Technical Memo – Temperature Readout System

February 2, 2021
Joshua Collins
Electronics Development Group

Abstract

The temperature system described in this document is used to gather temperature data of the MPPC board during experiments. The system includes three separate boards and a computer that are connected via cables. The three board are a MPPC Board, an Adapter Interface Board, and an Arduino UNO. The resistance of a thermistor installed on the MPPC board responds to temperature and allows the digitization of the temperature. The resistance is sampled by a circuit on the Adapter Interface Board and read by an analog to digital converter (ADC) on the Arduino UNO. The UNO translates the digital data into readable figures that are sent to an attached computer. Software on the computer records the sent data and saves it on .TXT files for future data analysis.

System Overview

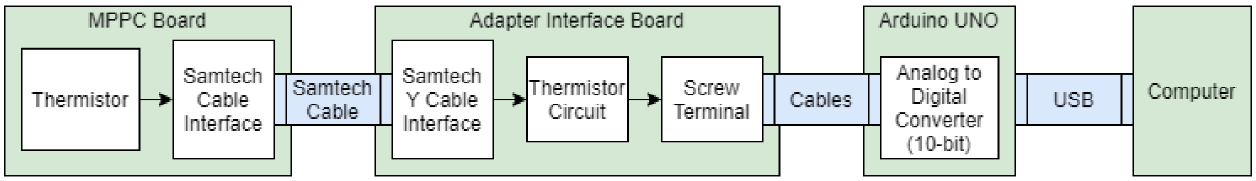


Figure 1 – Temperature System Diagram

The temperature system has four major subsystems: the MPPC Board, the Adapter Interface Board, the Arduino UNO, and a computer.

The MPPC Board includes several silicon photomultiplier chips (SiPMs) and two thermistors. The SiPMs are used to collect data in physics experiments. The resistance of the thermistors responds to variations in temperature. By using a voltage divider, temperature data can be collected and correlated to the data collected by the SiPMs.

The Adapter Interface Board is used to interface with the MPPC board and provide electrical connections to external devices. The board includes a part of the voltage divider circuit used to measure the temperature and a screw terminal to interface hook-up wires to the Arduino UNO

The Arduino UNO is a commercial microcontroller board that includes sets of digital and analog input/outputs (I/O) that can be interfaced with through stacked headers. Hook-up wires can be inserted into the headers to provide an electrical connection to the various I/Os. A 10-bit ADC is included on the Arduino UNO. The ADC samples the output of the temperature system and converts the voltage into a digital value. A calibration equation in the Arduino UNO's code converts the digital value into Celsius.

The Arduino UNO can communicate with a computer through an USB interface. Software can be created, modified, and uploaded to the Arduino UNO through the Arduino IDE. The standalone program RealTerm reads the serial output of the Arduino UNO and saves the data to .TXT files for future data analysis.

Electronic Hardware – MPPC Board

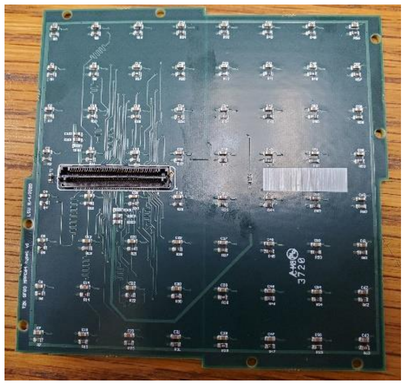


Figure 2 – A MPPC Board

The MPPC Board is a data collection PCB designed to capture light intensity data through an array of SiPM chips. The outputs of the SiPM chips are highly dependent on temperature. To normalize the outputs, the temperature must also be recorded. This is accomplished via two Semitec 103KT1608T-1P thermistors.

The resistance of a thermistor varies with temperature. As temperature decreases, the resistance increases. Conversely, when temperature increases, the resistance decreases. The nominal resistance at room temperature is 10kΩ. The relationship between temperature and resistance is shown in the graph below.

