Module Two Assignment: Temperature Logging

Nicolas Miller

09/22/2024

1 Requirements

As specified in the assignment document, this project has been design to satisfy the following requirements:

- Select and calibrate a temperature sensor
- Capture temperature data and convert it to Fahrenheit
- Log temperature readings at a periodic interval
- Store the temperature data alongside timestamps in a CSV file
- Record temperature readings for:
  - An initial stabilization period at room temperature
  - A 5-minute period inside of a refrigerator
  - A 5-minute period after being moved from the refrigerator to allow temperature normalization back to room temperature
- Round Robin with Interrupts design
- Download temperature data to host

2 Design

Most of my time with this project was spent on the hardware design, which presented some interesting challenges. Because temperature had to be logged inside of a refrigerator, it wasn't feasible to use a USB serial connection throughout the entire logging process, so I had to come up with a design that stored temperature data and transmitted it over serial when a serial connection was established.

I toyed around with the idea of using an ESP8266 and transmitting data over a web server, but I felt that the risk of connection failures inside of the refrigerator had the potential to cause a lot of headaches and debugging time.

Instead, I opted to use an Arduino Uno R3, a MicroSD Card component, a BME280 sensor, and a button.

I acknowledge that the BME280 isn't the ideal choice for an "accurate" temperature sensor. As described in the module's data sheet, it's stated outright that the temperature sensor on the BME280, while calibrated at the factory, is primarily used to offer temperature compensation for the onboard pressure and humidity sensors. I did have a DHT11 component on hand that I planned to use at first, but my module was broken, and I wasn't able to repair it or get a new one in time. So I had to use the BME280.

Regarding calibration, the BME280 is stated to come calibrated from the factory. Based on comparisons with my air conditioning settings and the readings from the sensor, the factory calibration appears to be sufficiently accurate for the purposes of this project. For instance, with the thermostat set to 73°F, the BME280 consistently recorded a temperature of around 74°F. This slight difference is expected, as the thermostat setting reflects the target temperature rather than the exact ambient temperature in the room. Given this small variation, no further calibration was deemed necessary for the scope of this project.

In general, the hardware operates as follows: The BME280 logs temperature every 10 seconds after an internal timer interrupt is triggered. That temperature is written to a CSV file on the MicroSD Card. When a serial connection is established and the button on the breadboard is pressed, a hardware interrupt is used to flush the contents of the CSV file to the serial port.

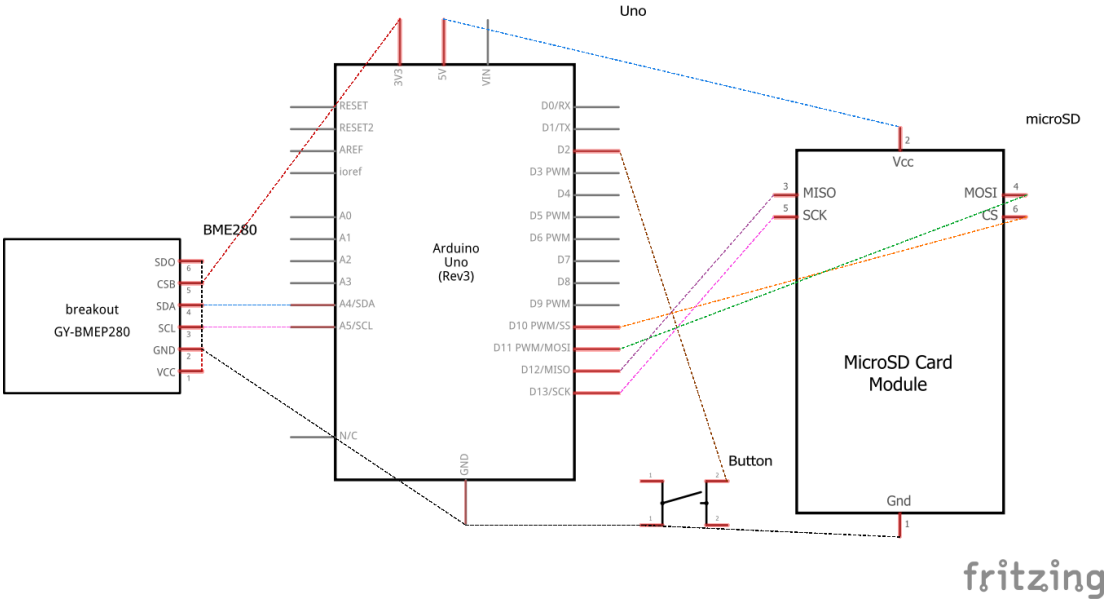
For the software design, I wanted to keep the `loop()` function clean and really lean into the Round Robin with Interrupts design. As hinted at in the previous paragraph, the main loop is composed of two primary conditional checks that check if a temperature logging flag or a button press flag have been set.

