Project Report

Thermocouple Reader for Heat Transfer

Version 1.3

December 15, 2024

Authors: Elizabeth Anthony, Jonathan Behrens, Rory Iwaneczko, Thomas Lea, Noah Richert

Version History

Version	Date	Description
1.0	11/7/2024	The Thermocouple for Heat Transfer team developed an initial map of the project report, covering a complete overview of the project's design process.
1.1	11/26/2024	Noah Richert adjusted the formatting and structure of the map to reflect feedback from Camilo Giraldo. The updated version was submitted as the final map for additional feedback.
1.2	12/04/2024	The Thermocouple for Heat Transfer team developed the initial report submission according to the map's structure. Elizabeth Anthony covered the project Background, Noah Richert addressed the design and validation of the code, Rory Iwaneczko discussed the device housing and budget, Jonathan Behrens described usage of the TCR. Updates were made to the Abstract and Conclusion by Thomas Lea.
1.3	12/15/2024	The Thermocouple for Heat Transfer team updated the report according to feedback by Camilo Giraldo. Edits were made to improve clarity, create proper citations, and ensure results are up to date. The completed report was submitted as the final project report.

Table of Contents

Table of Contents 3

Abstract 4

Background 4

Code Design and Validation 5

Device Housing and Cost 7

Using Thermocouple Reader 9

Conclusion 10

Bibliography 10

Appendix – Arduino Code 11

Abstract

The current heat transfer lab uses thermocouple readers made by LabVIEW and Texas Instruments to measure how temperature changes over time. Unfortunately, this equipment can cost up to $5,000. However, a cheaper system with a cost of around $230 has been designed and coded. This system's thermocouple reader has been verified as accurate to within ±0.5°C in environments; such as ice water, boiling water, and room temperature. Additionally, a transient conduction experiment, where the temperature on the inside of a non-temperature-conducting sphere is recorded over a period of 30 minutes, has been run with this system confirming its ability to function over time and display live data. Overall, this new Arduino system is simple to use and would work as an effective replacement for the LabVIEW system.

Background

In many Heat Transfer labs, students must record up to six temperature readings over varying periods. The Heat Transfer class at Messiah University is looking to implement a new temperature data acquisition system for these lab experiments for around $230, which would be less expensive than the current system. Currently, the class uses a LabVIEW and Texas Instruments system for such experiments. However, this system costs about $2,000 per hardware unit, plus an additional $3,000 for licenses to the software for multiple students. This price is higher than the Engineering Department would prefer to pay for the system, so it is searching for a new alternative. The new system must be able to record temperature and time data on at least 6 channels simultaneously. One potential option for building an alternative system would be to use thermocouples with an Arduino board.

Based on the criteria and information required of the new system, a list of the necessary specifications for the system was created. It was determined that it would be simplest to use an Arduino Uno microcontroller with the Arduino coding software. Due to the specifications of lab experiments, the system must be able to read six thermocouples accurately, within ±0.5°C. It also must be able to collect both time and temperature data at a frequency of 100 Hz for accuracy. Since this data collection frequency is much faster than the frequency at which the data must be recorded, the system must include an averaging feature which would allow the 100 Hz collected data to be averaged every second such that the average data is displayed every second (or, at 1 Hz). The system should record the average time that corresponds to each averaged temperature reading. It would also be beneficial to allow the user to define both the sampling rate and the averaging rate, so that these values could be easily changed without significantly altering the code. The system must also be able to display live data in some manner, as well as to save that data to a CSV or text file. Additionally, the finished system must be contained within a housing in order to protect its components from splashes of liquids used in the lab. This housing should be able to fit on a typical table and be openable without requiring specific tools.

This report aims to show the process of designing, coding, and manufacturing a lower-cost thermocouple reader system that will meet the specifications of the Heat Transfer class at Messiah University. Additionally, this report contains instructions to the user on how to properly use the system.

