Visvesvaraya National Institute of Technology (VNIT), Nagpur

Embedded Systems

Theory Tasks

Submitted by :
Sushant Jha (BT19ECE108)
Semester 5

Submitted to :
Dr. Ankit A. Bhurane
(Course Instructor)
Department of Electronics and Communication Engineering,
VNIT Nagpur

Contents

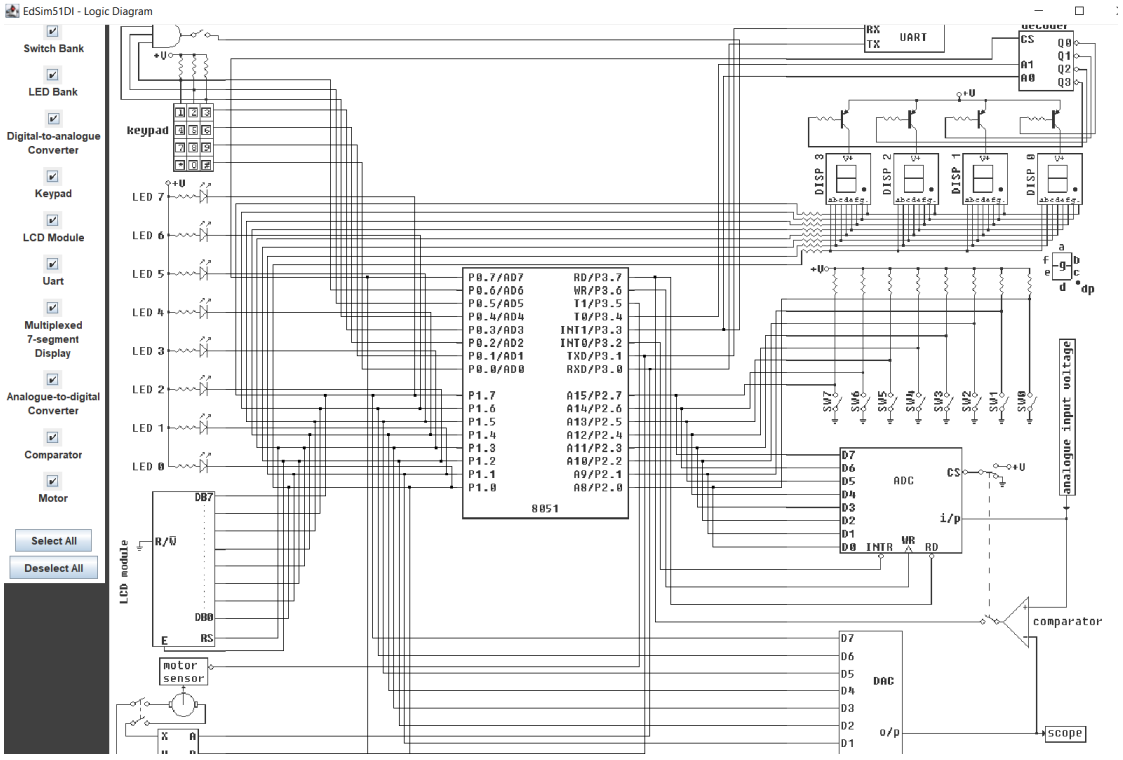
Experiment-1: Blink alternate LEDs on port-1 of 8051 with exactly 1sec delay.

Aim:: Blink alternate LEDs on port-1 of 8051 with exactly 1sec delay.

Requirements:: edsim51

Concept:: We are going to write a program which will glow alternate LEDs on the LED strip and each LED will turn ON and OFF with a gap of exactly 1 second.

Before we get started you need to know the connections with peripherals in edsim51 simulator, see the below image to get an idea:



Simulation Code with Comments:

```
%Code

BACK:
MOV A, #55H
MOV P1, A
ACALL DELAY
```

```
7  MOV A, #0AAH
8  MOV P1, A
9  ACALL DELAY
10 SJMP BACK
11
12
13 DELAY:
14 MOV B, #0EH
15 MOV TMOD, #11H
16 MOV TH0, #0FFH
17 MOV TL0, #0FFH
18 SETB TR0
19
20 T0: JNB TF0, T0
21 DJNZ B, DELAY
22 MOV TH1, #0EEH
23 MOV TL1, #0DDH
24 SETB TR1
25
26 T1: JNB TF1, T1
27 LJMP BACK
```

Output: As we can see from the figure the LED strip on the bottom left corner of the simulator glows alternatively.

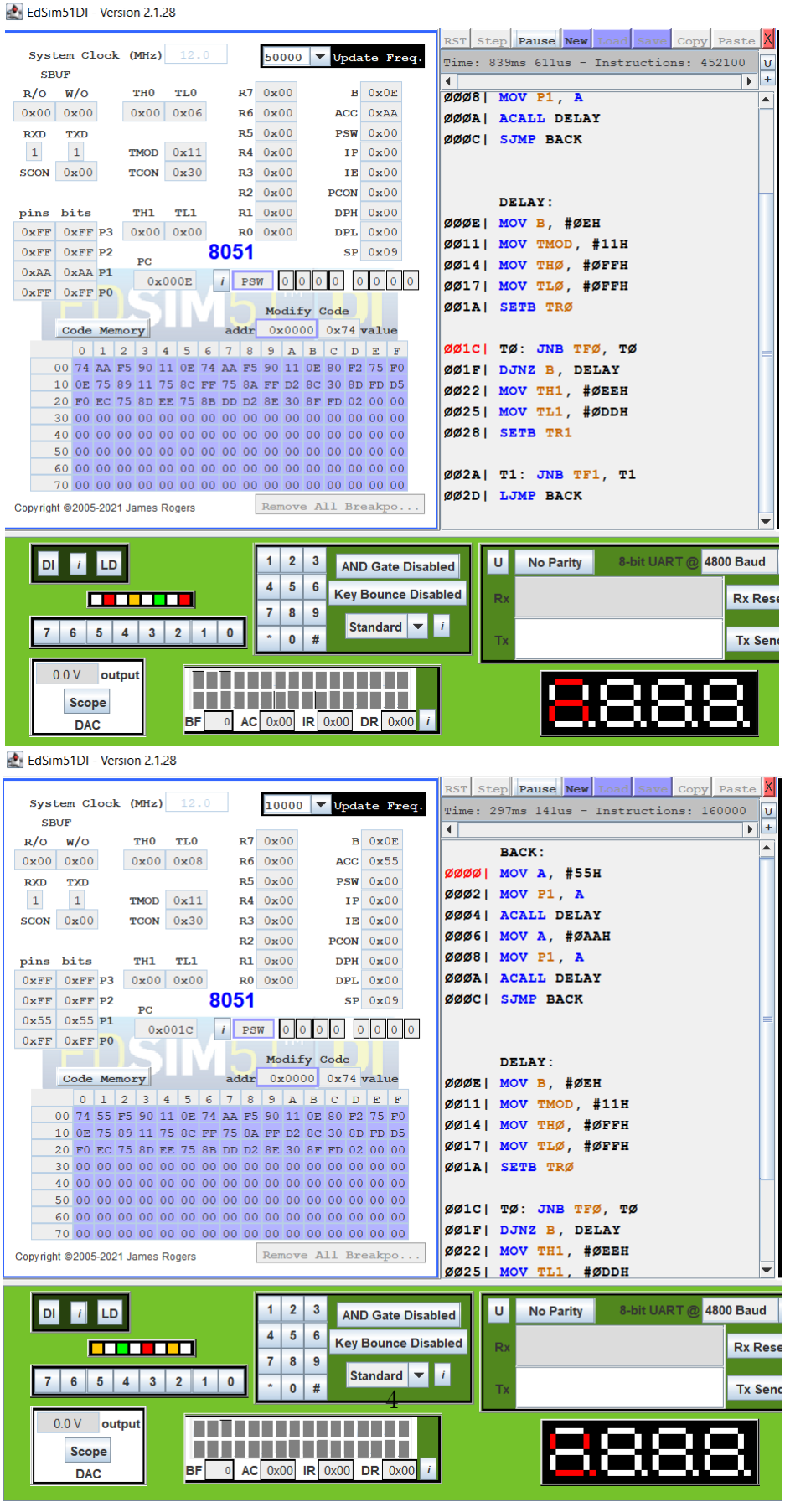


Figure 1: Alternate LEDs glowing.

Experiment-2: Display 0 to 9 on 7 segment display with precise 1 sec delay.

Aim:: Display 0 to 9 on 7 segment display with precise 1 sec delay.