The temperature logging flag uses an internal timer interrupt. Because I don't think we are able to use an internal timer interrupt for a 10s timer, I use a 1s timer interrupt that only enables the temperature logging flag after 10 iterations through the interrupt routing. Once set, the BME280 logs the current temperature and writes it to the CSV file.

The button press flag uses an external hardware interrupt. A button on the breadboard is connected to digital pin 2 on the Arduino. When the button is pressed, the button press flag is set, and if a serial connection is present, the contents of the CSV file are sent to the serial port. To prevent false positives, from floating currents, an internal pull-up resistor is used on pin 2. I also use some debouncing timings to prevent false positives, which is something that caused me some problems during testing (e.g., hitting the button and data being transmitted twice or three times).

The rest of the code is straightforward. The `setup()` function defines a serial connection, initializes the BME280 and MicroSD card component, creates a CSV file if one is not already present on the SD card, and defines the interrupts. Temperature logging is done by reading the temperature from the BME280 and converting it to Fahrenheit, opening the CSV file, and writing the time in milliseconds and the temperature to the CSV file. Serial transmission is done by opening the CSV file and writing the contents to the serial port.

Circuit Diagram



3 Source Code

main.cpp

```
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#define BME280_I2C_ADDRESS 0x76  // Default I2C address for BME280
#define SD_CS_PIN 10             // Chip Select pin for SD card reader
#define BUTTON_PIN 2             // Interrupt button pin

Adafruit_BME280 bme;             // BME280 object
File dataFile;                   // File object for SD card

volatile bool buttonPressed = false;       // Track button state
volatile bool logTemperatureFlag = false; // Track if temperature should be
                                          // logged

unsigned long debounceTime = 0;           // Time of last button press
const unsigned long debounceDelay = 200; // Debounce delay in milliseconds

// Declare functions
void buttonISR();
void logTemperature();
void transferDataToSerial();

void setup()
{
  // Initialize Serial Communication
  Serial.begin(9600);

  // Initialize BME280 sensor
  if (!bme.begin(BME280_I2C_ADDRESS))
  {
    // Loop forever if BME280 initialization fails. Future refactors may add
    // retry logic or flashing LEDs to indicate failure.
    while (1);
  }

  // Initialize SD card
  if (!SD.begin(SD_CS_PIN))
  {
    // Loop forever if SD Card initialization fails. Future refactors may add
```

```
43      // retry logic or flashing LEDs to indicate failure.
44      while (1);
45    }
46
47    // Check if the csv file exists; otherwise create it and write header
48    if (!SD.exists("tempdata.csv"))
49    {
50      dataFile = SD.open("tempdata.csv", FILE_WRITE);
51      if (dataFile)
52      {
53        dataFile.println("Timestamp(ms),Temperature(F)");
54        dataFile.close();
55      }
56    }
57
58    // Set up an internal 1s timer interrupt. Used Arduino interrupts timer
59    // calculator tool deepbluembedded website to generate values and code.
60    cli();                  // Disable global interrupts
61    TCCR1A = 0;             // Set TCCR1A register to 0
62    TCCR1B = 0;             // Set TCCR1B register to 0
63    TCCR1B |= B00000100;    // Prescaler = 256
64    OCR1A = 62500;          // Timer Compare1A Register
65    TIMSK1 |= B00000010;    // Enable Timer COMPA Interrupt
66    sei();                  // Enable global interrupts
67
68    // Set up button as an interrupt. We use the internal pull-up resistor to
69    // ensure no false button presses detected. Interrupt triggers on a falling
70    // edge.
71    pinMode(BUTTON_PIN, INPUT_PULLUP);
72    attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), buttonISR, FALLING);
73  }
74
75  void loop()
76  {
77    // Check if it's time to take a temperature reading
78    if (logTemperatureFlag)
79    {
80      // Reset temperature flag
81      logTemperatureFlag = false;
82
83      // Log temperature to SD card
84      logTemperature();
85    }
86
87    // Check if button has been pressed and serial connection is available
88    if (buttonPressed)
```

```
89    {
90      // Reset button pressed flag
91      buttonPressed = false;
92
93      // Transfer data if serial connection is available
94      if (Serial)
95      {
96        // Transfer CSV data to serial
97        transferDataToSerial();
98      }
99    }
100 }
101
102 // Interrupt Service Routine for Timer1
103 ISR(TIMER1_COMPA_vect)
104 {
105   static int count = 0; // Counter for number of seconds; used to count to 10
106   count++;              // Increment counter
107   OCR1A += 62500;       // Advance The COMPA Register
108
109   if (count >= 10)
110   {
111     count = 0;
112     logTemperatureFlag = true;
113   }
114 }
115
116 // ISR function for button press
117 void buttonISR()
118 {
119   // Check debounce
120   if ((millis() - debounceTime) > debounceDelay)
121   {
122     buttonPressed = true;
123     debounceTime = millis();
124   }
125 }
126
127 // Function to read temperature and log to SD card
128 void logTemperature()
129 {
130   // Read temperature from BME280
131   float temperature = bme.readTemperature();
132
133   // Convert temperature to Fahrenheit
134   temperature = (temperature * 1.8) + 32;
```

