Automated Test Device for Temperature Sensor

DESIGN DOCUMENT

Team 4
Client: Dana Conrad
Advisor: Dr. Neihart
Members: Tony Haberkorn, Justin Garden, Michael Hurley, Samuel Estrada
Email: sddec24-04@iastate.edu
Website: https://sddec24-04.sd.ece.iastate.edu/#

Executive Summary

Sukup is developing a device to measure the temperature inside its grain dryers using resistive temperature devices (RTDs); they need a way to test the functionality of their device. The testing solution for Sukup's device must be able to test the accuracy of the chip used to calculate the temperature from the RTDs as well as test for various fault conditions.

With this solution, Sukup can ensure only fully functional and accurate devices make it to field use. If a faulty device were to be sent for use in the field, it could potentially lead to inaccurate measurements at critical temperatures. These critical temperatures could allow the grain dryer to overheat and potentially combust. This testing device could also be used to improve accuracy and testcases through its development.

In our approach, we decided to create a custom PCB that can simulate a range of voltages that corresponds to different temperatures and DAC codes. We used a series of onboard switches to change the testcases being performed to test open, short, and simulated temperatures. Our code starts with a python script that takes in users' inputs of device connected, desired testcase, and finally desired temperature simulation. The python script then transmits the corresponding data where we have our MCU code written in C to set switches and DAC to the specified case.

Our design addresses the clients need to test these fault conditions by providing the adequate stimuli of a short and open condition across the RTD lines. We can confirm this through our testing of the PCB outputs across these terminals to see an open and short circuit. The next steps to be taken in our project is to review the PCB design regarding the DAC to ensure that it is fully functional. We also would work through creating a python script that sweeps through all testcases in a more streamlined manner to open the user base to users with less experience with the technology.

Learning Summary

Development Standards & Practices Used

Practices utilized by all team members throughout this project include schematic design and PCB layout design. We also used practices regarding Python coding and C coding. The practice of teamwork to decide the best route to take, especially component selection, was a major part of this project. Communication with the client also had a strong hold throughout the project.

The main engineering standard we used was the IEEE code of ethics. Being fully honest with the client was a huge part. At the start of the project, we did not fully understand the entire scope of the project, and had many questions. We then talked extensively with our client to understand their needs, rather than get to work immediately and act like we knew what we were doing. Financial responsibility was also a big part. We received a rather large budget for this project, but instead of going out and spending all the money and utilizing the full budget, we kept our costs to only what we absolutely needed.

Summary of Requirements

Functional:

- Measure accuracy of RTD measurement chip (MAX31865)
- Simulate controllable voltage
- Test fault conditions
- Send results to computer UI
- Be able to run single or sweep of tests
- Automatic switching between tests

Physical:

- Compatible with standard USB to PC connection
- Desktop sized

UI:

- Display current test
- Display testing results in user friendly way

Applicable Courses from Iowa State University Curriculum

- EE 285 – Problem Solving Methods and Tools for Electrical Engineering
  - This course was our first course using C programming and was fundamental to our understanding of coding
- EE 230 – Electronic Circuits and Systems
  - This course taught us to troubleshoot circuits and how to use equipment like the oscilloscope.
- EE 333 – Electronic systems design
  - This course was utilized as it gave us an understanding of schematic and PCB design
- CPRE - 288 Embedded Systems
  - CPRE 288 gave insight into coding microcontrollers in C

New Skills/Knowledge acquired that was not taught in courses

We all learned many new skills throughout this project including the python language, additional knowledge in C language, troubleshooting methods, component selection and PCB design. We also got a lot of experience with reading datasheets, and how to find specific information on various components. We also gained critical knowledge of how to look through a component vendor's website, such as DigiKey, and find the component that best suits our needs. Hand soldering components was also learned and improved upon when using small outline components.

Table of Contents

1. Introduction 6

1.1. Problem Statement 6

1.2. Intended Users 7

2. Requirements, Constraints, And Standards 7

2.1. Requirements & Constraints 7

2.2. Engineering Standards 8

3 Project Plan 8

3.1 Project Management/Tracking Procedures 8
3.2 Task Decomposition 8
3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria 8
3.4 Project Timeline/Schedule 9
3.5 Risks and Risk Management/Mitigation 10
3.6 Personnel Effort Requirements 10
3.7 Other Resource Requirements 12
4 Design 13
4.1 Design Context 13
4.1.1 Broader Context 13
4.1.2 Prior Work/Solutions 13
4.1.3 Technical Complexity 14
4.2 Design Exploration 14
4.2.1 Design Decisions 14
4.2.2 Ideation 14
4.2.3 Decision-Making and Trade-Off 15
4.3 Final Design 15
4.3.1 Overview 15
4.3.2 Detailed Design and Visual(s) 16
4.3.3 Functionality 26
4.3.4 Areas of Challenge 26
4.4 Technology Considerations 27
5 Testing 28
5.1 Unit Testing 28
5.2 Interface Testing 32
5.3 Integration Testing 32
5.4 System Testing 33
5.5 Regression Testing 33
5.6 Acceptance Testing 33
5.7 User Testing 34
5.9 Results 34
6 Implementation 34

6.1 Design Analysis 34
7 Professional Responsibility 35
7.1 Areas of Responsibility 35
7.2 Project Specific Professional Responsibility Areas 36
7.3 Most Applicable Professional Responsibility Area 36
8 Conclusions 37
8.1 Summary of Progress 37
8.2 Value Provided 37
8.3 Next Steps 37
9 References 37
10 Appendices 39
Appendix 1 – Operation Manual 39
Appendix 2 – Code 41
Appendix 3 - Team 43

List of figures/tables/symbols/definitions

Figure 1 Gannt Chart (Page 12)

Figure 2 High Level Block Diagram (Page 19)

Figure 3 Host PC Python Script (Page 20)

Figure 4 User interface in command terminal running a test (Page 21)

Figure 5 MCU High Level Block Diagram (Page 22)

Figure 6 DriverLib Testcase Example (Page 23)

Figure 7 KiCAD Schematic (Page 24)

Figure 8 PCB Layout (Page 25)

Figure 9 MAX chip equivalent circuit (Page 26)

Figure 10 RTD resistance vs temperature (Page 26)

Figure 11 SPI Initialization (Page 27)

Figure 12 DAC Clear/Initialization (Page 28)

Figure 13 Code for running temperature test (Page 29)

Figure 14 UART Communication initialization (Page 33)
Figure 15 MCU Test code Example (Page 34)
Figure 16 Data transmission with SPI (Page 35)
Figure 17 Data transmission for 70-degree test case (Page 36)
Figure 18 Connection terminals on test kit board (Page 45)
Figure 19 Snippet of Host PC Python script (Page 47)

Figure 20 Example of running the python script (Page 47)

1. Introduction

1.1. Problem Statement

Sukup is developing a device to measure the temperature inside its grain dryers using resistive temperature devices (RTDs); they need a way to test the functionality of their device. The testing solution for Sukup's device must be able to test the accuracy of the chip used to calculate the temperature from the RTDs. The solution must also test fault conditions of the device, such as over/under voltage, open/short circuit conditions, as well as confirm the board communicates with the outside world correctly through Modbus protocol.

With this solution, Sukup can ensure only fully functional and accurate devices make it to field use. If a faulty device were to be sent for use in the field, it could potentially lead to inaccurate measurements at critical temperatures. These critical temperatures could allow the grain dryer to overheat and potentially combust. This testing device could also be used to improve accuracy and testcases through its development.

The largest issue during development is the test devices own accuracy. If this accuracy is not accounted for, temperature devices that are not suitable to be installed into a dryer could pass all the tests, resulting in a faulty temperature device in the field. The accuracy of the test device will be accounted for by considering the tolerance the components used during its design. The test device will show a failed test if its own error carries outside the acceptable bounds for a given test.

1.2. Intended Users

● Sukup Electrical Engineer

○ The Sukup Electrical Engineer is responsible for designing, programming, and creation of PCB for RTD testing. The skills the electrical engineer has are well rounded in circuit design, PCB best practices, and power electronics. The user needs to have a way to test their RTD designs quickly and effectively. The electrical engineer will benefit from the design by being able to increase production rates and verify designs quickly.

● Technician

○ The Technician is involved with assembly and quality assurance processes of Sukup's RTD circuit boards. They deal with building and testing circuitry post-construction; the technician plays a role in ensuring that the boards are tested thoroughly and pass inspection before being implemented in the field.

2. Requirements, Constraints, And Standards

2.1. Requirements & Constraints

Functional:

● Measurement Accuracy: Verify the accuracy of the measurement chip (MAX31865).

● RTD Simulation: Simulate RTD resistance to test different scenarios.

● Fault Condition Testing: Test fault conditions of short and open circuits.

● RS485 Compatibility: Ensure compatibility with RS485 communication.

● Modbus Communication Testing: Ensure testing of Modbus communication protocols.

● Data Transmission: Send test results to a computer for analysis.

● Test Flexibility: Allow the device to run specific tests individually or execute a comprehensive sweep of all tests.

Physical:

● Power Supply: The device should be compatible with standard wall outlets.

● USB Connectivity: Include a standard USB-A connector for data transfer and programming.

● Form Factor: Design should have the device be desktop sized.

UI:

● Computer Interface: Display current test run and results associated with tests performed

● Use Command Terminal to execute testcases

2.2. Engineering Standards

● Standard Reference Designations for Electrical and Electronics Parts and Equipment (IEEE Std 200-1978)

○ Standard used for best practices in designating parts used for our design.

