```
#include <WiFi.h>
#include <esp_now.h>
#include <Keypad.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// ================================
// OLED SETUP
// ================================
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1
#define SDA_PIN 8
#define SCL_PIN 9

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// ================================
// Transmitter MAC Address (ESP8266)
// ================================
uint8_t senderMAC[] = {0xEC, 0xFA, 0xBC, 0xCA, 0x2A, 0xF3};

// ================================
// Data Structure
// ================================
typedef struct struct_message {
  float temperature;
} struct_message;

struct_message incomingData;
bool dataReceived = false;
unsigned long lastPacketTime = 0;
#define DATA_TIMEOUT 2000

// ================================
// Keypad Setup
// ================================
#define ROWS 4
#define COLS 4
char keyMap[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
```

```
  {'#','0','*','D'}
};
byte rowPins[ROWS] = {35, 36, 37, 39};
byte colPins[COLS] = {20, 21, 45, 40};
Keypad keypad = Keypad(makeKeymap(keyMap), rowPins, colPins, ROWS, COLS);

// ================================
// Thresholds
// ================================
float lowerThreshold = 0.0;
float upperThreshold = 100.0;
bool thresholdsSet = false;
bool inThresholdInput = false;

// ================================
// BUZZER ADDITION
// ================================
#define BUZZER_PIN 10
bool buzzerActive = false;
unsigned long lastBuzzToggle = 0;
bool buzzState = false;

// ================================
// LED PINS
// ================================
#define GREEN_LED 1
#define YELLOW_LED 2
#define RED_LED 42

// Red LED blink
unsigned long lastRedBlink = 0;
bool redState = false;

// ================================
// Callback when data is received
// ================================
void OnDataRecv(const esp_now_recv_info_t *recv_info, const uint8_t *incomingDataBytes, int
len) {
  memcpy(&incomingData, incomingDataBytes, sizeof(incomingData));
  dataReceived = true;
  lastPacketTime = millis();

  if (!inThresholdInput) {
```

```
    Serial.printf("📡 Received from %02X:%02X:%02X:%02X:%02X:%02X → Temperature:
%.2f\n",
                  recv_info->src_addr[0], recv_info->src_addr[1], recv_info->src_addr[2],
                  recv_info->src_addr[3], recv_info->src_addr[4], recv_info->src_addr[5],
                  incomingData.temperature);

    if (thresholdsSet) {
      float temp = incomingData.temperature;
      if (temp < lowerThreshold) Serial.println(" 🔻 Low Temp");
      else if (temp > upperThreshold) Serial.println(" 🔺 High Temp");
      else Serial.println("✅ In Range");

      // BUZZER CONTROL
      if (temp < lowerThreshold || temp > upperThreshold) {
        buzzerActive = true;
      } else {
        buzzerActive = false;
        digitalWrite(BUZZER_PIN, LOW);
      }
    }
  }
}

// ================================
// Function to input a float value via keypad
// ================================
float inputFloatValue(const char* prompt, bool isLower) {
  Serial.print(prompt);
  String inputStr = "";
  bool negative = false;
  bool decimalPointUsed = false;

  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0,0);
  display.print(isLower ? "Lower Threshold:" : "Upper Threshold:");
  display.display();

  while (true) {
    char key = keypad.getKey();
    if (key) {
      if (key == 'A') {
        if (!negative && inputStr.length()==0) { negative = true; Serial.print("-"); }
      } else if (key == 'B') {
```

```
      if (!decimalPointUsed) {
        decimalPointUsed = true;
        if (inputStr.length()==0) { inputStr += "0"; Serial.print("0"); }
        inputStr += ".";
        Serial.print(".");
      }
    } else if (key >= '0' && key <= '9') { inputStr += key; Serial.print(key); }
    else if (key == 'D') {
      if (inputStr.length() > 0) {
        char last = inputStr.charAt(inputStr.length()-1);
        inputStr.remove(inputStr.length()-1);
        Serial.print("\b \b");
        if (last == '.') decimalPointUsed = false;
      } else if (negative) { negative = false; Serial.print("\b \b"); }
    } else if (key == 'C') {
      Serial.println();
      display.clearDisplay();
      display.setTextSize(1);
      display.setCursor(0,0);
      display.print(isLower ? "Lower Threshold:" : "Upper Threshold:");
      display.setTextSize(2);
      display.setCursor(0,20);
      if (negative) display.print("-");
      display.print(inputStr);
      display.display();
      delay(500);
      break;
    }

    // Update value while typing
