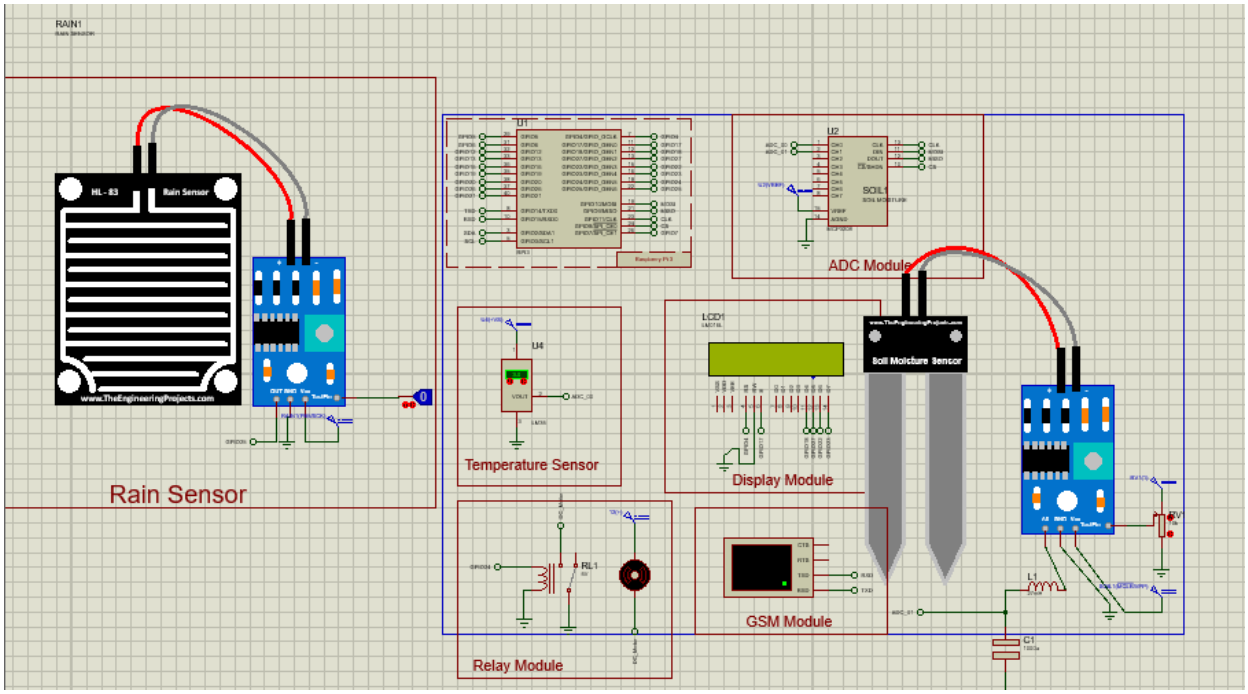
```
    HEADER='&field1={}&field2={}&field3={}&field4={}'.format(temp,moisture_level,motor_st
atus,rain_data)

    NEW_URL=URl+KEY+HEADER

    print(NEW_URL)

    data=urllib.request.urlopen(NEW_URL)

    print(data)



# Define delay between readings

delay = 5

lcd_init()

lcd_string("welcome ",LCD_LINE_1)

time.sleep(1)

lcd_byte(0x01,LCD_CMD) # 000001 Clear display

lcd_string("Smart Irrigation",LCD_LINE_1)

lcd_string("System ",LCD_LINE_2)

time.sleep(1)

lcd_byte(0x01,LCD_CMD) # 000001 Clear display

# Main function

def main () :

# Setup

 peripheral_setup()

 peripheral_loop()

 #Motor Status
```