Code Design and Validation

During the team's research, it was discovered that successful thermocouple projects have been completed with two different thermocouple readers, a single-input thermocouple reader (MCP9600, Adafruit) [1] and a four-input reader (MCP9601, Playing with Fusion) [2]. The team thus analyzed the two systems to determine which approach would be optimal for the project's objectives [3]. The Adafruit board is a compact and cost-effective thermocouple amplifier at $16 per reader that measures the temperature from one thermocouple and can be used in combination with Arduino to display temperatures on the Serial Monitor. The Playing with Fusion board, a similar reader design, is slightly larger but is capable of reading up to 4 thermocouples at once. Each board was briefly tested to confirm that they were capable of reading thermocouples and transmitting them to the Arduino Serial Monitor. After ensuring that both approaches are capable of meeting the client's needs, the team chose to move forward with the Playing with Fusion board as the reader design to incorporate into the system, primarily due to its space efficiency and ease of use [3].

Having chosen the Playing with Fusion board, the reader was integrated into a more complete circuit that could meet the needs of the client. This circuit consists of two Playing with Fusion readers, a multiplexer (TCA9548 I2C Multiplexer, Adafruit) [4], an SD card reader (MicroSD breakout board, Adafruit) [5], and an Arduino Uno [6]. The circuit (Figure 1) [7] was accompanied by an Arduino code that directs the reading of the thermocouples and the recording of the data [8](see the Appendix for the complete Arduino code). This code was developed and tested with an experimental setup (Figure 2) to verify the proper collection and recording of temperature data. The basic tests focused on the user-defined inputs of File Name, Sampling Frequency, and Averaging Rate, as well as the outputs of recording data to Excel and a text file using the SD card. The results of the tests confirmed that all inputs and outputs function as desired and meet the client's specifications.

