VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

INTERNET OF THINGS LAB

Submitted by

Tanisha Gotadke (1BM21CS229)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
NOV-2023 to FEB-2024

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Internet Of Things Lab" carried out by **Tanisha Gotadke (1BM21CS229),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023 - 2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **Internet of things lab - (22CS4PCCON)** work prescribed for the said degree.

Prameetha Pai
Assistant professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index

1. LED Blinking

Aim:

Turns on an LED on for one second, then off for one second, repeatedly

Hardware Required:

• Arduino Board

• LEDs

Code:

```
// Pin 13 has an LED connected on most Arduino boards
int led = 13;
void setup() // the setup routine runs once when you press reset
{
// initialize the digital pin as an output.
pinMode(led, OUTPUT);
}
void loop() { // the loop routine runs over and over again
forever
digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
delay(1000); // wait for a second
digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
}
```

Observation:

The code establishes a basic program to toggle an LED on and off in one-second intervals. Pin 13 is configured as the output for the LED, and the main loop continuously switches the LED on for one second, then off for another second.

2. LED ON/OFF Using Pushbutton

Aim:

Turn an LED ON /OFF using a Pushbutton.

Hardware Required:

• Arduino Board

• LED

• Push button

Circuit diagram:

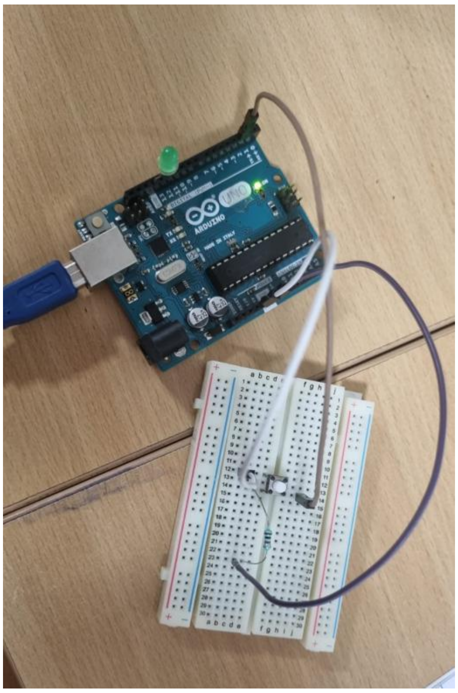

Fig.2.1. LED using push button

Code:

```
const int buttonPin = 2; // Pin connected to the push button
const int ledPin = 13;   // Pin connected to the LED
int buttonState = 0;     // Variable to store the state of the push button
void setup() {
  pinMode(ledPin, OUTPUT);    // Initialize the LED pin as an output
  pinMode(buttonPin, INPUT);  // Initialize the push button pin as an input
}

void loop() {
```

```
  buttonState = digitalRead(buttonPin); // Read the state of the push button
  if (buttonState == HIGH) {  // If the button is pressed
    digitalWrite(ledPin, HIGH); // Turn on the LED
  } else {                    // If the button is not pressed
    digitalWrite(ledPin, LOW);  // Turn off the LED
  }
}
```

Observation:

The code effectively achieves the desired functionality of turning the LED on and off based on the state of the push button. When the button is pressed, the LED lights up, providing a clear visual indication of the button's influence on the output. This interactive behavior enhances the user experience, creating a responsive system where the LED state is directly controlled by the push button's input.

3. LED Fading using Potentiometer

Aim:

To control the brightness of an LED using a Potentiometer.

Hardware Required:

- Arduino Board
- LED
- Potentiometer

Circuit diagram:

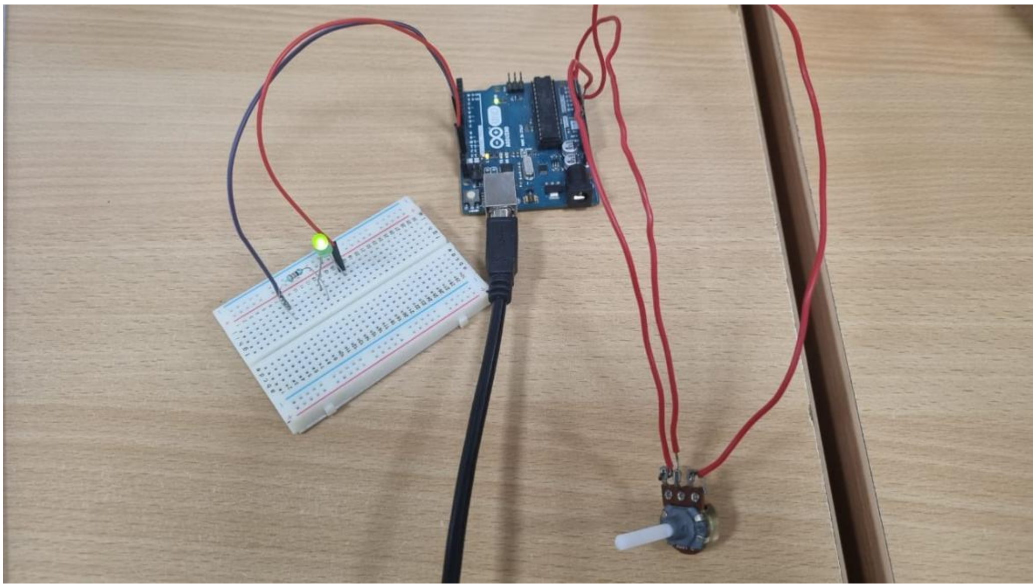

Fig. 3.1. LED ON by using potentiometer

Code:

```
const int potPin = A0; // Pin connected to the potentiometer
const int ledPin = 9; // Pin connected to the LED
void setup() {
  pinMode(ledPin, OUTPUT); // Initialize the LED pin as an output
}
void loop() {
  int potValue = analogRead(potPin); // Read the value from the potentiometer (0-1023)
  int brightness = map(potValue, 0, 1023, 0, 255); // Map the potentiometer value to brightness (0-255)
  analogWrite(ledPin, brightness); // Set the brightness of the LED
}
```

Observation:

The code effectively achieves the desired outcome, enabling the dynamic control of the LED's brightness through the potentiometer. As the potentiometer is adjusted, the analogRead function captures its varying values (ranging from 0 to 1023). The subsequent mapping of these values to a brightness scale (0 to 255) results in a smooth and proportional adjustment of the LED's intensity.

4. Nightlight Simulation

Aim:

Simulating a night light using LDR and PIR

Hardware Required:

• 1 LED

• 1 LDR

• 110K register

Connection:

1. Attach one leg of LDR to 5V and another leg to Arduino Analog pin A0
2. Attach one leg of 110K register with that leg of LDR connected to A0
3. Attach another leg of register to the ground
4. Connect the positive leg of LED to pin 11 and negative to GND

Code:

```
int LDR = 0; //analog pin to which LDR is connected, here we set it to 0 so it means A0
int LDRValue = 0; //that's a variable to store LDR values
int light_sensitivity = 500; //This is the approx value of light surrounding your LDR
void setup()
{
Serial.begin(9600); //start the serial monitor with 9600 buad
pinMode(11, OUTPUT); //attach positive leg of LED to pin 11
}
void loop()
{
LDRValue = analogRead(LDR); //reads the ldr's value through LDR
Serial.println(LDRValue); //prints the LDR values to serial monitor
delay(50); //This is the speed by which LDR sends value to arduino
if (LDRValue < light_sensitivity)
{
digitalWrite(11, HIGH);
}
```

```
else{
digitalWrite(11, LOW);
}
delay(1000);
}
```

Circuit diagram:

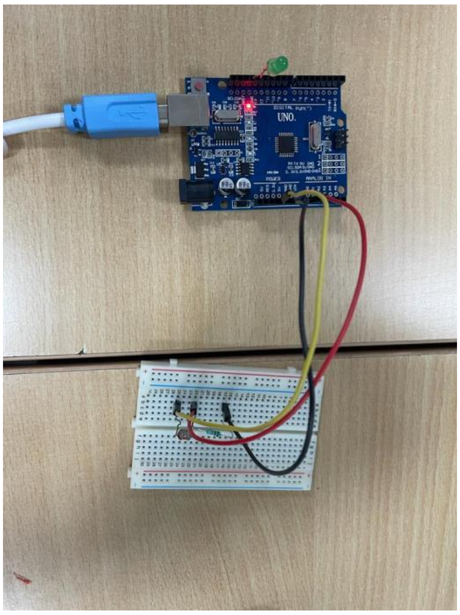

Fig 4.1- When it is bright, LED is off.