● Standard For Test Code for Resistance Measurement (IEEE Std 118-1978)

○ Standard used as a reference in our coding of resistance measurements

3 Project Plan

3.1 Project Management/Tracking Procedures

The project management style we have chosen is agile. A major reason for this is because as we have learned more about our client's wants and needs from the project, our requirements and design choices have changed.

An agile management style also allows us to focus on short term sprint goals that will allow us to better stay on track for finishing the project on time. A downside to this is that we don't have many hard deadlines which could allow us to become distracted and put too much focus on an area that doesn't require it. We can try to mitigate this by creating a Gannt chart that can be adjusted to keep sight of overall goals in the project.

3.2 Task Decomposition

In our task decomposition we decided to adopt an agile approach with weekly goals and trackable. We plan to meet every week to determine the progress made from each team member based on their last week's goals. In this we found it best to break our project into four phases project recap, project design, project construction and validation, and finally coding and gathering results.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

● Design a power supply for our test device

○ Our device will run using standard 5V given from our USB-A connection

● Effectively communicate with Sukup device

○ Our microprocessor will communicate with Sukup's board over the UART protocol to set testcases effectively

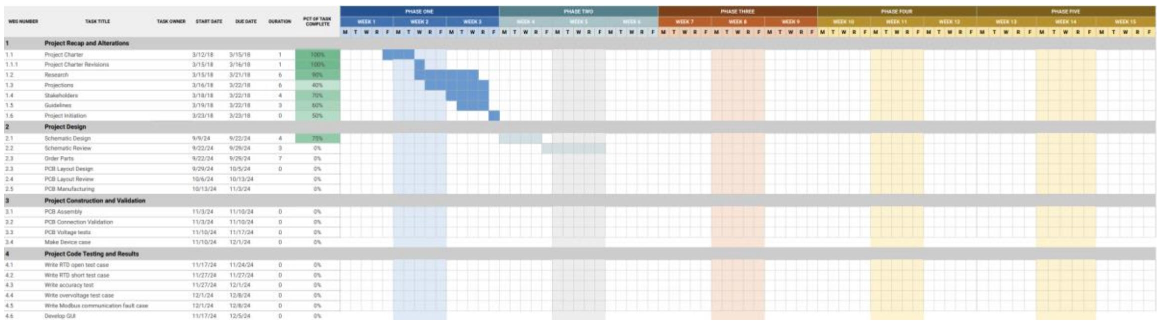
- Produce accurate voltage to simulate RTD resistance
  - Using a DAC to produce accurate voltages to meet resolution requirements defined by the client of within 5 degrees Fahrenheit
- Simulate/detect fault conditions
  - Force fault conditions to be met on Sukup's device to ensure the correct error is read by Sukup microprocessor.
- Display results
  - Display results of all tests through the command terminal received from Sukup's device
  - Process information to define if there are fault conditions seen

3.4 Project Timeline/Schedule

In our task decomposition we decided to adopt an agile approach with weekly goals and trackable progress through the use of a Gannt chart. This chart helps us focus on each team members goals and keep track of where we would like to be in our projects progress going into the following week. The Gantt Chart can be seen in figure 1. In this we break the project into four phases, project recap, project design, project construction and validation, and finally coding and gathering results.

Our project timeline is currently a bit flexible as discussed with our advisor. We begin with researching and understanding the board that was given to us by Sukup. This phase will take the majority of our first semester as it is critical in constructing a PCB that meets all the requirements set for us. We start by looking at the schematics and code shared so we can fully understand the data we are testing. From there we move onto designing our own PCB that we hope to have done by the end of the semester. This phase includes schematic creation, peer reviews, revisions, and PCB layout.

For each of these steps our team will meet with our advisor to make sure that our board is constructed properly and ready for fabrication. Once the board is fabricated, we can focus the majority of next semester on testing and revisions. This will help give us plenty of time to test our board against the Sukup board as well as make any revisions to the schematics based on our client's response.



WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
1	**Project Recap and Alterations**					
1.1	Project Charter		3/12/18	3/15/18	1	100%
1.1.1	Project Charter Revisions		3/15/18	3/16/18	1	100%
1.2	Research		3/15/18	3/21/18	6	90%
1.3	Projections		3/16/18	3/22/18	6	40%
1.4	Stakeholders		3/18/18	3/22/18	4	70%
1.5	Guidelines		3/19/18	3/22/18	3	60%
1.6	Project Initiation		3/23/18	3/23/18	0	50%
2	**Project Design**					
2.1	Schematic Design		9/9/24	9/22/24	4	75%
2.2	Schematic Review		9/22/24	9/29/24	3	0%
2.3	Order Parts		9/22/24	9/29/24	7	0%
2.3	PCB Layout Design		9/29/24	10/5/24	0	0%
2.4	PCB Layout Review		10/6/24	10/13/24		0%
2.5	PCB Manufacturing		10/13/24	11/3/24		0%
3	**Project Construction and Validation**					
3.1	PCB Assembly		11/3/24	11/10/24	0	0%
3.2	PCB Connection Validation		11/3/24	11/10/24	0	0%
3.3	PCB Voltage tests		11/10/24	11/17/24	0	0%
3.4	Make Device case		11/10/24	12/1/24	0	0%
4	**Project Code Testing and Results**					
4.1	Write RTD open test case		11/17/24	11/24/24	0	0%
4.2	Write RTD short test case		11/27/24	11/27/24	0	0%
4.3	Write accuracy test		11/27/24	12/1/24	0	0%
4.4	Write overvoltage test case		12/1/24	12/8/24	0	0%
4.5	Write Modbus communication fault case		12/1/24	12/8/24	0	0%
4.6	Develop GUI		11/17/24	12/5/24	0	0%

Figure 1 Gannt Chart

3.5 Risks and Risk Management/Mitigation

There can be a few risks seen when creating a PCB such as delays in fabrication. Each fabricator has a different fabrication time that we will be waiting on during our semester. We hope to help mitigate this by using this time to code on a microcontroller that will be the same as the one on our PCB. Overall, the biggest risk to our project is time, as there will be periods where we will not have control over how fast our board is being produced. Once we fabricated our board, we found that it would be most useful to write the microcontrollers code during this time as to keep progress during this down time.

Another risk we found that was not anticipated was the complexity of the microcontroller being used to initialize SPI and UART communication. Much of the team had little experience with the MSP430 and was more involved with Arduino's platform. We tried to mitigate this by meeting with our advisor to help understand the microcontrollers libraries and functionality.

3.6 Personnel Effort Requirements

This project will require a significant amount of effort from all members of the group. When creating any PCB design, it requires various people looking and making comments on schematic choices. By having peer reviews, we can answer questions and have a collaborative effort on our design choices. There is plenty of new learning and skills to be developed as all members of our group are EE majors with not much coding experience. There is a significant portion of the project that will require us to learn and develop in python and C.

Team Member	Anticipated Effort	Actual Effort
Samuel Estrada	I anticipate the effort requirement for the project will focus most heavily on the MSP430 microcontroller and its associated datasheets. I will most likely be putting in effort on learning to code in C and the required libraries for our project. I also anticipate some work in python to understand the requirements that are being sent over through our terminal script. In addition to coding skills, I expect to be working	In actuality, the project required much more coding skill development than that was previously anticipated. I was working most heavily with the microcontroller to create code that receives data from the host PC over UART and then passes it along to different functions within our script. It was required that I learn how to use the MSP430 Driver library in order to most effectively program our GPIO pins to switch between the cases. I did a bit of work with

	with KiCad in order to create a viable PCB for our project.	PCB design reviews but I was most focused on coding for this experience.
Tony Haberkorn	Looking at this project, I feel as though it will be very software intensive. It will have hardware as well, but have much more software. I anticipate that I will have to learn a lot of programming skills to do well in this project. I expect to gain skills in both python and C languages. On the hardware side, I also expect to learn about PCB design, as we are sure to need to create a PCB. At face value, the project description does not seem to be the most complex and intensive project, but I expect that to change and many issues will arise along the way during the completion of this project.	This project was a lot harder than I originally anticipated it to be. There is a lot more to it than meets the eye. On the hardware side, it took a lot of discussion on what the best method for simulating the RTD would be, and there were more components involved in doing that than I originally thought we would need. On the software side, I realized that I had forgotten much about C coding, and would have to essentially learn it all over again. As for python, I did not know the python language at all leading in to this project and did not have any experience with it. Learning python however, was not as hard as learning C for the first time however, because I already knew the basic fundamental concepts of programming.
Justin Garden	I am working mostly on PCB design and the communication between our MSP430 microcontroller and the DAC. While this is a new microcontroller, I will have to become familiar with, I do not anticipate this task to be too difficult or take a lot of time. The bulk of my time will be spent writing and testing the code to output the desired voltage from the DAC. I anticipate spending about 4 hours per week developing the code to output the correct voltage from the DAC, and another hour helping with PCB design.	In the end, I worked much more with the microcontroller than I thought I would have to. I ran into issues when initializing the SPI transmission from the MCU. The fixes were not very difficult to implement, but because I was not familiar with the MCU it took much longer to find what the real issues were than I had anticipated before. Testing also took more effort than I had originally anticipated. Some of the sources I was using to help me write the code seemed to be out dated or for a MCU that was in the same family but used a slightly different syntax. Because of this I had to spend much more time debugging

		and troubleshooting than I thought I would have to for a relatively simple block of code.
Michael Hurly	I anticipate mainly focusing on developing the PCB and helping to troubleshoot and build the PCB's. Most of my work will need to be done looking through the MCU and peripheral components to ensure everything works correctly together and we can get the desired functionality out of the pins we select. I think the most important skills will be working in circuit design CAD software and troubleshooting the board efficiently while looking at code and datasheets.	Overall, much more time was spent troubleshooting the board while consulting the datasheets than I expected due to some design errors on the physical board. The problems took some time to find but the fixes were easy to implement. Because of the overall design though getting good readings on some of the pins was more difficult than necessary and more test points should. Have been added.

