Please select your current program below:

☐ **Mechatronics Engineering**

Course Number	301
Course Title	MTE301
Semester/Year	Fall 2025
Section Number	[redacted]
Group Number	[redacted]

Final Project MTE 301

Assignment Title	Thermal Detection System
Submission Date	November 24th 2025
Due Date	November 24th 2025

Student Name	Student ID (xxxx1234)	Signature*
[redacted]	[redacted]	[redacted]
Aryan Billava	[redacted]	
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

(Note: Remove the first 4 digits from your student ID

**By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: https://www.torontomu.ca/senate/policies/academic-integrity-policy-60/.*

MTE 301 Final Project:
Thermal Detection System

Professor: Robin Chhabra
Section [redacted]

Table of Contents

Executive Summary

One of the most important physical quantities that we measure is temperature, which influences processes in engineering, manufacturing, consumer products, and safety-critical environments. Numerical displays can be difficult to interpret in fast paced or high risk environments. Our project proposes a robotic system that collects real time temperature data through digital sensors and then translates it into a visual safety indicator using a colour mapped 64 x 64 RGB matrix. The system enhances interpretability for users by converting the temperature values to a gradient of colours, along with clear warnings when reaching potentially harmful levels. A buzzer is integrated in the system to serve as an alarm for when the temperature exceeds 30°C, along with a caution message reading “Too Hot!” that will be displayed at the bottom to indicate the potential danger and reinforce user safety. Altogether, with the combination of visual and auditory cues, the system greatly improves user awareness and reduces the likelihood of accidental burns. The final implementation combines Arduino Mega based sensing and ESP32 driven display to form a fully functioning, user oriented mechatronic system, making temperature monitoring faster and more meaningful for both industrial and consumer applications.

Introduction

Background:

Temperature measurement plays a very important role in industrial processes, product safety, and everyday consumer applications. Typical thermometers do provide accurate readings; however, they are not always ideal in environments where quick decision making is required. In some high-risk thermal situations, even a brief delay in the interpretation of numerical values can lead to burns, equipment damage, or system failure. Prolonged exposure to temperatures around 45°C can cause injury, and temperatures ranging from 44-46°C can result in burns within 30 seconds. These realities significantly highlight the importance and need for temperature monitoring systems that communicate danger quickly and intuitively.

System Description:

The Thermal Detection System developed in this project provides a visual and auditory method for evaluating thermal safety. The temperature data is gathered using the D2180B digital sensors connected to the Arduino Mega, which handles gathering data, LCD output, and buzzer activation. The readings are transmitted to the ESP32 microcontroller, which then maps each value to a predefined colour gradient and displays it on the RGB matrix. The color representation on the matrix is shown with a dark blue/purple shade for temperatures 15°C and below, and then as the temperature increases, the color on the matrix transitions from dark blue/purple to deep red, shown at 30°C and above. A buzzer alerts the user when the temperature exceeds the threshold and flashes a “Too Hot!” warning at the bottom of the display. This combination provides users with multiple sensory cues, which enhance safety and accessibility.

Objectives:

The main objectives of the project focus on developing a responsive and accessible thermal detection system. First, the system must read the temperature values in real time from the environment, and it must do this accurately. Second, the readings must be converted into a clear colour based spectrum on the RGB matrix for a quick visual interpretation. Third, the design must incorporate safety features such as the buzzer alarm and the on screen warnings when the temperature exceeds the safe thresholds. Fourth, the system must aim to improve accessibility by reducing the reliance on only numerical temperature displays. Lastly, the system should demonstrate effective communication between the Arduino Mega and the ESP32 to ensure a smooth data transfer.

Statement of Work Completed:

Over the course of the project, the group successfully assembled and programmed a multi-microcontroller thermal detection system using an Arduino Mega and an ESP32. The Arduino Mega was programmed to read temperature values from the DS18B20 sensors, display the temperatures on the LCD, and activate the buzzer when temperatures went over 30°C. The communication between the Arduino Mega and the ESP32 was implemented using Serial1, which enabled the live temperature transmission. The ESP32 was then used to generate a colour gradient and warning system on the 64 x 64 RGB matrix. Hardware integration included soldering sensor leads, configuring a 5V / 4A power supply for the matrix, and resolving wiring and RAM limitations that were associated with using just the Arduino Mega alone. The system now reliably displays real time temperature and safety information.

Components Used

- Sunfounder Arduino Kit
    - Arduino Mega R3
    - Breadboard x1
    - Buzzer x1
    - Pendentimentor x1
    - Resistors x3
        i. Multiple resistors varying from 1k - 10k
- RGB 64x64 Matrix Kit Including
    - 16-bit color Ribbon Cable
    - Power cable
    - 64 x 64 Matrix Display
- Electrical tape
- Soldering Iron
- D2180B temperature Sensors x 5
    - Additional Soldering is required for the sensors
- Cables and adapters x 30
    - Jumper cables
    - Female to Female
    - Male to Male

- Male to Female

- Power transformer/ adapter
    - 5V and 4Amp power converter used to power the RGB matrix

Methodology

Arduino Mega:

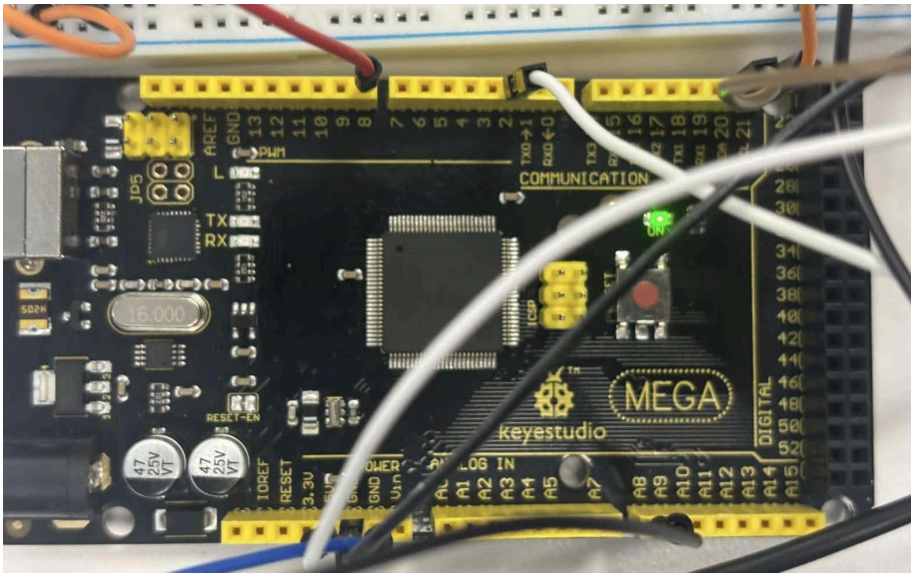


Figure #1: Image of the Arduino Mega R3 wired and functioning

The Arduino Mega acts as the data collection and safety alert system for the thermal detection system. It works with the Dallas D2180B digital temperature sensor, which processes the data and then outputs the real-time data into the LCD. With the Arduino Mega being coded, a condition was made to activate the buzzer if the temperature detected from the temperature sensor exceeds 30 degrees Celsius. This system uses a potentiometer to control the volume of the buzzer and is able to communicate with the ESP32, which is in turn able to display the

temperature to the RGB Matrix through the breadboard.

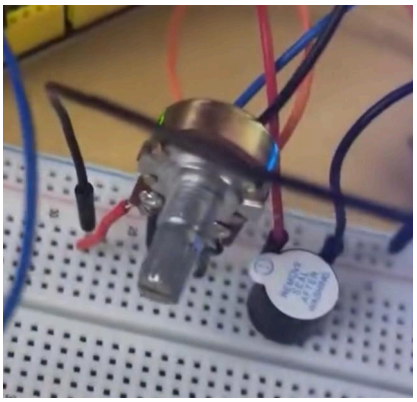


Figure #2: Close-up image of the buzzer and the potentiometer wired into the breadboard

1. **Temperature Acquisition:**

Figure #3 Displaying DS18B20 temperature sensor

a) Displaying the entire DS18B20 temperature sensor with its attachment pins

b) Displaying the DS18B20 temperature sensor integrated into our design and being held

The DS18B20 sensor communicates with the Arduino Mega with OneWire, which is found in the Dallas Temperature Library. With each loop of the cycle, the Arduino Mega

a. Check for a connection to the sensor
b. Requests for an updated temperature reading
c. The measured temperature is transferred into Mega as Celsius
d. Converts the temperature into Fahrenheit
e. Display both Celsius and Fahrenheit as the temperature output.

2. **LCD Display:**

Figure #4: This Figure shows the LCD displaying the temperature recorded from the DS18B20 temperature sensor in both Celsius and Fahrenheit.

A small 16x2 LCD screen displays the temperature in both Celsius and Fahrenheit. This can provide the temperature value to the user in a numerical form and can provide an

alternative display in case of failure of the matrix. In addition to displaying the temperature in both imperial and metric units, it makes the system more accessible.

3. **Communication with the ESP32**

The mega sends the temperature reading to the ESP32 through serial1 via the TX1 = pin 18. With each value being sent to the ESP32, which is then able to display the reading from the temperature sensor to the matrix in real time.

4. **Wiring Flowchart/Diagram of Breadboard**

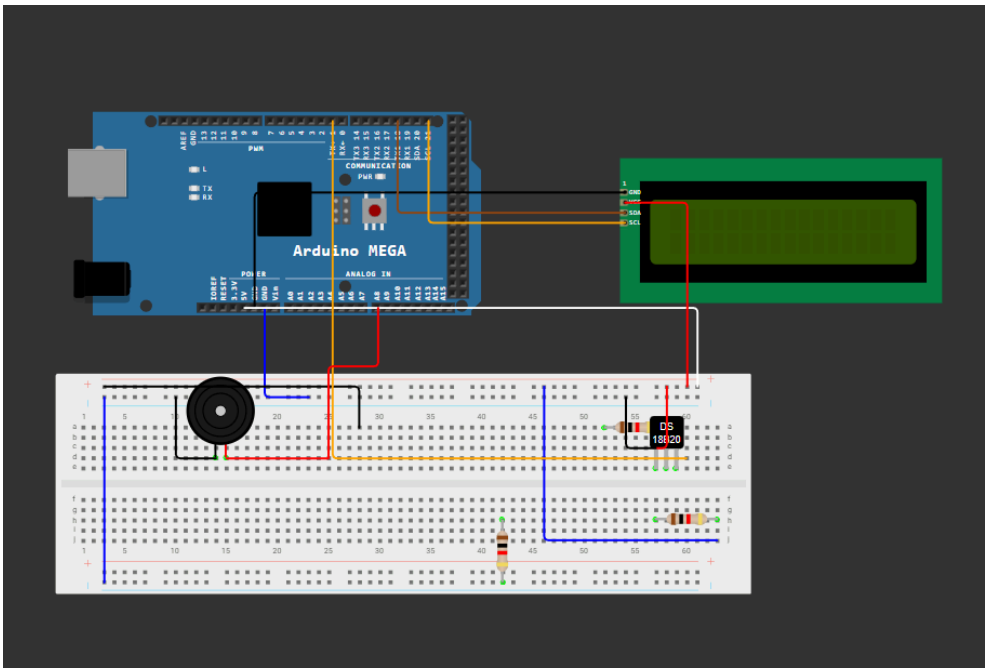


Figure #5: This figure shows the general wiring diagram of the breadboard with its connections to the Arduino Mega

The wiring diagram illustrates the general configuration for the Arduino Mega. This interface has a 16x2 I2C LCD, a DS18B20 temperature sensor probe, and a passive

buzzer. The Arduino Mega provides a 5V and GND connection to the breadboard's power rails, which also provide power to the I2C LCD, the DS18B20 probe, and any other components. The LCD is connected through the I2C bus with both the SDA and SCL pins wired directly to Arduino's communications pins (20 and 21, respectively). While the VCC and GND pins are connected to the 5V and GND rails. The DS18B20 probe is placed on the right side of the breadboard, and its VDD and GND pins are connected to the 5V and GND rails, and the data pin is connected to digital pin 2 on the Mega. Then the passive buzzer is wired with the positive terminal connected to Arduino digital pin 0 and the negative terminal to GND, which allows for the buzzer sound to go off.

