Visvesvaraya National Institute
of Technology, Nagpur

Embedded System
(ECL403)

Lab Report

Submitted By :
Aditya Wadichar (BT18ECE035)
Semester 5
Electronics and Communication Engineering Dept.

Submitted To :
Dr. Ankit A Bhurane
Course Instructor

Contents

Experiment-1: Running LED, Switches + LED

Problem Statement:: To glow LEDs in clockwise & anticlockwise fashion, and control LEDs by switches

LED in fashion: Assembly code

```
MOV A, #11111110B
HERE: MOV P1, A
RL A
CJNE A, #01111111B, HERE
THERE: MOV P1, A
RR A
CJNE A, #11111110B, THERE
JMP HERE
```

C code

```
#include<reg51.h>

// pattern of led panel to display
unsigned char led[] = { ... 0xFE, 0xFD, 0xFB, 0xF7, ... 0xEF, 0xDF, 0xBF, 0x7F};

void main()
{
    int i;
    while(1)  // for continuous blinking of leds
    {
        for(i = 0; i<8 ... ;i++) // anticlockwis
        {
            P1 = led[i];
        }

        for(i=7 ; i≥0 ;i−−) // clockwise
        {
            P1 = led[i];
        }
    }
}
```

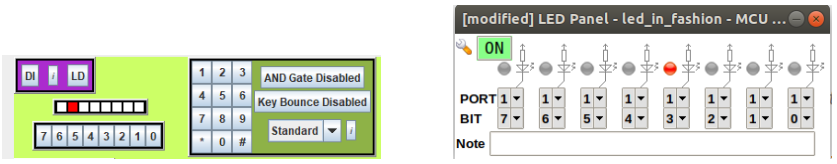
LEDs & switch: Assembly code

```
HERE: MOV P1, P2
JMP HERE
```

C code

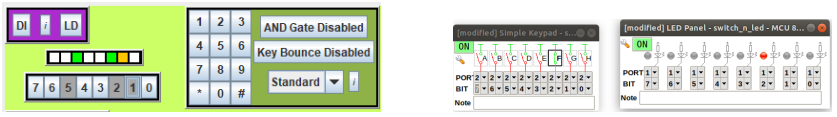
```
#include<reg51.h>
void main()
{
    while(1)
    {
        P1=P2;
    }
}
```

Output: The outputs are shown in Fig.1 & 2



(a) (b)

Figure 1: LEDs in fashion



(a) (b)

Figure 2: LEDs controlled by switches

Observation and Discussion: The leds are connected in common anode configuration. i.e led will glow if logic 0 is given to it. Hence to glow leds in cyclic manner, corresponding inputs are given to them.

Ports connected to switches are default in high state. When key is pressed, their state becomes low. Hence the values of Port 2 are directly copied to Port 1 to glow corresponding led.

Conclusion: The LEDs are glown in cyclic manner and also LEds are controlled by switches.

Experiment-2: DAC scope to display sinusoidal wave

Problem Statement:: Display a sine wave on DAC scope

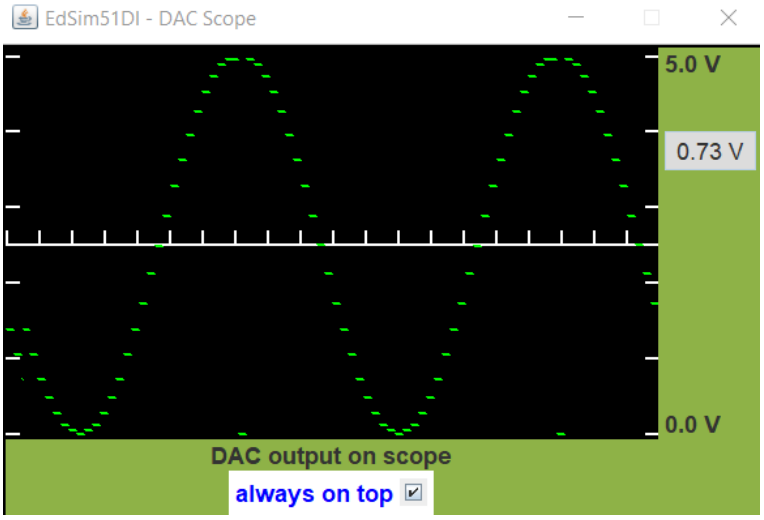
Code: Assembly code

```
MOV P0,00H  ; SETTING P0.7 PIN
MOV A, #00H
MOV DPTR, #LOOKUP
BACK: MOV B, #00H
LOOP: MOVC A, @A+DPTR
MOV P1,A
INC B
MOV A,B
CJNE A,#41, LOOP
JMP BACK
LOOKUP:
DB  0, 2, 6, 14, 24, 37, ...
    53, 70, 88, 108, 127, ...
    147, 167, 185, 202, 218, ...
    231, 241, 249, 253, 255, ...
    253, 249, 241, 231, 218, ...
    202, 185, 167, 147, 128, ...
    108, 88, 70, 53, 37, 24, ...
    14, 6, 2, 0

; DATASET OF COS FUNCTION
    SCALED FROM 0 TO 255
```

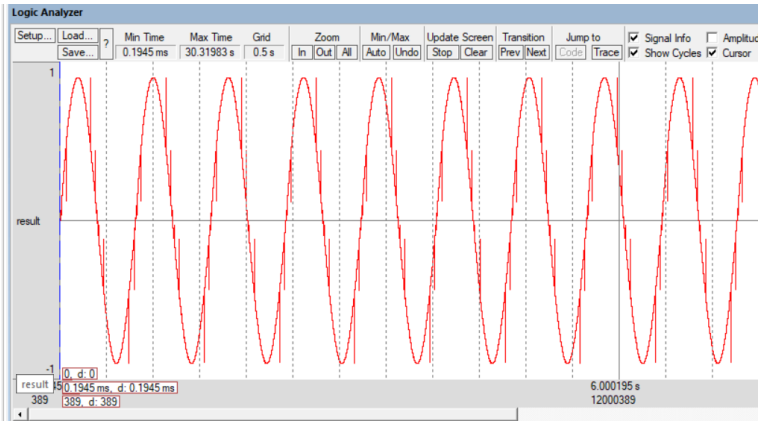
C Code

```
#include<reg51.h>
#include<math.h>
void main()
{
    while(1)
    {
        int i;
        double result;
        for (i=0; i≤360; i++)
        {
            result = ...
                sin(2*3.1415*i ...
                /360);   //
                points of sin
                wave
            // the variable
                result is
                analysed in
                logic analyser
        }
    }
}
```

Output: The output is shown in Fig. 3



(a)



(b)

Figure 3: DAC scope

Observation and Discussion: The sine wave gives output between 0 to 1, but DAC scope can shows values from 0 to 255. Hence 41 values of sine wave has been scaled to 0 to 255 and stored in database.

