Lab Report: Final Project – External PC Hardware Monitor

Lucas Porter

Spring 2025 TESY 3300

Trevor P. Robinson, PhD

Final Project – External PC Hardware Monitor

04/21/25

Abstract

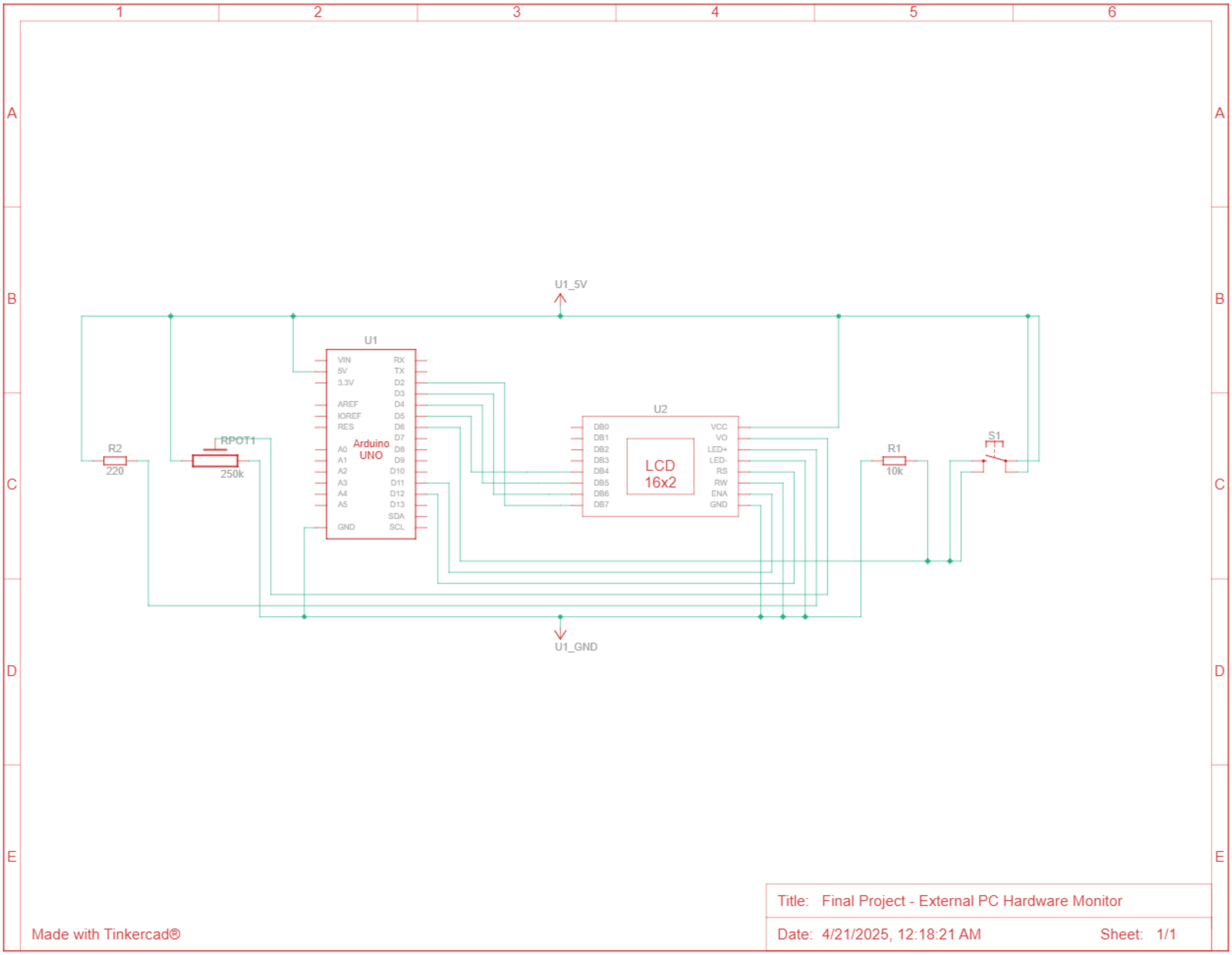
The goal of this lab was to create a small, standalone hardware monitor that displays real-time temperature readings on an LCD screen using an Arduino. The project was designed to teach how to combine sensor data, microcontroller programming, and user input into a complete system. Two approaches for gathering temperature data were considered: either wiring an external temperature sensor directly to the Arduino or reading onboard sensor data from the computer and sending it over USB serial communication. Due to hardware failure, the final implementation utilized onboard system sensors, gathered through a Python script, and transmitted to the Arduino. A button on the device allows users to cycle between different temperature metrics displayed on the LCD. Additionally, a maximum recorded temperature for each metric is shown, providing a quick view of peak system temperatures. An auto-scroll function was also added, automatically cycling through the different metrics every few seconds if the button is not pressed.