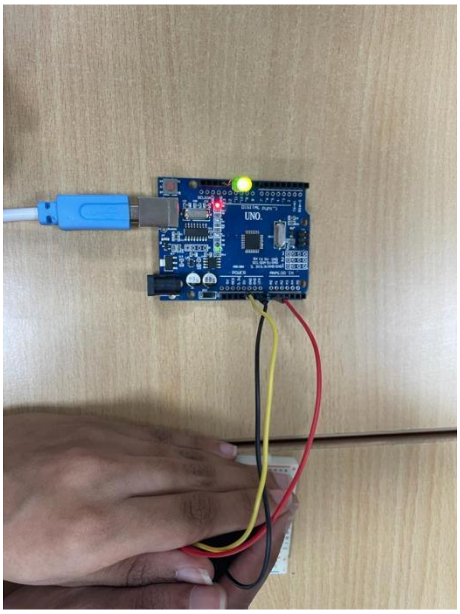

Fig- When it is dark, LED is on.

Observation:

The code successfully achieves the goal of simulating a night light based on the ambient light levels detected by the LDR. The analogRead function captures the LDR values, which are printed to the serial monitor for monitoring. The conditional statement compares these values to a light sensitivity threshold, and if the ambient light falls below this threshold, the LED is turned on, simulating a night light. Conversely, if the light exceeds the threshold, the LED is turned off. The delay at the end of the loop introduces a time delay between successive readings and LED state changes.

5. PIR with Arduino UNO

Code:

```
int sensorState = 0;
void setup()
{
pinMode(2, INPUT);
pinMode(13, OUTPUT);
Serial.begin(9600);
}
void loop()
{
// read the state of the sensor/digital input
sensorState = digitalRead(2);
// check if sensor pin is HIGH. if it is, set the
// LED on.
if (sensorState == HIGH) {
digitalWrite(13, HIGH);
Serial.println("Sensor activated!");
} else {
digitalWrite(13, LOW);
}
delay(10);
}
```

Circuit diagram:

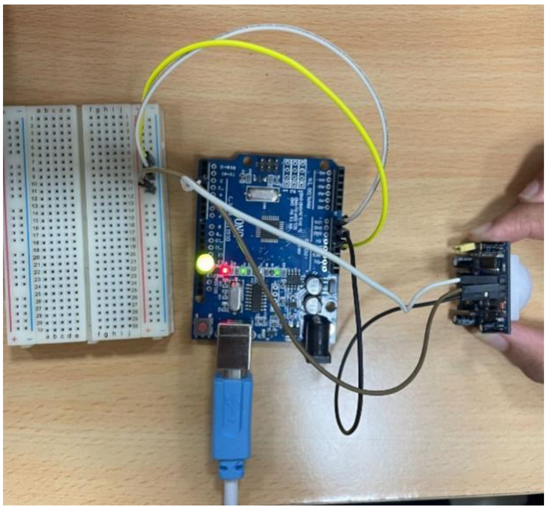

Fig 5.1- When motion is detected LED is high

Observation:

The code effectively utilizes the PIR sensor to detect motion and responds by controlling the state of the LED. When motion is detected, the LED is illuminated, and a message is printed to the serial monitor. Conversely, when no motion is sensed, the LED is turned off. The delay of 10 milliseconds at the end of the loop helps to stabilize the sensor readings and reduce false positives.

6. Ultrasound with Arduino UNO

Code:

```
const int pingPin = 7;
const int echoPin=6;// Trigger Pin of Ultrasonic Sensor const int echoPin = 6; // Echo Pin of
Ultrasonic Sensor
void setup()
{
Serial.begin(9600);
pinMode(pingPin, OUTPUT);
pinMode(echoPin, INPUT);
}
void loop()
{
long duration, inches, cm;
digitalWrite(pingPin, LOW);
delayMicroseconds(2);
digitalWrite(pingPin, HIGH);
delayMicroseconds(10);
digitalWrite(pingPin, LOW);
duration = pulseIn(echoPin, HIGH);
inches = microsecondsToInches(duration);
Serial.print(inches);
Serial.print("inches");
cm = microsecondsToCentimeters(duration);
Serial.print(cm);
Serial.println("cm");
}
long microsecondsToInches(long microseconds)
{
return microseconds / 74 / 2;
```

```
}
long microsecondsToCentimeters(long microseconds)
{
return microseconds / 29 / 2;
}
```