Whereas in logic analyser, 360 values of sine wave has been directly plotted, no scaling is done.

Conclusion: The sine wave has been displayed on DAC scope as well as logic analyser.

Experiment-3: Interfacing of 4x3 keypad

Problem Statement:: Detect the key pressed on keypad and indicate as binary pattern on LED array with defaults connections.

Code: Assembly code:

```
BACK: MOV P0, #11110111B ; CHECKING FOR FOURTH ROW

JNB P0.6, ONE
JNB P0.5, TWO
JNB P0.4, THREE

MOV P0, #11111011B ; CHECKING FOR THIRD ROW
JNB P0.6, FOUR
JNB P0.5, FIVE
JNB P0.4, SIX

MOV P0, #11111101B ; CHECKING FOR SECOND ROW
JNB P0.6, SEVEN
JNB P0.5, EIGHT
JNB P0.4, NINE

MOV P0, #11111110B ; CHECKING FOR FIRST ROW
JNB P0.5, ZERO
MOV A, #00H ; IF NO BUTTON IS PRESSED, ALL LEDS ARE ON
JMP HERE

ZERO: MOV A, #0FFH
JMP HERE
ONE: MOV A, #0FEH
JMP HERE
TWO: MOV A, #0FDH
JMP HERE
THREE: MOV A, #0FCH
JMP HERE
FOUR: MOV A, #0FBH
JMP HERE
FIVE: MOV A, #0FAH
JMP HERE
SIX: MOV A, #0F9H
JMP HERE
SEVEN: MOV A, #0F8H
JMP HERE
```

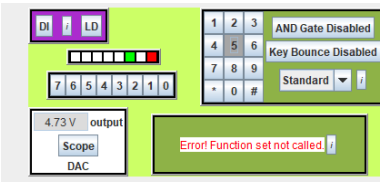
C code:

```
#include<reg51.h>
void main()
{
    while(1)
    {
        P0=0b11110111; // checking first row
        if (P0_6==0) // 1 is pressed
        {
            P1=0b11111110;
            break;
        }
        else if (P0_5==0) // 2 is pressed
        {
            P1=0b11111101;
            break;
        }
        else if (P0_4==0) // 3 is pressed
        {
            P1=0b11111100;
            break;
        }

        P0=0b11111011; // checking second row
        if (P0_6==0)  // 4 is pressed
        {
            P1=0b11111011;
            break;
        }
        else if (P0_5==0) // 5 is pressed
        {
            P1=0b11111010;
            break;
        }
        else if (P0_4==0) // 6 is pressed
```

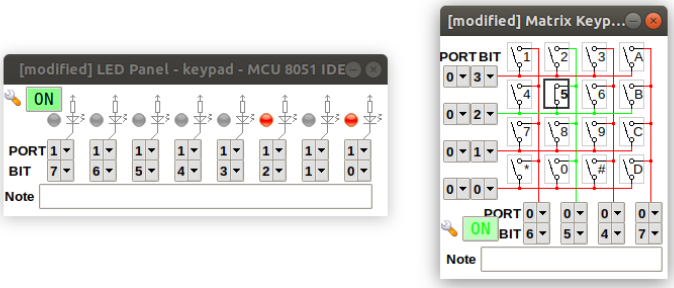
```
38 EIGHT: MOV A, #0F7H
39 JMP HERE
40 NINE: MOV A, #0F6H
41
42 HERE: MOV P1, A
43 JMP BACK
```

```
35          {
36              P1=0b11111001;
37              break;
38          }
39          P0=0b11111101; //
                checking third row
40          if (P0_6==0) // 7 is
                pressed
41          {
42              P1=0b11111000;
43              break;
44          }
45          else if (P0_5==0) //
                8 is pressed
46          {
47              P1=0b11110111;
48              break;
49          }
50          else if (P0_4==0) //
                9 is pressed
51          {
52              P1=0b11110110;
53              break;
54          }
55          P0=0b11111110; //
                checking last row
56          if (P0_5==0) // 0 is
                pressed
57          {
58              P1=0b11111111;
59              break;
60          }
61          P1=0b00000000; // if
                nothing is pressed,
                all leds will glow
62      }
63  }
```

Output: The output is shown in Fig. 4



(a)



(b)

Figure 4: Display of binary equivalent of keypad button pressed

Observation and Discussion: The first row from bottom of matrix keypad is connect to the P0.0 pin, second to the P0.1 pin, Third to the P0.2 pin and fourth to the P0.3 pin. Again, first column from right is connected to P0.4, second to P0.5 and third to P0.6.

We first set P0.3 as 0 and if we get P0.6 as 0 then we conclude that key 1 is pressed, if P0.5 is 0 then key 2 is pressed and if P0.4 is 0 then key 3 is pressed. Then we move on to the next row and similar process is followed to check for other keys.

After detecting the key, corresponding binary equivalent is displayed on LED panel. If no key is pressed then all LEDs will glow.

Conclusion: The matrix keypad has been interfaced with 8051 and binary equivalent of pressed button has been shown on LED panel.

Experiment-4: Interfacing LCD

Problem Statement:: Interface LCD with 8051 and write name on it.