Materials

- Arduino board
- Breadboard (medium)
- 16x2 LCD display
- Potentiometer (for LCD contrast)
- Momentary pushbutton
- 10kΩ resistor (pull-down resistor for button)
- Jumper wires (assorted)
- USB cable for Arduino

Procedure

Circuit Diagram

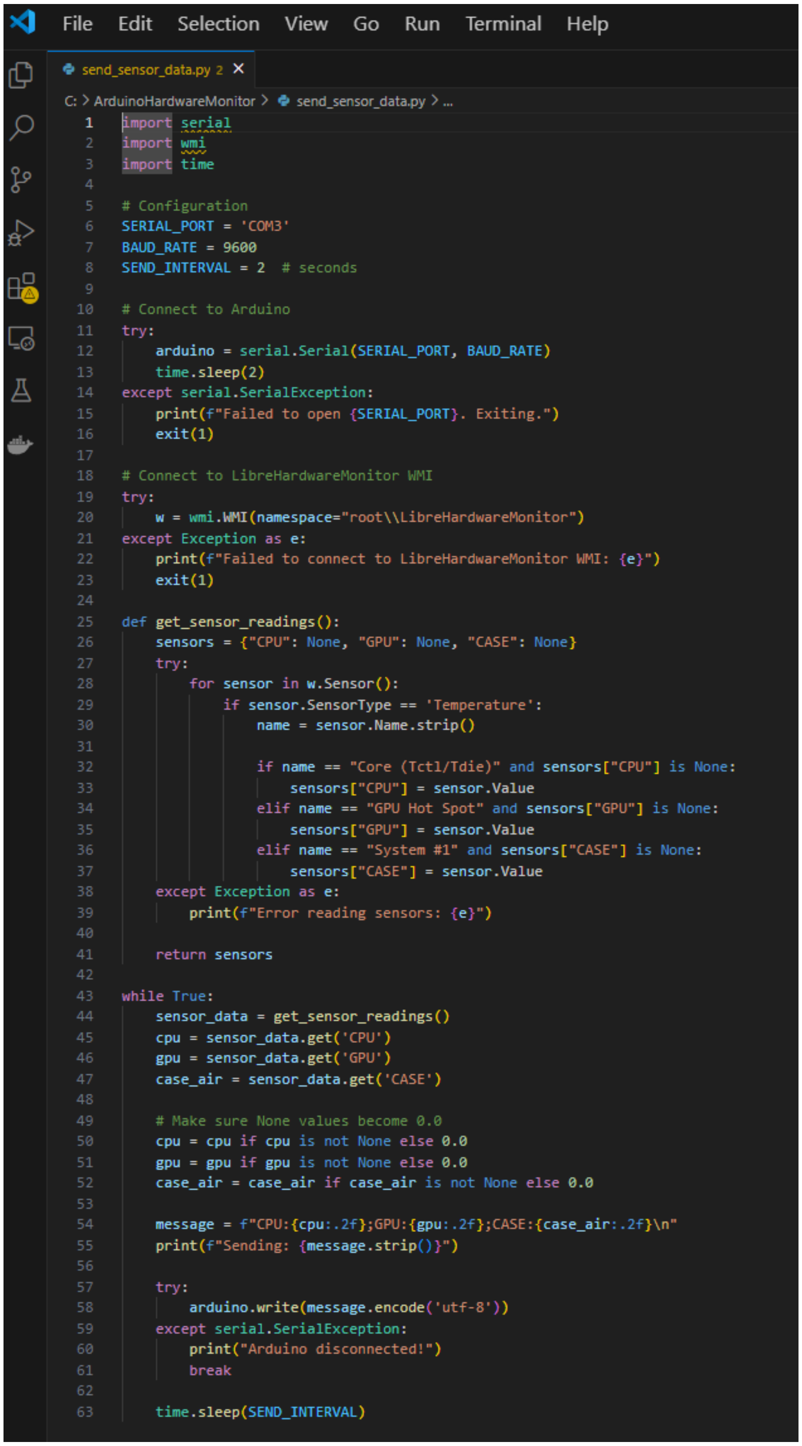


Steps

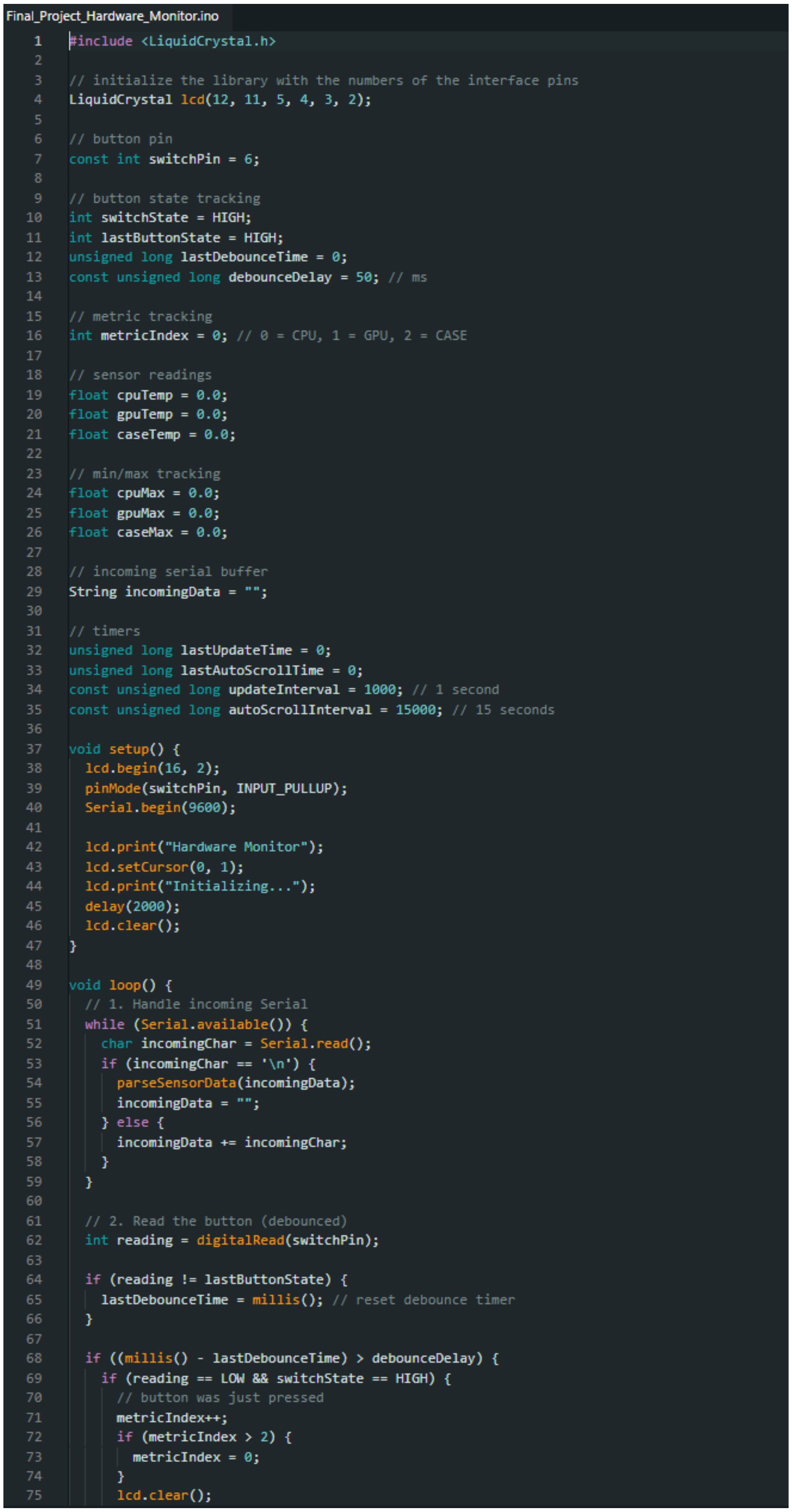
1. Assemble the circuit as shown in Project 11 (Crystal Ball) in the Arduino Beginner Projects book.
2. Replace the tilt sensor used in Project 11 with a momentary pushbutton.
3. Wire the button to the Arduino using a pull-down resistor:
    - Connect one side of the button to a digital input pin on the Arduino.
    - Connect the same side through a 10kΩ resistor to ground.
    - Connect the other side of the button to 5V.
4. Wire the LCD display using the same wiring as Project 11:
    - RS pin to Arduino pin 12
    - Enable pin to Arduino pin 11
    - D4 to Arduino pin 5
    - D5 to Arduino pin 4
    - D6 to Arduino pin 3
    - D7 to Arduino pin 2
    - Connect a potentiometer to adjust LCD contrast.
5. Program the Arduino to:
    - Initialize the LCD.
    - Read incoming serial data.
    - Parse sensor readings from serial.
    - Update the LCD to show CPU temperature, GPU temperature, or Case Air temperature.
    - Change the displayed metric each time the button is pressed.
    - Display the maximum recorded temperature for each metric.
    - Automatically cycle through the different metrics after a set period of time if the button is not pressed (auto-scroll).