    display.setTextSize(2);
    display.setCursor(0,20);
    if (negative) display.print("-");
    display.print(inputStr + "   ");
    display.display();
  }
  delay(10);
 }

 float value = inputStr.toFloat();
 if (negative) value = -value;
 return value;
}
```

```
// ================================
// Setup
// ================================
void setup() {
  Serial.begin(115200);

  pinMode(47, OUTPUT);
  digitalWrite(47, LOW);

  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);

  // LED setup
  pinMode(GREEN_LED, OUTPUT);
  pinMode(YELLOW_LED, OUTPUT);
  pinMode(RED_LED, OUTPUT);
  digitalWrite(GREEN_LED, LOW);
  digitalWrite(YELLOW_LED, LOW);
  digitalWrite(RED_LED, LOW);

  Wire.begin(SDA_PIN, SCL_PIN);

  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("❌ SSD1306 not found. Check wiring!");
    while(true);
  }

  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(10,20);
  display.print("OLED OK");
  display.display();
  delay(1000);

  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0,0);
  display.print("Boot complete...");
  display.display();
  delay(500);

  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) { Serial.println("❌ Error initializing ESP-NOW"); return; }
```

```
  esp_now_register_recv_cb(OnDataRecv);

  esp_now_peer_info_t peerInfo = {};
  memcpy(peerInfo.peer_addr, senderMAC, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo) != ESP_OK) { Serial.println("❌ Failed to add ESP8266 as
peer"); return; }

  Serial.println("✅ ESP32-S3 Receiver Ready");
  Serial.println("Press '*' to enter threshold input mode.");
}

// ================================
// Loop
// ================================
void loop() {
  char key = keypad.getKey();

  if (key == '*') {
    buzzerActive = false;
    digitalWrite(BUZZER_PIN, LOW);

    display.clearDisplay();
    display.setTextSize(2);
    display.setCursor(20,20);
    display.print("RESET");
    display.display();
    delay(800);

    inThresholdInput = true;
    Serial.println("\n--- Enter threshold mode ---");

    lowerThreshold = inputFloatValue("Enter LOWER threshold: ", true);
    upperThreshold = inputFloatValue("Enter UPPER threshold: ", false);

    if (upperThreshold < lowerThreshold) {
      float tmp = upperThreshold;
      upperThreshold = lowerThreshold;
      lowerThreshold = tmp;
      Serial.println("Swapped thresholds because upper < lower.");
    }
```

```
    thresholdsSet = true;
    inThresholdInput = false;

    Serial.printf("Thresholds set → Lower: %.2f  Upper: %.2f\n", lowerThreshold,
upperThreshold);
    Serial.println("--- Returning to real-time temperature output ---");
  }

  // BUZZER BEEPING LOGIC
  if (buzzerActive) {
    if (millis() - lastBuzzToggle > 500) {
      buzzState = !buzzState;
      digitalWrite(BUZZER_PIN, buzzState ? HIGH : LOW);
      lastBuzzToggle = millis();
    }
  }

  // TIMEOUT CHECK FOR NO DATA
  if(dataReceived && millis() - lastPacketTime > DATA_TIMEOUT) dataReceived = false;

  // ================================
  // OLED DISPLAY & LED LOGIC
  // ================================
  display.clearDisplay();

  if(inThresholdInput) {
      digitalWrite(GREEN_LED, LOW);
      digitalWrite(RED_LED, LOW);
      digitalWrite(YELLOW_LED, LOW);
  } else if(!dataReceived) {
      display.setTextSize(2);
      display.setCursor((SCREEN_WIDTH - 6*7*2)/2, (SCREEN_HEIGHT - 16)/2);
      display.print("NO DATA");
      digitalWrite(YELLOW_LED, HIGH);
      digitalWrite(GREEN_LED, LOW);
      digitalWrite(RED_LED, LOW);
  } else {
      digitalWrite(YELLOW_LED, LOW);

      display.setTextSize(1);
      display.setCursor(0,0);
      display.print("Lower:");
      display.print(lowerThreshold,1);
      display.print("  Upper:");
```

```
    display.print(upperThreshold,1);

    display.setTextSize(3);
    String tempStr = String(incomingData.temperature,1);
    if (incomingData.temperature < 0) tempStr = "-" + tempStr; // negative fix
    int16_t x1, y1;
    uint16_t w, h;
    display.getTextBounds(tempStr, 0, 0, &x1, &y1, &w, &h);
    display.setCursor((SCREEN_WIDTH - w)/2, (SCREEN_HEIGHT - h)/2);
    display.print(tempStr);

    // LED logic
    if(incomingData.temperature >= lowerThreshold && incomingData.temperature <=
upperThreshold) {
        digitalWrite(GREEN_LED, HIGH);
        digitalWrite(RED_LED, LOW);
    } else {
        digitalWrite(GREEN_LED, LOW);
        if(millis() - lastRedBlink >= 500) {
            redState = !redState;
            digitalWrite(RED_LED, redState);
            lastRedBlink = millis();
        }
    }
  }

  display.display();
  delay(50);
}
```