ESP 32 and RGB Matrix:

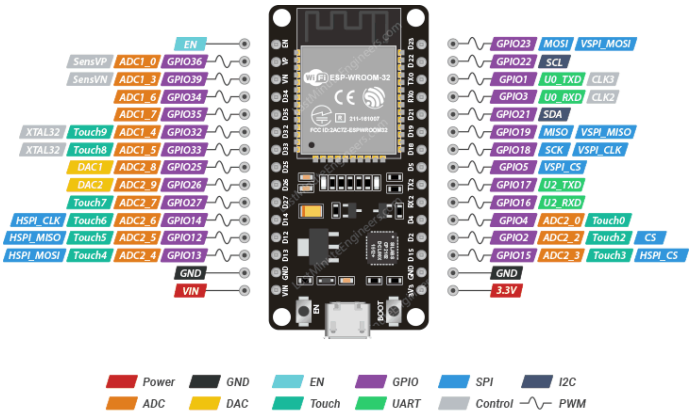


Figure #6: An image of the ESP32 and all of its pins

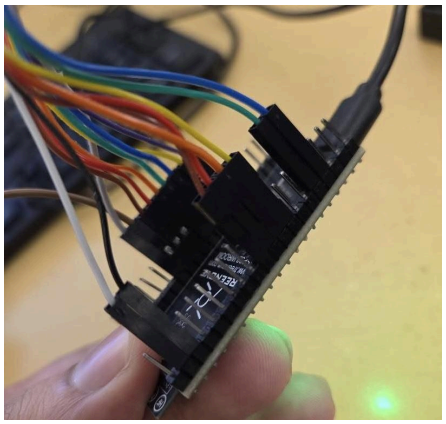


Figure #7: Wiring of the ESP32

The ESP 32 module is responsible for converting the pure numerical value from the Arduino Mega wand and then representing that data visually with colors on the 64 x 64 RGB matrix. This methodology focuses on three core tasks: aquarium data, visual display of data, and safety display.

Data Acquisition from the Arduino Mega

The ESP 32 communicates to the Arduino Mega with its Mega serial pin. The mega sends the temperature readings as plain text to the ESP32, while the ESP32:

1. Reads the incoming characters from the MegaSerial in real time
2. Stores the characters and data as a serial buffer
3. Only displaying the most recent temperature value named “lastTemp”, as to only display the most accurate data to the user

Color mapping

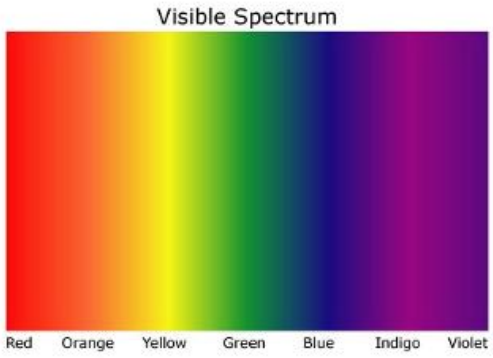


Figure #8 This figure shows the color spectrum and in which the color display was programmed, with violet and purple representing the coldest colors, changing colors in the spectrum up to red.

To create the visual spectrum display, the ESP32 maps each temperature value to a specific color with

- Temperatures below 15C shown as dark purple
- And the Temperature is changed with a color gradient, changing a shade every 3 degrees to show the transition of temperature

- Temperatures above 30 °C are stuck to a deep red color, along with the buzzer activation and the too hot display.

Displaying the structure:

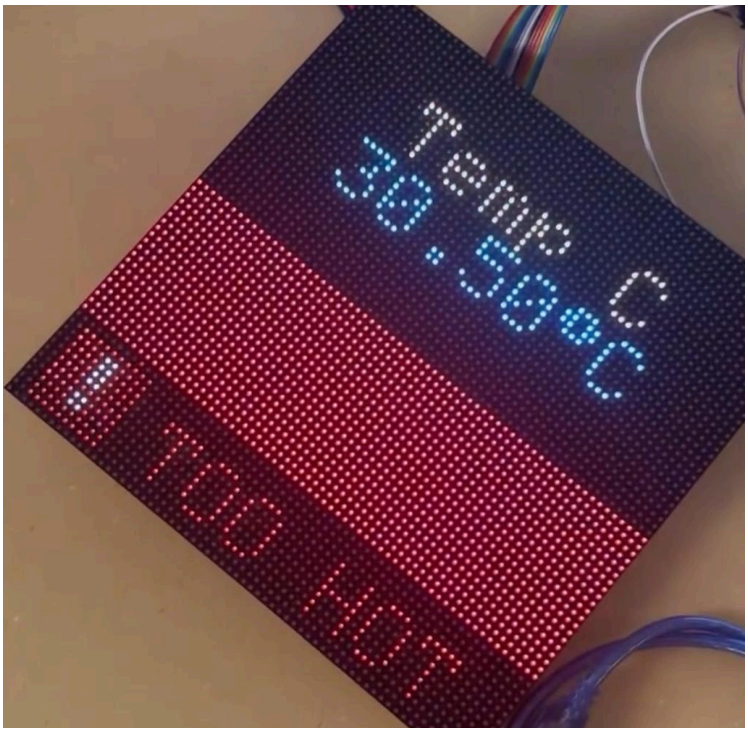


Figure #9: This image shows all of the elements in the Matrix, once the temperature exceeds the threshold, leading to the warning display "TOO HOT".