3.7 Other Resource Requirements

Other resources that were required for project completion were:

- Code Composer Studio
- MSP430 Driver Library Import
- Python IDE and associated libraries
- ETG testing equipment
- KiCad PCB schematic and layout development

4 Design

4.1 Design Context

4.1.1 Broader Context

Looking at the broader context of our design we can see the environmental and economic impact of our design choices. It can be a bit difficult to see the broader context for our design as it is a one-off board created to test a variety of other boards. With this design however, it would allow for Sukup to test their boards and make sure they are sending out fully functional devices.

These devices could impact grain ignition that could cause safety hazards due to the fire produced. We also will see a financial impact as there would no longer be a need for technicians to test these devices so often. The test cases created would solve these issues and return information specific to that board, reducing man hours needed during production.

4.1.2 Prior Work/Solutions

Looking at the prior work done on our project we noticed that the design that was created for most of the schematics had not been analyzed or questioned in quite some time. The schematic was created by a contractor that did not document the design choices and was handed off multiple times. There were a few questions from our contact at Sukup that we decided to take on. There were components that were unsure of the impact on measurements that were tested and verified. Looking at the other competitors on the market we were able to identify three major competitors in the market. These competitors all offer methods in testing equipment that our contact at Sukup could use. After listing out the benefits and drawbacks, we believe that we offer a level of flexibility that they do not.

	Modbus	Applicos	Labjack	Our Board
Pros	Developers of the software that we are testing Allows users to define their own series of tests	Has tests and test equipment developed Software is well developed and has resources	Free GUI Devices that already test wanted information	Specially designed for a specific application Can offer the flexibility needed by our client
Cons	Only a software simulation Not available in hardware to test individual boards	Mostly focuses on mixed signal applications Less flexibility in tests that can be ran Software code base not available for editing Does not test Modbus communication protocol	No support for desired communication type	Won't have the UI sophistication that our competitors have Test cases will be a bit simpler than those on offer from others

4.1.3 Technical Complexity

The complexity of the design showcases the expertise that was developed across Iowa State's Curriculum. Design requires knowledge of power supplies, usage of microcontrollers, and the functionality of different voltage monitors. We are also using code to help run test cases as well as return information about our system. Device interconnections must communicate with each other using different communication protocols. Knowledge of PCB design and common practices used to produce a functioning board are also required.

4.2 Design Exploration

4.2.1 Design Decisions

There have been various design choices that had to be made in order to successfully accomplish our target goal. Our first decision was the method we wanted to communicate with the board. We explored a few ideas in communicating such as looking at devices similar to our competitors. We however came to the conclusion of communicating through a daughter board that could be plugged into multiple boards under tests. This custom PCB would allow us to have more flexibility and control over the test we are able to run. With our PCB created we felt that board could easily be replicated or transferred over for other testing methods that the user may want to perform. The success of this board allowed us to test different fault conditions by probing our switches output and not solely relying on test code.

The next decision to be made was the approach we wanted to take in order to simulate the RTDs resistance. The RTDs resistance needed to be simulated for a testing range in order to verify that the board under test was behaving correctly. The decision was to try and simulate a voltage as the MAX chip takes a voltage reference that is used to perform its measurements. The success of this design decision allowed us to perform different temperature cases and map voltages to a specific temperature through a look up table.

The third choice we will have to make is the DACs we will want to use for simulating a voltage to the MAX chip. There is a wide range of DACs available ranging from different bit resolutions that will be crucial in accurately simulating a given voltage. We have selected a DAC that has a 16-bit resolution as this allows our step size to be small enough to differentiate between multiple temperatures that the RTD responds with. The success of this design choice was crucial in giving accurate measurement for the test cases as the threshold between voltages is in a small enough range to warrant the resolution.

4.2.2 Ideation

The design decision that we spent the most time on was the method in which we wanted to simulate the RTDs resistance. We went through an option of simulating a voltage directly with a power supply, analog potentiometers, digital potentiometer in parallel, 10-bit DAC, and a 16-bit DAC. Each of these choices had discussions over what would be the best method for testing. These options all provide a way to simulate the voltage seen in place of the RTD but have different temperature resolutions. After speaking with our advisor, we decided to use a 16-bit DAC that allows for high resolution and small temperature step sizes.

4.2.3 Decision-Making and Trade-Off

Each of the options had pros and cons to them. Starting with simulating the voltage directly we would be able to easily simulate the voltage we liked. This would be the easiest and quickest way to accomplish our goal. However, the con is that it would be the slowest way as you would need to connect all power supply cables for each test and physically change the power supply.

The next design choice we considered was using analog potentiometers. This would all allow us to change the resistance on the board to the values we want. It allows for a small resolution size as we can tune the resistance. The issue with this approach is that it is also quite slow. You would need to adjust the potentiometer each time you wanted to take a new measurement as well as take measurements of the potentiometer's resistance.

The next option was using a digital potentiometer in parallel. This method would allow us to quickly change the resistance of the potentiometer using code rather than physically tuning the potentiometer. The issue with this design is that the most TAPs we could find was 256 which is not enough resolution to support our test range. The change in resistances between TAPs was just too large to get an accurate sweep of voltages.

Our final choice was either using a 10-bit DAC or 16-bit DAC. These two offer the benefit of simulating a voltage using code so we can quickly change the voltage that is being output. The DAC allows for quick testing and accurate measurements. The issues we had to consider for the two is that we needed to check if the step sizes would be small enough for our desired resolution. The tradeoff between a 10 bit and 16-bit DAC would be the price and resolution. A 10-bit DAC would be cheaper but have a larger resolution while a 16-bit DAC would have a smaller resolution but be more expensive.

4.3 Final Design

4.3.1 Overview

In our final design we wanted to provide the ability for test cases to communicated from the host PC to our custom PCB. Our custom PCB would then be able to simulate a voltage to Sukup's board for testing. Along with this, Sukup's board would run communication through a RS-485 converter that gave details of the tests results. Much of the code to take on selecting test cases and voltage look up tables are handled by the host PC and then transmitted to the MCU to set our boards conditions.

We included the following sub systems:

- Host Personal Computer
    - Used to receive user commands and transmit data for various test cases
- Data Converter

○ Used to translate from USB to rs485 signals to receive test result data

- Power

○ Input power from the Launchpads USB port that can be used for sub-systems

- MCU

○ Used to set desired testcases by changing switch ports and transferring data to DAC unit

- On Board Power Regulation

○ Used to further step-down voltages to different desired levels and provide a constant and consistent reference voltage for DAC

- DAC

○ Used to drive simulated readings into Sukup's host board and verify temperature range is accurate