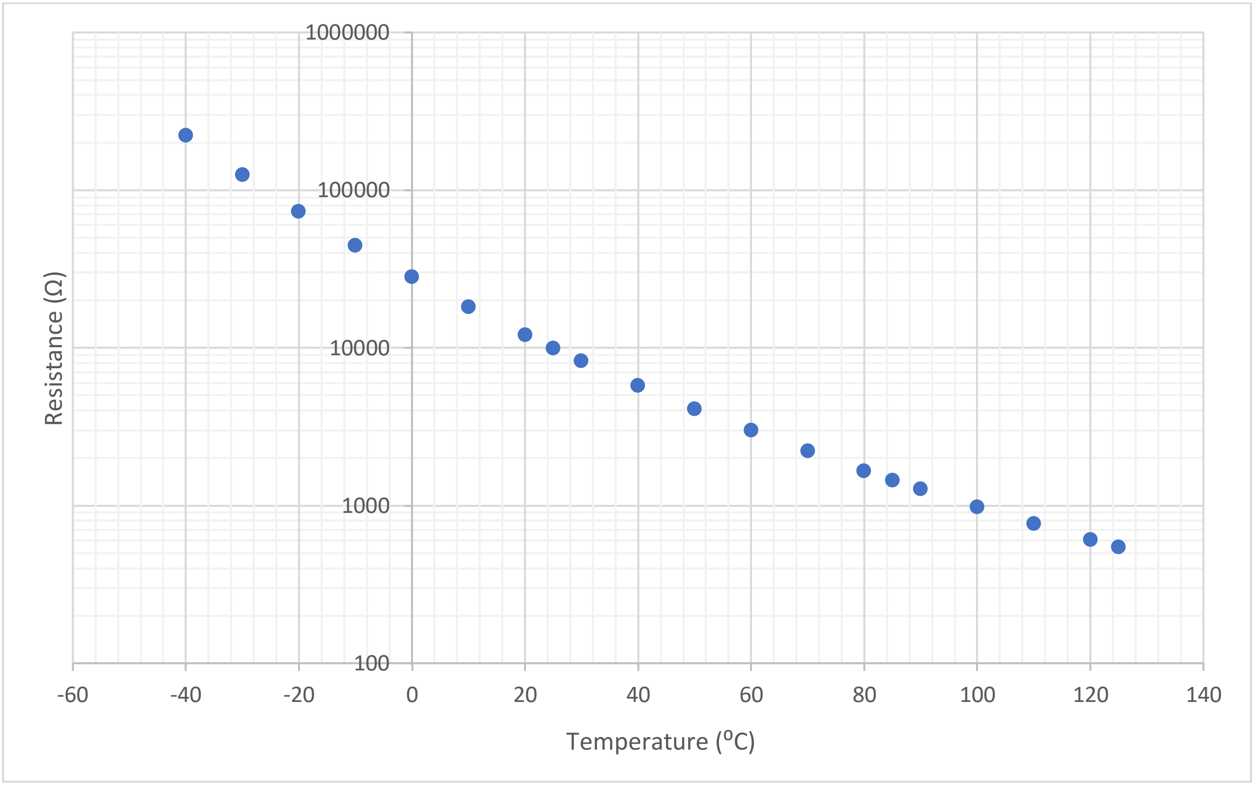
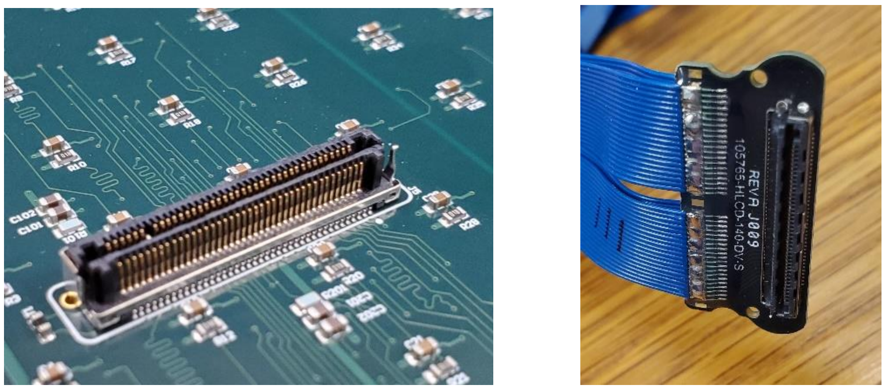


Figure 3 – Thermistor Resistance vs. Temperature

A high-speed Samtech LSHM-140-02.5-L-DV-A-S-K-TR series socket serves as the primary electrical interface for the MPPC board. The LSHM socket is mounted directly onto the board and connects to a Samtech HLCD series cable. The cable supplies power to the board, allows data to be collected from the SiPMs, and provides an electrical connection to the thermistors attached on the board.