The screen is divided into three main sections

1. Upper third (Temperature Display)

The upper half of the display is reserved only for the temperature reading and text

.The label "Temp C" in bright yellow text

.The numeric value in cyan (eg, 28.75 °C)

2. Displaying the color/Gradient

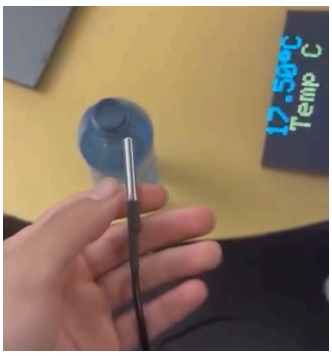
The middle section of the matrix displays the only animated vertical wave pattern. With each column, a sine wave is refreshed based on the frame counter. The wave value controls how the base colors blend the main color at the temperature with a slightly warmer color to represent a gradient effect. This allows multiple colors, which show moving colors, that show the current temperature, and create an animation that is easier to interpret than slight changes in a shade of color.

3. Bottom Third of Display (Heat warning)

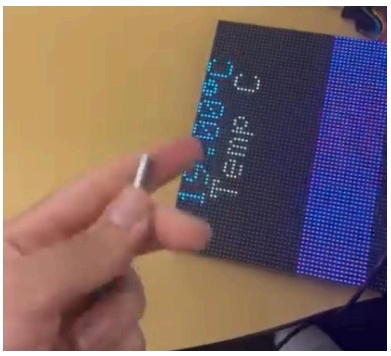
The bottom half of the panels is reserved for a safety warning, and this lower third is only used for this purpose when the temperature is above 30 degrees Celsius.

- When the temperature exceeds 30 degrees Celsius, the section flashes
    - A red block of text with a while exclamation mark appears on the left, similar to that of a standard caution sign seen on other products
    - A red text block can be seen displaying “TOO HOT”; this is to alert the user visually that a dangerous temperature has been reached, with the frame counter being toggled on and off to flash and make this message even more visible to the user

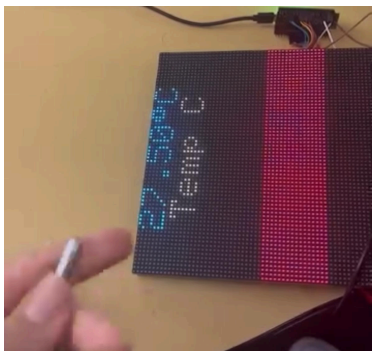
Stop Motion of System:



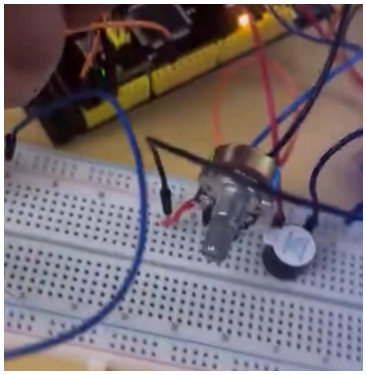
System turned on



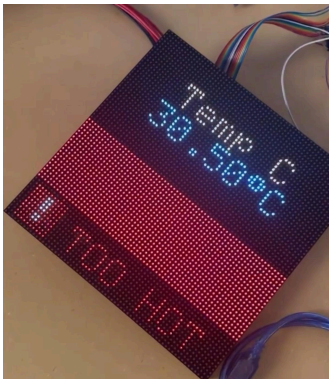
Sensors being warmed up



Temperature gradually increases



Buzzer alarm goes off after exceeding 30°C



Caution message appears after exceeding 30°C

Pseudocodes:

For Arduino Mega:

```
// --- CONSTANTS & VARIABLES ---
DEFINE Pin 2 as OneWireBus (Sensor)
DEFINE Pin 8 as Buzzer (Output/PWM)
DEFINE Pin A8 as Potentiometer (Input)
DEFINE Threshold_Temp as 30.0 Celsius

OBJECTS: LCD_Display, Temperature_Sensor

// --- SETUP  ---
BEGIN SETUP
    INITIALIZE Serial_Port_0 (Connection to PC)
    INITIALIZE Serial_Port_1

    INITIALIZE LCD_Display
    TURN ON LCD Backlight
    DISPLAY "Init Sensor..." on LCD

    START Temperature_Sensor

    IF (Sensor Address is NOT found) THEN
        DISPLAY "Error: No Sensor" on LCD
    ELSE
        DISPLAY "Sensor Found!" on LCD
    END IF

    SET Sensor Resolution to 9-bit

    CONFIGURE Buzzer_Pin as OUTPUT
    CONFIGURE Potentiometer_Pin as INPUT
    ENSURE Buzzer is OFF

    WAIT 2 seconds
    CLEAR LCD

END SETUP

// --- MAIN LOOP ---
BEGIN LOOP
    // 1. Get Temperature Data
    REQUEST temperature from Sensor
```

```
          READ temp_C from Sensor

          // 2. Error Handling
          IF (temp_C is "Disconnected_Value") THEN
              DISPLAY "Sensor Error" on LCD
              TURN OFF Buzzer
              WAIT 1 second
              RETURN to start of LOOP
          END IF

          // 3. Input Processing (Volume Control)
          READ analog value from Potentiometer (0 to 1023)
          CALCULATE buzzer_volume = MAP Potentiometer value to PWM
range (0 to 255)

          // 4. Alarm Logic
          IF (temp_C > 30.0) THEN
              // Turn Buzzer ON with volume determined by Potentiometer
              WRITE PWM signal (buzzer_volume) to Buzzer_Pin
          ELSE
              // Temperature is safe
              TURN OFF Buzzer
          END IF

          // 5. Data Transmission
          PRINT temp_C to Serial_Port_0 (for PC Debugging)
          PRINT temp_C to Serial_Port_1 (Sending data to ESP32)

          // 6. Update Display
          SET Cursor to Row 0
          PRINT "Temp: " + temp_C + "C"

          CALCULATE temp_F = Convert temp_C to Fahrenheit
          SET Cursor to Row 1
          PRINT "Temp: " + temp_F + "F"

          // 7. Delay
          WAIT 0.5 seconds
      END LOOP
```