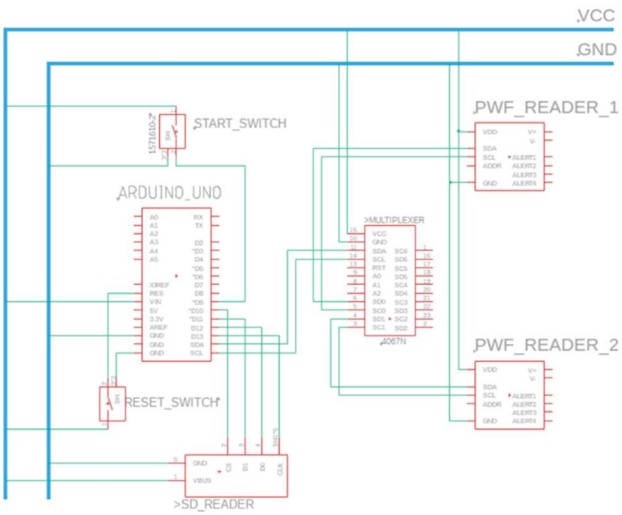


Figure 1: Circuit Schematic

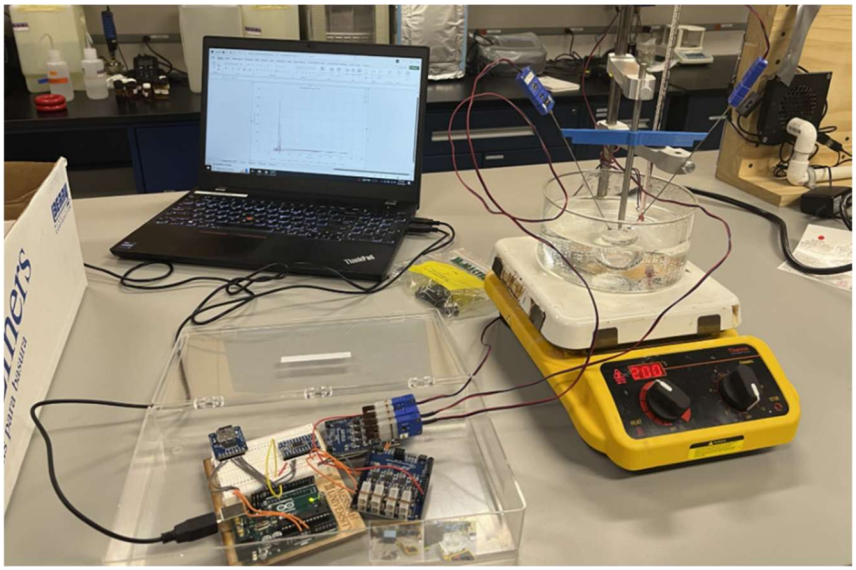
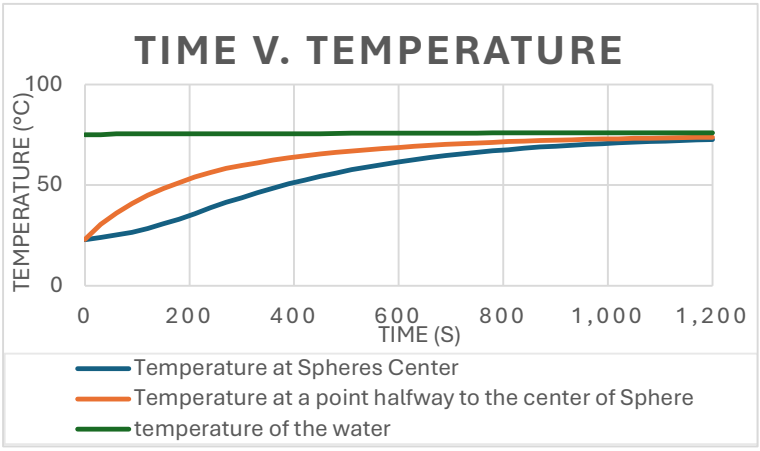
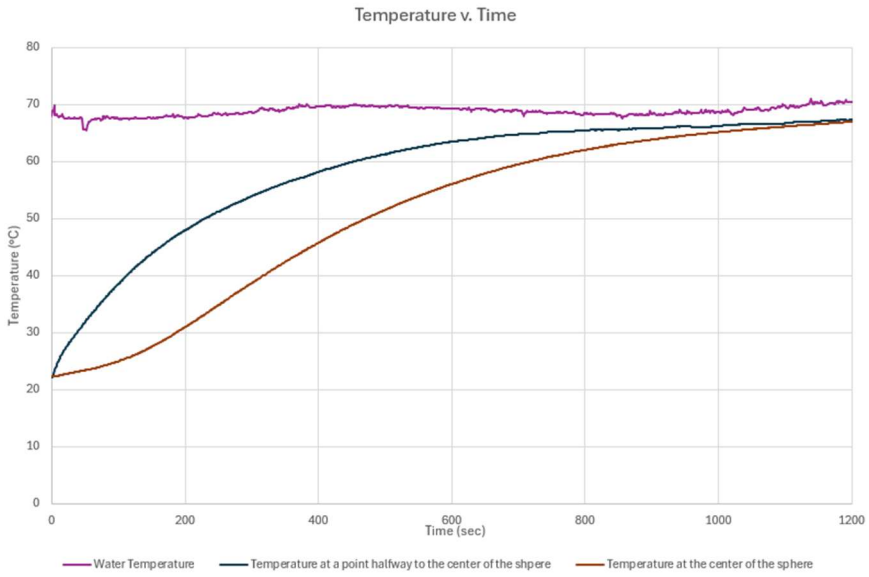


Figure 2: Experimental testing setup.

In addition to basic verification tests, the thermocouple reading system was validated by performing a transient conduction experiment [9]. This experiment consists of submerging a non-temperature-conducting glass sphere in water at a constant temperature (Figure 2) and periodically recording the temperature both at the center of the sphere and a point halfway to the center of the sphere for 20-30 minutes after submersion. The data gathered by the Arduino system showed comparable results to previous runs of this experiment, indicating successful collection and transfer of the data. A sample set of data (Graph 1) can be compared to the Arduino reader's output (Graph 2) to verify that the data reflects the expected behavior of the temperature over time. As can be seen from Graph 1 and Graph 2, the temperature at the halfway point shows a steep incline at the start in both graphs while the temperature at the centers begins increasing slowly but speeds up as time goes on.



Graph 1: Time v. Temperature of a sphere in 75°C Water (Previous Experiments)



Graph 2: Time v. Temperature of a sphere in 70°C water (as measured by Arduino)

Device Housing and Cost

Upon validating the software, all components were securely enclosed within a custom-designed housing that integrates the system's electrical elements to simplify user interaction and protect the components from environmental hazards and potential damage caused by direct handling (Figure 3). The housing consists of a three-walled main structure that accommodates components in tailored slots, resembling the layout of a PC case. The front face, depicted to the left in Figure 3, provides access to essential component interfaces for external connections. The back panel shown on the right side of Figure 3, slides into place and fully encloses the system. To ensure proper heat dissipation, the back panel features angled openings that allow airflow while minimizing exposure to environmental risks. Additionally, pins are strategically positioned behind the components to prevent dislodgment when users interact with the interfaces. For additional specifications refer to "Design and Implementation of a Housing for Thermocouple Data Acquisition System" [1].