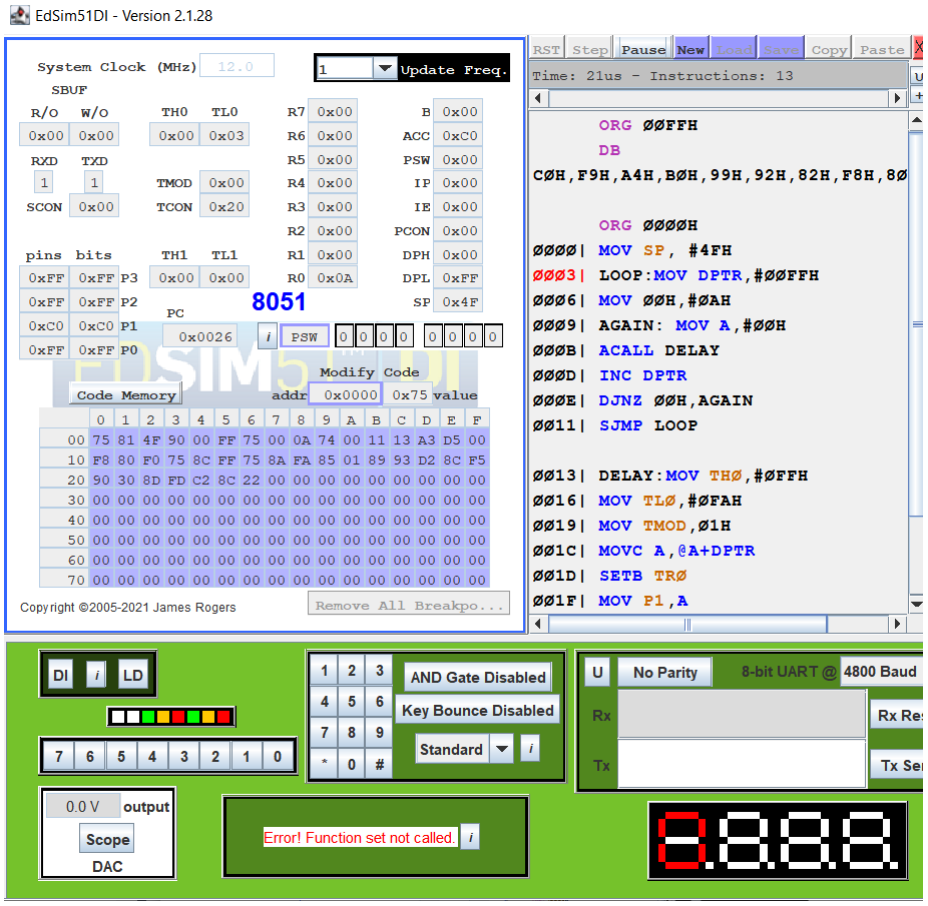
Requirements:: edsim51

Concept:: We are going to write a program which will display with precise 1 sec delay on the 7 segment display the digits from 0 to 9 continuously..

Simulation Code with Comments:

```
30
31 %Code
32
33 ORG 00FFH
34 DB 0C0H,0F9H,0A4H,0B0H,099H,092H,082H,0F8H,080H,090H
35
36 ORG 0000H
37 MOV SP, #4FH
38 LOOP:MOV DPTR,#0FFH
39 MOV 00H,#0AH
40 NEXT: MOV A,#00H
41 ACALL DELAY
42 INC DPTR
43 DJNZ 00H,NEXT
44 SJMP LOOP
45
46 DELAY:
47 MOVC A,@A+DPTR
48 MOV P1,A
49 MOV B, #0EH
50 MOV TMOD, #11H
51 MOV TH0, #0FFH
52 MOV TL0, #0FFH
53 SETB TR0
54
55 T0: JNB TF0, T0
56 DJNZ B, DELAY
57 MOV TH1, #0EEH
58 MOV TL1, #0DDH
59 SETB TR1
60 RET
```

Output: As we can see from the figure the 7-segment display in the bottom of the simulator, the digits are being displayed one by one. Since both the 7-segment display and the LED strip are connected to the same port on the simulator hence, the LED strip also glows up giving us the binary representation of the digits being displayed.



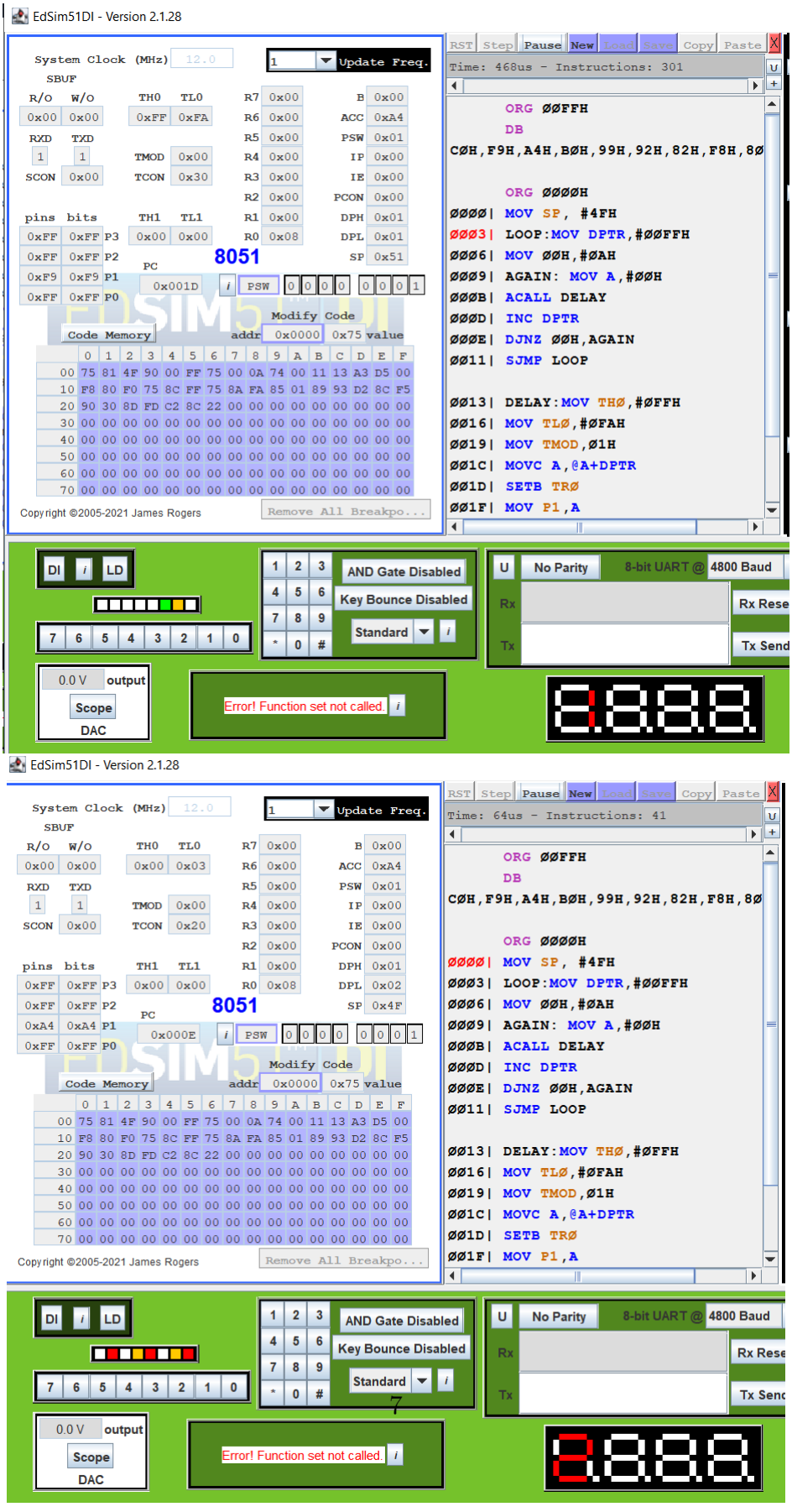


Figure 2: ’0’ , ’1’ , ’2’ Digits being displayed on the 7-segment display.

Experiment-3: Display the digit entered into the 7-segment display onto the keypad.

Aim: Display the digit entered into the 7-segment display onto the keypad.

Requirements: edsim51

Concept: We are going to write a program which will glow alternate LEDs on the LED strip and each LED will turn ON and OFF with a gap of exactly 1 second.

Simulation Code with Comments:

```
%Code
ORG 00FFH
DB 0C0H, 0F9H, 0A4H, 0B0H, 099H, 092H, 082H, 0F8H, 080H, 090H

ORG 0000H
MAIN:   MOV A,#00H
        MOV P3,A
        MOV A,#0FFH
        MOV P0,A
ACALL KEY
SJMP MAIN

ZERO_0: MOV P1,#0C0H
        RET
ONE_1:  MOV P1,#0F9H
        RET
TWO_2:  MOV P1,#0A4H
        RET
THREE_3:MOV P1,#0B0H
        RET
FOUR_4: MOV P1,#099H
        RET
FIVE_5: MOV P1,#092H
        RET
SIX_6:  MOV P1,#082H
        RET
SEVEN_7:MOV P1,#0F8H
        RET
EIGHT_8:MOV P1,#080H
        RET
NINE_9: MOV P1,#090H
        RET
HASH:   MOV P1,#080H
        RET
STAR:   MOV P1,#0BFH
```

```
98          RET
99
100 KEY:    CLR P0.0
101         JNB P0.4,HASH
102         JNB P0.5,ZERO_0
103         JNB P0.6,STAR
104         SETB P0.0
105         CLR P0.1
106         JNB P0.4,NINE_9
107         JNB P0.5,EIGHT_8
108         JNB P0.6,SEVEN_7
109         SETB P0.1
110         CLR P0.2
111         JNB P0.4,SIX_6
112         JNB P0.5,FIVE_5
113         JNB P0.6,FOUR_4
114         SETB P0.2
115         CLR P0.3
116         JNB P0.4,THREE_3
117         JNB P0.5,TWO_2
118         JNB P0.6,ONE_1
119         SETB P0.3
120         RET
```

Output: As we can see from the figure the 7-segment display displays the digit that is pressed in the keypad.