Code: Assembly code

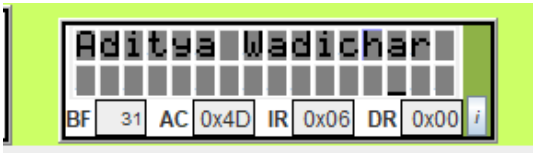
```
CLR P0.3 ; CLEAR RS
MOV P1, #38H; Function set (2 line
SETB P0.2; Set EN high
CALL DELAY
CLR P0.2; Clear EN
MOV P1, #0EH ; cursor and display on
SETB P0.2
CALL DELAY
CLR P0.2
MOV P1, #06H ; move cursor
SETB P0.2
CALL DELAY
CLR P0.2

SETB P0.3

; the string to be displayed is in lookup table
MOV A, #00H
MOV DPTR, #LOOKUP
MOV B, #00H
LOOP: MOVC A, @A+DPTR
MOV P1,A
SETB P0.2
CALL DELAY
CLR P0.2
INC B
MOV A, B
CJNE A, #16, LOOP
RET

DELAY:
MOV R0, #0F0H
DJNZ R0, $
RET

LOOKUP:
DB "Aditya Wadichar"
```

C code

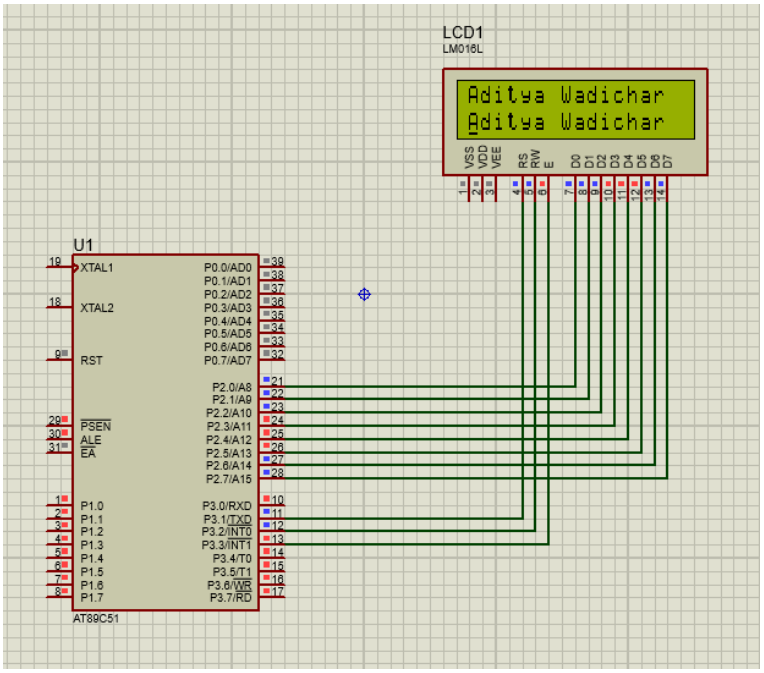
```
#include<reg51.h>
#include<string.h>

void LCD_set(unsigned char set);
void LCD_input(unsigned char *input);
void delay();


sbit rs=P3^1;  // Register select pin
sbit rw=P3^2;  // read/write pin
sbit en=P3^3;  // enable pin

void main()
{
    LCD_set(0x38);  // Setting LCD 2 lines, 5*7 matrix
    LCD_set(0x0E);  // cursor & display on
    LCD_set(0x06);  // increment cursor position

    LCD_input("Aditya Wadichar");
    LCD_set(0xC0);
    delay();


}

void LCD_set(unsigned char set)
{
    rs = 0;  // command mode
    rw = 0;  // write mode select
    P2 = set;  // command
    en = 1;
    delay();
    en = 0;
    return;
}

```

```
36 void LCD_input(unsigned
       char *input)
37 {
38     int i;
39     int l = strlen(input);
40     for (i=0; i<l; i++)   //
           string to be displayed
41     {
42         rw = 0;  // write mode
43         rs = 1;  // Data mode
44         P2 = input[i];  //
               sending character
                one by one
45         en = 1;
46         delay();
47         en = 0;
48     }
49     return;
50 }
51
52 void delay()
53 {
54     int i = 1000;
55     while (i>=0)
56     {
57         i--;
58     }
59     return;
60 }
```

Output: The outputs are shown in Fig. 5



(a)



(b)

Figure 5: Name on LCD display

Observation and Discussion: The LCD has been set to 2 lines 5×7 matrix mode. Required instructions are given to initialize LCD. In assembly language, the string to be displayed is stored in database and accessed through DPTR to display. In c language code the string is stored in variable and when string ends, next line command is given to LCD so as even if code executes continuously, we can see name displayed on screen without any distortion as it will rewrite the string on second line and not distort any other display.

Conclusion: The LCD is interfaced and name has been displayed on LCD.

Experiment-5: Interfacing Motor

Problem Statement:: Interface the motor with 8051 microcontroller and display the no. of rotations of motor on seven segment display

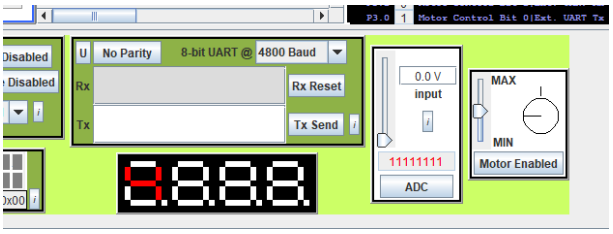
Code: Assembly code

```
MOV TMOD, #50H ; COUNTER IN MODE 1
SETB TR1 ; START COUNTER

MOV DPTR, #DATA

SETB P3.0
CLR P3.1 ; FORWARD ROTATION OF MOTOR
LOOP:
MOV A, TL1 ; MOVE COUNT IN A
CJNE A, #0AH, DISP ; JUMP TO DISP IF COUNT IS LESS THAN 9
MOV TL1, #00H ; RESTART THE COUNT
CLR A

DISP:
MOVC A, @A+DPTR ; DISPLAY THE COUNT ON SEVEN SEGMENT
MOV P1, A
JMP LOOP

; DATA OF LEDS FOR SEVEN SEGMENT DISPLAY
DATA:
DB 11000000B, 11111001B, ...
    10100100B, 10110000B, ...
    10011001B, 10010010B, ...
    10000010B, 11111000B, ...
    10000000B, 10010000B
```