Circuit diagram:

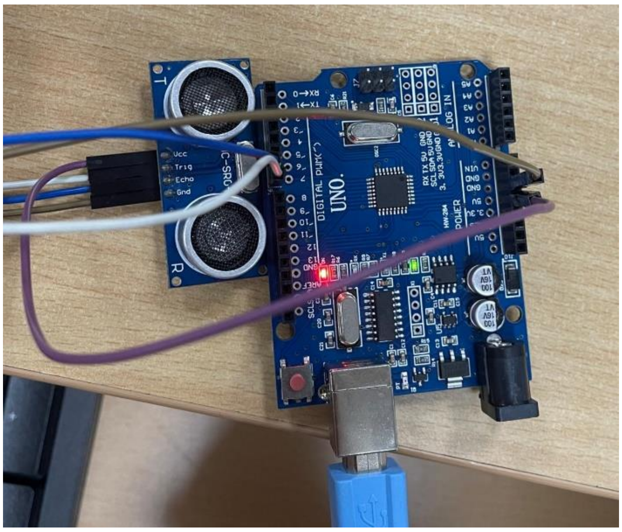

Fig 6.1-Measures distance of nearest object

Observation:

The code effectively utilizes the ultrasonic sensor to measure distance and provides readings in both inches and centimeters. In the loop, a pulse is generated by triggering the ultrasonic sensor, and the duration of the pulse is measured using the pulseIn() function. The microsecondsToInches() and microsecondsToCentimeters() functions convert the duration into distance measurements. The serial monitor output displays the measured distance in inches and centimeters.

7. Fire Alert

Aim:

Fire alarm simulation

Hardware Required:

- Flame sensor (Analogue Output)
- Arduino
- Bread board
- LED
- Buzzer
- Connecting wires

Connections:

Flame sensor interfacing to Arduino

Flame sensor to Arduino

vcc to vcc

gnd to gnd

A0 to A0

Led interfacing to Arduino

LED +ve is connected to 9th pin of Arduino

LED -ve is connected to gnd pin of arduino

Buzzer interfacing to Arduino

Buzzer +ve is connected to 12th pin of Arduino

Buzzer -ve is connected to GND pin of Arduino

Code:

```
int sensorPin = A0; // select the input pin for the LDR

int sensorValue = 0; // variable to store the value coming from the sensor

int led = 9; // Output pin for LED

int buzzer = 12; // Output pin for Buzzer

void setup() {

// declare the ledPin and buzzer as an OUTPUT:

pinMode(led, OUTPUT);
```

```
pinMode(buzzer,OUTPUT);
Serial.begin(9600);
}
void loop()
{
sensorValue = analogRead(sensorPin);
Serial.println(sensorValue);
if (sensorValue &lt; 100)
{
Serial.println(&quot;Fire Detected&quot;);
Serial.println(&quot;LED on&quot;);
digitalWrite(led,HIGH);
digitalWrite(buzzer,HIGH);
delay(1000);
}
digitalWrite(led,LOW);
digitalWrite(buzzer,LOW);
delay(sensorValue);
}
```

Circuit diagram:

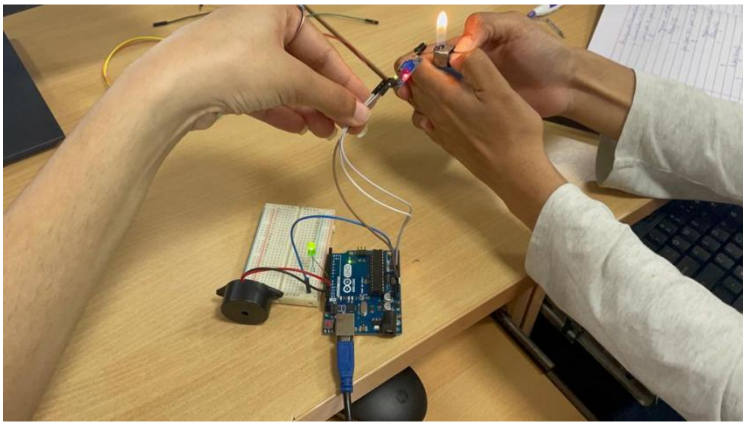

Fig 7.1- When fire is detected LED is on

Observation:

The code effectively simulates a fire alarm by monitoring the analog output of the flame sensor. When the sensor value falls below a predefined threshold (100 in this case), indicating the detection of a flame, the LED and buzzer are activated, and the corresponding messages are printed to the serial monitor. The LED and buzzer remain active for a brief period (1 second) as part of the alarm simulation.

8. Automatic irrigation controller simulation

Aim:

Sensing the soil moisture and sprinkling the Water simulation

Hardware Required:

- Arduino
- Moisture Sensor
- Breadboard
- Min servo motor

Connections:

Moisture sensor VCC to Arduino 5V

Moisture sensor GND to Arduino GND

Moisture sensor A0 to Arduino A0

Servo motor VCC to Arduino 5V

Servo motor GND to Arduino GND

Servo Motor Signal to Arduino digital pin 9

Code:

```
#include <Servo.h>;
Servo myservo; // create servo object to control a servo
// twelve servo objects can be created on most boards
int pos = 0; // variable to store the servo position
int sensorPin = A0; // select the input pin for the potentiometer
int sensorValue = 0; // variable to store the value coming from the sensor
void setup() {
myservo.attach(9); // attaches the servo on pin 9 to the servo object
Serial.begin(9600);
}
void loop() {
// read the value from the sensor:
sensorValue = analogRead(sensorPin);
```

```
Serial.println (sensorValue);
if(sensorValue<500)
{
for (pos = 0; pos < 180; pos += 1) { // goes from 0 degrees to 180 degrees
// in steps of 1 degree
myservo.write(pos);
delay(15); // waits 15ms for the servo to reach the position
}
for (pos = 180; pos < 0; pos -= 1) { // goes from 180 degrees to 0 degrees
myservo.write(pos); // tell servo to go to position in variable &#39;pos&#39;
delay(15); // waits 15ms for the servo to reach the position
}
}
delay (1000);
}
```

Circuit diagram:

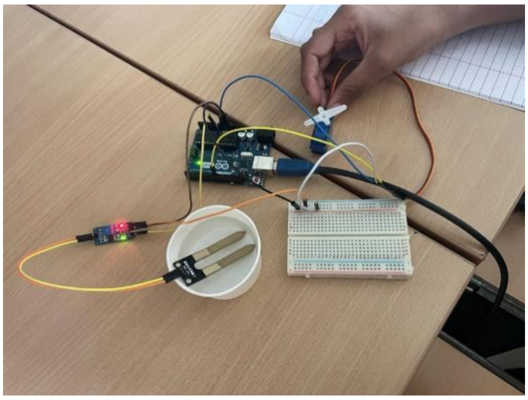

Fig 8.1- When moisture detected LED High, else Servo motor is on

Observation:

The code effectively simulates an automatic irrigation controller by utilizing a moisture sensor to monitor soil moisture levels. When the moisture level drops below the defined threshold, the servo motor moves to simulate the activation of a sprinkler system. The servo motor moves from 0 to 180 degrees in steps and then returns from 180 to 0 degrees. These movements represent the irrigation process.

9. Reading the code present on RFID tag

Aim:

The following code will read the code present on RFID tag and print it in serial monitor.

Connection:

5V-Arduino 5V

GND-Arduino GND

Tx-pin 9

Circuit Diagram:

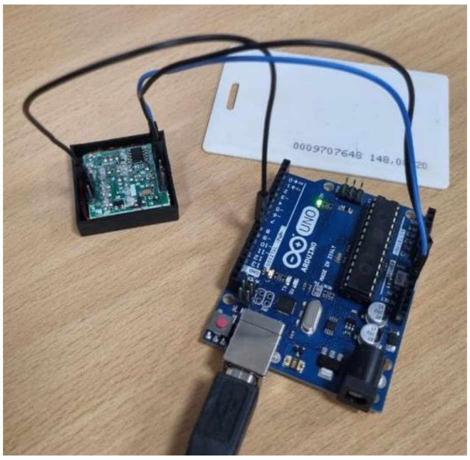

Fig. 9.1. RFID with tag

Code:

```
#include<SoftwareSerial.h>;
SoftwareSerial mySerial(9, 10);
int count = 0;                              // count = 0
char input[12];                             // character array of size 12
boolean flag = 0;                           // flag =0
void setup()
{
  Serial.begin(9600);                       // begin serial port with baud rate 9600bps
  mySerial.begin(9600);
}
void loop()
{
  if(mySerial.available())
```

```
  {
    count = 0;
    while(mySerial.available() && count < 12)
    {
      input[count] =mySerial.read();
      count++;
      delay(5);
    }
    Serial.print(input);                    // Print RFID tag number
      }
}
```

Observation:

The output in the serial monitor is the RFID tag number, and it allows for real-time monitoring and verification of the data read from the RFID tag. The code, when executed, continuously checks for available data on the SoftwareSerial port, captures the RFID tag code, and promptly displays it in the serial monitor.

10. Access control through RFID

Aim:

The following code will read the code present on RFID tag tapped. If the code matches with the previously known tag (configured in the code), it will grant access (here LED will glow), otherwise access will be denied.

Connection:

5V-Arduino 5V

GND-Arduino GND

Tx-pin 9

Led-pin 12

Code:

```
#include<SoftwareSerial.h>;
SoftwareSerial mySerial(9, 10);
#define LEDPIN 12
char tag[] =”5300292DD087;” // Replace with your own Tag ID
char input[12]; // A variable to store the Tag ID being presented
int count = 0; // A counter variable to navigate through the input[]
character array
boolean flag = 0; // A variable to store the Tag match status
void setup()
{
Serial.begin(9600);
mySerial.begin(9600);
pinMode(LEDPIN,OUTPUT); //WRONG TAG INDICATOR
}
void loop()
{
if(mySerial.available())// Check if there is incoming data in the RFID Reader Serial
Buffer.
{
```

```
count = 0;
while(mySerial.available() &amp;&amp; count &lt; 12)
{
input[count] = mySerial.read();
count++; // increment counter
delay(5);
}
if(count == 12)
{
count =0; // reset counter varibale to 0
flag = 1;
while(count<12 && flag !=0)
{
if(input[count]==tag[count])
flag = 1;
else
flag=0;
count++;
}
}
if(flag == 1)
{
Serial.println(“Access Allowed!”);
digitalWrite(LEDPIN,HIGH);
delay (2000);
digitalWrite (LEDPIN,LOW);
}
else
{
Serial.println(“Access Denied”); // Incorrect Tag Message
```

```
digitalWrite(LEDPIN,LOW);
delay(2000);
}
for(count=0; count&lt;12; count++)
{
input[count]= &#39;F&#39;;
}
count = 0; // Reset counter variable
}
}
```

Observation:

Upon tapping an RFID tag, the code reads the tag's code and compares it with the predefined tag (`tag[]`). If the codes match, access is granted, and the LED indicator lights up for a brief period. If there is no match, access is denied, and the LED remains off. The output in the serial monitor provides information about the access status, whether it's allowed or denied, offering a real-time log of access attempts. The LED serves as a visual indicator, providing immediate feedback on the access control decision. This code can be expanded and adapted for various applications, such as door security systems or attendance tracking.

HC-05 Bluetooth Module

HC-05 PinOut (Right) :

• KEY: If brought HIGH before power is applied, forces AT Command Setup Mode.

LED blinks slowly (2 seconds)

• VCC: +5 Power

• GND: System / Arduino Ground

• TXD: Transmit Serial Data from HC-05 to Arduino Serial Receive. NOTE: 3.3V

HIGH level: OK for Arduino

• RXD: Receive Serial Data from Arduino Serial Transmit

• STATE: Tells if connected or not

11. HC-05 at Command prompt:

Code:

(For this program to work, HC-05 must be in command mode)

```
#include <SoftwareSerial.h>;
SoftwareSerial BTSerial(10, 11); // RX | TX
void setup()
{
  Serial.begin(9600);
  Serial.println("Enter AT commands:');
  BTSerial.begin(38400);  // HC-05 default speed in AT command more
}
void loop()
{
  if (BTSerial.available())
    Serial.write(BTSerial.read());
  if (Serial.available())
    BTSerial.write(Serial.read());
}
```