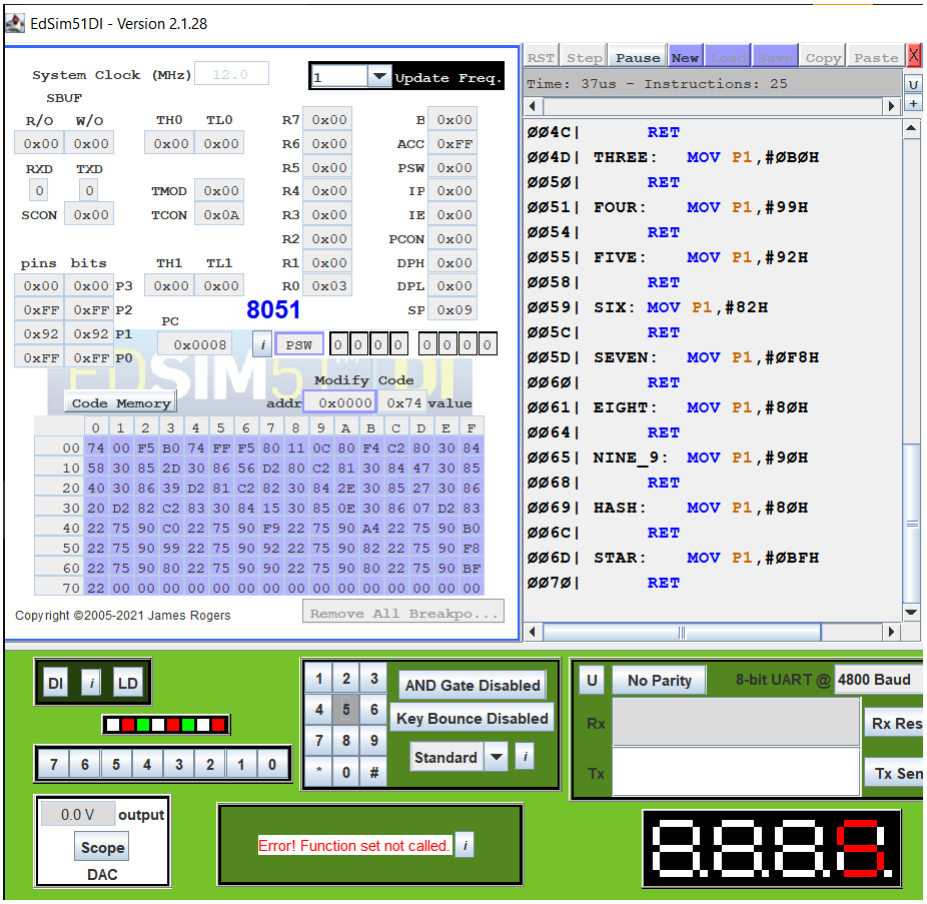


Figure 3: Key '5' being pressed and displayed on the display.

Experiment-4: Interface stepper motor with 8051.

Aim: Interface stepper motor with 8051.

Requirements: edsim51

Explanation: We are going to write a program which will rotate a dc stepper motor.

Simulation Code with Comments:

```
%Code
ORG 0000H
MOV A,#0CCH
LOOP:   MOV P2,A
        RR A
        LCALL DELAY
        SJMP LOOP

DELAY:  MOV TH0,#0FAH
        MOV TL0,#0FAH
        MOV TMOD,02H
        SETB TR0
        LEBEL: JNB TF0,LEBEL
        CLR TR0
        RET
```

Explanation: We are going to write a program which will rotate a dc stepper motor by some fixed amount say 70 degrees. The stepper motor here has a step angle of 2 degrees.

Simulation Code with Comments:

```
%Code
ORG 0000H
MOV A,#0CCH
MOV R3 ,#35
LOOP:   MOV P2,A
        RR A
        LCALL DELAY
        DJNZ R3, LOOP
HERE:   SJMP HERE

DELAY:  MOV TH0,#0FAH
        MOV TL0,#0FAH
```

```
151         MOV TMOD,02H
152         SETB TR0
153         LEBEL: JNB TF0,LEBEL
154         CLR TR0
155         RET
```

Output: In the first case the stepper motor rotates with a certain rpm, in the second case however we move the motor by a fixed angle by calculating the number of steps, here steps = angle total / step size = 70 / 2 = 35.

- Since the stepper motor can be applied power with different terminals, we can adjust the RPM as well as power/Torque of the motor as per our requirement.
- Since stepper motor can be moved with a minimum angle of step size, hence it is possible to rotate it with any amount of angle which is an integral multiple of step size.

Experiment-5: Interface 16 x 2 LCD with 8051.

Aim: To display 'VNIT' on a 16x2 LCD pannel using edsim51 simulator for 8051.

Requirements:: Edsim51

Explanation: In this task we will interface a 16X2 LCD with the 8051 using Edsim51 simulator, and we will display 'VNIT' on it. To do this we will first store the characters of 'VNIT' in the data memory and then send them one by one to the LCD following the LCD interfacing protocols.

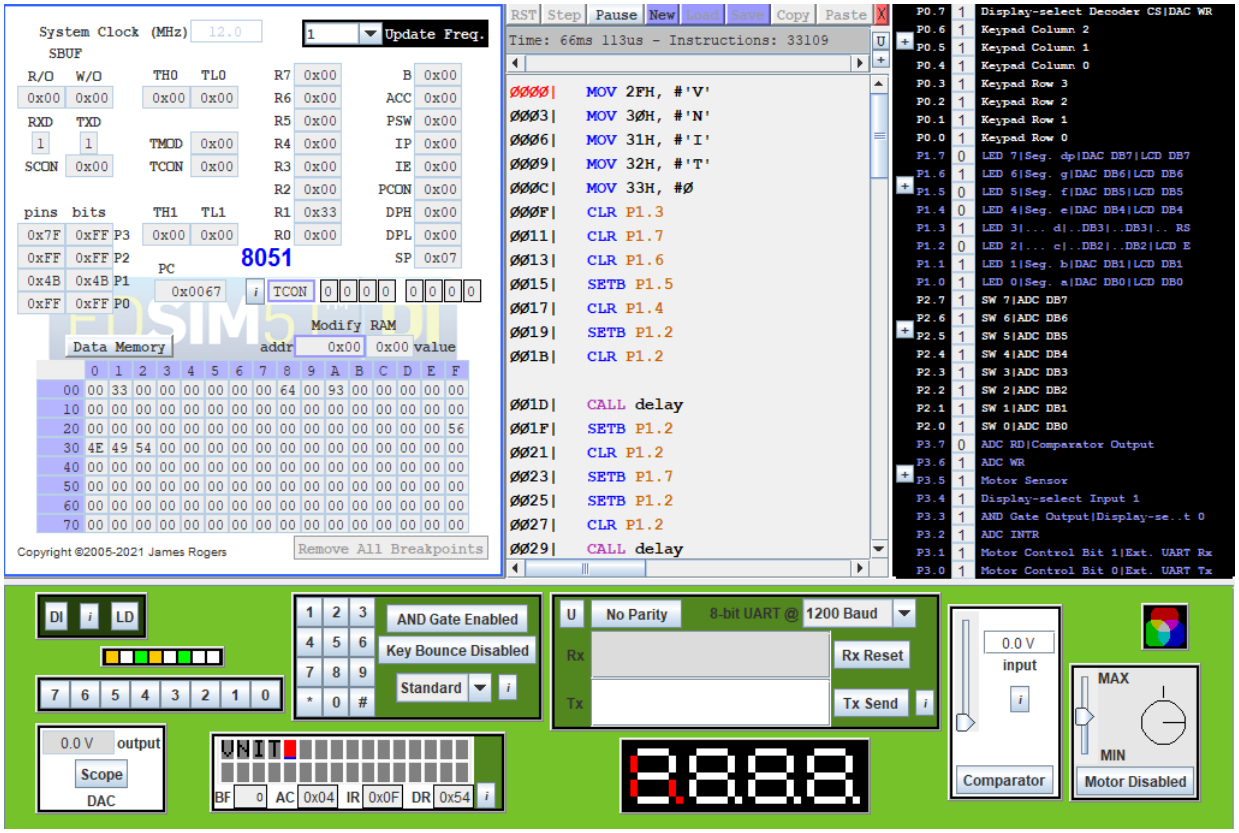
Simulation Code with Comments:

```
%Code
    MOV 2FH, #'V'
    MOV 30H, #'N'
    MOV 31H, #'I'
    MOV 32H, #'T'
    MOV 33H, #0
    CLR P1.3
    CLR P1.7
    CLR P1.6
    SETB P1.5
    CLR P1.4
    SETB P1.2
    CLR P1.2

    CALL delay
    SETB P1.2
    CLR P1.2
    SETB P1.7
    SETB P1.2
    CLR P1.2
    CALL delay
    CLR P1.7
    CLR P1.6
    CLR P1.5
    CLR P1.4

    SETB P1.2
    CLR P1.2

    SETB P1.6
    SETB P1.5

    SETB P1.2
    CLR P1.2
```

```
190
191      CALL delay
192
193      CLR P1.7
194      CLR P1.6
195      CLR P1.5
196      CLR P1.4
197
198      SETB P1.2
199      CLR P1.2
200
201      SETB P1.7
202      SETB P1.6
203      SETB P1.5
204      SETB P1.4
205
206      SETB P1.2
207      CLR P1.2
208      CALL delay
209
210      SETB P1.3
211      MOV R1, #2FH
212  loop:
213      MOV A, @R1
214      JZ finish
215      CALL sendCharacter
216      INC R1
217      JMP loop
218
219  finish:
220      JMP $
221
222  sendCharacter:
223      MOV C, ACC.7
224      MOV P1.7, C
225      MOV C, ACC.6
226      MOV P1.6, C
227      MOV C, ACC.5
228      MOV P1.5, C
229      MOV C, ACC.4
230      MOV P1.4, C
231
232      SETB P1.2
233      CLR P1.2
234
235      MOV C, ACC.3
236      MOV P1.7, C
237      MOV C, ACC.2
238      MOV P1.6, C
```

```
239       MOV C, ACC.1
240       MOV P1.5, C
241       MOV C, ACC.0
242       MOV P1.4, C
243
244       SETB P1.2
245       CLR P1.2
246
247       CALL delay
248
249  delay:
250       MOV R0, #50
251       DJNZ R0, $
252       RET
```