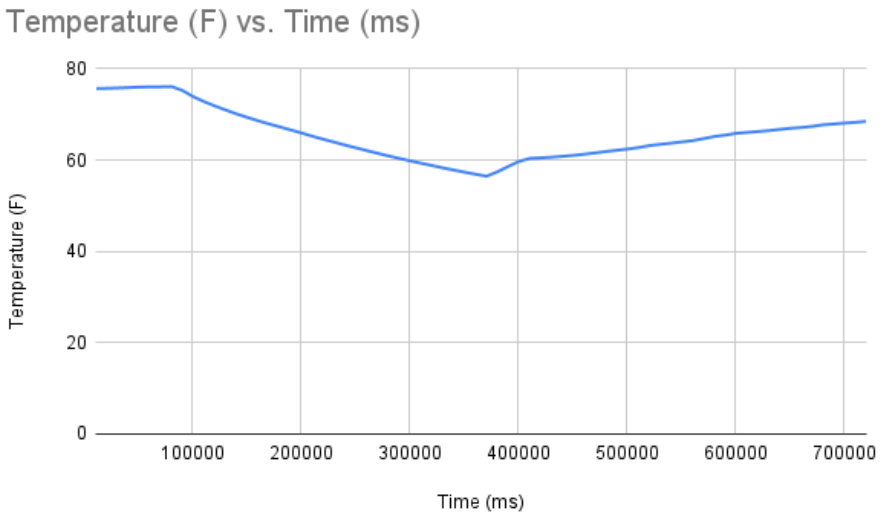
```
135
136   // Open the file stored on the SD card for writing
137   dataFile = SD.open("tempdata.csv", FILE_WRITE);
138   if (dataFile)
139   {
140     // Log timestamp and temperature to file
141     dataFile.print(millis());
142     dataFile.print(",");
143     dataFile.println(temperature);
144     dataFile.close();
145   }
146   else
147   {
148     Serial.println("Error opening tempdata.csv for logging");
149   }
150 }
151
152 // Function to transfer CSV data to serial
153 void transferDataToSerial()
154 {
155   // Open the file stored on the SD card for reading
156   dataFile = SD.open("tempdata.csv");
157
158   if (dataFile)
159   {
160     // Flush contents of data file to serial
161     while (dataFile.available())
162     {
163       Serial.write(dataFile.read());
164     }
165
166     // Close the file
167     dataFile.close();
168   }
169   else
170   {
171     Serial.println("Error opening tempdata.csv for data transfer");
172   }
173 }
```

4 Video

A video demonstration of the project can be found at the following link: https://youtu.be/fBPzude1SVE

5 Data

Here are the results of the temperature logging process:



The results demonstrate gradual adjustments to ambient temperature, which was expected. We see a steady, stabilized temperature at the beginning of the graph, which begins to gradually drop after the unit was placed inside of the refrigerator. Interestingly, there is a small spike in the middle of the graph at the point when the unit was moved out of the refrigerator. Based on some quick research I did online, I think this can be attributed to hysteresis.

After removing the unit from the refrigerator, we see a gradual rise in temperature as the sensor readjusts to room temperature. Worth noting is that the temperature does not return to the stable room temperature during that five minute period. This indicates that the BME280 takes much longer to adjust to increases in ambient temperature than it does in decreases.

6 Sharing Statement

I will share this entire PDF submission in the sharing discussion board.