(a) (b)

Figure 4 – (a) A Samtech LSHM series socket. (b) A Samtech HLCD series cable

Electronic Hardware – Interface Adapter Board

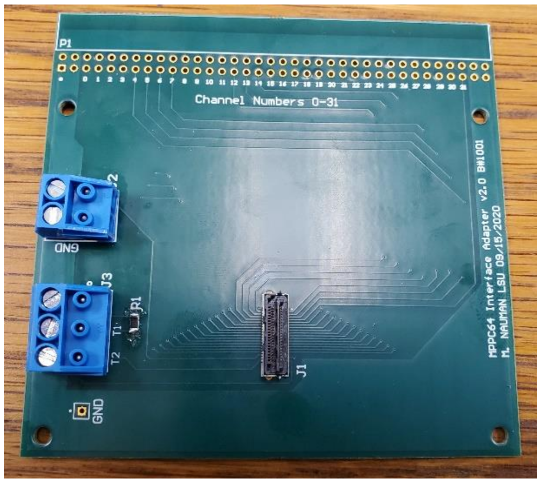
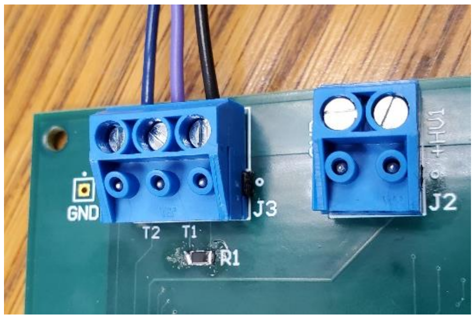


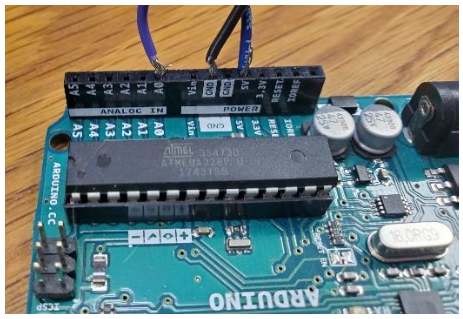
Figure 5 – Interface Adapter Board

The Interface Adapter Board is a custom design PCB board used to interface with the MPPC board. It includes a Samtech LSHM-120-02.5-L-DV-A-S-TR series socket to interface with a Samtech HLCD-Y series cable and a screw terminal to interface with the Arduino UNO. It also includes the thermistor readout circuit.

Power is supplied to the thermistor readout circuit from the Arduino UNO at screw terminal J3. Terminal T2 is connected to the +5V rail on the UNO. Pin 3 (Unlabeled) is connected to the ground of the Arduino. T1 is connected to the ADC Channel 0 or 1.



(a)



(b)

Figure 6 – Wire connection between the Interface Adapter board and the Arduino UNO. T2 connects to +5V, T1 connects to Analog Channel A0, and the unlabeled pin connects to GND.

The thermistor readout circuit is a voltage divider. +5V and GND are supplied externally from the Arduino UNO. A resistance of 2kΩ was chosen for R1 so that the voltage output will remain mostly linear between the temperatures of 0 to 100ºC. As the resistance of the thermistor changes, the output of the circuit will vary. As temperature increases, the voltage will begin to decrease. The relationship between output voltage and temperature is shown in the graph below.