Output: The simulator shows this output:



Explanation:

- To interface an LED we need to follow some protocols, these are explained below:
- First select the LED.
- Then Reset and Set EN to latch the data to the display.
- Send the data serially to the LCD.

Complete guide to ESP-32.

Details of ESP-32: ESP-32 is an on-chip microcontroller with:

a) Dual core processor

b) Inbuilt Wi-Fi, Bluetooth

c) 520 KB internal SRAM

d) 4 MB external flash

e) 30 pin board

f) Micro-USB interface.

These features allow ESP-32 to act as a amazing device for IOT based applications. Using these features of ESP-32 we are going to simulate some useful tasks which include:

1) Using touch input to control inbuilt LED and send the same to serial monitor.

2) Switching the inbuilt LED using Bluetooth.

3) Using ESP-32 in Wi-Fi station mode.

4) Send weather data to ThingSpeak cloud by using ESP-32 in Wi-Fi client mode.

5) Home automation- Switching LED using Telegram.

6) Home automation- Switching LED using Google assistant.

7) Using RTOS for dual-core management.

8) Using RTOS for dual-task management.

These tasks help us to garner an idea of all the features provided by the ESP-32, and also opens your mind to the immense number of new ideas that can simplify life as we know it.

For all these tasks we need ESP32 board and open-source Arduino Software (IDE) for writing code and USB-A to micro-USB cable for uploading code to ESP32.

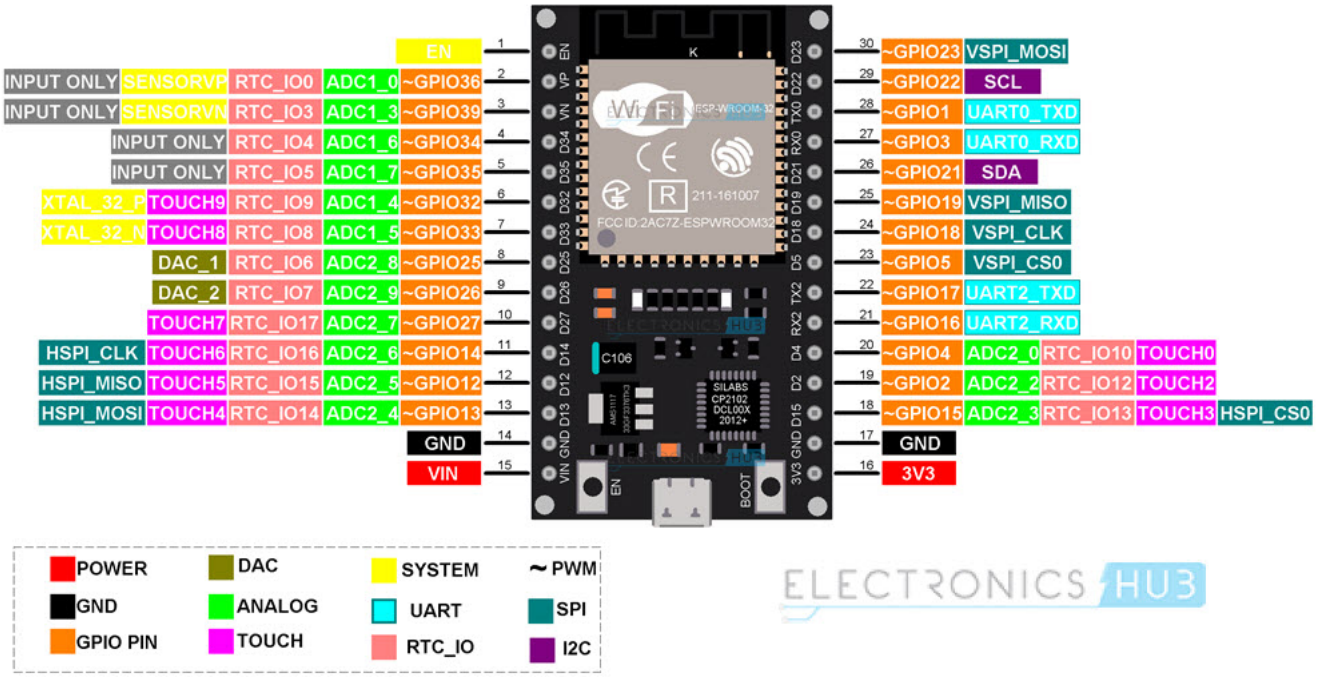


Figure 4: I/O description of ESP32

Task 1: Using touch input to control inbuilt LED and send the same to serial monitor.

Aim: We are going to read touch from any of the 9 touch sensing pins available on ESP32 board.

Concept: The touch pins on ESP32 act as a capacitive sensor, which means that when these pins are touched, the voltage read from these pin drop to voltage of the finger. This voltage change can be used to our advantage for changing the state of inbuilt LED present on the Pin - 2 of the ESP32.

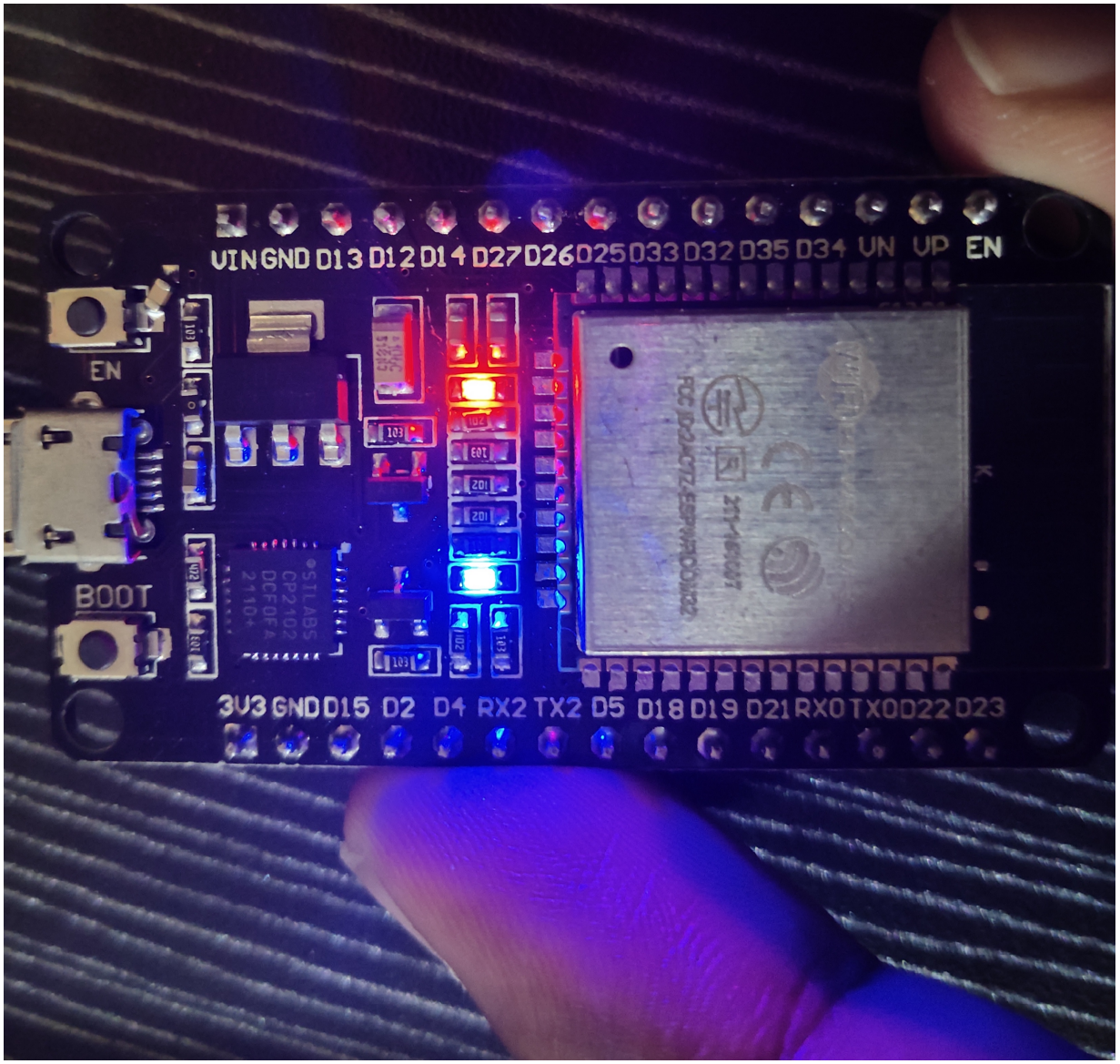
Simulation Code with Comments:

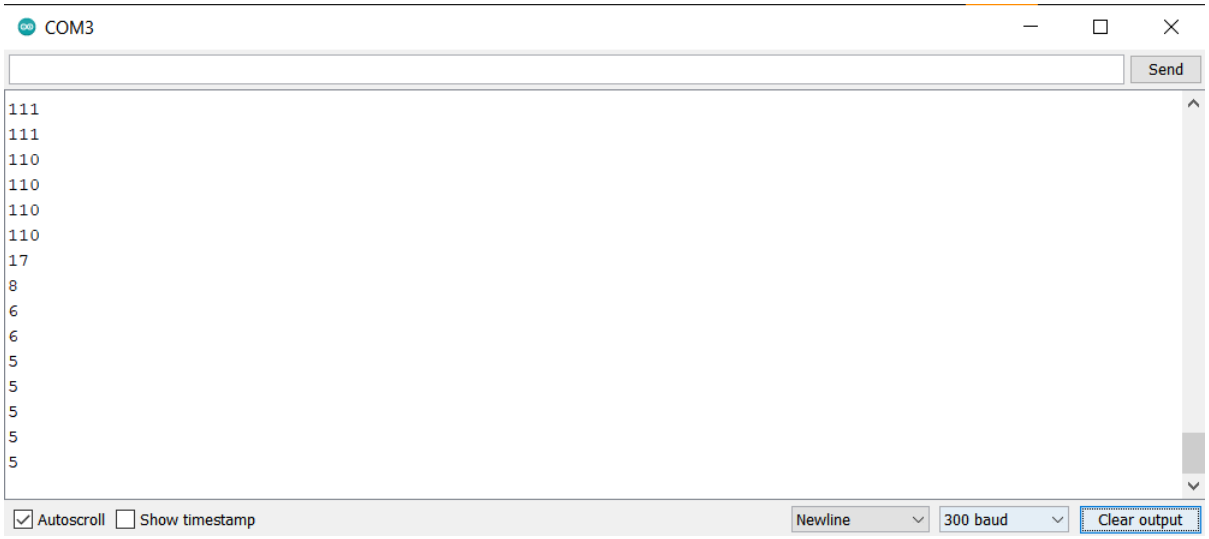
```
%Code

int t;
void setup()
{
    // the code here runs once on boot:
    pinMode(4,INPUT);
    pinMode (2, OUTPUT);
    Serial.begin(300);
}

void loop()
{
    // the code here runs repeatedly:
    t = touchRead(4);
    Serial.println(t);
    if (t < 90)
    digitalWrite(2,HIGH);
    else
    digitalWrite(2,LOW);
}
```

Output: The blue color LED on the ESP32 glows on the board whenever the pin -4 is touched.

Explanation: The blue color LED on the ESP32 is connected to the pin -2 and the pin -4 can act as a capacitive sensor. Using these we set the sensor as input and the LED as the output. 'touchRead()' is an inbuilt function for returning the voltage of the pin in mV (integer), we store this in a predefined integer t, and the compare the voltage(t) with our desired voltage to get required sensitivity.

The data from the pin can also be sent to the PC using micro-USB. This is achieved by setting a baud rate and then listening to it on that baud rate through serial monitor. Here we are using baud rate of 300. Note that the available baud rates for communications are in multiple of '2'. Higher baud rate can be used for faster data transfer.




COM3
Send
111
111
110
110
110
110
17
8
6
6
5
5
5
5
5
Autoscroll
Show timestamp
Newline
300 baud
Clear output

Task 2: Switching the inbuilt LED using Bluetooth.

Aim: We will extend the previous task by taking the input from bluetooth instead of touch.

Requirements: Serial Bluetooth terminal app on smartphone or PC and relay module.

Concept: ESP32 has inbuilt Bluetooth which can be used to send and receive data wirelessly. We use this functionality by sending '0' on serial Bluetooth app for turning OFF the LED and '1' for turning it ON. The remaining part of switching LED remains same as the previous work.

Simulation Code with Comments:

```
%Code

#include "BluetoothSerial.h"
BluetoothSerial SerialBT;
char r;
void setup() {
    // the code here runs once on boot:
    SerialBT.begin("ESP-32");
    pinMode(2,OUTPUT);
}

void loop() {
    // the code here runs repeatedly:
    r = SerialBT.read();
    if (r == '1'){
      digitalWrite(2,HIGH);}
    if (r == '0'){
      digitalWrite(2,LOW);}
 }
```

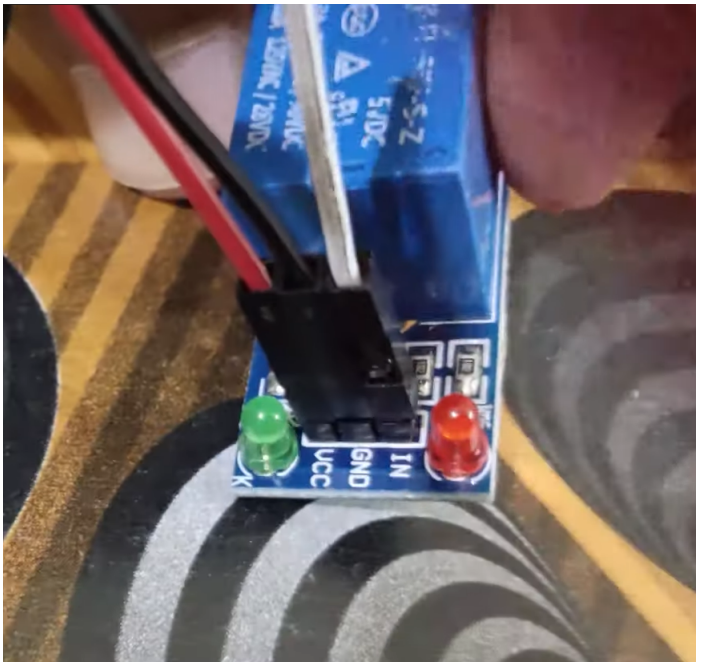
Output: The LED glows when '1' is sent to the ESP32 and turns OFF when '1' is sent.

Going Further: The above functionality can be extended to some practical use by connecting a Bulb to ESP32 using relay. The code remains similar but the connections need to be done to relay. The working model is shown here on YouTube.

```
%Code
```

```
296
297 #include "BluetoothSerial.h"
298 BluetoothSerial SerialBT;
299 char r;
300 void setup() {
301     // the code here runs once on boot:
302     SerialBT.begin("ESP-32");
303     pinMode(4,OUTPUT);
304 }
305
306 void loop() {
307     // the code here runs repeatedly:
308     r = SerialBT.read();
309     if (r == '1'){
310       digitalWrite(4,HIGH);}
311     if (r == '0'){
312       digitalWrite(4,LOW);}
313  }
```

Connections: The connections are shown in the images:



Task3: Using ESP-32 in Wi-Fi station mode.

Aim: To run ESP32 as a server/station which hosts site. Then interact with this sever to change state of LED..

Concept: ESP32 can be used in station mode wherein a webpage can be stored on it and will be accessable to anyone who connects to it using the SSID and password of the ESP32.

Simulation Code with Comments:

```
%Code
#include "WiFi.h"
#define LED0 2
const char* ssid = "personal";
const char* password = "0123456789";
WiFiServer server(80);
String html ="<!DOCTYPE html> \
<html> \
<body> \
<center><h1>LED control using esp as server to host a ...
    webpage</h1></center> \
<form> \
<button name=\"LED\" button style=\"color:green\" value=\"ON\" ...
    type=\"submit\">LED0 ON</button> \
<button name=\"LED\" button style=\"color=red\" value=\"OFF\" ...
    type=\"submit\">LED0 OFF</button><br><br> \
</form> \
</body> \
</html>";


void Connect_WiFi()
{
WiFi.begin(ssid, password);
while(WiFi.status() != WL_CONNECTED)
{
delay(100);
}
}

void setup()
{
Serial.begin(115200);
pinMode(LED0, OUTPUT);
digitalWrite(LED0, LOW);
Serial.print("Setting soft access point mode");
```

```
348 WiFi.softAP(ssid, password);
349 IPAddress IP = WiFi.softAPIP();
350 Serial.print("AP IP address: ");
351 Serial.println(IP);
352 server.begin();
353 }
354
355
356 void loop()
357 {
358 WiFiClient client=server.available();
359 if(client)
360 {
361 String request = client.readStringUntil('\r');
362 if(request.indexOf("LED0=ON") != -1)digitalWrite(LED0, HIGH);
363 if(request.indexOf("LED0=OFF") != -1)digitalWrite(LED0, LOW);
364 client.print(html);
365 request="";
366 }
367 }
```

Output: Webpage opened by connecting to ESP32 which is in station mode:

Explanation:

- We can access the webpage by writing the IP adress of the server in the address bar.
- We have wriiten a basic html code which displays this heading with two buttons on the webpage.