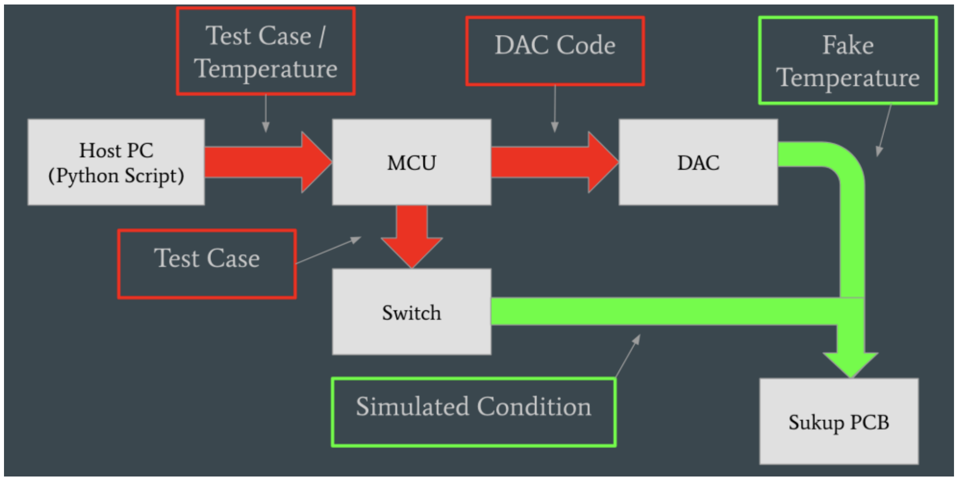
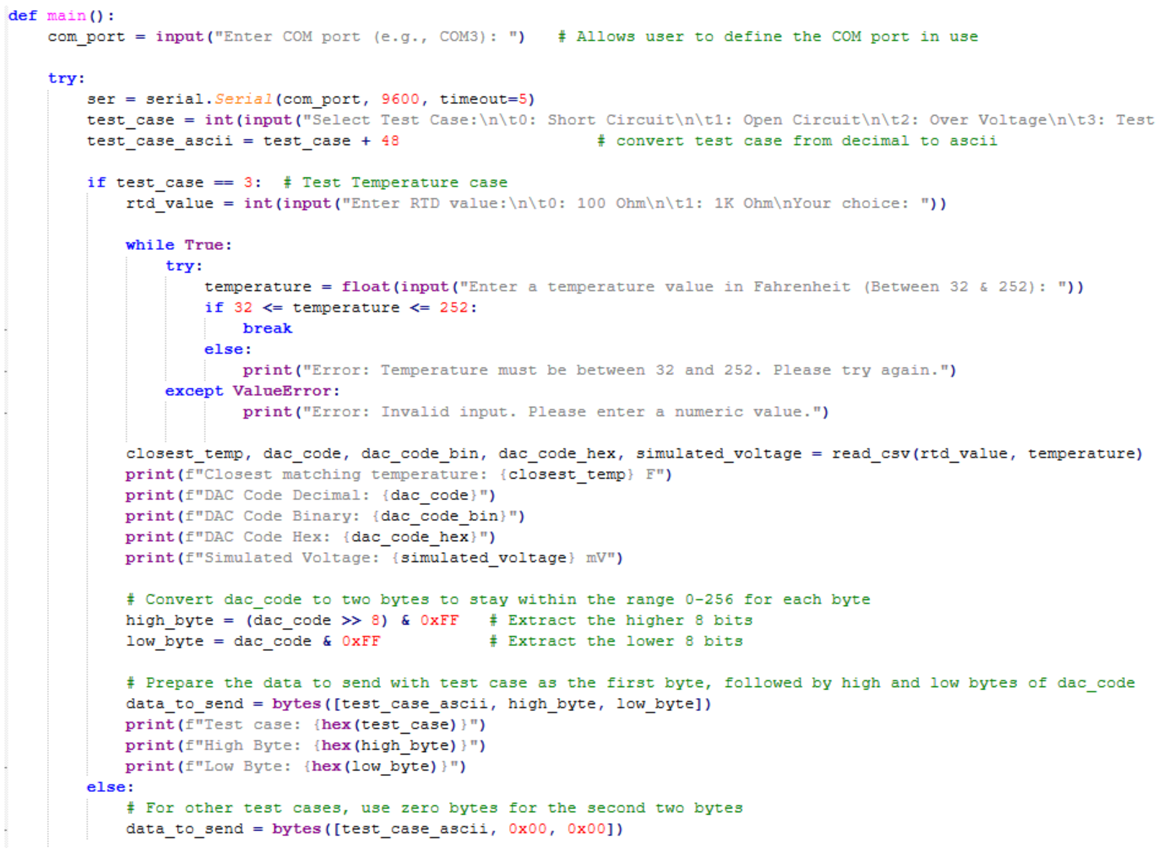


Figure 2 High Level Block Diagram

4.3.2 Detailed Design and Visual(s)

Working on the user interface between the host PC and the MSP430 was determined to be best implemented as a python code. Some of our team members did not know python before working on this project and only had a brief knowledge of C. The python code was determined to be the best because it is much easier to work with the COM ports in python, than it is in C. It is also easier to display information and create a simple user interface in the command terminal with python.

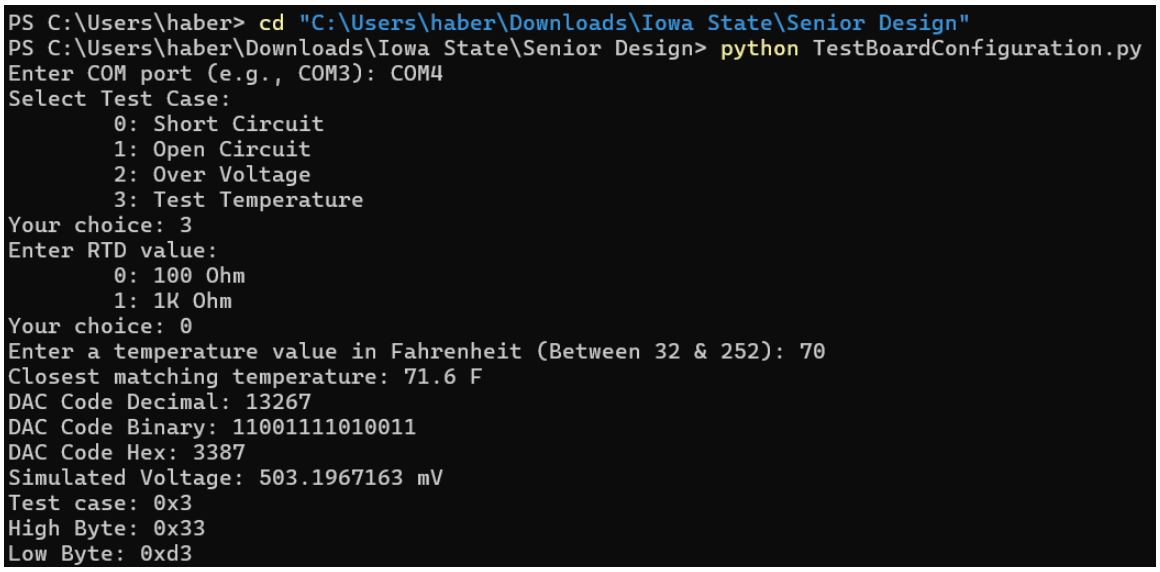
The code works by asking the user a series of questions such as the COM port in use, the test case that is desired, and the temperature desired. For specific temperature values, the code pulls in values from a CSV file, and populates the bytes to send. The communication between the host PC and the launchpad is 3 bytes. The first byte is the test case, and the second two are the 16-bit DAC code.



```python
def main():
    com_port = input("Enter COM port (e.g., COM3): ")   # Allows user to define the COM port in use

    try:
        ser = serial.Serial(com_port, 9600, timeout=5)
        test_case = int(input("Select Test Case:\n\t0: Short Circuit\n\t1: Open Circuit\n\t2: Over Voltage\n\t3: Test
        test_case_ascii = test_case + 48                        # convert test case from decimal to ascii

        if test_case == 3:  # Test Temperature case
            rtd_value = int(input("Enter RTD value:\n\t0: 100 Ohm\n\t1: 1K Ohm\nYour choice: "))

            while True:
                try:
                    temperature = float(input("Enter a temperature value in Fahrenheit (Between 32 & 252): "))
                    if 32 <= temperature <= 252:
                        break
                    else:
                        print("Error: Temperature must be between 32 and 252. Please try again.")
                except ValueError:
                        print("Error: Invalid input. Please enter a numeric value.")

            closest_temp, dac_code, dac_code_bin, dac_code_hex, simulated_voltage = read_csv(rtd_value, temperature)
            print(f"Closest matching temperature: {closest_temp} F")
            print(f"DAC Code Decimal: {dac_code}")
            print(f"DAC Code Binary: {dac_code_bin}")
            print(f"DAC Code Hex: {dac_code_hex}")
            print(f"Simulated Voltage: {simulated_voltage} mV")

            # Convert dac_code to two bytes to stay within the range 0-256 for each byte
            high_byte = (dac_code >> 8) & 0xFF    # Extract the higher 8 bits
            low_byte = dac_code & 0xFF            # Extract the lower 8 bits

            # Prepare the data to send with test case as the first byte, followed by high and low bytes of dac_code
            data_to_send = bytes([test_case_ascii, high_byte, low_byte])
            print(f"Test case: {hex(test_case)}")
            print(f"High Byte: {hex(high_byte)}")
            print(f"Low Byte: {hex(low_byte)}")
        else:
            # For other test cases, use zero bytes for the second two bytes
            data_to_send = bytes([test_case_ascii, 0x00, 0x00])
```

Figure 3 Host PC Python Script

After sending the data to the MCU, the python script will wait for an echo back from the MCU. When the echo returns, the python code with check it against the data that it sent. If the echo is the same as the data sent, then the communication was successful. If the echo does not match the data sent, or the script does not receive an echo within 5 seconds of sending the data, it will display an error message.



```
PS C:\Users\haber> cd "C:\Users\haber\Downloads\Iowa State\Senior Design"
PS C:\Users\haber\Downloads\Iowa State\Senior Design> python TestBoardConfiguration.py
Enter COM port (e.g., COM3): COM4
Select Test Case:
        0: Short Circuit
        1: Open Circuit
        2: Over Voltage
        3: Test Temperature
Your choice: 3
Enter RTD value:
        0: 100 Ohm
        1: 1K Ohm
Your choice: 0
Enter a temperature value in Fahrenheit (Between 32 & 252): 70
Closest matching temperature: 71.6 F
DAC Code Decimal: 13267
DAC Code Binary: 11001111010011
DAC Code Hex: 3387
Simulated Voltage: 503.1967163 mV
Test case: 0x3
High Byte: 0x33
Low Byte: 0xd3
```

Figure 4 User interface in command terminal running a test

In working with the MSP430, there was a requirement to learn C and its associated libraries. During our undergrad course work in electrical engineering, we had limited experience in working with C at a level higher than register. When working through this design it was found most beneficial to create a block diagram in breaking down the tasks needed to create a functional system. In this block diagram we break down our code into different states. The first state is to put out MCU into low power mode in which all the GPIO pins are set low in preparation for a command to be received.

When the user is prepared to send a command, we enter a wake state that prepares our MCU to set GPIO pins. The MCU then receives a byte from the host computer that corresponds to the test case the user wants to select. If the byte sent is 0 it will go into our short circuit testcase and set the associated GPIO pins. Similarly, if the byte is 1, we go into an open circuit test, if 2 we do an overvoltage testcase, and finally if 3 we set a temperature simulation. The temperature testcase has more associated setup with it, in this an additional setup there is a SPI initialization as well as DAC initialization.