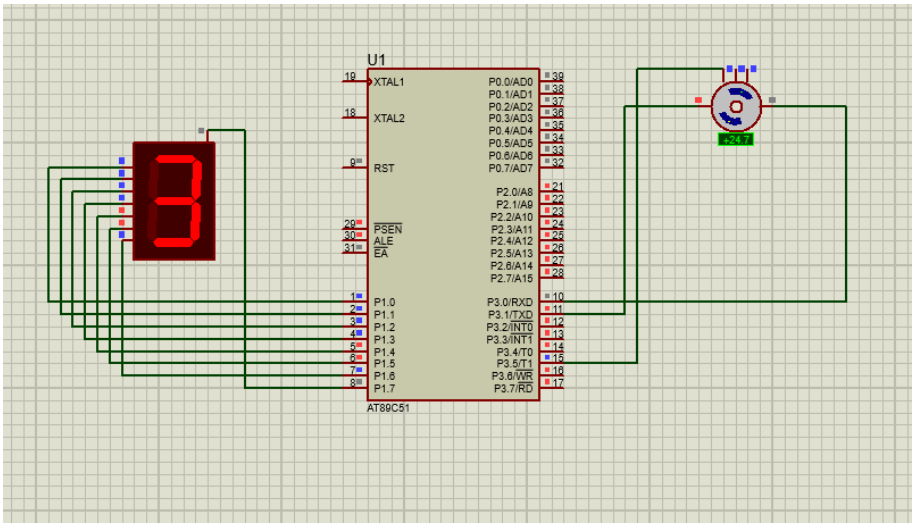
C code

```
#include<reg51.h>

sbit zero = P3^0;
sbit one = P3^1;

void main()
{
    int i;
    // Data for seven segment display
    unsigned char disp[] = ...
        {0xC0, 0xF9, 0xA4, ...
        0x99, 0xB0, 0x92, ...
        0x82, 0xF8, 0x80, 0x90};
    TMOD = 0x50;  // counter in mode 1
    TR1 = 1;      // Start counter

    zero = 0;
    one = 1;  // forward motion of motor

    while(1)
    {
        i = TL1;
        if (i==10)
        {
            TL1 = 0x00; // reset to zero if count exceeds 9
        }
        P1 = disp[i];
    }
}
```

Output: The outputs are shown in Fig. 6



(a)



(b)

Figure 6: Interfacing of Motor

Observation and Discussion: The timer 1 is used as counter to count the number of rotations of motor. The motor sensor is connected to P3.5 pin. It becomes 0 when motor completes one revolution and causes the increment in count. The count is displayed on seven segment display by using stored values for glowing leds to show required count. If count exceeds than 9, we reset it to 0.

Conclusion: The motor has been interfaced with 8051 and count of revolutions has been displayed by using edsim and proteus software

Experiment-6: Generating square wave of 2KHz

Problem Statement:: In this experiment we have to generate square wave of 2KHz frequency. Also count the waves for particular time duration to verify the frequency of wave

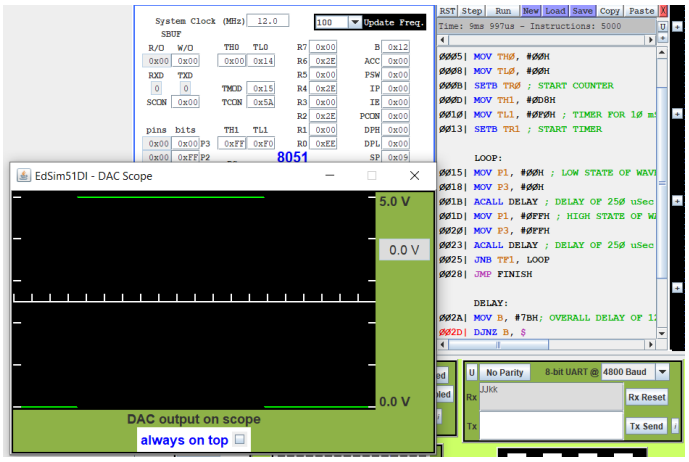
Code: Assembly code

```
CLR P0.7 ; ENABLE DAC

MOV TMOD, #15H ; TIMER 1 IN MODE 1 AND TIMER 0 AS COUNTER IN MODE 1
MOV TH0, #00H
MOV TL0, #00H
SETB TR0 ; START COUNTER
MOV TH1, #0D8H
MOV TL1, #0F0H ; TIMER FOR 10 mS
SETB TR1 ; START TIMER

LOOP:
MOV P1, #00H ; LOW STATE OF WAVE
MOV P3, #00H
ACALL DELAY ; DELAY OF 250 uSec
MOV P1, #0FFH ; HIGH STATE OF WAVE
MOV P3, #0FFH
ACALL DELAY ; DELAY OF 250 uSec
JNB TF1, LOOP
JMP FINISH

DELAY:
MOV B, #7BH; OVERALL DELAY OF 125 uSec INCLUDING OTHER INSTRUCTIONS
DJNZ B, $
RET

FINISH:
END
```

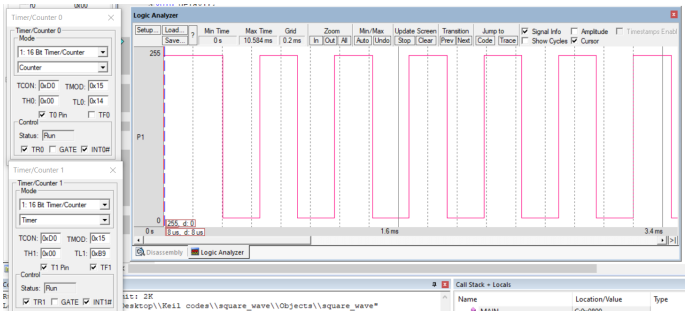
C code

```
#include<reg51.h>
void delay();
void main()
{
    TMOD = 0x15;    /*TIMER 1 IN MODE 1 AND TIMER 0 AS COUNTER IN MODE 1 */
    TH0 = 0x00;
    TL0 = 0x00;
    TR0 = 1;        //START COUNTER

    TH1 = 0xD8;
    TL1 = 0xF0;     // TIMER FOR 10 mS
    TR1 = 1;        // START TIMER

    while(TF1 != 1)
    {

        P1 = 0x00;    // LOW STATE
        P3 = 0x00;
        delay();
        P1 = 0xFF;    // HIGH STATE
        P3 = 0xFF;
        delay();

    }
}

void delay()
{
    int t = 39;
    while(t>=0)
    {
        t--;
    }
}
```

Output: The outputs of the DAC and logic analyser are shown in Fig. 7



(a)



(b)

Figure 7: Square wave of 2 KHz

Observation and Discussion: We have 2 timers. Timer 0 is used as counter for counting the waves and timer 1 is used to notify after 10 ms time. The count obtained in 10 ms × 100 is our counted frequency of wave.

We set B = 7BH, i.e 123 and decrement it till 0. Since DJNZ is 2 cycle instruction, along with other instructions used in loop it gives overall delay of 250 us. The 250 us of off time & 250 us of on time generates the wave of 2KHz frequency.

After execution of program we get count of 14H in timer 1 i.e 20. Hence frequency of wave is 20×100=2KHz

Conclusion: The 2 KHz square wave has been generated and displayed on DAC scope as well as on logic analyser. The frequency of wave is also verified by counting.

Experiment-7: Serial Transmission

Problem Statement:: In this experiment we have to send our name as sample data via serial transmission in mode 0 and mode 1.