- When these buttons are pressed the value corresponding to those buttons are changed.

Task 4: Send DHT11 weather data to ThingSpeak cloud by using ESP-32 in Wi-Fi client mode.

Aim: We will take Temperature , Humidity and Heat index reading using DHT11 sensor and send the data to cloud (thingSpeak) using its API in ESP32.

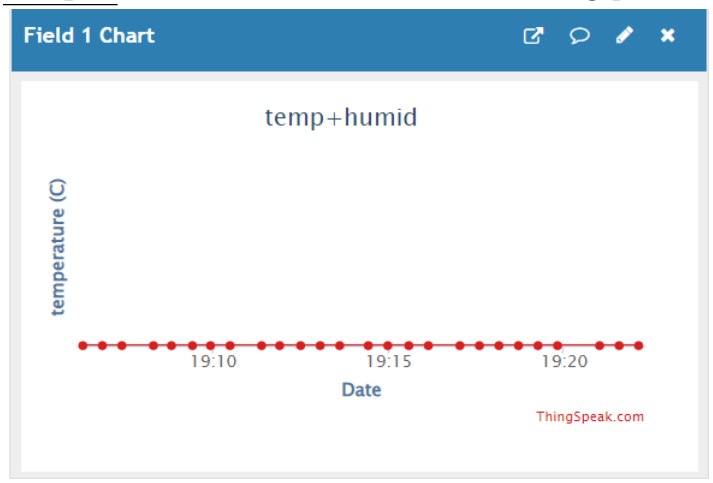
Concept: DHT11 sensor is bent for weather monitoring. It can communicate serially to ESP32. We install the libraries for functions related to DHT11 and then take the readings. By default the sensor takes readings in Fahrenheit so we need to convert them to Celsius. To get acquainted with some common terms:
Temperature : Measurement of actual hotness.
Humidity: Measurement of the concentration of water in air. Heat index: Temperature that is felt by a human body.

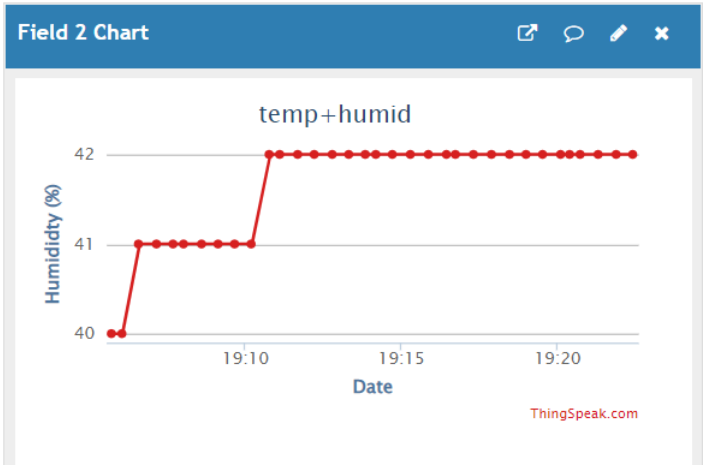
Simulation Code with Comments:

```
%Code
#include "WiFi.h"
#include "ThingSpeak.h"
#include "DHT.h"

#define DHTPIN 4
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

#define CHANNEL_ID 1550217
#define API_KEY "JRUYCT5JJPX7DLGZ"
#define WIFI_TIMEOUT_MS 10000
#define WIFI_NETWORK "PERSONAL"
#define WIFI_PASSWORD "0123456789"
#ifdef __cplusplus

WiFiClient client;

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

uint8_t temprature_sens_read();
void setup() {
    WiFi.mode(WIFI_STA);
    WiFi.begin(WIFI_NETWORK, WIFI_PASSWORD);
```

```
399     ThingSpeak.begin(client);
400     Serial.begin(9600);
401     Serial.println(F("DHTxx test!"));
402     dht.begin();
403 }
404 void loop() {
405     delay(2000);
406     float h = dht.readHumidity();
407     // Read temperature as Celsius (the default)
408     float t = dht.readTemperature();
409     // Read temperature as Fahrenheit (isFahrenheit = true)
410     float f = dht.readTemperature(true);
411 
412     // Check if any reads failed and exit early (to try again).
413     if (isnan(h) || isnan(t) || isnan(f)) {
414       Serial.println(F("Failed to read from DHT sensor!"));
415       return;
416     }
417     // Compute heat index in Fahrenheit (the default)
418     float hif = dht.computeHeatIndex(f, h);
419     // Compute heat index in Celsius (isFahreheit = false)
420     float hic = dht.computeHeatIndex(t, h, false);
421 
422     Serial.print(F("Humidity: "));
423     Serial.print(h);
424     Serial.print(F("%  Temperature: "));
425     Serial.print(t);
426     Serial.print(F[U+FFFD]'C));
427     Serial.print(f);
428     Serial.print(F[U+FFFD]EHeat index: "));
429     Serial.print(hic);
430     Serial.print(F[U+FFFD]'C));
431     Serial.print(hif);
432     Serial.println(F[U+FFFD]'F));
433 
434     ThingSpeak.writeField(CHANNEL_ID,2,h,API_KEY);
435     delay(500);
436     ThingSpeak.writeField(CHANNEL_ID,1,t,API_KEY);
437     delay(500);
438 }
```

Output: The data can be viewed on thingspeak cloud:





Explanation:

- We need to make a channel on Thingspeak with two fields: one for the humidity sensor and the other for the temperature.
- Then get the API key and use it ESP32 to write data to the cloud.
- The library for DHT11 needs to be installed for using the inbuilt functions for reading the data off the sensor.
- To see the working in real time head over to my YouTube video.

Task 5: Home automation- Send touch sensor data through Telegram.

Aim: We are going to sense input from Pin-4 of ESP32 and send it to user through Telegram API and Telegram bot.

Concept: We have read data as touch input in the 1st task here, ESP32 will be connected to the internet and we will send the touch data to Telegram bot on PC/phone using Telegram API on ESP32.
We can customize the message as per situation, here it is 'Touch detected!' and 'Nope!' for when the pin is touched and not, respectively.

Simulation Code with Comments:

```
%Code

#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>
#include <ArduinoJson.h>

// network details
const char* ssid = "personal";
const char* password = "0123456789";

// BOT initialization
#define BOTtoken "2016834462:AAGjfMeU1IGvbm63f_9pJwaf-BJw5fb0DDQ"

#define CHAT_ID "994003044"

WiFiClientSecure client;
UniversalTelegramBot bot(BOTtoken, client);

const int touchsensor = 4; // PIR Motion Sensor
int V;
void setup() {
  Serial.begin(115200);

  // PIR Motion Sensor mode INPUT_PULLUP
  pinMode(touchsensor, INPUT);

  // Attempt to connect to Wifi network:
  Serial.print("Connecting Wifi: ");
  Serial.println(ssid);

  WiFi.mode(WIFI_STA);
```

```
472   WiFi.begin(ssid, password);
473   client.setCACert(TELEGRAM_CERTIFICATE_ROOT); // Add root ...
          certificate for api.telegram.org
474
475   while (WiFi.status() != WL_CONNECTED) {
476     Serial.print(".");
477     delay(500);
478   }
479
480   Serial.println("");
481   Serial.println("WiFi connected");
482
483 }
484
485 void loop() {
486    if (V<80){
487     bot.sendMessage(CHAT_ID, "Touch detected!!", "");
488     Serial.println("Touch Detected");
489     delay(2000);
490     V = touchRead(4);
491     }
492     else{
493     bot.sendMessage(CHAT_ID, "Nope..!!", "");
494     Serial.println("Nope");
495     delay(2000);
496     V = touchRead(4);
497       }
498 }
```

Output: The telegram messages by bot looks like:

Explanation:

- To simulate the same using ESP32 you need to first setup BOT using botfather-bot available on telegram.
- You also need to install UniversalTelegramBot for using inbuilt functions required to setup and send data to the bot.
- Then you need to get the ID of your bot.
- Then you need to send data to this bot using the ID.

Task 6: Home automation- Switching LED using Google assistant.

Aim: We will use some keywords in google assistant for switching LED.

Concept: We can control esp32 from anywhere via internet connection. We have demonstrated this in previous tasks, we extend this functionality by sending the data via google assistant.
Since google assistant is capable of taking text input as well as audio input which it can convert it to text with speech recognition we can use both text and speech keywords for sending some data to ESP32.
So here we send data to Adafruit IO through google assistant, it passes the data to ESP32 and also stores the data on IFTTT. You can also follow this site for the task too.