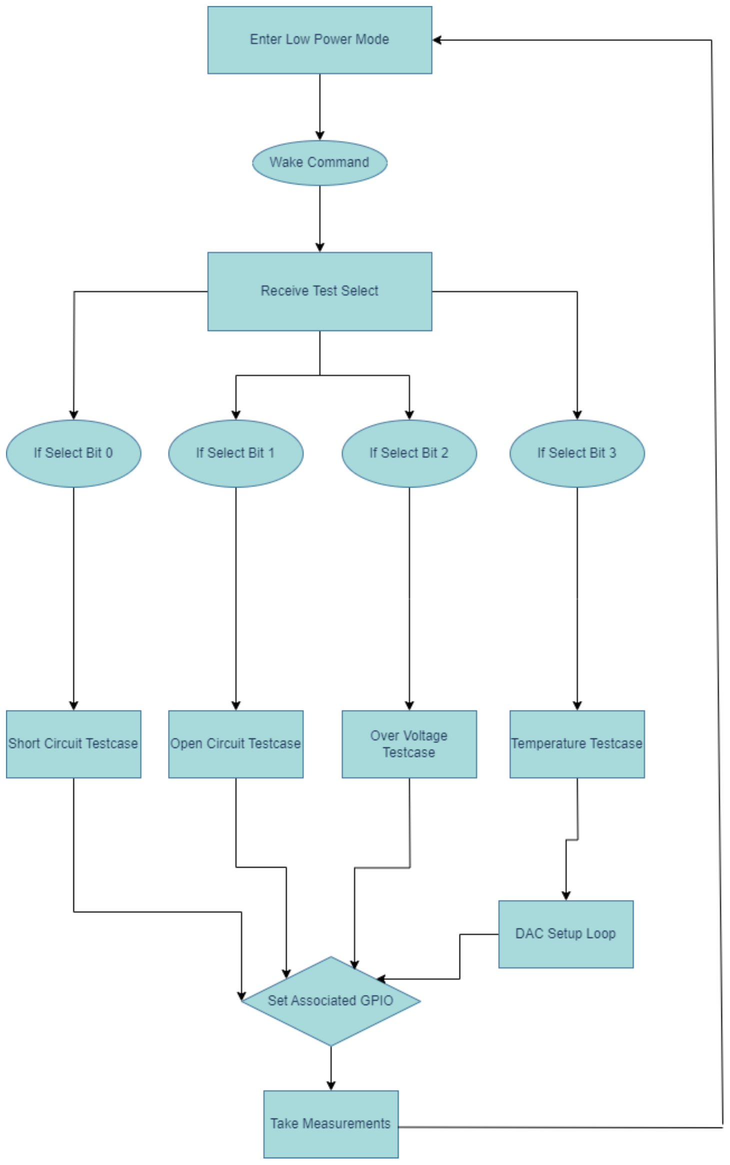
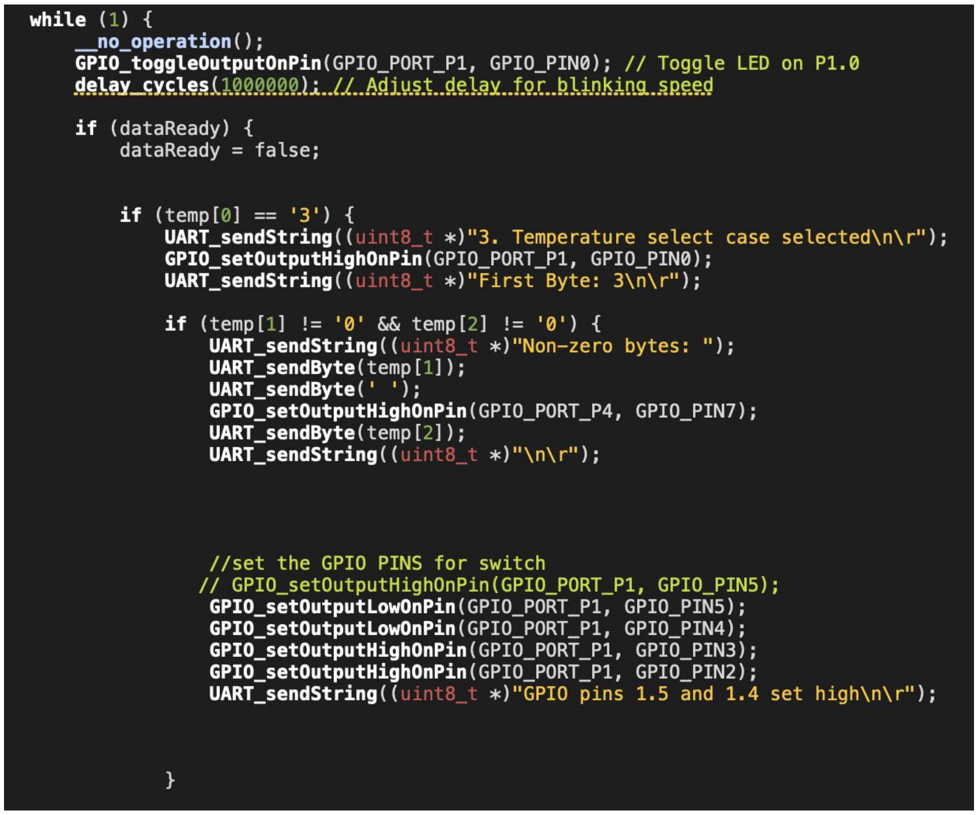


Figure 5 MCU High Level Block Diagram

With the block diagram created, we began programming the MCU to implement this functionality. In this we use the DriverLib import to code within code composer studio. The communication is initialized with UART between the host PC and the MCU. The code begins with setting much of the peripherals for clock, ports, and SPI. In the example shown, the code checks if data is ready and if so, begins its loop. This shows the if statement pertaining to setting a temperature simulation, it takes the first byte and checks if it is 3 and then takes the following two bytes to process into the DAC. A further breakdown of the code can be found in the appendix.



```
while (1) {
    __no_operation();
    GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0); // Toggle LED on P1.0
    delay_cycles(1000000); // Adjust delay for blinking speed

    if (dataReady) {
        dataReady = false;


        if (temp[0] == '3') {
            UART_sendString((uint8_t *)"3. Temperature select case selected\n\r");
            GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
            UART_sendString((uint8_t *)"First Byte: 3\n\r");

            if (temp[1] != '0' && temp[2] != '0') {
                UART_sendString((uint8_t *)"Non-zero bytes: ");
                UART_sendByte(temp[1]);
                UART_sendByte(' ');
                GPIO_setOutputHighOnPin(GPIO_PORT_P4, GPIO_PIN7);
                UART_sendByte(temp[2]);
                UART_sendString((uint8_t *)"\n\r");



                //set the GPIO PINS for switch
               // GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN5);
                GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN5);
                GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN4);
                GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN3);
                GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN2);
                UART_sendString((uint8_t *)"GPIO pins 1.5 and 1.4 set high\n\r");



            }
```

Figure 6 DriverLib Testcase Example

Following our block outline and MCU selection we began creating our PCB schematic and component selection. Due to the tight tolerances needed on our output for accuracy, a 5V, 16bit DAC with a 2.5V reference voltage was selected to give us steps as low as 2.4mV. This also influenced the switch selection used to switch between the desired tests. This ended up in the selection of a 900mOhm 4:1 and 650mOhm 2:1 switch. Finally, the datasheets for the DAC and voltage reference were consulted to select the remaining peripheral components required, an op-amp and passives.

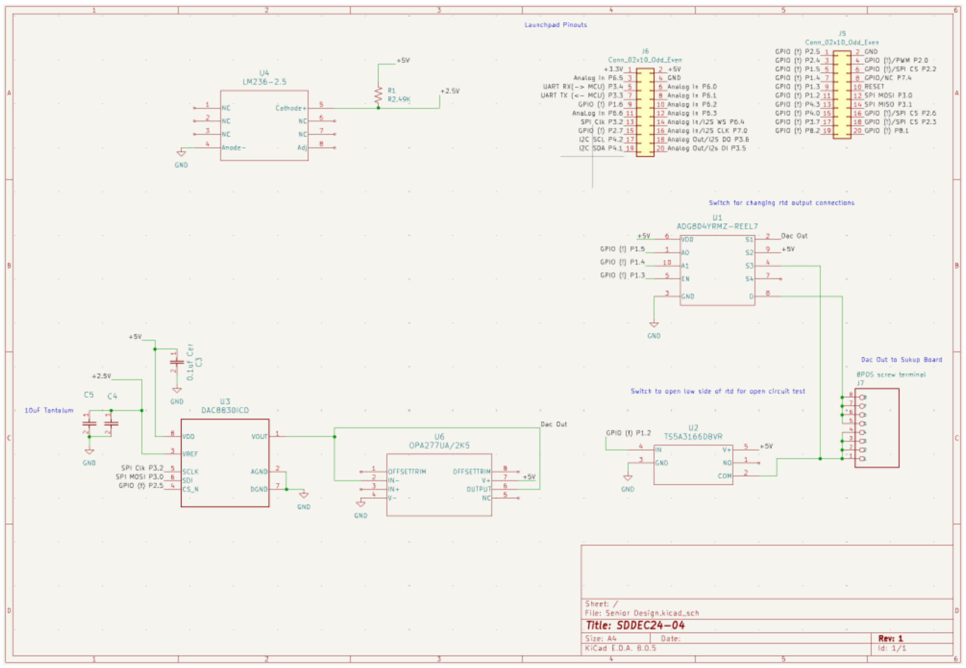


Figure 7 KiCAD Schematic

After schematic review, the board layout work was done along with selection of the screw terminal output block to allow 4 RTD probe sensors to be tested simultaneously. The layout had a few considerations associated with it such as component placement and routing practices.