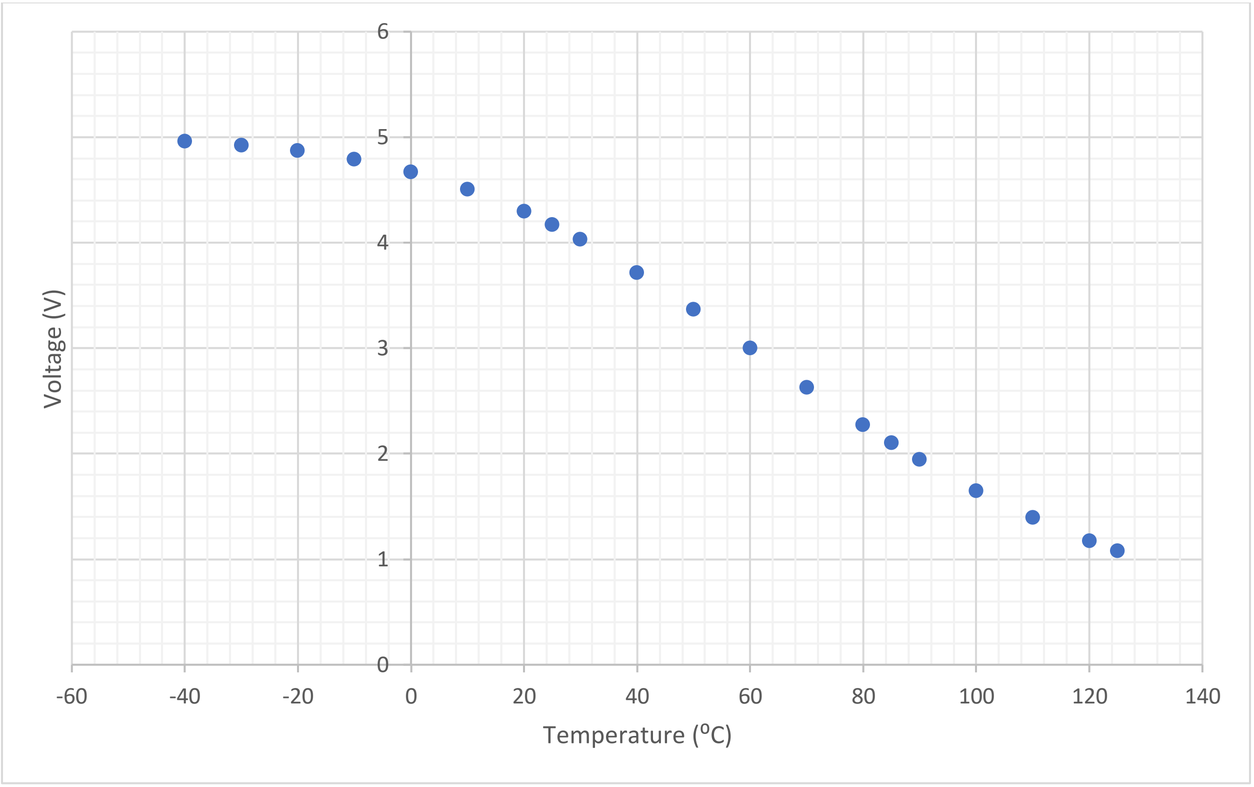


Figure 6 – Output Voltage vs Temperature

Electronic Hardware – Arduino UNO

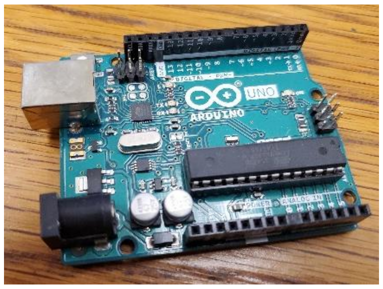


Figure 7 – An Arduino UNO

The Arduino UNO is a commercial microprocessor development board that utilizes an ATMEGA328P microcontroller. It provides interfaces to power, digital I/Os, and analog inputs to an ADC. Inputs and outputs are accessed through stackable header pins.

Hookup wire is used to connect the Interface Adapter Board to the Arduino. +5 VDC and GND are provided to the adapter board through this interface. The output of the temperature monitor is connected to the ADC inputs at A0 or A1.

Software is uploaded to the Arduino UNO through a USB cable. The program is coded using the Arduino IDE. Uploads to the Arduino UNO are performed in the IDE.

Software Systems – Arduino Code

The Arduino UNO runs a software program that samples the voltage on the ADC channels, converts that value into a temperature value, and outputs that value to the USB serial port.

The program begins by initializing the ADC channels and opening the USB serial port at 9600 baud. It then initializes the variables used to store the ADC values, the timers, and the time elapsed. A summary of the variable initialized and their purpose in the code is shown in the table below.

Table 1 – Variable Names and Purposes

Variable Name	Type	Purpose
temp1	Float	Holds the value of the temperature sensor sampled on channel 0
temp2	Float	Holds the value of the temperature sensor sampled on channel 1
timer1	Unsigned Long	Holds the current value of millis(). Used to calculate timeElapsed
timer2	Unsigned Long	Holds the past value of millis(). Used to calculate timeElapsed
timeElapsed	Float	Used to determine if 1 second has passed. Calculated by subtracting timer2 from timer1.

After initialization, the code begins an infinite while loop. In this loop, the timer is updated every loop by calling the millis() command. The millis() command returns the amount of milliseconds passed since the Arduino was powered on. The value of timeElapsed is calculated by subtracting the current time stored in the variable timer1 from the past time stored in the variable timer2. The code then checks if timeElapsed is greater than 1000. If not, the timers are updated and a new timeElapsed is calculated.

If timeElapsed is greater than 1000, the data capture process begins. The voltages at the ADC channels are read and converted into a 10-bit value. The value is then converted to degrees Celsius by using a linear calibration equation. The values are then output onto the USB serial port. Timer2 is then set to the value of timer1, the value of timer1 is set to the current time, and a new timeElapsed is calculated. The loop then begins again.