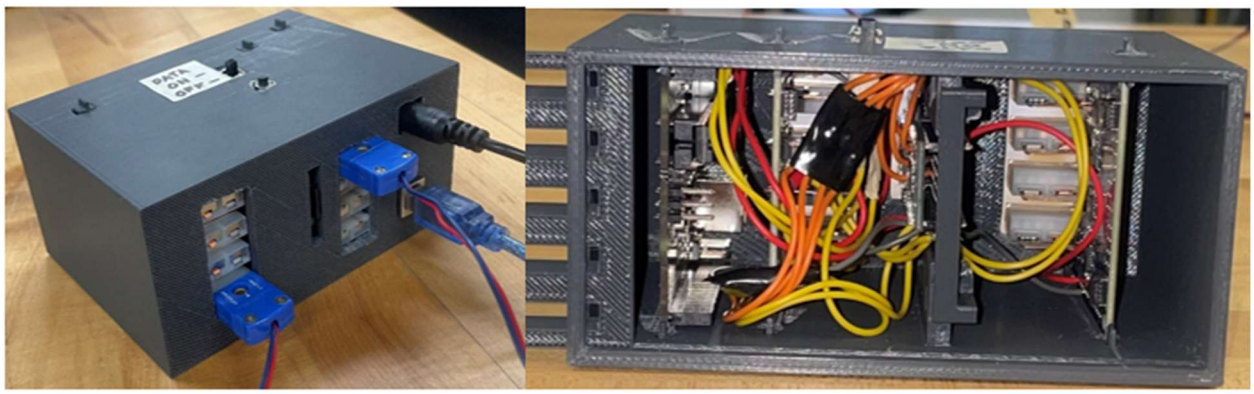


Figure 3: Fully Assembled Housing and System (Front and Back)

The bill of materials for the thermocouple reading system prototype includes two thermocouple reader boards [7], one multiplexer [6], one micro SD reader [5], one Adafruit micro SD card [3],and one Arduino Uno Rev3 microcontroller [4], bringing the total cost to $229. Additional components such as soldering material, wires, switches, thermocouples, and 3D printing material were provided by the Messiah University Engineering Department and were not factored into the total cost. Extra thermocouple reader boards, Arduino, and multiplexers were included to assemble a backup system, ensuring the reliability of the design. The total cost of the project, including the backup system and additional components, is $423.

Table 1: Bill of Materials

Item No.	Part Name	Specifications	Vendor	Quantity (Project)	Quantity (System)	Price (Project) ($)	Price (System) ($)
1	Thermocouple Reader Boards	Playing With Fusion SEN-30011	Playing With Fusion	4	2	360	180
2	Adafruit TCA9548A 1-to-8 I2C Multiplexer Breakout	I2C Multiplexer	Adafruit Industries	3	1	21	7
3	Arduino Uno Rev3	Arduino Microcontroller	Arduino	1	1	24	24
4	128MB Micro SD Memory Card	Standard Storage	Adafruit Industries	1	1	10	10
5	MicroSD Card Breakout Board+	External MicroSD Card Reader	Adafruit Industries	1	1	8	8
	Grand Total					$423	$229

Using Thermocouple Reader

One of the primary focus areas of this project was developing an easy-to-use system in a lab environment. This was a factor in our selection of the PWF system, the Arduino, and especially in how we wrote the code. This section will break down the process allowing the user to record accurate data on our system. The following steps will be provided in lab procedures:

1. Before connecting to the system, ensure the external data switch is in the "OFF" position. Open an Excel file with Data Streamer, as shown in Figure 5, and set the following settings.

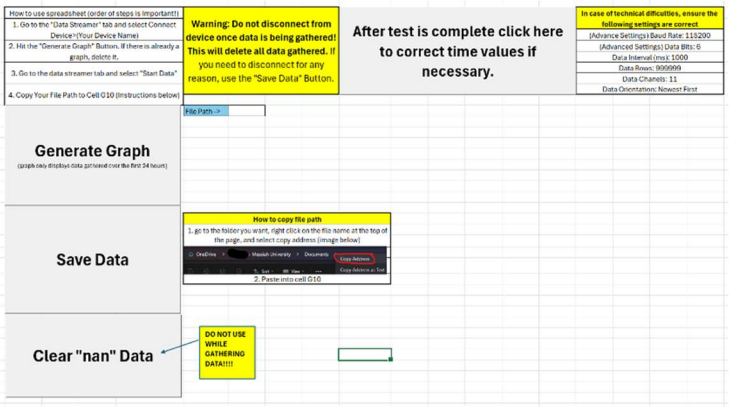


Figure 4: Cover page of the template data streamer sheet.