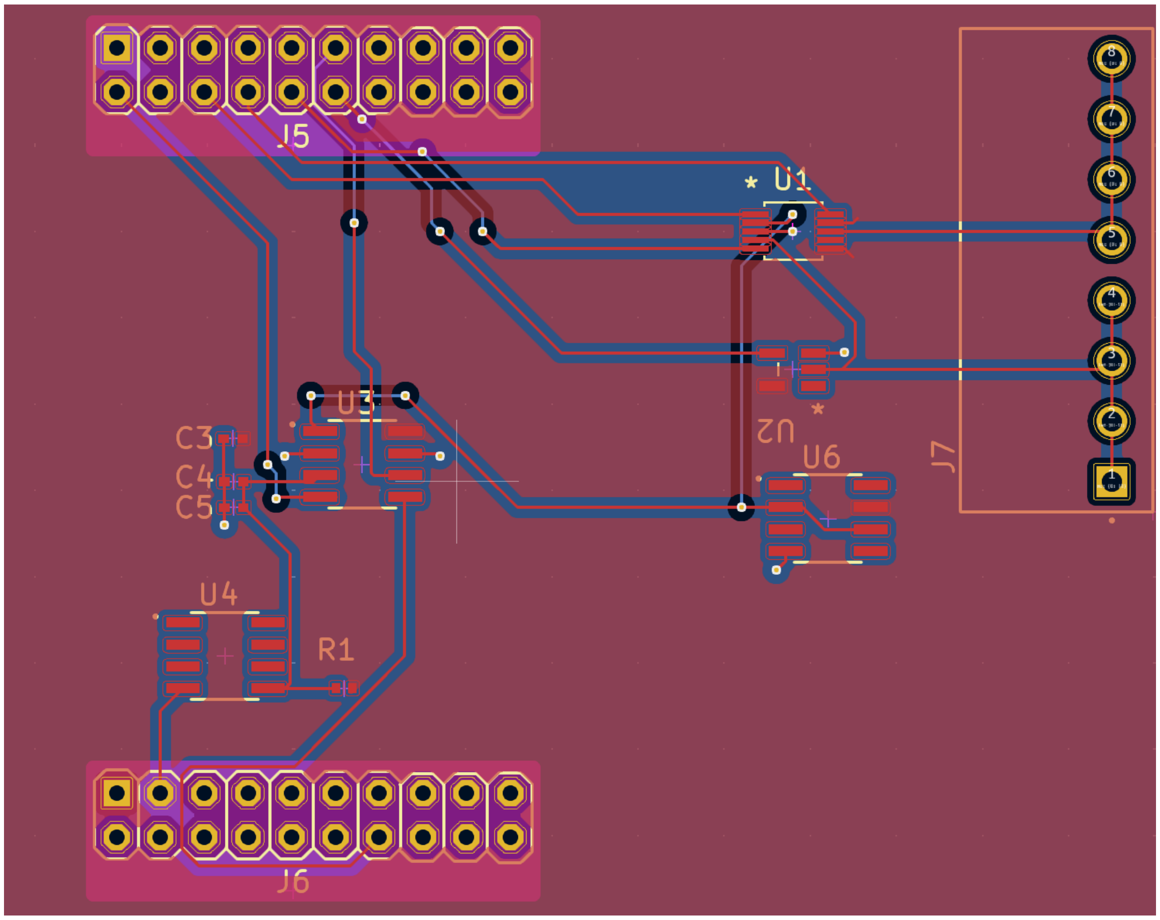


Figure 8 PCB Layout

To test a desired temperature, we first needed to figure out how the MAX chip on Sukup's calculates the temperature from the RTD. The MAX chip uses a bias voltage and reference resistor to produce a known current, I and voltage drop, Vref. It then checks the voltage drop across the RTD against Vref to determine the temperature of the environment. By knowing Io, we can use the fact that the resistance of the RTDs is linear for our range of temperatures to calculate the expected voltage drop across the RTDs. This voltage drop is what we will use to simulate a desired temperature by producing it with the DAC.

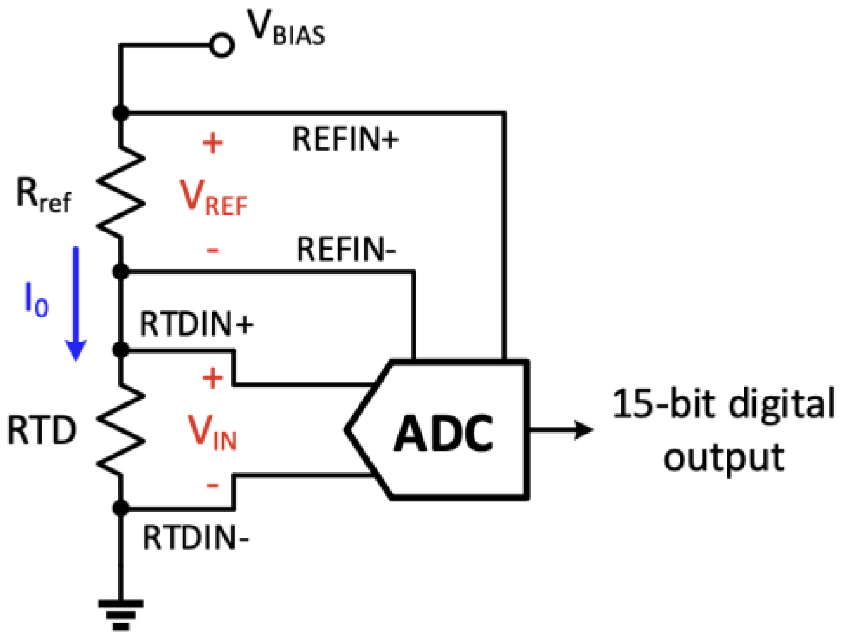


Figure 9 MAX chip equivalent circuit

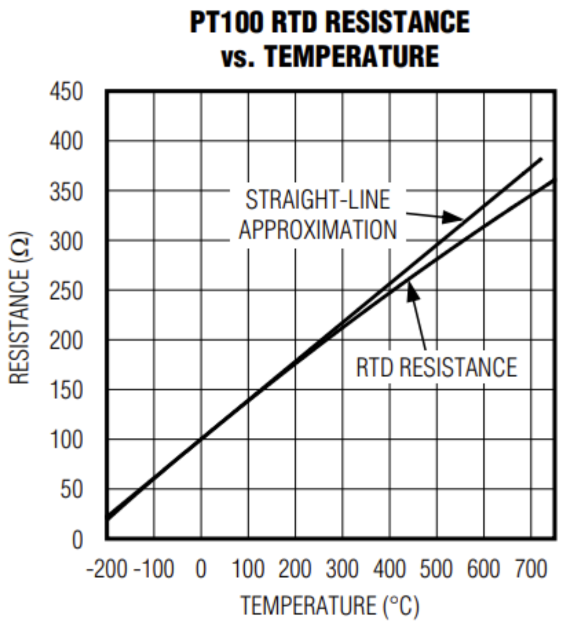
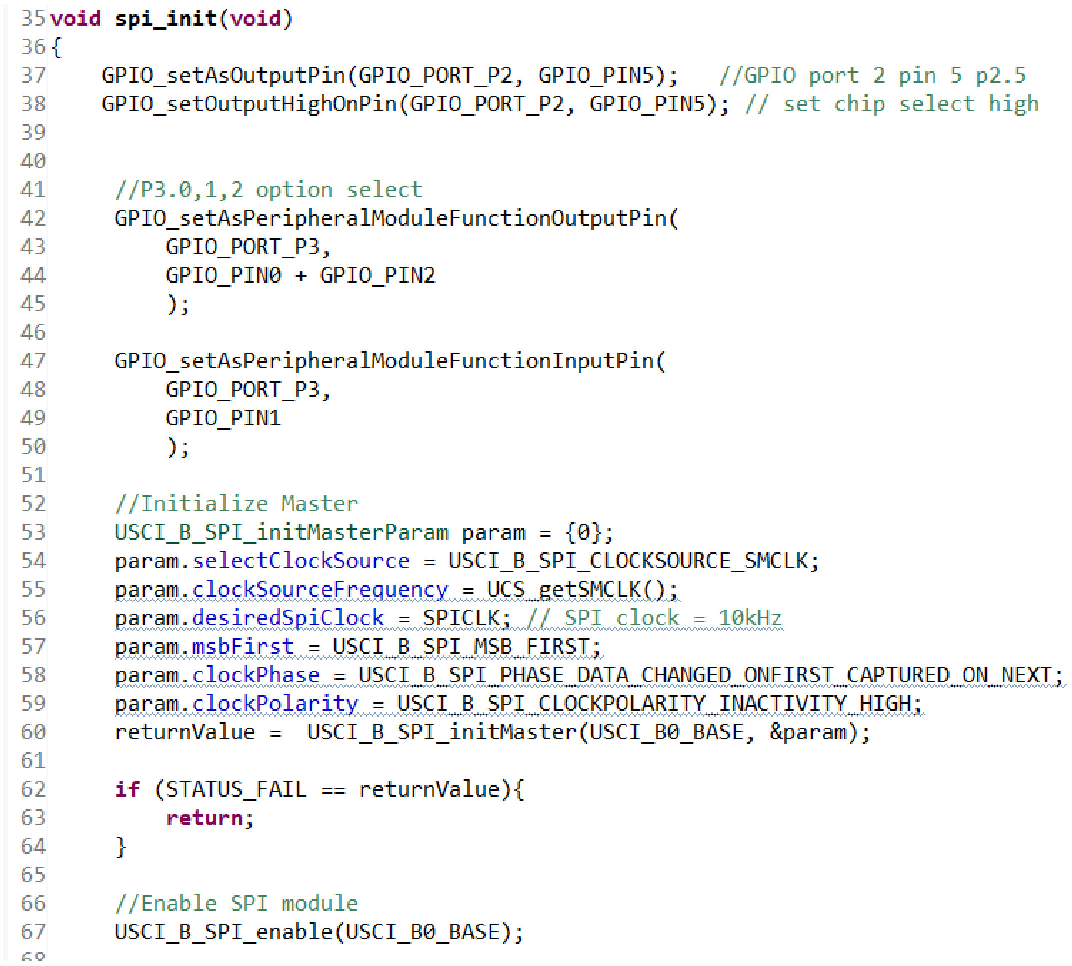