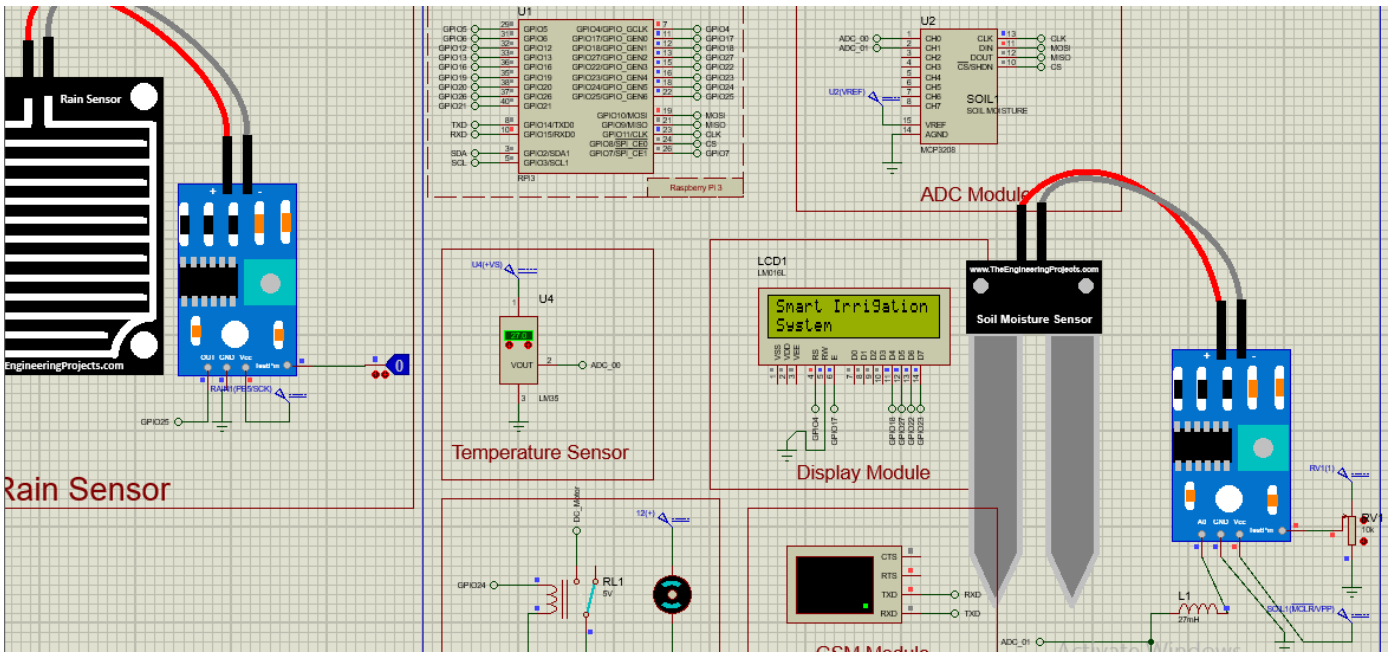
```
 motor_status = 0
# Infinite loop
 while 1 :

  temp_level = ReadChannel(temp_channel)
  temp       = ConvertTemp(temp_level,2)


  # Print out results
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  lcd_string("Temperature  ",LCD_LINE_1)
  lcd_string(str(temp),LCD_LINE_2)
  time.sleep(0.5)


  moisture_level = ReadChannel(Moisture_channel)
  # Print out results
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  lcd_string("Moisture Level  ",LCD_LINE_1)
  lcd_string(str(moisture_level),LCD_LINE_2)
  time.sleep(0.5)
  rain_data = GPIO.input(Rain_sensor)
  #Send data on thing speak server
  thingspeak_post(temp,moisture_level,motor_status,rain_data)
  if((temp > 25)  and  (moisture_level < 100) and (rain_data != True)) :
```