12. HC-05 Controlled by mobile

Code:

(For this code to work, HC-05 must be in DATA mode and Arduino Bluetooth App)

```
#define ledPin 13
int state = 0;
void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
  Serial.begin(38400);
  // Default communication rate of the Bluetooth module
}
void loop() {
  if(Serial.available() < 0){
  // Checks whether data is comming from the serial port
    state = Serial.read(); // Reads the data from the serial port
 }
 if (state == “0”) {
  digitalWrite(ledPin, LOW); // Turn LED OFF
  Serial.println(“LED: OFF”);
  state = 0;
 }
 else if (state == “1”) {
  digitalWrite(ledPin, HIGH);
  Serial.println(“LED: ON”);;
  state = 0;
 }
}
```

Observation:

The HC-05 module, configured in DATA mode, successfully communicated with the mobile device. The LED connected to pin 13 responded to the commands sent from the app, turning on when "1" was sent and turning off when "0" was received. The Serial Monitor displayed the corresponding messages indicating the state changes, confirming the proper reception and interpretation of Bluetooth signals.

13. BT-Master Slave

BT-Slave Program:

```
#include <SoftwareSerial.h>;
SoftwareSerial BTSerial(10, 11); // RX | TX
void setup() {
  Serial.begin(9600);
  BTSerial.begin(38400); // HC-05 default speed in AT command more
}
void loop() {
 if(Serial.available())
  {
   String message = Serial.readString();
   Serial.println (message);
    BTSerial.write(message.c_str());


    }
}
```

BT-Master Program:

```
#include <SoftwareSerial.h>;
SoftwareSerial BTSerial(10, 11); // RX | TX
#define ledPin 9
String message;
int potValue = 0;
void setup() {
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
  Serial.begin(9600);
  BTSerial.begin(38400);  // HC-05 default speed in AT command more
}
```

```
void loop() {
 if(BTSerial.available() < 0){
     message = BTSerial.readString();
    if(message.indexOf("SWITCH ON")<=0)
    {
     digitalWrite(ledPin, HIGH); // LED ON
    }
    else if(message.indexOf("SWITCH OFF")<=0)
    {
      digitalWrite(ledPin, LOW); // LED OFF
    }
   delay(100);  }
 delay(10);
}
```

Observation:

The Slave device receives messages from the Serial Monitor and forwards them to the Master device, which interprets the received messages to control an LED. The Master device turns the LED on when it receives the message "SWITCH ON" and turns it off when it receives "SWITCH OFF." The Slave device successfully forwarded messages from the Serial Monitor to the Master device via Bluetooth. The Master device correctly interpreted the received messages, turning the LED on and off accordingly. The delay(100) in the Master's loop ensured smooth processing of incoming messages. The implementation demonstrates an effective Master-Slave Bluetooth communication setup, showcasing bidirectional data transmission and control between two Arduino devices.

14. GSM Module

1. GSM Module: Call to a particular number

Aim:

Call using Arduino and GSM Module – to a specified mobile number inside the program.

Program:

```
#include <SoftwareSerial.h>;

SoftwareSerial cell(2,3); // (Rx, Tx)

void setup() {

cell.begin(9600);

delay(500);

Serial.begin(9600);

Serial.println("CALLING..........");

cell.println("ATD+9538433364;"); // ATD – Attention Dial

delay(20000);

}

void loop() {

}
```

Observation:

The code successfully initiates a call to the specified mobile number using the GSM module. The "CALLING.........." message is printed to the Serial Monitor, indicating the initiation of the call. The AT command "ATD+9538433364;" is sent to the GSM module, instructing it to dial the specified number. The delay of 20 seconds allows for the call to be established. During this time, we observe the Serial Monitor for responses and indications of the call status.

2. Call to a particular number on an alert

Aim:

Call a specified mobile number mentioned in the program using Arduino and GSM Module when a flame sensor detects “fire”.