For the ESP 32 and Matrix:

```
Begin
```

```
//Initialize
-Initialize the connection to the Arduino Mega3
-Configure the RGB Matrix display
-Set the brightness level


Main Loop

WHILE program is running:

IF there is data from Arduino on Serial1:
      Read characters individually
If Newline is received
      Convert value to temperature value in celsius
Clear Buffer

IF tempC changed from last reading
      Update top text area:
      -Clear top half of the screen and print “Temp C”
      -Print updated number value below Temp C

//To Decide color for wave
IF tempC < 15*C:
       baseColor = PURPULE
Else If tempC between 15C and 30C
      Map tempC linearly between the BLUE and REd
      Basecolor = blended color

ELSE ( tempC >= 30C)
      baseColor = RED

//Draw Animated wave in middle section:
FOR each column xx from 0 to panel width:
      Compute the value using the sin of the value with the frame
counter
      Use wave value to blend, the original color to be cooler and
warmer around the baseColor

//Warning Sight at Bottom
IF tempC  < 30C:
      Clear bottom bar (no warning)

Else  tempC >= 30C
      Use frameCounter to make the bar flash on and off
```

```
IF flash is ON this frame
      -Draw red block with a WHITE exclamation mark to the left
      -Print “TOO HOT” text in bright red on the right
Else
      Clear bottom bar
END WHILE

END Program
```

Challenges

1. **Complicated Integration of Hardware**

   Original Issue: Mixing 5V (Mega) and 3.3V (ESP32) logic is dangerous and complex.

   Revised:

   a. **Microcontroller Integration and Logic Level Shifting**: Integrating the Arduino Mega with the ESP32 presented a significant hardware challenge due to voltage mismatches. The Arduino Mega operates on 5V logic, whereas the ESP32 operates on 3.3V. Direct connection poses a risk of damaging the ESP32. To resolve this, we implemented a voltage divider circuit (using resistors) to step down the signals safely. Furthermore, establishing a "common ground" between the two separate power rails was critical to ensure a stable reference voltage for communication.

2. **Powering the High-Current Matrix**

   Original Issue: The matrix needs 4A, connectors didn't match, and wires had to be stripped. **Revised:**

   a. **Power Supply Implementation for LED Matrix:** The 64x64 LED Matrix has a high current draw, requiring a dedicated 5V, 4A power supply; the standard microcontroller USB power is insufficient for this load. A key physical challenge was the incompatibility between the matrix's pre-installed wiring and the external power adapter. We had to modify the hardware by cutting and stripping the matrix power cables to interface them securely with the DC power adapter using screw terminals.

3. **<u>Adaptation from Arduino Mega to ESP32</u>**

   Original Issue: The Mega didn't have enough RAM, so you switched to ESP32, which improved performance but complicated wiring. **Revised:**

   a. **Transition to ESP32 for Display Management:** Initially, the project attempted to drive the RGB matrix using the Arduino Mega. However, testing revealed that the Mega's onboard RAM was insufficient to buffer the frame data for a 64x64 resolution display. Consequently, the display driver role was offloaded to an ESP32. While this shift significantly improved the refresh rate and resolution, it required a complete redesign of the wiring schematic, as the ESP32 pinout and library requirements differ fundamentally from the Arduino architecture. This distributed system approach allows the Mega to handle sensor data while the ESP32 handles high-speed graphics.

4. **<u>Software challenges:</u>**

   Original Issue: Coordinates were wrong (images misaligned) and Serial communication failed at first. **Revised:**

   a. **Matrix Mapping and Coordinate Systems**:A major software hurdle was the pixel mapping on the LED matrix. Due to the complex internal wiring of 64x64 panels (often utilizing "snake" or "zigzag" patterns), initial code tests resulted in misaligned text and scattered images. We had to iteratively adjust the coordinate mapping in the software to ensure the temperature data and graphical elements aligned correctly on the physical display.

   b. **Inter-Device Serial Communication:** Establishing reliable data transfer between the Arduino Mega and the ESP32 required multiple iterations. Early tests showed

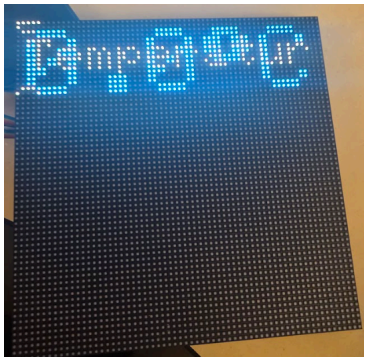
data corruption and packet loss due to the logic level differences and power instability. By refining the UART serial communication protocol and stabilizing the power distribution, we successfully achieved a consistent data stream where the Mega transmits sensor readings and the ESP32 parses them for display in real-time.

Conclusions

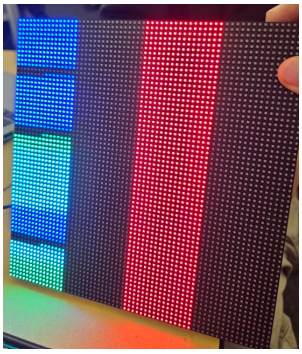
In conclusion, the Thermal Detection System successfully takes real-time temperature and integrates it with an intuitive visual and auditory safety interface, which meets the project's objectives and demonstrates successful and effective communication between the Arduino Mega and ESP32 microcontrollers. The system displays temperature values reliably, generates a color-mapped gradient on the RGB matrix, and activates both a buzzer that serves as an alarm and a caution warning "Too Hot!" when thresholds are exceeded, offering a clear improvement over numerical only displays. Throughout the project, the entire group was able to gain practical experience with embedded programming, sensor communication, hardware integration, and iterative debugging, especially when resolving the power requirements and adapting the RGB matrix to the ESP32. Some improvements that can be made include supporting multiple sensors, enabling adjustable warning thresholds, refining the colour mapping logic, and streamlining the power and wiring through custom PCBs. Future work may involve adding wireless connectivity, remote monitoring, or advanced visualization features that expand the system's functionality and real world applications. Overall, the system delivers a reliable and accessible thermal monitoring solution that demonstrates a strong engineering design and provides a solid foundation for future designs.