Figure 10 RTD resistance vs temperature

Before we can send any data to the DAC to produce a voltage, we need to initialize the SPI transmission on the MCU. This was easiest to do by using the driver library instead of at the register level. This is done by first setting pins to the correct directions and peripheral functions.

Pins P3.0 and P3.2 are our data and clock pins, respectively, so they are set as peripheral outputs, while P3.1 is our master in pin and is set as a peripheral input. Even though we don't actually communicate back and forth with a slave device, we still initialize the master input because it is good practice. After the pins are set correctly, we initialize the master using functions from the driver library. We don't need to transmit our data insanely fast so we chose to set the SPI clock to 10 kHz. Finally, we finish up the SPI initialization by enabling the SPI module. Below is the code used for SPI initialization.



```c
void spi_init(void)
{
    GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN5);    //GPIO port 2 pin 5 p2.5
    GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN5); // set chip select high


     //P3.0,1,2 option select
     GPIO_setAsPeripheralModuleFunctionOutputPin(
         GPIO_PORT_P3,
         GPIO_PIN0 + GPIO_PIN2
         );

     GPIO_setAsPeripheralModuleFunctionInputPin(
         GPIO_PORT_P3,
         GPIO_PIN1
         );

     //Initialize Master
     USCI_B_SPI_initMasterParam param = {0};
     param.selectClockSource = USCI_B_SPI_CLOCKSOURCE_SMCLK;
     param.clockSourceFrequency = UCS_getSMCLK();
     param.desiredSpiClock = SPICLK; // SPI clock = 10kHz
     param.msbFirst = USCI_B_SPI_MSB_FIRST;
     param.clockPhase = USCI_B_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT;
     param.clockPolarity = USCI_B_SPI_CLOCKPOLARITY_INACTIVITY_HIGH;
     returnValue =  USCI_B_SPI_initMaster(USCI_B0_BASE, &param);

     if (STATUS_FAIL == returnValue){
         return;
     }

     //Enable SPI module
     USCI_B_SPI_enable(USCI_B0_BASE);
```

Figure 11 SPI Initialization

We end the SPI and DAC initialization by clearing the DAC register. To do this we first set the chip select low which allows the DAC to read and hold the data that is sent to it. We can only send 8 bits at a time through the SPI so to clear the DAC we send one byte of zeros followed by another byte of zeros then set the chip select high again to latch the register and update the DAC to output zero volts. We use delays between our two transmissions to allow all the data in one byte to be sent before the next byte is sent, and a final short delay to allow the DAC to settle before anything else done on the MCU. Below is the code for clearing the DAC register and setting the output to zero.

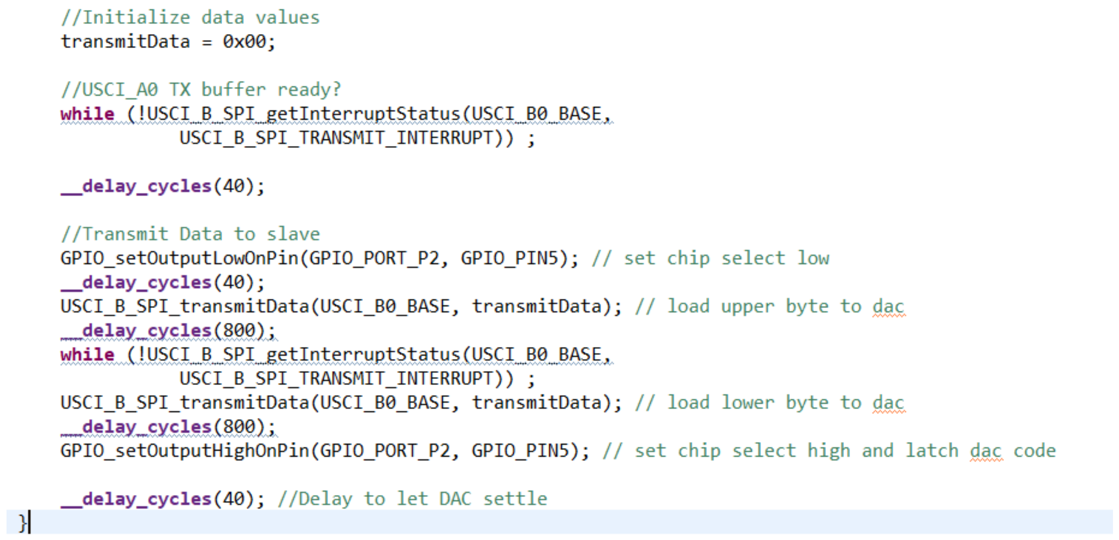
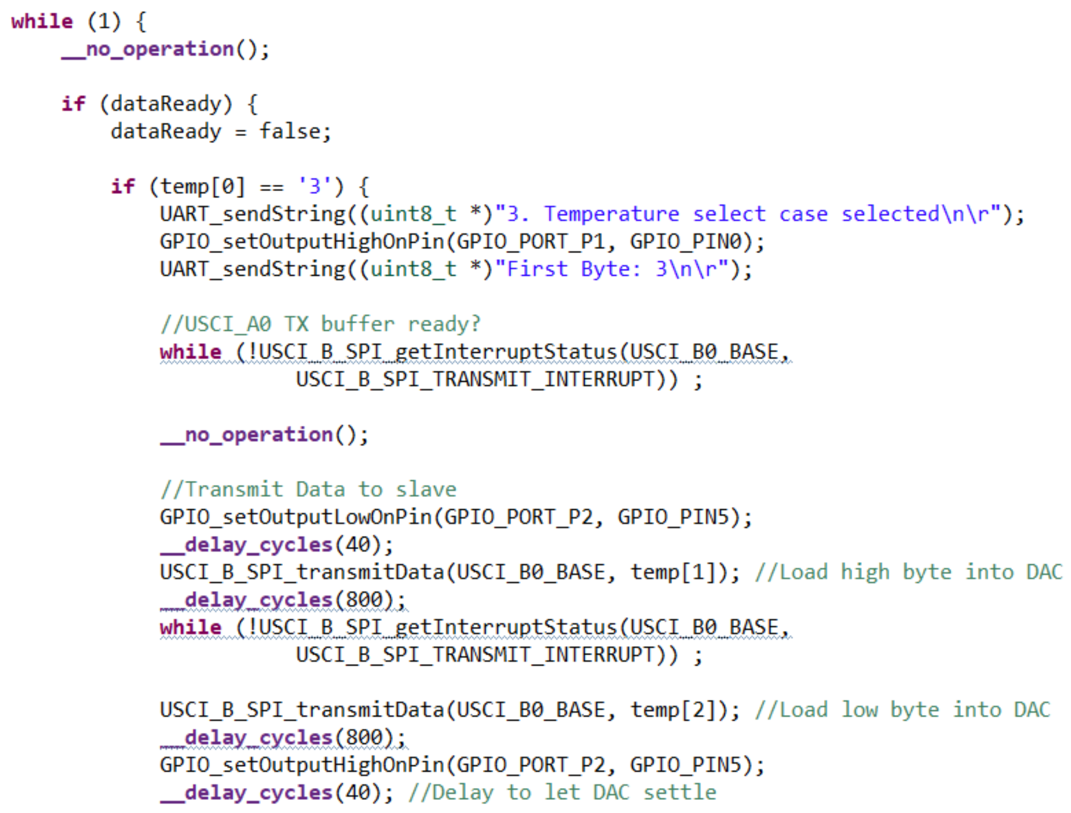

```
    //Initialize data values
    transmitData = 0x00;

    //USCI_A0 TX buffer ready?
    while (!USCI_B_SPI_getInterruptStatus(USCI_B0_BASE,
               USCI_B_SPI_TRANSMIT_INTERRUPT)) ;

    __delay_cycles(40);

    //Transmit Data to slave
    GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN5); // set chip select low
    __delay_cycles(40);
    USCI_B_SPI_transmitData(USCI_B0_BASE, transmitData); // load upper byte to dac
    __delay_cycles(800);
    while (!USCI_B_SPI_getInterruptStatus(USCI_B0_BASE,
               USCI_B_SPI_TRANSMIT_INTERRUPT)) ;
    USCI_B_SPI_transmitData(USCI_B0_BASE, transmitData); // load lower byte to dac
    __delay_cycles(800);
    GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN5); // set chip select high and latch dac code

    __delay_cycles(40); //Delay to let DAC settle
}
```