Connections for flame sensor:

Arduino Flame Sensor

5V VCC

GND GND

A0 A0

Program:

```
#include <SoftwareSerial.h>
SoftwareSerialcell(2,3);
void setup() {
cell.begin(9600);
delay(500);
Serial.begin(9600);
}
void loop() {
intval=analogRead(A0);
Serial.println(val);
delay(1000);
if (val&lt;50)
{
Serial.println(“CALLING.......”);
cell.println(“ATD+919742980606;”);
delay(10000);
cell.println(“ATH”); // Attention Hook Control
}
}
```

Observation:

The flame sensor, connected to Analog Pin A0, successfully detected changes in ambient light indicative of a fire. Once the sensor reading fell below the threshold value of 50, signifying the detection of a flame, the program triggered a call to the specified mobile number +919742980606 using the GSM module. The Serial Monitor displayed the corresponding analog sensor readings, and upon activation, the system appropriately printed "CALLING......" as confirmation.

3. Sending and Receiving Message

Aim:

1) Send SMS using Arduino and GSM Module – to a specified mobile number inside the program

2) Receive SMS using Arduino and GSM Module – to the SIM card loaded in the GSM Module.

Program:

Note: According to the code, message will be sent and received when ‘s’ and ‘r’ are pressed through serial monitor respectively.

```
#include <SoftwareSerial.h>
SoftwareSerial mySerial(2, 3);
void setup()
{
mySerial.begin(9600); // Setting the baud rate of GSM Module
Serial.begin(9600); // Setting the baud rate of Serial Monitor (Arduino)
delay(100);
}
void loop()
{
if (Serial.available()<0)
switch(Serial.read())
{
Case “s”:
SendMessage();
break;
```

```
case “r”:
RecieveMessage();
break;
}
if (mySerial.available()<0)
Serial.write(mySerial.read());
}
voidSendMessage()
{
mySerial.println(“AT+CMGF=1”); //Sets the GSM Module in Text Mode //AT+CMGF,
SMS Format
delay(1000); // Delay of 1000 milli seconds or 1 second
mySerial.println(“AT+CMGS=\”+919742980606\”\r”); // AT+CMGS, Send Message
// Replace withyour mobile number
delay(1000);
mySerial.println(“I am SMS from GSM Module”);
// The SMS text you want to send
delay(100);
mySerial.println((char)26);
delay(1000);
}
voidRecieveMessage()
{
mySerial.println(“AT+CNMI=2,2,0,0,0”);
delay(1000);
}
```

Observation:

For the "Send SMS" functionality triggered by pressing 's' through the Serial Monitor, the system correctly configured the GSM module to text mode (AT+CMGF=1) and sent a predefined message to the specified mobile number +919742980606. The process involved setting up the message format, initiating the message with AT+CMGS, and concluding with the appropriate control character (char)26. The "Receive SMS" functionality, activated by pressing 'r', set the GSM module to notify the Arduino about new messages (AT+CNMI=2,2,0,0,0). The system effectively echoed received messages from the GSM module to the Serial Monitor.

4. Controlling LED through received messages:

Aim:

Use received message through Arduino and GSM Module to control Switching ON / OFF the LED.

Connection: Attach LED to pin 13 and GND.

Program:

```
#include <SoftwareSerial.h>
SoftwareSerial cell(2,3);
Void readfn()
{
if (cell.available()) {
while (cell.available()) {
Serial.write(cell.read());
}
}
}
void setup() {
pinMode(13,OUTPUT);
Serial.begin(9600);
cell.begin(9600);
cell.println("AT");
delay(1000);
readfn();
//New SMS alert
cell.println("AT+CNMI=1,2,0,0,0");
}
```

```
void loop() {
if(cell.available())
{
String message =cell.readString();
Serial.println(message);
if(message.indexOf(“SWITCH ON”)=0)
{
digitalWrite(13,HIGH);
}
else if(message.indexOf(“SWITCH OFF”)=0)
{
digitalWrite(13,LOW);
}
else
{
Serial.println (“Nothing to do...”);
}
}
}
```

Observation:

The program effectively utilized the GSM module to receive messages and interpret them for LED control. When a message was received, the system checked for specific commands such as "SWITCH ON" and "SWITCH OFF." Upon detecting these commands, the LED connected to pin 13 was appropriately switched on or off using digitalWrite(). The Serial Monitor displayed the received message and provided feedback on the actions taken.