Data Format

The data is output on the Arduino USB serial port. Two values are sent through this channel – the value of the first thermistor and the value of the second. They are delimited by spaces. RealTerm applies a timestamp to the data record when the value was received by the program. These values are saved to as ASCII characters to a .TXT file.

Table 2 – Data record Format

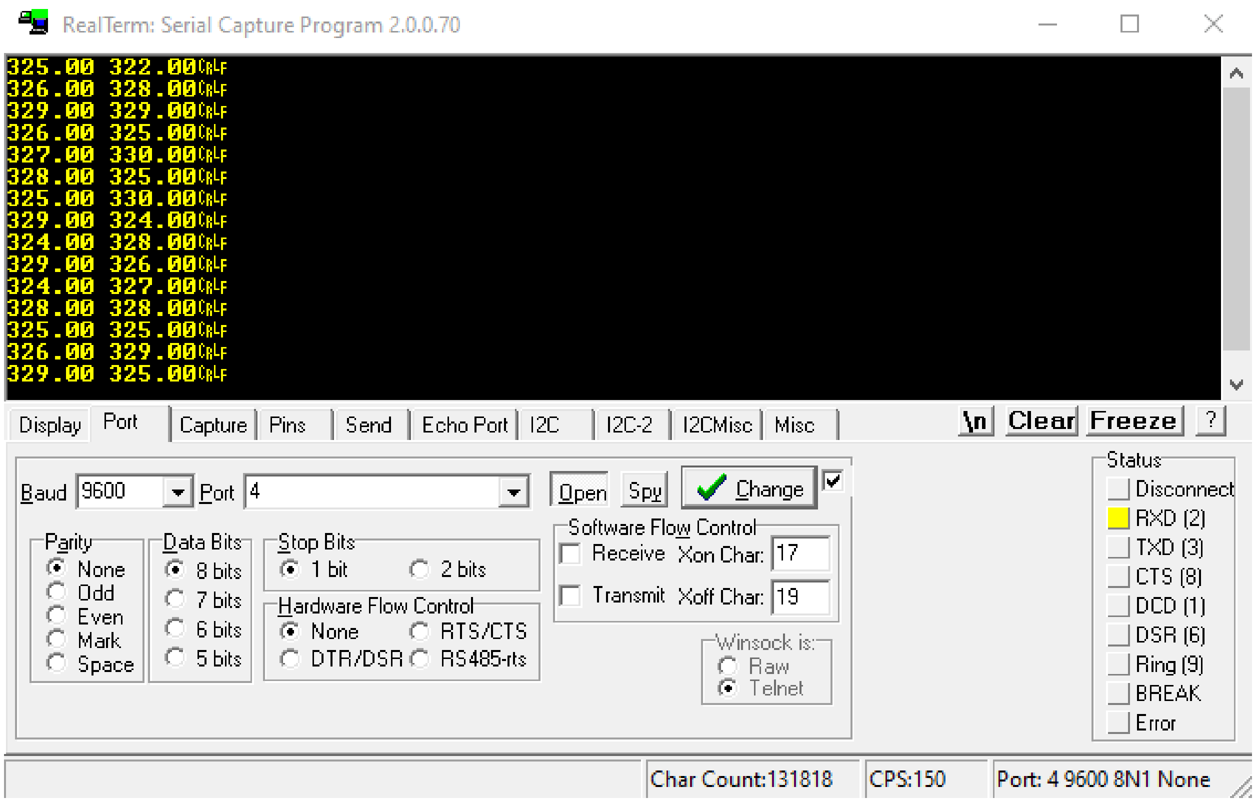
Timestamp	Delimiter	Temperature Value 1	Delimiter	Temperature Value 2
"1/28/2021 2:40:52 PM"	,	22.58	,	22.58

Software Systems – RealTerm

RealTerm is a free serial capture program that allows the user to directly view data passed on a serial port. The user can select which serial port to listen to and display the data on the screen. The program also allows for data capture. If a file path is specified, the program can overwrite or append the file with the data captured on the serial port. RealTerm can append a timestamp to each line saved to the data file. This timestamp utilizes the clock of the computer the program is run on.

To begin viewing serial data on a port, first select the “Port” tab. The data format and serial protocol must be known to successfully listen to the data stream. The baud rate must match the rate specified in the Arduino program. Additionally, the serial port the Arduino is connected to must be specified. RealTerm does not show ports that are not in use. Therefore, the Arduino must be plugged into an USB port before RealTerm is opened.