2. Follow the instructions in the top left corner of the document.
3. During the experiment's setup, the system should be connected to a computer that runs Excel and Arduino IDE using a USB-B cable. Connecting to power is not necessary.
4. Insert any thermocouples you wish to use for your experiment. Location in the ports does not matter but match the Copper and Constantin pins to the matching-colored side of the socket.
5. Open Arduino IDE with the code provided by your lab instructor and set the following variables as needed for the test you wish to run. (Figure 6) Changing the filename is not necessary but could be helpful as a reference. You must have this file already on the SD card.

```
8
9   //--------------------------------------------------------------------//
10
11  // User-Defined Inputs
12  int SamplingFreq = 100;        // Sampling Frequency [Hz]
13  float AvgFrequency = 1;        // Average Frequency [Hz]
14  char filename[] = "test.txt";  // Name cannot be longer than XXXX.txt
15
16  //--------------------------------------------------------------------//
17
```

Figure 5: User-defined variables for the Arduino UNO code.

6. Ensure the MicroSD card has been inserted securely, and the system has been set up. Open the Serial Monitor on Arduino IDE and check that it is set to 15200 baud. You should see an update from the computer that has identified your thermocouples and that there is an SD card present. If there is an error, please recheck your connections.

7. Once all systems have been checked and you are ready to test, flip the data switch to “ON.”
8. Watch the data on Excel once the test has concluded, or if you need to stop the test, switch the data switch to “OFF” and ensure no new data is transmitted before altering the connections, then press the “RESET” button.
9. Once your test has been completed, you may export your data as a file from Excel or the MicroSD card in the TCR.

Conclusion

In summary, the Project Management Thermocouple Team was able to assemble a working prototype and is on schedule to complete our goals by the end of the semester. The device is equipped with a working code and adjustable parameters, provides accurate data readings, and can track temperature over time. This device is also much cheaper than the system currently being used by the heat transfer lab; only costing $230. The Arduino-based system meets our goals and specifications.

Bibliography

[1] A. Industries, "Adafruit MCP9600 I2C Thermocouple Amplifier," 2024.

[2] I. Playing With Fusion, "Universal Thermocouple Digital I2C MCP9601," 2024.

[3] J. Behrens, "Thermocouple System: Alternatives and Decisions," 2024.

[4] Adafruit Industries, "Adafruit TCA9548A 1-to-8 I2C multiplexer breakout," 11 September 2015. [Online]. Available: https://learn.adafruit.com/adafruit-tca9548a-1-to-8-i2c-multiplexer-breakout/overview.

[5] Adafruit Industries, "Micro SD Card Breakout Board Tutorial," 31 July 2013. [Online]. Available: https://learn.adafruit.com/adafruit-micro-sd-breakout-board-card-tutorial/download.

[6] Arduino, "Arduino Uno REV3," [Online]. Available: https://store-usa.arduino.cc/products/arduino-uno-rev3.

[7] N. Richert, "Testing of Thermocouple Code and User-Defined Variables," 2024.

[8] E. Anthony, "Coding & Averaging System Design," 2024.

[9] T. Lea, "Thermocouple System in the Lab," 2024.

[10] R. Iwaneczko, "Design and Implementation of a Housing for Thermocouple Data Acquisition System," Mechanicsburg, 2024.

[11] Adafruit Industries, "128MB Micro SD Memory Card," [Online]. Available: https://www.adafruit.com/product/5250. [Accessed 15 December 2024].

[12] J. Steinlage, "4-Channel T-Type Thermocouple Digital I2C Interface MCP9601 Breakout," 13 Mach 2021. [Online]. Available: https://www.playingwithfusion.com/productview.php?pdid=119.

[13] N. Instruments, "Test and Measurement Systems," 2024.