Other requirements: Account on IFFT, Adafruit IO
Google assistant enabled device

Setup: Adafruit setup:

1) Create a new feed with desired name (preferably the operation name)

2) Create new block containing an ON-OFF switch.

3) Get the Username and Key (will be used in ESP32 setup).

IFTTT (If this then that) setup:

1) Create new applet.

2) Set up the assistant keywords and name.

3) Include Adafruit mqtt library in ESP32 IDE.

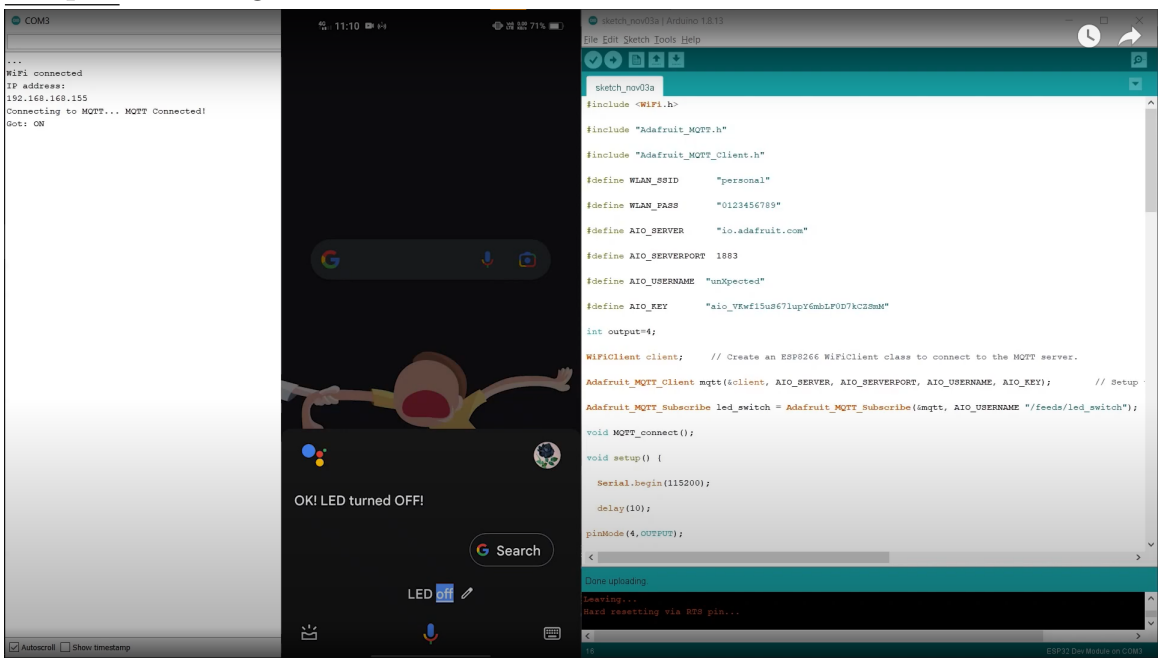
Simulation Code with Comments:

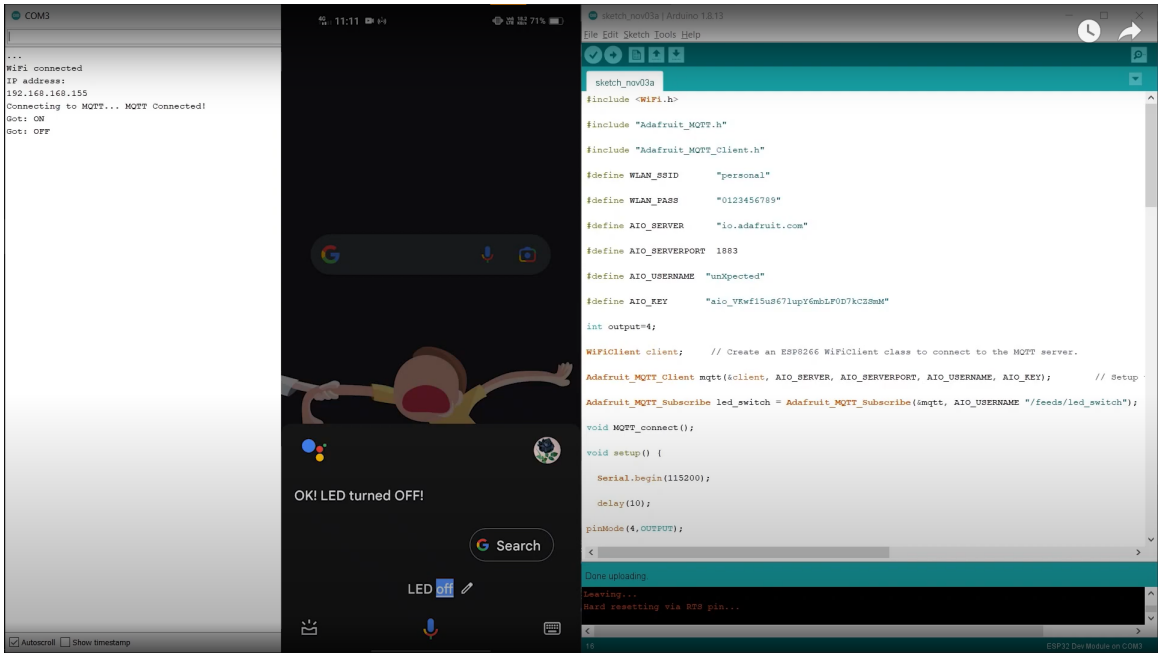
```
%Code

#include <WiFi.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
#define WLAN_SSID       "personal"
#define WLAN_PASS       "0123456789"
#define AIO_SERVER      "io.adafruit.com"
```

```
#define AIO_SERVERPORT  1883
#define AIO_USERNAME  "unXpected"
#define AIO_KEY       "aio_VKwf15uS67lupY6mbLF0D7kCZSmM"
int output=4;
WiFiClient client;
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, ...
    AIO_USERNAME, AIO_KEY);
Adafruit_MQTT_Subscribe led_switch = ...
    Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/led_switch");
void MQTT_connect();


void setup() {
  Serial.begin(115200);
  delay(10);
pinMode(4,OUTPUT);
 // Connect to WiFi access point.
  Serial.println(); Serial.println();
  Serial.print("Connecting to ");
  Serial.println(WLAN_SSID);
  WiFi.begin(WLAN_SSID, WLAN_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
 Serial.println("WiFi connected");
  Serial.println("IP address: "); Serial.println(WiFi.localIP());
  mqtt.subscribe(&led_switch);
}
uint32_t x=0;


void loop() {
   MQTT_connect();
Adafruit_MQTT_Subscribe *subscription;
  while ((subscription = mqtt.readSubscription(5000))) {
    if (subscription == &led_switch) {
      Serial.print(F("Got: "));
      Serial.println((char *)led_switch.lastread);
       if (!strcmp((char*) led_switch.lastread, "ON"))
      {
        digitalWrite(4, HIGH);
      }
      else
      {
        digitalWrite(4, LOW);
      }
    }
```

```
555     }
556   }
557   void MQTT_connect() {
558     int8_t ret;
559     // Stop if already connected.
560     if (mqtt.connected()) {
561       return;
562     }
563    Serial.print("Connecting to MQTT... ");
564   uint8_t retries = 3;
565     while ((ret = mqtt.connect()) != 0) { // connect will return 0 ...
          for connected
566         Serial.println(mqtt.connectErrorString(ret));
567         Serial.println("Retrying MQTT connection in 5 seconds...");
568         mqtt.disconnect();
569         delay(5000);  // wait 5 seconds
570         retries--;
571         if (retries == 0) {
572           // basically die and wait for WDT to reset me
573           while (1);
574         }
575     }
576     Serial.println("MQTT Connected!");
577   }
```

Output: Working:





Explanation:

1) As expected the assistant data is relayed to ESP32 using the Adafruit API.

2) We observe very high latency (10s approx) this is expected because of the multi-level operation at work here.

3) Since the Adafruit provides different kinds of buttons/ storage types,we can extend this task to other kinds of applications too, say controlling speed of motor.

Task 7: Using RTOS for dual-task management.

Aim: We will run two completely separate tasks on two cores of ESP32 using RTOS.

Concept: RTOS - Real Time Operating System.
RTOS provides efficient pipelining for multi-core processing.
Since ESP32 is a double core processor it provides us the flexibility to run separate tasks on two cores at the same time. Using this ability we will try to execute a simple code wherein two counters will run on the two cores of ESP32.
Note that by default ESP32 runs any task on core-0.