For this system, the serial data is sent at 9600 baud with no parity or hardware flow control. It has 1 stop bit and 8 data bits. Once all parameters have been set, hit the “Open” button. You should begin to see data fill the terminal window.



```
325.00 322.00CRLF
326.00 328.00CRLF
329.00 329.00CRLF
326.00 325.00CRLF
327.00 330.00CRLF
328.00 325.00CRLF
325.00 330.00CRLF
329.00 324.00CRLF
324.00 328.00CRLF
329.00 326.00CRLF
324.00 327.00CRLF
328.00 328.00CRLF
325.00 325.00CRLF
326.00 329.00CRLF
329.00 325.00CRLF
```

Baud 9600 Port 4

Parity: [✓ None] Odd Even Mark Space

Data Bits: [✓ 8 bits] 7 bits 6 bits 5 bits

Stop Bits: [✓ 1 bit] 2 bits

Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR RS485-rts

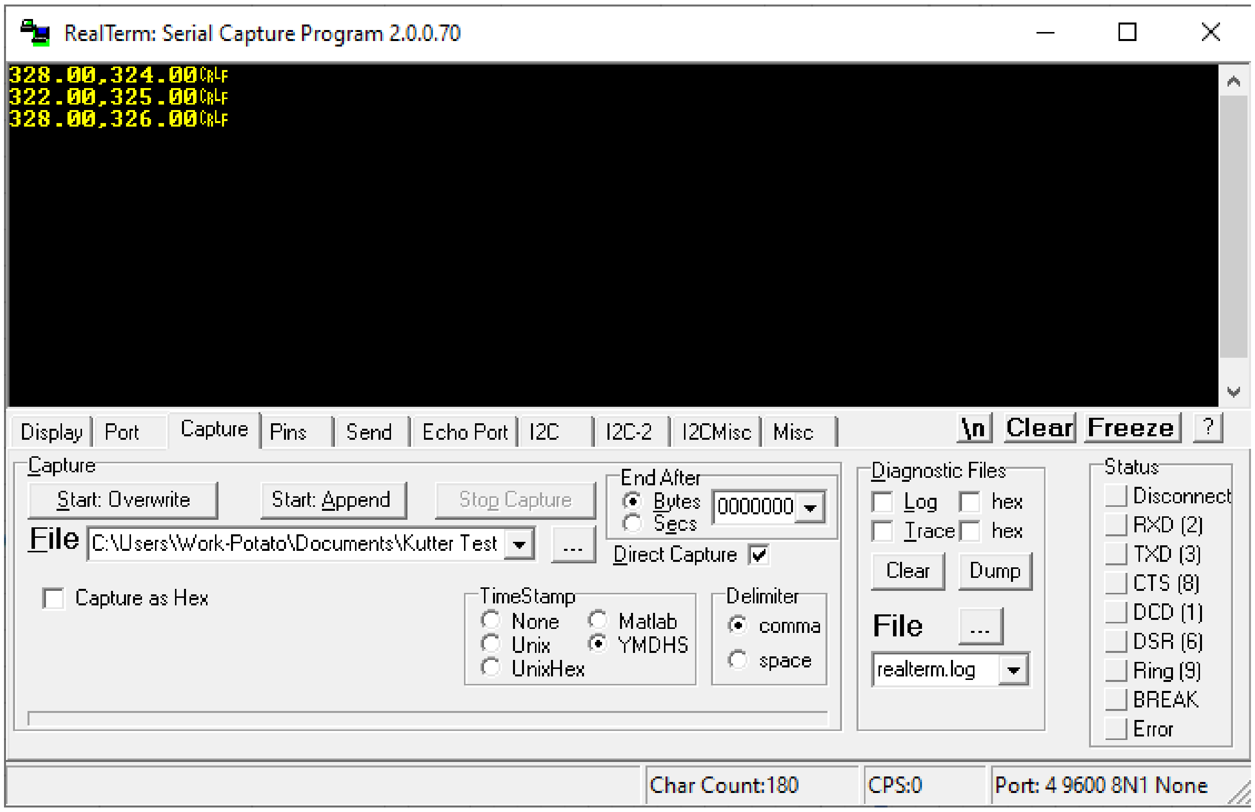
Software Flow Control: Receive Xon Char: 17 Transmit Xoff Char: 19

Winsock is: Raw [✓ Telnet]

Char Count:131818 CPS:150 Port: 4 9600 8N1 None

Figure 8 – RealTerm “Port” terminal window

Once the port has been opened and data is being displayed in the terminal window, data capture can begin. First save an empty .TXT file in the path folder you wish the data to be saved in. Then open the “Capture” tab. Specify the file path to the .TXT file you just created in the “File” text box by either manually typing the file path or selecting the “…” button and selecting the file path from the file explorer. Then select “YMDHS” button on the Timestamp. This will append the computer’s local time to the data every time a serial line is saved.