Appendix – Arduino Code

```
//Libraries to be used
#include <Wire.h>
#include <PWFusion_Mcp960x.h>
#include <Adafruit_I2CDevice.h>
#include <Adafruit_I2CRegister.h>
#include <SPI.h>
#include <SD.h>

//-------------------------------------------------------------------------//

// User-Defined Inputs
int SamplingFreq = 100;     // Sampling Frequency [Hz]
float AvgFrequency = 1;     // Average Frequency [Hz]
char filename[] = "test.txt";  // Name cannot be longer than XXXX.txt

//-------------------------------------------------------------------------//

// Average Variables
int count = 0;
unsigned long millis_rec_all = 0;
unsigned long millis_rec;
float time_window = 1 / AvgFrequency;
float avg_time;

//Frequency to Delay Conversion
float FreqDelay = (0.0011 * SamplingFreq * SamplingFreq) - 0.311 * SamplingFreq + 22.429;

//SD Card Variables
const int chipSelect = 10;
const int switch_pin = 9;
int switch_val;
int last_switch_val = LOW;
File myFile;

//Defining address for multiplexer
int TCAADR = 0x70;

//Define number of thermocouples per system
const int num_thermocouples = 4;

//Defining objects for both thermocouples
Mcp960x thermo_1[num_thermocouples];  //System 1
```

```
Mcp960x thermo_2[num_thermocouples];  //System 2

//Define addresses for both thermocouples thermocouple systems
const uint8_t mcp9601Addr[] = { 0, 1, 4, 7 };

//Define output temperatures
float T0_1;       //T0 from system 1 (after average)
float T0_1_all = 0;  //Cumulative sum of T0 sys 1 for Avg purposes
float T0_1_curr;    //Current value of T0 sys 1
int count_T0_1;     //number of data points from T0 sys 1

float T1_1;       //T1 from system 1
float T1_1_all = 0;  //Cumulative sum of T1 sys 1 for Avg purposes
float T1_1_curr;    //Current value of T1 sys 1
int count_T1_1;     //number of data points from T1 sys 1

float T4_1;       //T4 from system 1
float T4_1_all = 0;  //Cumulative sum of T4 sys 1 for Avg purposes
float T4_1_curr;    //Current value of T4 sys 1
int count_T4_1;     //number of data points from T4 sys 1

float T7_1;       //T1 from system 1
float T7_1_all = 0;  //Cumulative sum of T7 sys 1 for Avg purposes
float T7_1_curr;    //Current value of T7 sys 1
int count_T7_1;     //number of data points from T7 sys 1

float T0_2;       //T0 from system 2
float T0_2_all = 0;  //Cumulative sum of T0 sys 2 for Avg purposes
float T0_2_curr;    //Current value of T0 sys 2
int count_T0_2;     //number of data points from T0 sys 2

float T1_2;       //T1 from system 2
float T1_2_all = 0;  //Cumulative sum of T1 sys 2 for Avg purposes
float T1_2_curr;    //Current value of T1 sys 2
int count_T1_2;     //number of data points from T1 sys 2

float T4_2;       //T4 from system 2
float T4_2_all = 0;  //Cumulative sum of T4 sys 2 for Avg purposes
float T4_2_curr;    //Current value of T4 sys 2
int count_T4_2;     //number of data points from T4 sys 2

float T7_2;       //T1 from system 2
float T7_2_all = 0;  //Cumulative sum of T7 sys 2 for Avg purposes
float T7_2_curr;    //Current value of T7 sys 2
int count_T7_2;     //number of data points from T7 sys 2

//Timer variables
unsigned long millis_curr;
unsigned long millis_last;

void setup() {

 //Starting baudrate
 Serial.begin(115200);

 //Initialize Wire library
 Wire.begin();

 //Initiating multiplexer with both systems
 tcaselect(0);  //set system 1 to TCA pin 0
 tcaselect(1);  //set system 1 to TCA pin 1

 //Waiting for serial to open
 while (!Serial) {
```

```
 ;  // wait for serial port to connect. Needed for native USB port only
}