Appendices

Table A: Documentation and pictures of the process of building this Project



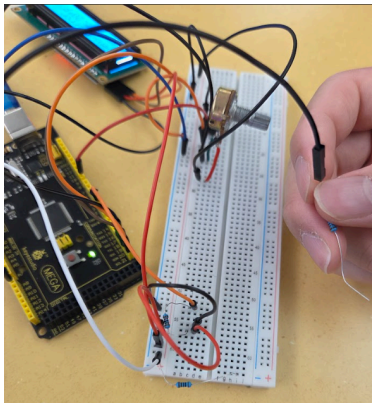
(9)



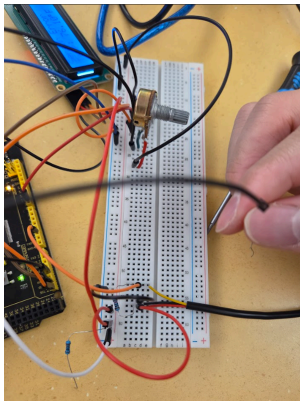
(10)

(11)

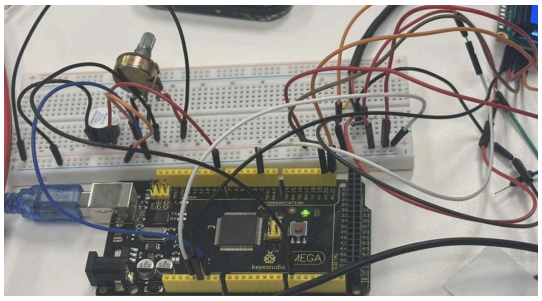
Figures 9 - 11 show technical errors that occurred when trying to display the desired visuals on the matrix



(12)

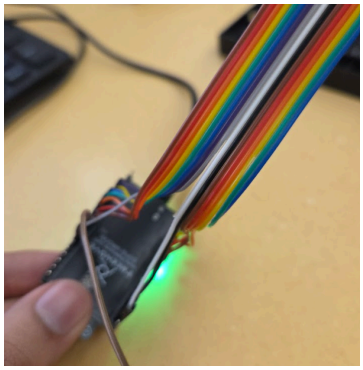


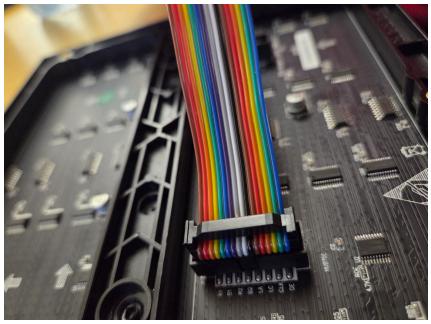
(13)

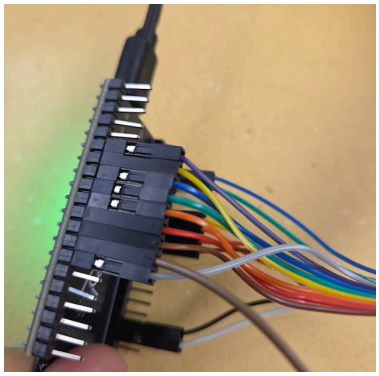


(14)

Figures 12 -14 show complications that arose with wire connections on the breadboard and microcontrollers







(15) *(16)* *(17)*

Figures 15 - 17 show challenges with the wiring and connection of the ESP32 and the matrix

Link to test video:
https://drive.google.com/file/d/1pFnSGyxI8_zatG1DRpwE_3j52XeJZJkf/view?usp=drive_link

Table B: Codes for Both Microcontrollers

Arduino Mega Code

```
// Include the libraries we need to run this code
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

#define ONE_WIRE_BUS 2
#define BUZZER_PIN 8        // PWM pin on Arduino Mega
#define POT_PIN A8          // Potentiometer for volume control For the buzzer

LiquidCrystal_I2C lcd(0x27, 16, 2);
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
DeviceAddress insideThermometer;

void setup(void)
{
```

```
  //The serial to the buzzer
  Serial.begin(9600);

  // Defening Serial1 to ESP32 (TX1 = pin 18)
  Serial1.begin(9600);

  lcd.init();
  lcd.backlight();
  lcd.print("Init Sensor....");

  sensors.begin();

    //Display message for sensor
  if (!sensors.getAddress(insideThermometer, 0)) {
    lcd.setCursor(0,1);
    lcd.print("Error: No Sensor");
  } else {
    lcd.setCursor(0,1);
    lcd.print("Sensor Found!");
  }

  sensors.setResolution(insideThermometer, 9);

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

  delay(2000);
  lcd.clear();
}

void loop(void)
{
  sensors.requestTemperatures();
  float tempC = sensors.getTempC(insideThermometer);

  if (tempC == DEVICE_DISCONNECTED_C)
  {
    lcd.setCursor(0,0);
    lcd.print("Sensor Error ");
```

```
  analogWrite(BUZZER_PIN, 0);  //Turning OFF the buzzer
  delay(1000);
  return;
}

//Reading the Poetinmeter values
int potValue = analogRead(POT_PIN);

//Mapping TO PWM VALUE (0–255):
int buzzerPower = map(potValue, 0, 1023, 0, 255);

//Controlling the buzzer with the Potentiometer
if (tempC > 30.0) {
    //Volume control with POTENTIOMETER
  analogWrite(BUZZER_PIN, buzzerPower);
} else {
  analogWrite(BUZZER_PIN, 0); //Turning off the buzzer if it's below the threshold
}

// SERIAL OUTPUT
Serial.println(tempC);

//What the ESP32 reads
Serial1.println(tempC);

//The display for the LCD skin
lcd.setCursor(0, 0);
lcd.print("Temp: ");
lcd.print(tempC);
lcd.print((char)223);
lcd.print("C ");

//Reading the display into Fahrenheit for LCD
lcd.setCursor(0, 1);
lcd.print("Temp: ");
lcd.print(DallasTemperature::toFahrenheit(tempC));
lcd.print((char)223);
lcd.print("F ");

delay(500);
```