Simulation Code with Comments:

```
%Code

int c1 = 0;
int c2 = 0;
void t1(void *par){
  for(;;){
    Serial.print("Task1 running on core ");
    Serial.println(xPortGetCoreID());
    Serial.print("Task 1 count:");
    Serial.print(c1++);
    Serial.println();
    vTaskDelay(1000);
  }
}

void t2(void *par){
  for(;;){
  digitalWrite(2,HIGH);
  delay(1000);
  digitalWrite(2,LOW);
  delay(1000);
  }

}

void setup() {
  // runs on setup:
  pinMode(2,OUTPUT);
  Serial.begin(9600);
  xTaskCreatePinnedToCore(
    t1,       // Task function
    "Task 1", // Task Name
    1000,     // Stack size
```

```
612     NULL,      // Parameter for the task
613     1,         // Priority, high for high no.
614     NULL,      // Task handle
615     0);        // Core
616
617  xTaskCreatePinnedToCore(
618     t2,        // Task function
619     "Task 2", // Task Name
620     1000,      // Stack size
621     NULL,      // Parameter for the task
622     1,         // Priority, high for high no.
623     NULL,      // Task handle
624     1);        // Core
625 }
626 void loop() {
627   // no code required here, code already in loop from setup
628 }
```

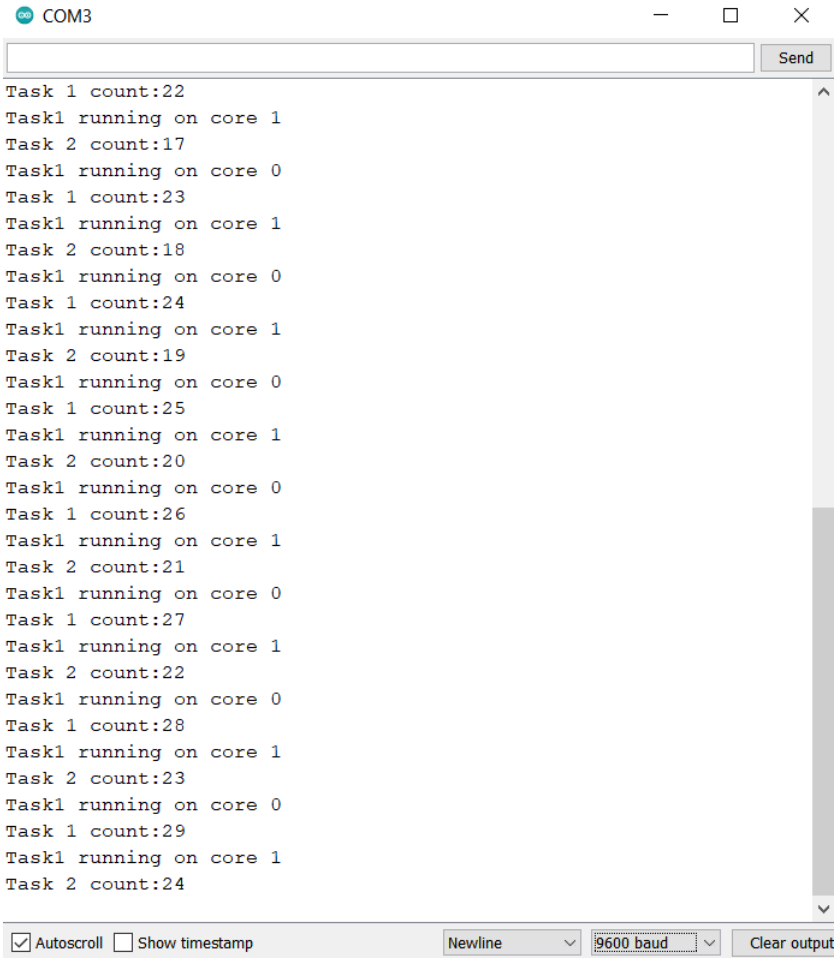
Extension: Now we can even pause and resume the task of one core based on the condition of other task. To do this vTaskSuspend(), vTaskResume()can be used. The code for the same looks like this:

Simulation Code with Comments:

```
630 %Code
631
632 int c1 = 0;
633 int c2 = 0;
634 TaskHandle_t t2h = NULL;
635 void t1(void *par){
636   for(;;){
637     Serial.print("Task1 running on core ");
638     Serial.println(xPortGetCoreID());
639     Serial.print("Task 1 count:");
640     Serial.print(c1++);
641     if (c1 == 10){
642       vTaskSuspend(t2h);
643     }
644     if (c1 == 15){
645       vTaskResume(t2h);
646     }
647     Serial.println();
648     vTaskDelay(1000);
649   }
650 }
651
652 void t2(void *par){
```

```
653   for(;;){
654     Serial.print("Task1 running on core ");
655     Serial.println(xPortGetCoreID());
656     Serial.print("Task 2 count:");
657     Serial.print(c2++);
658     Serial.println();
659     vTaskDelay(1000);
660   }
661 }
662
663 void setup() {
664   // put your setup code here, to run once:
665   pinMode(2,OUTPUT);
666   Serial.begin(9600);
667   xTaskCreatePinnedToCore(
668     t1,       // Task function
669     "Task 1", // Task Name
670     1000,     // Stack size
671     NULL,     // Parameter for the task
672     1,        // Priority, high for high no.
673     NULL,     // Task handle
674     0);       // Core
675
676  xTaskCreatePinnedToCore(
677     t2,       // Task function
678     "Task 2", // Task Name
679     1000,     // Stack size
680     NULL,     // Parameter for the task
681     1,        // Priority, high for high no.
682     &t2h,     // Task handle
683     1);       // Core
684 }
685 void loop() {
686   // no code here, since looping in setup itself:
687 }
```

Output: The serial monitor shows the count:



Explanation:

- In the multicore programming more than one process is executed at once on a single chip.
- We stop the counter-2 when the counter-1 reaches 10 and then resume when the counter reaches 15.
- This kind of functionality is very advantageous in processing and is used in every modern processor.
- Other commands can be found on the RTOS website.

Task 8: Using RTOS for dual-task management with separate cores for input and output.

Aim: We extend the last task even further by taking an integer input form the user on core-1 and then using that data to control the blinking of led using core-2.

Concept:

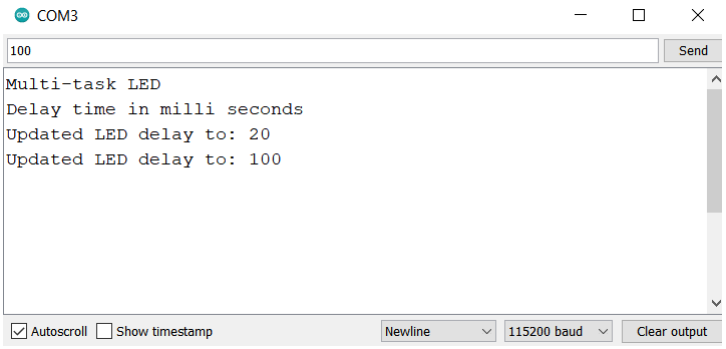
Simulation Code with Comments:

```
%Code

// Needed for atoi()
#include <stdlib.h>

#if CONFIG_FREERTOS_UNICORE
  static const BaseType_t app_cpu = 0;
#else
  static const BaseType_t app_cpu = 1;
#endif
static const uint8_t buf_len = 20;
static const int led_pin = 2;

static int led_delay = 500;   // ms


// Task: Blink LED at rate given by the user
void FlickerLED(void *parameter) {
  while (1) {
    digitalWrite(led_pin, HIGH);
    vTaskDelay(led_delay / portTICK_PERIOD_MS);
    digitalWrite(led_pin, LOW);
    vTaskDelay(led_delay / portTICK_PERIOD_MS);
  }
}



void serialIN(void *parameters) {

  char c;
  char buf[buf_len];
  uint8_t idx = 0;

  // Clear whole buffer
  memset(buf, 0, buf_len);

```

```
726   // Loop forever
727   while (1) {
728
729     // Read characters from serial
730     if (Serial.available() > 0) {
731       c = Serial.read();
732
733       // Update delay variable and reset buffer if we get a ...
          newline character
734       if (c == '\n') {
735         led_delay = atoi(buf);
736         Serial.print("Updated LED delay to: ");
737         Serial.println(led_delay);
738         memset(buf, 0, buf_len);
739         idx = 0;
740       } else {
741
742         // Only append if index is not over message limit
743         if (idx < buf_len - 1) {
744           buf[idx] = c;
745           idx++;
746         }
747       }
748     }
749   }
750 }
751
752
753
754 void setup() {
755   pinMode(led_pin, OUTPUT);
756
757   // Configure serial and wait a second
758   Serial.begin(115200);
759   vTaskDelay(1000 / portTICK_PERIOD_MS);
760   Serial.println("Multi-task LED");
761   Serial.println("Delay time in milli seconds");
762
763   // Start blink task
764   xTaskCreatePinnedToCore(
765             FlickerLED,
766             "Flicker LED",
767             1024,
768             NULL,
769             1,
770             NULL,
771             app_cpu);
772
773   // Start serial read task
```

```
774   xTaskCreatePinnedToCore(
775               serialIN,
776               "Seial in",
777               1024,
778               NULL,
779               1,
780               NULL,
781               app_cpu);
782
783   // Delete "setup and loop" task
784   vTaskDelete(NULL);
785 }
786
787 void loop() {
788   // looping in setup itself.
789 }
```

Output: The output looks like this while adjusting frequency of the LED:



Explanation:

- Here we have used two cores for two separate tasks, one for taking input other for blinking LED.
- We could do this on the same core but then the core will have to alternate between the tasks which will create more delay while taking input and also the blinking time will vary very slightly.
- This will happen because the cores will be busy for some cycles in doing the other task.