// see if the card is present and can be initialized:
if (SD.begin(chipSelect)) {
 Serial.println("SD card is present & ready");
} else {
 Serial.println("SD card missing or failure");
 while (1)
  ;  //wait here forever
}
Serial.println("card initialized.");

//Declaring pins
pinMode(switch_pin, OUTPUT);

//Initiating the thermocouples for both systems
for (int i = 0; i < num_thermocouples; i++) {

 //Calling systems to initiate
 tcaselect(0);
 thermo_1[i].begin(mcp9601Addr[i]);  //System 1
 tcaselect(1);
 thermo_2[i].begin(mcp9601Addr[i]);  //System 2

 //Checking for connection on system 1
 tcaselect(0);
 if (thermo_1[i].isConnected()) {

  //Thermocouple was found in system 1
  Serial.print(F("Found MCP9601 system 1 at address "));
  Serial.println(i);

  //Set the type and resolution
  thermo_1[i].setThermocoupleType(TYPE_T);
  thermo_1[i].setResolution(RES_18BIT, RES_0p0625);

 } else {

  //Send an error message
  Serial.print(F("ERROR: Unable to connect to MCP9601 system 1 at address "));
  Serial.println(i);

  //Trap code in infinite loop
  while (1)
   ;
 }

 //Checking for connection on system 2
 tcaselect(1);
 if (thermo_2[i].isConnected()) {

  //Thermocouple was found in system 2
  Serial.print(F("Found MCP9601 system 2 at address "));
  Serial.println(i);

  //Set the type and resolution
  thermo_2[i].setThermocoupleType(TYPE_T);
  thermo_2[i].setResolution(RES_18BIT, RES_0p0625);

 } else {

  //Send an error message
  Serial.print(F("ERROR: Unable to connect to MCP9601 system 2 at address "));
  Serial.println(i);
```

```
    //Trap code in infinite loop
    while (1)
     ;
   }
  }

  //Check for SD file & Error
  if (!SD.exists(filename)) {
   Serial.println("Requested file doesn't exist.");
   while (1)
    ;
  }

}

//Setting TCA output channel
void tcaselect(uint8_t channel) {
 if (channel > 7) return;
 Wire.beginTransmission(TCAADR);
 Wire.write(1 << channel);
 Wire.endTransmission();
}

void loop() {

 myFile = SD.open(filename, FILE_WRITE);

 switch_val = digitalRead(switch_pin);

 if (switch_val == HIGH) {

  //Updating millis_curr
  millis_curr = millis();
  millis_last = millis_curr;

  //Initializing counters and other cumulative variables
  count = 0;
  T0_1_all = 0;
  T1_1_all = 0;
  T4_1_all = 0;
  T7_1_all = 0;
  T0_2_all = 0;
  T1_2_all = 0;
  T4_2_all = 0;
  T7_2_all = 0;
  millis_rec_all = 0;
  count_T0_1 = 0;
  count_T1_1 = 0;
  count_T4_1 = 0;
  count_T7_1 = 0;
  count_T0_2 = 0;
  count_T1_2 = 0;
  count_T4_2 = 0;
  count_T7_2 = 0;

  //Getting access to while loop for averaging feature
  while (millis_curr - millis_last < time_window * 1000.0) {

   //Time Recording
   if (switch_val == HIGH && last_switch_val == LOW) {
    millis_curr = 0;
    millis_rec = 0;
    millis_rec_all = 0;
```

```
}

millis_curr = millis();
millis_rec = millis_curr;
millis_rec_all = millis_rec_all + millis_rec;


//Working on system 1
tcaselect(0);

//Going over all thermocouples in system 1
for (int i = 0; i < num_thermocouples; i++) {

 //Only recording data when the sensor in system 1 is ready
 if (thermo_1[i].getStatus() == READY) {

  //Recording all wired thermocouples in system 1
  if (i == 0) {  //Recording thermocouple 0
   T0_1_curr = thermo_1[i].getThermocoupleTemp();
   T0_1_all = T0_1_all + T0_1_curr;
   count_T0_1 = count_T0_1 + 1;

  } else if (i == 1) {  //Recording thermocouple 1
   T1_1_curr = thermo_1[i].getThermocoupleTemp();
   T1_1_all = T1_1_all + T1_1_curr;
   count_T1_1 = count_T1_1 + 1;
  } else if (i == 2) {  //Recording thermocouple 4