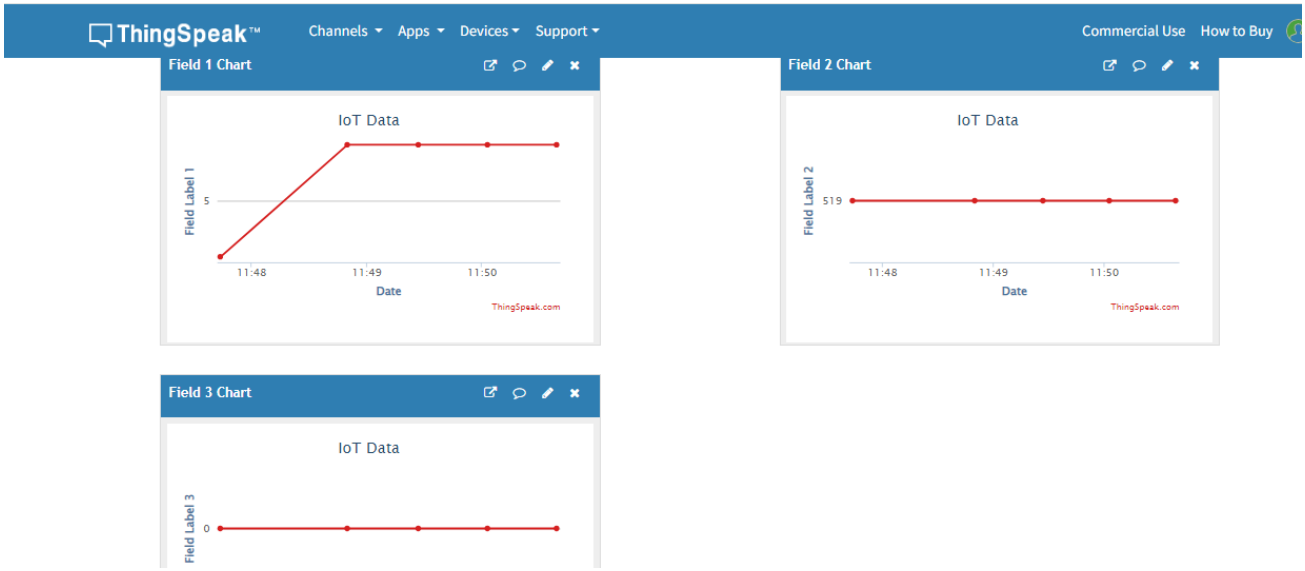
```
 GPIO.output(Relay_pin, True)
 lcd_byte(0x01,LCD_CMD) # 000001 Clear display
 lcd_string("Motor Start ",LCD_LINE_1)
 pio.uart.println("AT")
 pio.uart.println("AT+CMGF=1")
 pio.uart.println("AT+CMGS=\"+919865533668\"\r")
 pio.uart.println("Motor Started")
 motor_status = 1
 time.sleep(0.5)
else:
 GPIO.output(Relay_pin, False)
 lcd_byte(0x01,LCD_CMD) # 000001 Clear display
 lcd_string("Motor Stop ",LCD_LINE_1)
 if(rain_data == True):
    lcd_string("Rain Detected ",LCD_LINE_2)
 pio.uart.println("AT")
 pio.uart.println("AT+CMGF=1")
 pio.uart.println("AT+CMGS=\"+919865533668\"\r")
 pio.uart.println("Motor Stop")
 motor_status = 0
 time.sleep(0.5)
```

```
  pass

# Command line execution

if __name__ == '__main__' :

   main()
```

OUTPUT:




Rain Sensor
EngineeringProjects.com
Rain Sensor
U1
Raspberry Pi 3
U2
ADC Module
SOIL
SOIL MOISTURE
MCP3208
U4
Temperature Sensor
LCD1
LM016L
Smart Irrigation System
Display Module
www.TheEngineeringProjects.com
Soil Moisture Sensor
DC_Motor
RL1
5V
GSM Module
L1
27mH
RV1
10k


ThingSpeak™
Channels
Apps
Devices
Support
Commercial Use
How to Buy
Field 1 Chart
IoT Data
Field Label 1
5
11:48
11:49
11:50
Date
ThingSpeak.com
Field 2 Chart
IoT Data
Field Label 2
519
11:48
11:49
11:50
Date
ThingSpeak.com
Field 3 Chart
IoT Data
Field Label 3
0