Mode 0: Assembly code

```
MOV SCON, #00H  ; transmission in mode 0

MOV A, #00H
MOV DPTR, #LOOKUP
MOV B, #00H
LOOP: MOVC A, @A+DPTR
MOV SBUF, A   ; sending character
JNB TI, $     ; waiting for transmission
CLR TI     ; reseting the flag
INC B
MOV A, B
CJNE A, #06, LOOP
JMP FINISH

LOOKUP:
DB "Aditya"

FINISH:
END
```

C code

```
#include<reg51.h>
void main()
{
    unsigned int i=0;
    char name[] = "Aditya";
    SCON = 0x00;   // mode 0
    for (i=0; i<sizeof(name); i++)
    {
        SBUF = name[i];  // sending character
        while(TI==0);    // wait for transmission to complete
        T1 = 0;          // reset the flag after transmission
    }
}
```

Mode 1: Assembly Code

```
MOV TMOD, #20H  ; Timer 1 in mode 2
MOV TH1, #0FAH  ; buad rate of 4800
MOV SCON, #40H  ; transmission in mode 1
SETB TR1

MOV A, #00H
MOV DPTR, #LOOKUP
MOV B, #00H
LOOP: MOVC A, @A+DPTR
MOV SBUF, A   ; sending character
JNB TI, $     ; waiting for transmission
```

C code

```
#include<reg51.h>
void main()
{
    unsigned int i=0;
    char name[]= "Aditya";   // character
    TMOD = 0x20; ...                // Timer 1 in mode 2
    TH1 = 0xFD; ...                 // count for 3
    SCON = 0x40;             // transmission in mode 1
    TR1 = 1;                 //
```

```
12 CLR TI      ; reseting the flag
13 INC B
14 MOV A, B
15 CJNE A, #06, LOOP
16 JMP FINISH
17
18
19 LOOKUP:
20 DB "Aditya"
21
22 FINISH:
23 END
```

```
            start timer 1
10
11
12
13      for (i=0; i<
            sizeof(name); i++)
14      {
15          SBUF = name[i];      ...
                // send
            character
16          while(TI==0);        ...
                // wait till
            transmission of one
             character is
            complete
17          TI = 0;              ...
                // reset the
            flag
18
19      }
20  }
```

Output: The output of assembly code of mode 0 is not shown as edsim does not have required baud rate available. The output of C code in Keil is shown in Fig. 8 and the output of both assembly and C code in mode 1 are shown in Fig 9

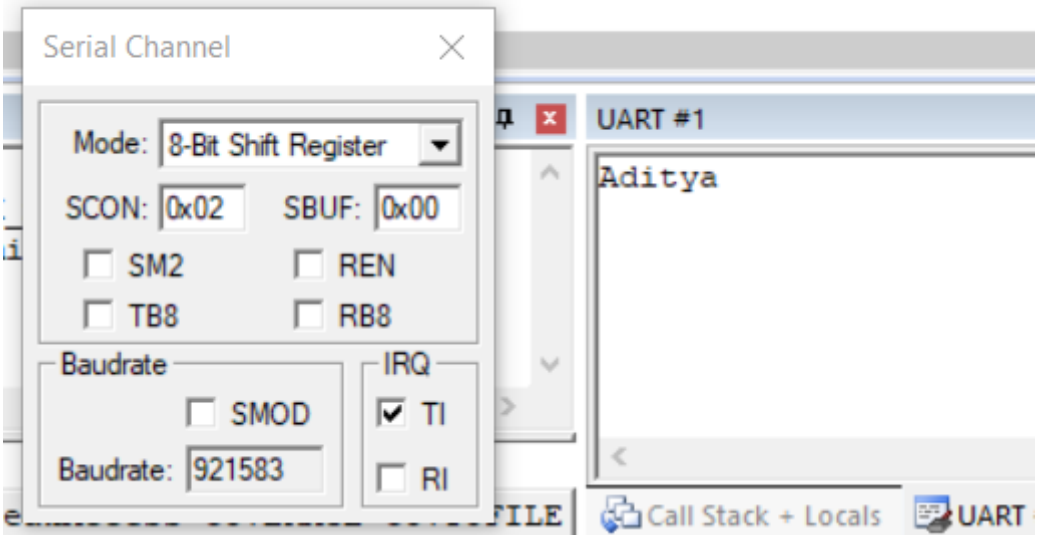
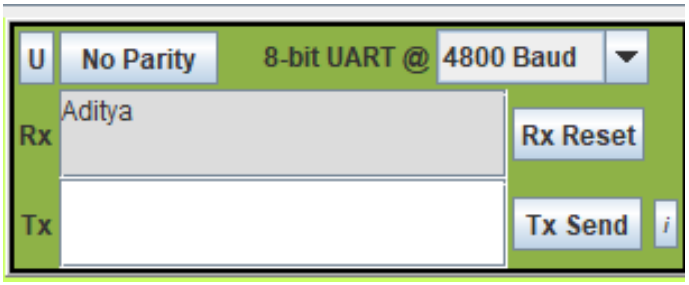
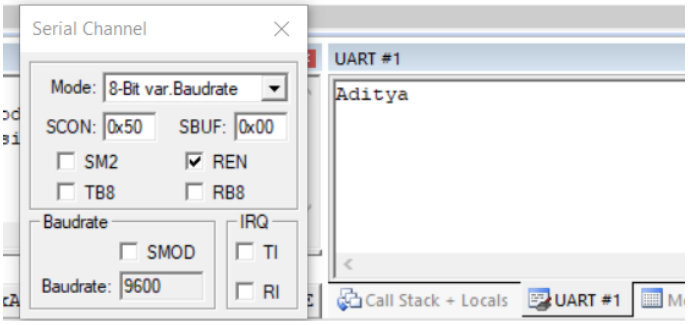


Figure 8: Mode 0



(a)



(b)

Figure 9: Mode 1

Observation and Discussion: To set the serial transmission in mode 0, SCON is set as 00H. Then the characters have been sent one by one to SBUF and waited till transmission if one character is complete. When the transmission is complete, the TI flag becomes 1 and then it is again set to 0 for further transmission.

For mode 1, SCON is set to 40H. Timer 1 is used in mode 2 for transmission in the required baud rate. In C code baud rate is 9600. This is achieved by loading the count of FDH in TH1. In assembly code, the baud rate is 4800 as a baud rate of 9600 is not available to see an output in edsim. This is done by loading the count of FAH in TH1. Rest procedure is the same as mode 0.