   T4_1_curr = thermo_1[i].getThermocoupleTemp();
   T4_1_all = T4_1_all + T4_1_curr;
   count_T4_1 = count_T4_1 + 1;
  } else if (i == 3) {  //Recording thermocouple 7
   T7_1_curr = thermo_1[i].getThermocoupleTemp();
   T7_1_all = T7_1_all + T7_1_curr;
   count_T7_1 = count_T7_1 + 1;
  }
 }
}

//Working on system 2
tcaselect(1);

//Going over all thermocouples in system 2
for (int i = 0; i < num_thermocouples; i++) {

 //Only recording data when sensor in system 2 is ready
 if (thermo_2[i].getStatus() == READY) {

  //Recording all wired thermocouples in system 2
  if (i == 0) {  //Recording thermocouple 0
   T0_2_curr = thermo_2[i].getThermocoupleTemp();
   T0_2_all = T0_2_all + T0_2_curr;
   count_T0_2 = count_T0_2 + 1;
  } else if (i == 1) {  //Recording thermocouple 1
   T1_2_curr = thermo_2[i].getThermocoupleTemp();
   T1_2_all = T1_2_all + T1_2_curr;
   count_T1_2 = count_T1_2 + 1;
  } else if (i == 2) {  //Recording thermocouple 4
   T4_2_curr = thermo_2[i].getThermocoupleTemp();
   T4_2_all = T4_2_all + T4_2_curr;
   count_T4_2 = count_T4_2 + 1;
  } else if (i == 3) {  //Recording thermocouple 7
   T7_2_curr = thermo_2[i].getThermocoupleTemp();
   T7_2_all = T7_2_all + T7_2_curr;
   count_T7_2 = count_T7_2 + 1;
```

```
    }
   }
  }

  //Update the counter
  count = count + 1;

  //Apply delay
  delay(FreqDelay);  //Equation for delay based on user defined frequency
}

//Calculating Average of Temperature Readings & Time
T0_1 = T0_1_all / count_T0_1;
T1_1 = T1_1_all / count_T1_1;
T4_1 = T4_1_all / count_T4_1;
T7_1 = T7_1_all / count_T7_1;
T0_2 = T0_2_all / count_T0_2;
T1_2 = T1_2_all / count_T1_2;
T4_2 = T4_2_all / count_T4_2;
T7_2 = T7_2_all / count_T7_2;
avg_time = millis_rec_all / (count * 1000.0);

//PRINTING ALL DATA TO SERIAL MONITOR

//Print timer
Serial.print(avg_time);
Serial.print(",");

//Printing readings from system 1
Serial.print(T0_1);
Serial.print(",");  //Serial.print(T0_CJ_1); Serial.print(",");
Serial.print(T1_1);
Serial.print(",");  //Serial.print(T1_CJ_1); Serial.print(",");
Serial.print(T4_1);
Serial.print(",");  //Serial.print(T4_CJ_1); Serial.print(",");
Serial.print(T7_1);
Serial.print(",");  //Serial.print(T7_CJ_1); Serial.print(",");

//Printing readings from system 2
Serial.print(T0_2);
Serial.print(",");  //Serial.print(T0_CJ_2); Serial.print(",");
Serial.print(T1_2);
Serial.print(",");  //Serial.print(T1_CJ_2); Serial.print(",");
Serial.print(T4_2);
Serial.print(",");  //Serial.print(T4_CJ_2); Serial.print(",");
Serial.print(T7_2);
Serial.print(",");  //Serial.println(T7_CJ_2);

//PRINTING ALL DATA TO SD CARD

//Print timer
myFile.print(avg_time);
myFile.print(",");

//Printing readings from system 1
myFile.print(T0_1);
myFile.print(",");  //myFile.print(T0_CJ_1); myFile.print(",");
myFile.print(T1_1);
myFile.print(",");  //myFile.print(T1_CJ_1); myFile.print(",");
myFile.print(T4_1);
myFile.print(",");  //myFile.print(T4_CJ_1); myFile.print(",");
myFile.print(T7_1);
myFile.print(",");  //myFile.print(T7_CJ_1); myFile.print(",");
```

```
  //Printing readings from system 2
  myFile.print(T0_2);
  myFile.print(","); //myFile.print(T0_CJ_2); myFile.print(",");
  myFile.print(T1_2);
  myFile.print(","); //myFile.print(T1_CJ_2); myFile.print(",");
  myFile.print(T4_2);
  myFile.print(","); //myFile.print(T4_CJ_2); myFile.print(",");
  myFile.print(T7_2);
  myFile.print(","); //myFile.println(T7_CJ_2);

  //Stop recording if turned off
  if (switch_pin == LOW) {
   myFile.close();
   Serial.println("File is Done");
  }
 }
}
```