6. On the computer, create a Python script to:
    - Use the wmi module to access onboard sensor data from LibreHardwareMonitor.
    - Format the temperature data.
    - Send it over the USB serial port to the Arduino at 9600 baud.
7. Set up a Windows Scheduled Task to run the Python script silently at system startup.
8. Test the hardware monitor by restarting the computer and verifying that the Arduino receives and displays the updated sensor data automatically.

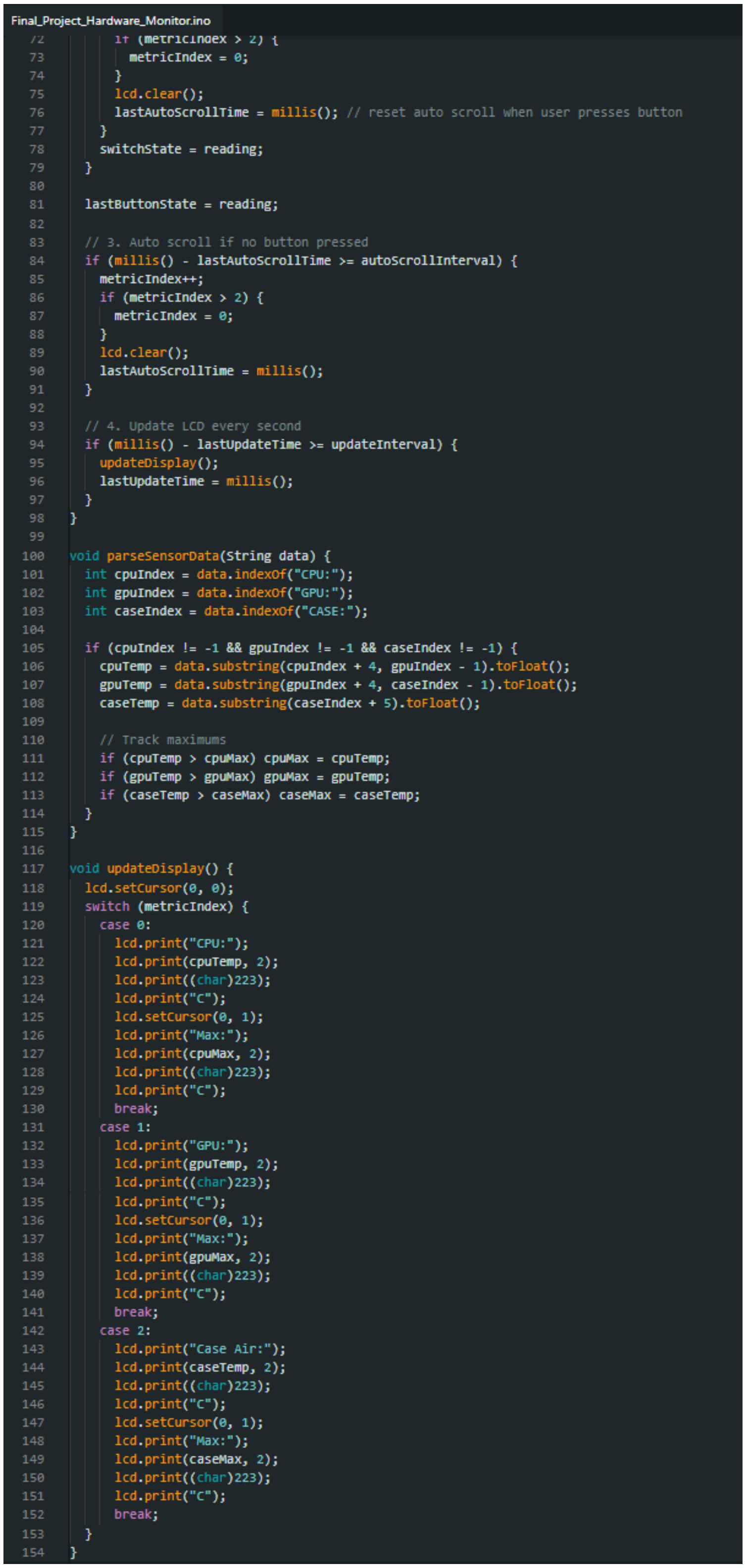
Code



```python
import serial
import wmi
import time

# Configuration
SERIAL_PORT = 'COM3'
BAUD_RATE = 9600
SEND_INTERVAL = 2  # seconds

# Connect to Arduino
try:
    arduino = serial.Serial(SERIAL_PORT, BAUD_RATE)
    time.sleep(2)
except serial.SerialException:
    print(f"Failed to open {SERIAL_PORT}. Exiting.")
    exit(1)

# Connect to LibreHardwareMonitor WMI
try:
    w = wmi.WMI(namespace="root\\LibreHardwareMonitor")
except Exception as e:
    print(f"Failed to connect to LibreHardwareMonitor WMI: {e}")
    exit(1)

def get_sensor_readings():
    sensors = {"CPU": None, "GPU": None, "CASE": None}
    try:
        for sensor in w.Sensor():
            if sensor.SensorType == 'Temperature':
                name = sensor.Name.strip()

                if name == "Core (Tctl/Tdie)" and sensors["CPU"] is None:
                    sensors["CPU"] = sensor.Value
                elif name == "GPU Hot Spot" and sensors["GPU"] is None:
                    sensors["GPU"] = sensor.Value
                elif name == "System #1" and sensors["CASE"] is None:
                    sensors["CASE"] = sensor.Value
    except Exception as e:
        print(f"Error reading sensors: {e}")

    return sensors

while True:
    sensor_data = get_sensor_readings()
    cpu = sensor_data.get('CPU')
    gpu = sensor_data.get('GPU')
    case_air = sensor_data.get('CASE')

    # Make sure None values become 0.0
    cpu = cpu if cpu is not None else 0.0
    gpu = gpu if gpu is not None else 0.0
    case_air = case_air if case_air is not None else 0.0

    message = f"CPU:{cpu:.2f};GPU:{gpu:.2f};CASE:{case_air:.2f}\n"
    print(f"Sending: {message.strip()}")

    try:
        arduino.write(message.encode('utf-8'))
    except serial.SerialException:
        print("Arduino disconnected!")
        break

    time.sleep(SEND_INTERVAL)
```



```
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// button pin
const int switchPin = 6;

// button state tracking
int switchState = HIGH;
int lastButtonState = HIGH;
unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 50; // ms

// metric tracking
int metricIndex = 0; // 0 = CPU, 1 = GPU, 2 = CASE

// sensor readings
float cpuTemp = 0.0;
float gpuTemp = 0.0;
float caseTemp = 0.0;

// min/max tracking
float cpuMax = 0.0;
float gpuMax = 0.0;
float caseMax = 0.0;

// incoming serial buffer
String incomingData = "";

// timers
unsigned long lastUpdateTime = 0;
unsigned long lastAutoScrollTime = 0;
const unsigned long updateInterval = 1000; // 1 second