}

ESP32 Code

```
//Libraries needed for the code
#include < ESP32 -HUESP32B75-MatrixPanel-I2S-DMA.h>
#include <Adafruit_GFX.h>
#include <math.h>

// The Configuration for the Panel
const uint16_t PANEL_WIDTH  = 64;
const uint16_t PANEL_HEIGHT = 64;
const uint8_t  PANELS_IN_CHAIN = 1;


//Space of 12 rows reserved for Caution Warning at the bottom of the matrix
const int CAUTION_HEIGHT = 12;


MatrixPanel_I2S_DMA *dma_display = nullptr;


//Serial from Mega
HardwareSerial MegaSerial(1);


String serialBuffer;
float lastTemp = 0.0;
float prevDisplayedTemp = -999;


uint32_t frameCounter = 0;


//Color Constants
uint16_t RED, BLUE, PURPLE, WHITE;


//Blending both colors
uint16_t blendColor(uint16_t c1, uint16_t c2, float ratio) {
```

```
    uint8_t r1 = (c1 >> 11) & 0x1F;
    uint8_t g1 = (c1 >> 5)  & 0x3F;
    uint8_t b1 =  c1        & 0x1F;


    uint8_t r2 = (c2 >> 11) & 0x1F;
    uint8_t g2 = (c2 >> 5)  & 0x3F;
    uint8_t b2 =  c2        & 0x1F;


    uint8_t r = r1 + (r2 - r1) * ratio;
    uint8_t g = g1 + (g2 - g1) * ratio;
    uint8_t b = b1 + (b2 - b1) * ratio;


    return dma_display->color565(r << 3, g << 2, b << 3);
}


//Temperature and the color to color
uint16_t getTemperatureColor(float t) {
    if (t < 15.0) return PURPLE;
    if (t > 30.0) t = 30.0;


    float x = (t - 15.0) / 15.0;


    return blendColor(BLUE, RED, x);
}


//Centering the temperature readings
int centerX(const char *text, int size = 1) {
    int16_t x1, y1;
    uint16_t w, h;


    dma_display->setTextSize(size);
    dma_display->getTextBounds(text, 0, 0, &x1, &y1, &w, &h);
```

```
    return (PANEL_WIDTH - w) / 2;
}

//Ensuring Text At the Top remains static with no animations or flickering
void drawTop(float t) {
    dma_display->fillRect(0, 0, PANEL_WIDTH, PANEL_HEIGHT / 2, 0);

    int topHalfHeight = PANEL_HEIGHT / 2;
    int lineHeight = 8;
    int blockHeight = 2 * lineHeight;
    int startY = (topHalfHeight - blockHeight) / 2;

    dma_display->setTextSize(1);
    dma_display->setTextColor(dma_display->color565(255, 255, 0));
    dma_display->setCursor(centerX("Temp C"), startY);
    dma_display->print("Temp C");

    char buffer[12];
    sprintf(buffer, "%.2f%cC", t, 247);
    dma_display->setTextColor(dma_display->color565(0, 255, 255));
    dma_display->setCursor(centerX(buffer), startY + lineHeight);
    dma_display->print(buffer);
}

//Wave animation/blend
void drawWave(uint16_t baseColor) {
    uint16_t warmer = blendColor(baseColor, RED, 0.25);
    uint16_t cooler = blendColor(baseColor, BLUE, 0.25);

    int waveStartY = PANEL_HEIGHT / 2;
    int waveHeight = PANEL_HEIGHT / 2 - CAUTION_HEIGHT;  //Room left at the
bottom
```

```
    for (int x = 0; x < PANEL_WIDTH; x++) {
      float wave = sin((x + frameCounter * 0.3) * 0.15);
      float norm = (wave + 1.0) / 2.0;


      uint16_t c1 = blendColor(cooler, baseColor, norm);
      uint16_t finalColor = blendColor(c1, warmer, norm);


      dma_display->drawFastVLine(
        x,
        waveStartY,
        waveHeight,
        finalColor
      );
    }
}


//Caution Bar at the Bottom
void drawCaution(float t) {
    int y0 = PANEL_HEIGHT - CAUTION_HEIGHT;


    //Leave the bottom of the bar empty if tempC is below the threshold (30 °C)
    if (t < 30.0) {
      dma_display->fillRect(0, y0, PANEL_WIDTH, CAUTION_HEIGHT, 0);
      return;
    }


    //Flashing frames
    bool on = ((frameCounter / 8) % 2) == 0;


    if (!on) {

      //Clearing the bar when it's off
```

```
            dma_display->fillRect(0, y0, PANEL_WIDTH, CAUTION_HEIGHT, 0);
            return;
        }


        //Keeping the background of the matrix black
        dma_display->fillRect(0, y0, PANEL_WIDTH, CAUTION_HEIGHT, 0);


        //Red caution block warning on the left
        uint16_t darkRed = dma_display->color565(180, 0, 0);
        int signX = 2;
        int signW = 10;
        dma_display->fillRect(signX, y0 + 1, signW, CAUTION_HEIGHT - 2, darkRed);


        //Adding White Exclamation mark in the middle of the red warning block
        int exX = signX + signW / 2 - 1;
        int exTop = y0 + 2;
        int exH = CAUTION_HEIGHT - 6;
        dma_display->fillRect(exX, exTop, 2, exH - 2, WHITE);
        dma_display->fillRect(exX, exTop + exH - 1, 2, 2, WHITE);


        //Displaying "TOO HOT" text in red
        dma_display->setTextSize(1);
        dma_display->setTextColor(RED);
        dma_display->setCursor(16, y0 + 3);
        dma_display->print("TOO HOT");
    }



    void setup() {
        Serial.begin(115200);
        MegaSerial.begin(9600, SERIAL_8N1, 33, -1);


        HUB75_I2S_CFG mxconfig(PANEL_WIDTH, PANEL_HEIGHT,
PANELS_IN_CHAIN);
```

```
  //PIn MAP configuration to the ESP32 pins
  mxconfig.gpio.r1 = 25;
  mxconfig.gpio.g1 = 26;
  mxconfig.gpio.b1 = 27;


  mxconfig.gpio.r2 = 14;
  mxconfig.gpio.g2 = 12;
  mxconfig.gpio.b2 = 13;


  mxconfig.gpio.a  = 23;
  mxconfig.gpio.b  = 22;
  mxconfig.gpio.c  = 5;
  mxconfig.gpio.d  = 17;
  mxconfig.gpio.e  = 32;


  mxconfig.gpio.clk = 16;
  mxconfig.gpio.lat = 4;
  mxconfig.gpio.oe  = 15;



  dma_display = new MatrixPanel_I2S_DMA(mxconfig);
  dma_display->begin();
  dma_display->setBrightness8(40);


  RED     = dma_display->color565(255, 0, 0);
  BLUE    = dma_display->color565(0, 0, 180);
  PURPLE  = dma_display->color565(150, 0, 150);
  WHITE   = dma_display->color565(255, 255, 255);


  dma_display->fillScreen(0);
  drawTop(0);
}
```

```
//While Loop
void loop() {


   while (MegaSerial.available()) {
      char c = MegaSerial.read();


      if (c == '\n' || c == '\r') {
         if (serialBuffer.length() > 0) {
            lastTemp = serialBuffer.toFloat();
            serialBuffer = "";
         }
      } else {
         serialBuffer += c;
      }
   }


   if (fabs(lastTemp - prevDisplayedTemp) >= 0.01) {
      drawTop(lastTemp);
      prevDisplayedTemp = lastTemp;
   }


   uint16_t baseColor = getTemperatureColor(lastTemp);
   drawWave(baseColor);


   //Printing the TOO HOT bar when the temp exceeds the limit
   drawCaution(lastTemp);

   frameCounter++;
}
```