Conclusion: The name has been transmitted by serial transmission in both mode 0 & mode 1.

Experiment-8: I^{2}C Communication

Problem Statement:: In this experiment we have to write a sample data in a given memory location of slave by I^{2}C communication.

Mode 0: C code

```
#include <reg66x.h> // include file for P89C66X microcontrollers
void main()
{
S1CON = 0x44;
STA = 1;         // START
while (!SI);  // wait until SI = 1 to confirm
STA = 0;         // START
S1DAT = 0xAA; // send slave address + W
SI = 0;              // clear SI bit
while (!SI);  // wait until SI = 1 to confirm
S1DAT = 0x20; // send data 25H
SI = 0;              // clear SI bit
while (!SI);  // wait until SI = 1 to confirm
STO = 1;              // set STO to generate STOP condition
while (1);
}
```

Output: The output of I^{2}C interface is shown in the Fig 10

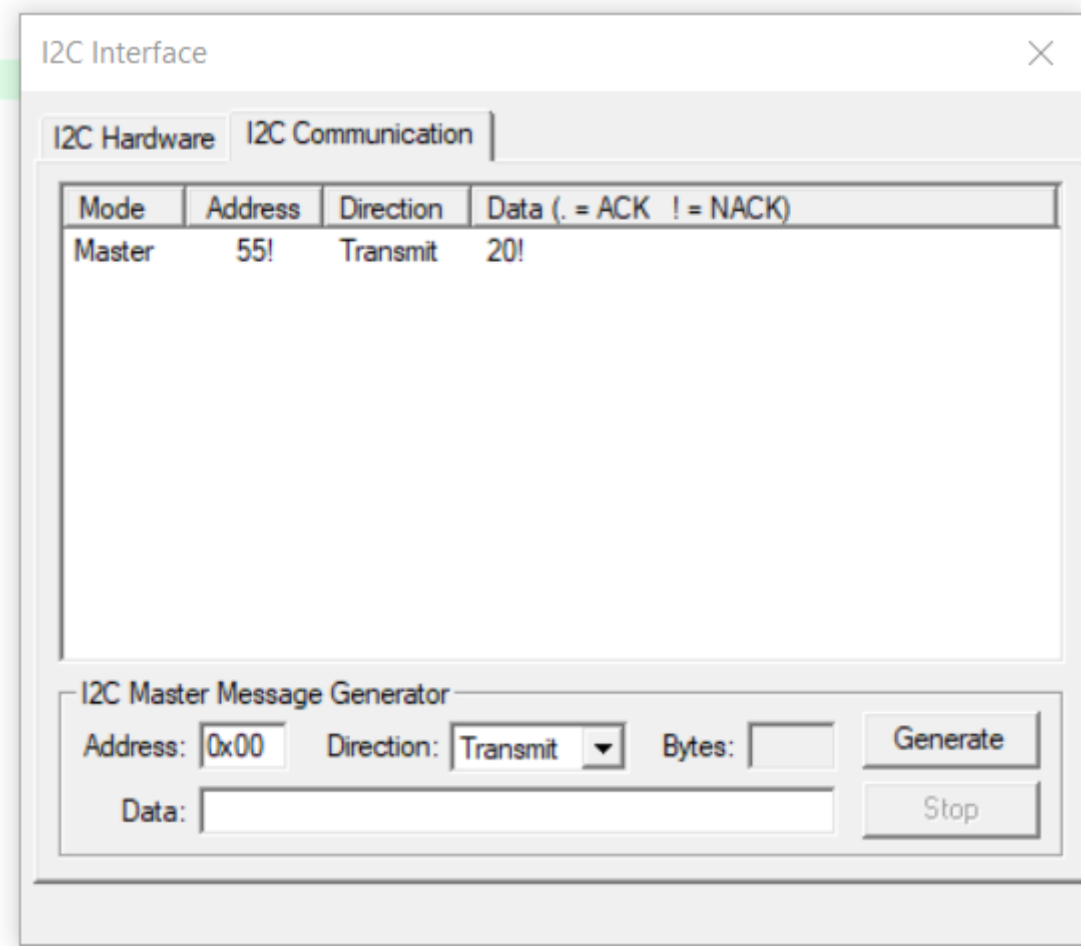


Figure 10: I^2C Communication

Observation and Discussion: The S1CON register is set accordingly to enable I^2C Communication. Then STA is set to be 1 to start communication. First the address in which data is to be written is sent. We wait here until transmission is completed by checking the status of SI. Then the data which is to be written is sent and to complete the communication, STA is set to be 0.

Conclusion: The sample data is sent to specified memory location of slave by I^2C Communication

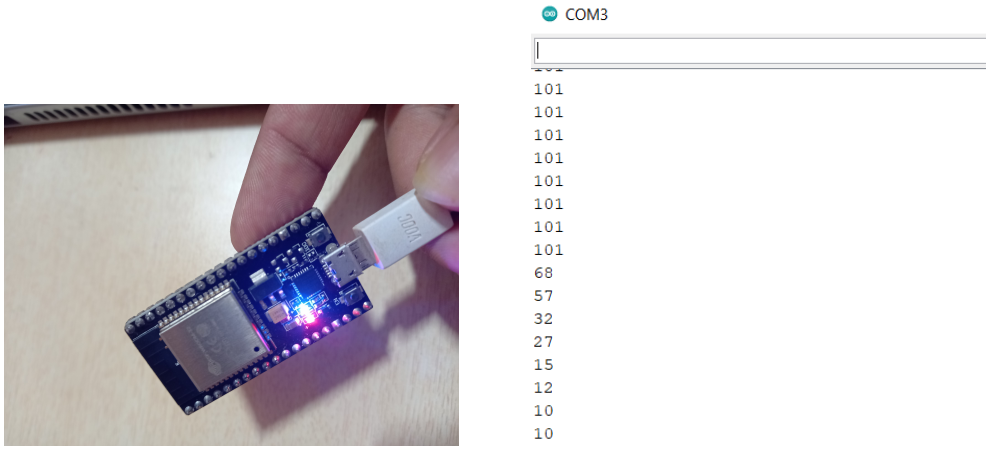
Experiment-9: Touch sensor in ESP32

Problem Statement: In this experiment we have to control built in led of esp32 by touch sensor present in it and also display the sensor reading on serial monitor

Code:

```
int touch;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200); // baud rate
  pinMode (4, INPUT);   // setting pin 4 as input
  pinMode (2, OUTPUT);  // setting pin 2 as output to show ...
      output on built in led

}

void loop() {
  // put your main code here, to run repeatedly:
  touch= touchRead(4);  // reading the values
  Serial.print(touch);  // printing values on serial monitor
  Serial.print('\n');
  if (touch<80) digitalWrite (2, HIGH);  // display of output on LED
  else digitalWrite (2, LOW);
  delay(500);
}
}
```

Output: The output of led and sensor reading on serial monitor are shown in figure 11. To watch video illustration click here



(a) (b)

Figure 11: Touch sensor

Observation and Discussion: When there is no touch to the pin sensor readings are nearly constant at 101 but when we touch finger to the pin sensor reading drops down to 10 to 20 according to extent of touch. The cutoff of touch is set to 80 so that even small touch will be detected and led will glow while touch is detected.