const unsigned long autoScrollInterval = 15000; // 15 seconds

void setup() {
  lcd.begin(16, 2);
  pinMode(switchPin, INPUT_PULLUP);
  Serial.begin(9600);

  lcd.print("Hardware Monitor");
  lcd.setCursor(0, 1);
  lcd.print("Initializing...");
  delay(2000);
  lcd.clear();
}

void loop() {
  // 1. Handle incoming Serial
  while (Serial.available()) {
    char incomingChar = Serial.read();
    if (incomingChar == '\n') {
      parseSensorData(incomingData);
      incomingData = "";
    } else {
      incomingData += incomingChar;
    }
  }

  // 2. Read the button (debounced)
  int reading = digitalRead(switchPin);

  if (reading != lastButtonState) {
    lastDebounceTime = millis(); // reset debounce timer
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading == LOW && switchState == HIGH) {
      // button was just pressed
      metricIndex++;
      if (metricIndex > 2) {
        metricIndex = 0;
      }
      lcd.clear();
```



```
      if (metricIndex > 2) {
        metricIndex = 0;
      }
      lcd.clear();
```

```
      lastAutoScrollTime = millis(); // reset auto scroll when user presses button
    }
    switchState = reading;
  }

  lastButtonState = reading;

  // 3. Auto scroll if no button pressed
  if (millis() - lastAutoScrollTime >= autoScrollInterval) {
    metricIndex++;
    if (metricIndex > 2) {
      metricIndex = 0;
    }
    lcd.clear();
    lastAutoScrollTime = millis();
  }

  // 4. Update LCD every second
  if (millis() - lastUpdateTime >= updateInterval) {
    updateDisplay();
    lastUpdateTime = millis();
  }
}

void parseSensorData(String data) {
  int cpuIndex = data.indexOf("CPU:");
  int gpuIndex = data.indexOf("GPU:");
  int caseIndex = data.indexOf("CASE:");

  if (cpuIndex != -1 && gpuIndex != -1 && caseIndex != -1) {
    cpuTemp = data.substring(cpuIndex + 4, gpuIndex - 1).toFloat();
    gpuTemp = data.substring(gpuIndex + 4, caseIndex - 1).toFloat();
    caseTemp = data.substring(caseIndex + 5).toFloat();

    // Track maximums
    if (cpuTemp > cpuMax) cpuMax = cpuTemp;
    if (gpuTemp > gpuMax) gpuMax = gpuTemp;
    if (caseTemp > caseMax) caseMax = caseTemp;
  }
}

void updateDisplay() {
  lcd.setCursor(0, 0);
  switch (metricIndex) {
    case 0:
      lcd.print("CPU:");
      lcd.print(cpuTemp, 2);
      lcd.print((char)223);
      lcd.print("C");
      lcd.setCursor(0, 1);
      lcd.print("Max:");
      lcd.print(cpuMax, 2);
      lcd.print((char)223);
      lcd.print("C");
      break;
    case 1:
      lcd.print("GPU:");
      lcd.print(gpuTemp, 2);
      lcd.print((char)223);
      lcd.print("C");
      lcd.setCursor(0, 1);
      lcd.print("Max:");
      lcd.print(gpuMax, 2);
      lcd.print((char)223);
      lcd.print("C");
      break;
    case 2:
      lcd.print("Case Air:");
      lcd.print(caseTemp, 2);
      lcd.print((char)223);
      lcd.print("C");
      lcd.setCursor(0, 1);
      lcd.print("Max:");
      lcd.print(caseMax, 2);
      lcd.print((char)223);
      lcd.print("C");
      break;
  }
}
```