Table C: Table Of Figures From Report

Figure #	**Title**
1	Arduino Mega R3 wiring
2	Wiring of the buzzer and the potentiometer into the breadboard
3 a)	Stock image of the DS18B20 sensor, showing the attachment pins
3 b)	Integrated DS18B20 sensor being held in hand while connected to the system
4	The LCD shows the accurate temperature from the sensor
5	Online diagram of all of the connections of the ESP32
6	Wired ESP 32, with all of the correct pins according to the diagram
7	Visible Color spectrum, which influenced the color gradient for the system
8	This figure shows all of the elements past the threshold of 30 degrees Celsius, including the warning text
9-10-11	These figures show technical errors that occurred when trying to display the desired visuals on the matrix
12-13-14	These figures show complications that occurred with breadboard and microcontrollers
15-16-17	These figures show challenges with the wiring and connection of the ESP32 and the matrix

References

[1] I. Yakimush, “IharYakimush/Arduino-temperature-control-events: Arduino thermometer with onChange event based on DS18B20 sensor,” GitHub, https://github.com/IharYakimush/arduino-temperature-control-events (accessed Nov. 24, 2025).

[2] 7semi-solutions, “7SEMI-solutions/7semi-DS18B20-arduino-library: This Arduino Library provides support for the 7semi DS18B20 temperature sensor module, a waterproof 1-wire digital sensor capable of measuring temperatures from -55°C to +125°C with 9–12-bit resolution. It's ideal for weather monitoring, thermal logging, and embedded sensing applications.,” GitHub, https://github.com/7semi-solutions/7Semi-DS18B20-Arduino-Library (accessed Nov. 24, 2025).

[3] T. Adafruit, “Adafruit/Adafruit-GFX-Library: Adafruit GFX Graphics Core Arduino Library, this is the ‘core’ class that all our other graphics libraries derive from,” GitHub, https://github.com/adafruit/Adafruit-GFX-Library (accessed Nov. 24, 2025).

[4] “LiquidCrystal,” Docs.arduino.cc, https://docs.arduino.cc/libraries/liquidcrystal/#Usage/Examples (accessed Nov. 24, 2025).

[5] L. E. Staff, “Interfacing DS18B20 1-wire digital temperature sensor with Arduino,” Last Minute Engineers, https://lastminuteengineers.com/ds18b20-arduino-tutorial/ (accessed Nov. 24, 2025).

[6] A. Z. Jones, “What is the visible light spectrum?,” ThoughtCo, https://www.thoughtco.com/the-visible-light-spectrum-2699036 (accessed Nov. 24, 2025).

[7] M. Short and J. E. B, “How to use a Breadboard,” How to Use a Breadboard - SparkFun Learn, https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/all (accessed Nov. 24, 2025).

[8] J. Hylén, “Getting Started with Arduino Mega 2560 Rev3,” Docs.arduino.cc, https://docs.arduino.cc/tutorials/mega-2560/getting-started/ (accessed Nov. 24, 2025).

[9] Jithendra et al., “Getting started with the ESP32 Development Board,” Random Nerd Tutorials, https://randomnerdtutorials.com/getting-started-with-esp32/ (accessed Nov. 24, 2025).

[10] “RGB-matrix-P3-64X64,” RGB-Matrix-P3-64x64 - Waveshare Wiki, https://www.waveshare.com/wiki/RGB-Matrix-P3-64x64 (accessed Nov. 24, 2025).