Conclusion: The led has been controlled by touch sensor and sensor readings have been displayed on serial monitor

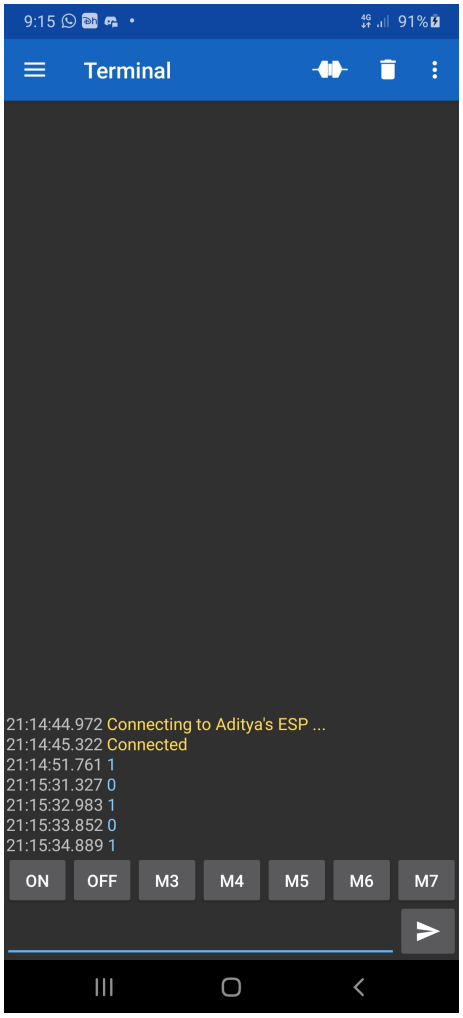
Experiment-10: LED control using Bluetooth

Problem Statement: In this experiment we have to control in built led of esp32 via bluetooth using suitable app

Code:

```
#include "BluetoothSerial.h"
BluetoothSerial BT;
char led;

void setup()
{
  // put your setup code here, to run once:
BT.begin("Aditya's ESP");  // deviece name
pinMode( 2, OUTPUT);       // built in led as output
}

void loop()
{
  // put your main code here, to run repeatedly:
  led = BT.read();    // reading the command sent via bluetooth
  if (led == '1')     // showing output on led
  {
    digitalWrite( 2, HIGH);
  }
  if ( led == '0')
  {
    digitalWrite( 2, LOW);
  }
}
```

Output: The output of in-built led of esp32 is shown in the figure 12. To watch video illustration click here



(a) Serial Bluetooth Terminal App

(b) ESP32

Figure 12: Bluetooth Communication

Observation and Discussion: ESP has inbuilt Bluetooth which can be used to communicate over certain range. Here we are using Serial Bluetooth Terminal android application to send commands to esp via Bluetooth using mobile. If we send '1' through the terminal, the led turns on and if we send '0' through the terminal, the led turns off. Further we can create buttons in application to turn on and off the led.

Conclusion: The built in led of ESP32 has been controlled by mobile app via Bluetooth

Experiment-11: LED control using Wi-Fi

problem Statement: In this experiment we have to control inbuilt led using Web page via WiFi of ESP32

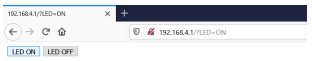
Code:

```
#include "WiFi.h"

const char* ssid = "Aditya-ESP";
const char* Password = "Aditya";

// ngrok for globa testing

WiFiServer server(80);
String html ="<!DOCTYPE html>\
<html>\
<body>\
<form>\
<button name=\"LED\" button value=\"ON\" type=\"submit\">LED ... ON</button> \
<button name=\"LED\" button value=\"OFF\" type=\"submit\">LED ... OFF</button> \
</form> \
</body \
</html>";


void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  pinMode(2, OUTPUT);
  digitalWrite(2, LOW);
  WiFi.softAP(ssid, Password);
  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP Address: ");
  Serial.print(IP);
  server.begin();

}

void loop() {
  // put your main code here, to run repeatedly:
  WiFiClient client = server.available();
  if(client)
  {
    String request = client.readStringUntil('\r');
    if (request.indexOf("LED=ON")≥0)digitalWrite(2, HIGH);
```

```
40      if (request.indexOf("LED=OFF")>=0)digitalWrite(2, LOW);
41      client.print(html);
42      request="";
43    }
44
45  }
```

output: The output of in-built led and web-page is shown in the figure 13. To watch video illustration click here

(a) Serial Monitor



(b) Web page

(c) ESP32

Figure 13: WiFi Communication

Observation and Discussion: We have created a web page using html which is stored in the code in the form of string. The web page which we want to display on browser is not available unless we give the IP address on which it is created. To know this IP address we print it on serial monitor. This web page has two buttons which controls the led of ESP32.

Conclusion: The led has been controlled Via WiFi through webpage

Experiment-12: Hall Sensor

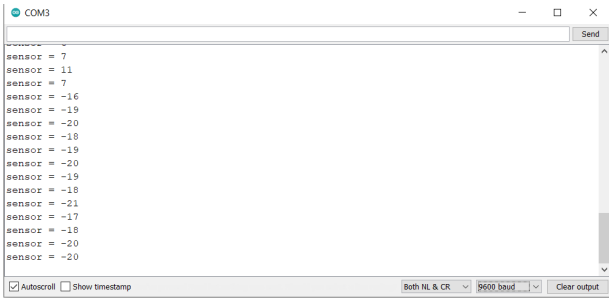
Problem Statement: To display readings of hall sensor on serial monitor and control led using magnet.