Troubleshooting

One major issue encountered during the lab was hardware failure. While attempting to wire up an external temperature sensor, two of the sensor's leads broke off. The sensor could not be repaired, and the kit only came with a single sensor, leaving no replacement available. To solve this, I decided to switch from using an external sensor to reading temperature data directly from the computer's onboard sensors. This required writing a Python script to gather system temperatures from LibreHardwareMonitor and send them over USB serial to the Arduino. Setting up serial communication, ensuring LibreHardwareMonitor was fully running before starting the script, and dealing with Windows COM port permissions were all small challenges that were solved with troubleshooting and patience.

Conclusion

Completing this lab required me to bridge my knowledge of both software and hardware. My background is mainly in software, so I naturally leaned toward finding a software-based solution when the hardware sensor failed. Instead of stopping the project or waiting for replacement parts, I wrote a Python script to collect the onboard sensor data from the motherboard, GPU, and other components. I set up a scheduled task in Windows to launch the Python script silently in the background when the system boots. This allowed the Arduino to immediately start receiving and displaying real-time temperatures on the LCD screen after startup, creating a seamless experience.

The functionality of the final device includes being able to press a button to cycle through different temperature readings, as well as automatically cycling through metrics if no button is pressed for a few seconds. I also added functionality to track and display the maximum recorded temperature for each sensor, allowing me to easily monitor peak system conditions.

In the future, I could improve the project by building a custom case to house the Arduino, LCD, and button neatly. Adding more buttons could allow easier navigation, like selecting which sensors to add or remove from the display. Expanding the number of sensors, improving the graphical display, or even logging temperature history could also be great future improvements. This project really showed me how combining software and hardware can create a functional and useful tool.