RealTerm: Serial Capture Program 2.0.0.70

```
328.00,324.00CRLF
322.00,325.00CRLF
328.00,326.00CRLF
```

File: C:\Users\Work-Potato\Documents\Kutter Test

End After: [✓ Bytes] Secs 0000000

Direct Capture ✓

TimeStamp: None Matlab Unix [✓ YMDHS] UnixHex

Delimiter: [✓ comma] space

Diagnostic Files File: realterm.log

Char Count:180 CPS:0 Port: 4 9600 8N1 None

Figure 9 – RealTerm "Capture" terminal window

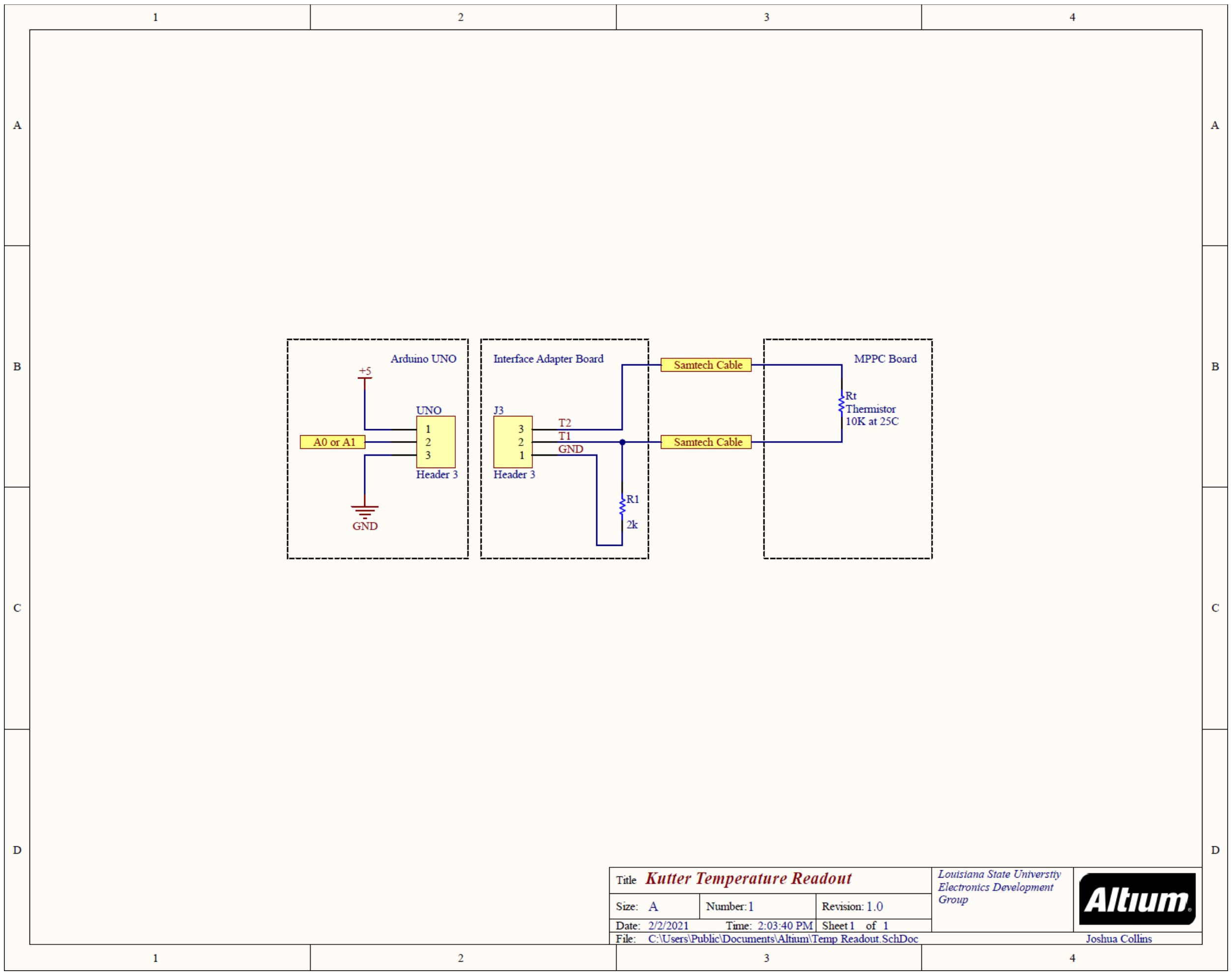
There are two option for saving data. "Start:Overwrite" will delete all data in the specified file path and begin writing data to that file. "Start:Append" will open the file and begin adding lines of data to the end of the file. While RealTerm is operating in either Overwrite or Append mode, no data will be updated on the terminal window.