Code:

```
int val = 0;
void setup() {
  Serial.begin(9600);  // baud rate
  pinMode (2, OUTPUT); // inbuilt led as output
    }

void loop() {
  val = hallRead();  // reading sensor values
  if (val <0) digitalWrite (2, HIGH);
  else digitalWrite (2, LOW);
  // print the results to the serial monitor:
  Serial.print("sensor = ");
  Serial.println(val);
  delay(500);
}
```

Output: The output of led and serial monitor are shown in the figure 14. To watch video illustration click here

(a) Magnet near Hall sensor



(b) Serial monitor

Figure 14: Hall Sensor

Observation and Discussion: The ESP32 has inbuilt hall sensor which detects presence of magnet near it. If the magnet is not present near, it gives readings nearly about 10. When we bring magnet near to ESP32 the value decreases or increases depending on the pole which is near to hall sensor. The change in reading value also depends on the magnetic power of magnet. In this experiment we are turning on the led if values becomes negative.

Conclusion: The readings of hall sensor are displayed on serial monitor and led is controlled using magnet.

Experiment-13: Temperature Sensor

Problem Statement: To display readings of inbuilt temperature sensor in ESP32

Code:

```
#ifdef __cplusplus
extern "C" {
#endif
uint8_t temprature_sens_read();
#ifdef __cplusplus
}
#endif
uint8_t temprature_sens_read();

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

void loop() {
  Serial.print("Temperature: ");

  // Convert raw temperature in F to Celsius degrees
  Serial.print((temprature_sens_read() - 32) / 1.8);
  Serial.println(" C");
  delay(5000);
}
```

Output: The output of serial monitor is shown in figure 15. To watch video illustration click here

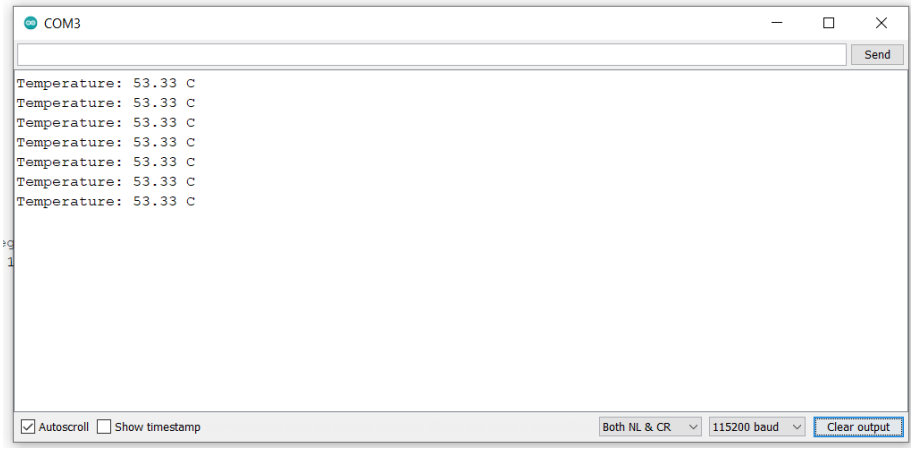


Figure 15: Serial monitor showing temperature reading

Observation and Discussion: The inbuilt temperature sensor gives internal temperature of ESP32 and not the surrounding temperature. The reading of inbuilt temperature sensor is constant every time and that is 53.3°C. This is because from version V2.2 of 2018, the manufacturers removed inbuilt temperature sensor from ESP32 as given in the datasheet. The Older version that has temperature sensor will give correct reading.

Conclusion: The readings of inbuilt temperature sensor has been displayed on serial monitor.

Experiment-14: Cloud storage

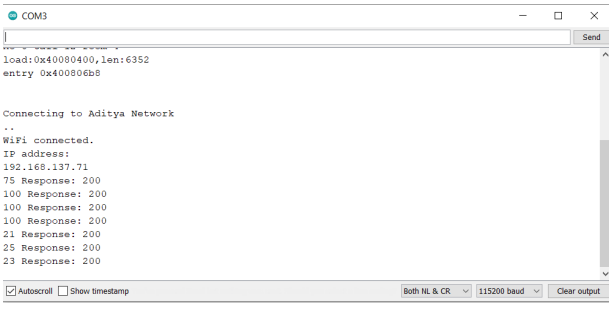
Problem Statement: To send the sensor readings to ThingSpeak cloud storage.

Code:

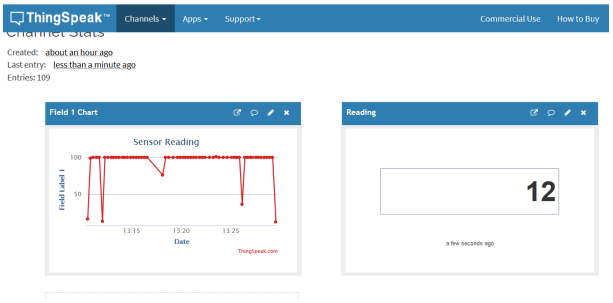
```
#include <WiFi.h>
#include <HTTPClient.h>


const char* ssid     = "Aditya Network";
const char* password = "987654321";

const char* serverName = "https://api.thingspeak.com/update";
String apiKey = "R20ZU3URLYZSHXDE";


WiFiServer server(80);

void setup(){
    Serial.begin(115200);
    pinMode(2, OUTPUT);
    pinMode(4, INPUT);

    // We start by connecting to a WiFi network

    Serial.println();
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

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

    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}

int touch;

void loop(){
 if(WiFi.status()== WL_CONNECTED){
  HTTPClient http;
```

```
44    http.begin(serverName);
45    int touch = touchRead(4);
46    if(touch<20) digitalWrite(2, HIGH);
47    else digitalWrite(2, LOW);
48
49    String DataSent = "api_key=" + apiKey + "&field1=" + ...
          String(touch);
50    int Response = http.POST(DataSent);
51    Serial.print(touch);
52    Serial.print(" Response: ");
53    Serial.println(Response);
54    http.end();
55   }
56  }
```

Output: The screenshot of serial monitor and ThingSpeak page is shown in the figure 16. To watch video illustration click here.



(a) Serial Monitor



(b) ThingSpeak platform

Figure 16: Cloud Storage

Observation and Discussion: ThingSpeak is a online cloud storage platform in which one can store any values in real time without consuming local space and can

access it whenever needed. Here we are updating the readings obtained to inbuilt touch sensor of ESP32 and plotting graph on ThingSpeak to analyse the variation. Here there is particular delay while updating the values on ThingSpeak.

Conclusion: The sensor readings have been sent to ThingSpeak cloud storage.