Software Systems – Arduino IDE

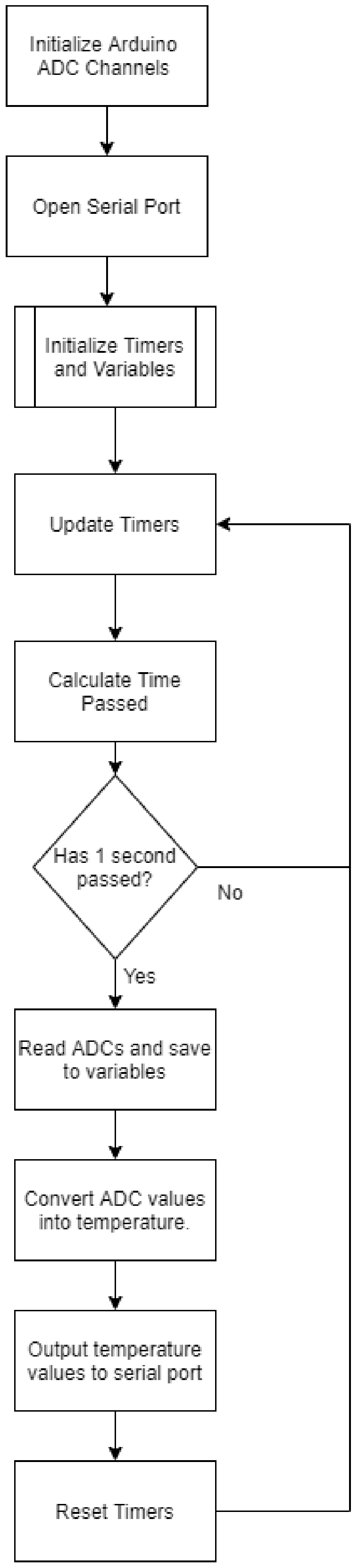
The Arduino IDE is a free integrated development environment used to program Arduino products. The program allows for the creation and modification of code. The Arduino runs on modified C programming. Tutorials on using the Arduino language are provided online.

To upload code using the Arduino IDE, the user must use the dropdown Tools menu to specify that an Arduino UNO is used and specify which serial port the Arduino is plugged into. The program will not upload to the Arduino if the wrong Arduino product or serial port is selected. Once the serial port is selected, the code can be uploaded. Code cannot be uploaded if RealTerm is viewing or saving data from the serial port the Arduino is plugged into.

Appendix A – Temperature Readout Schematic



Appendix B – Arduino Software Flowchart


Initialize Arduino ADC Channels
Open Serial Port
Initialize Timers and Variables
Update Timers
Calculate Time Passed
Has 1 second passed?
No
Yes
Read ADCs and save to variables
Convert ADC values into temperature.
Output temperature values to serial port
Reset Timers

Appendix C – Calibration Table

Temperature (ºC)	Thermistor Resistance (Ω)	Output Voltage (V)	ADC Value
10*	**18250**	**4.51**	**923**
11	17291	4.48	918
12	16600	4.46	914
13	15940	4.44	910
14	15308	4.42	906
15	14703	4.40	901
16	14123	4.38	897
17	13569	4.36	892
18	13038	4.34	888
19	12530	4.31	883
20*	**12140**	**4.29**	**879**
21	11576	4.26	873
22	11129	4.24	868
23	10701	4.21	863
24	10291	4.19	857
25*	**10000**	**4.17**	**853**
26	9520	4.13	846
27	9159	4.10	840
28	8812	4.08	835
29	8480	4.05	829
30*	**8283**	**4.03**	**825**
31	7855	3.99	816
32	7562	3.95	810
33	7280	3.92	803
34	7010	3.89	797
35	6750	3.86	790

*Highlighted values are gathered from the thermistor data sheet. Other values were calculated using a power fit equation.