Figure 12 DAC Clear/Initialization

With the DAC cleared, it is now ready to be loaded with a new value corresponding to a desired test temperature. The code for this looks very similar to the code used to clear the DAC; the main difference is that instead of loading the DAC register with all zeros, it is loaded with the 16-bit code corresponding to the chosen temperature. This 16-bit DAC code is found from our table of temperatures, the corresponding RTD value, and the expected voltage drop across the RTD. Below is the code used to produce the voltage needed to test a given temperature.



```
while (1) {
    __no_operation();

    if (dataReady) {
        dataReady = false;

        if (temp[0] == '3') {
            UART_sendString((uint8_t *)"3. Temperature select case selected\n\r");
            GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
            UART_sendString((uint8_t *)"First Byte: 3\n\r");

            //USCI_A0 TX buffer ready?
            while (!USCI_B_SPI_getInterruptStatus(USCI_B0_BASE,
                       USCI_B_SPI_TRANSMIT_INTERRUPT)) ;

            __no_operation();

            //Transmit Data to slave
            GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN5);
            __delay_cycles(40);
            USCI_B_SPI_transmitData(USCI_B0_BASE, temp[1]); //Load high byte into DAC
            __delay_cycles(800);
            while (!USCI_B_SPI_getInterruptStatus(USCI_B0_BASE,
                       USCI_B_SPI_TRANSMIT_INTERRUPT)) ;

            USCI_B_SPI_transmitData(USCI_B0_BASE, temp[2]); //Load low byte into DAC
            __delay_cycles(800);
            GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN5);
            __delay_cycles(40); //Delay to let DAC settle
```

Figure 13 Code for running temperature test

4.3.3 Functionality

The designs functionality operates with the user downloading the python files provided by Sukup's test team. Following this our board is connected to the test board through the screw terminal lines, replacing the RTDs. The user then locates the python script for selecting a testcase and executes the program. A prompt will appear for the user to select a test case and if desired, a temperature to simulate. Then on the same host PC Sukup’s provided code then checks the temperature that is being received as well as for the other testcases seen at the RTD terminals.

4.3.4 Areas of Challenge

Some areas of challenge that were faced when designing our system was that we were finding issues with the various interfaces and capabilities of our components. Initially we were hoping to script completely in python to be most compatible with Sukup's code base. We however found issue in this when working with our microcontroller, the MSP430. This microcontroller is coded in C and required additional libraries and imports to be functional with our 16-bit DAC. These two coding languages proved difficult to jump between as the MSP would not be able to host any python code to be executed on boot and could not handle the conversion and processing needed to simulate various temperatures.

Our solution to this was reimagine how our board would connect and communicate between all of the interfaces. We decided to establish two connections between Sukup's board, our board, and the host PC. This solution would allow us to do much of the heavy lifting for data processing and user input exclusively on the host PC in python. The MSP430 would then set all of GPIO pins and DAC initialization through C.

Another area of challenge that was faced was initializing the SPI on the MCU. In order to have the DAC produce a voltage to simulate a temperature, the SPI on the MCU had to be used to send data to the DAC register. Originally, we were doing this on the register level of the MCU, but ran into issues due to lack of familiarity with the MCU and inexperience in writing code on the register level.

Our solution for this was to switch to using the driver library which has functions specifically for initializing the SPI, so we didn't have to worry about being wrong on the register level. After we switched to using the driver library it was much easier to finish up the SPI initialization so we were able to transmit data 8 bits at a time.

4.4 Technology Considerations

- Voltage Regulators
  - Can produce a lot of heat
  - Limits the voltage supplied to components in our device
- Stable Voltage Reference
  - Use of a stable voltage reference helps the DAC produce accurate voltages when step sizes are small
  - Is not capable in powering other subsystems in the circuit
  - Requires special consideration in PCB floor planning
- Digital to analog converter
  - Best way to produce accurate voltage that can be adjusted through code
  - Need higher bit converter to improve result resolution such as a 16-bit DAC
  - With higher bit DACs the price increases significantly and produces a more expensive board overall
- RS485 Converter

  - ○ Already used for existing Sukup device
  - ○ Ease to setup and establish communication through USB
  - ○

- ● MSP430 Microcontroller
  - ○ Microcontroller is ideal for low power implementation
  - ○ Allows for direct communication with DAC and setting of GPIO pins
  - ○ Advisor would be able to provide the most aid in this family of microcontrollers
  - ○ Microcontroller can be difficult to understand and requires learning of C
  - ○ Not capable of running complex calculations or data processing as easily as languages like python is able to

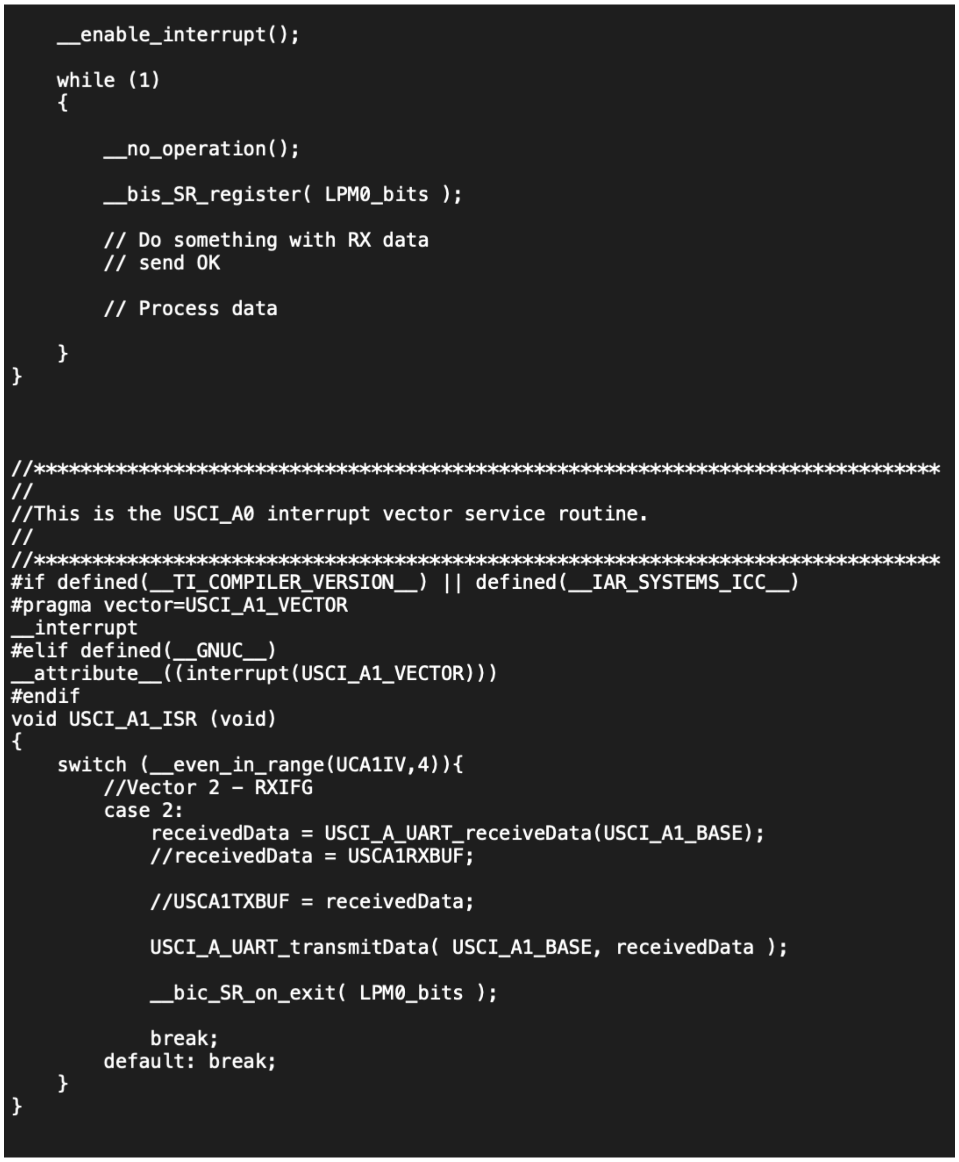
5 Testing

5.1 Unit Testing

The testing for the python code relatively simple compared to other aspects of this project. The method we used for testing was to send the data generated by the script over a USB to the MCU. The MCU had an echo program flashed onto it, so the python script would wait for that echo, and then check it to ensure it was the same as the data it sent to the MCU. This was great for the initial testing, but it needed a bit more for the next stages of integration. The second stage of testing was to ensure that the MCU could correctly pick up the desired test case. To do this, we used two LEDs on the launchpad, and the lit up as a binary code between 0 and 3, and if that binary code matched the test case, it was good to go.

The last stage of testing involved sending a temperature value to the MCU and using an oscilloscope to read the binary code that was sent, and ensure it is correct. The only problem that arose during the testing of the python script was in the first stage when we were waiting for the echo. The code was saying that it received a correct echo, but when we printed the data out in the command terminal, it the echo did not match the data that was being sent. Luckily, we realized that it was not the data that was incorrect, but it was being displayed incorrectly. After decoding the data and converting it into the correct numbering system, it worked very well during the remainder of testing.

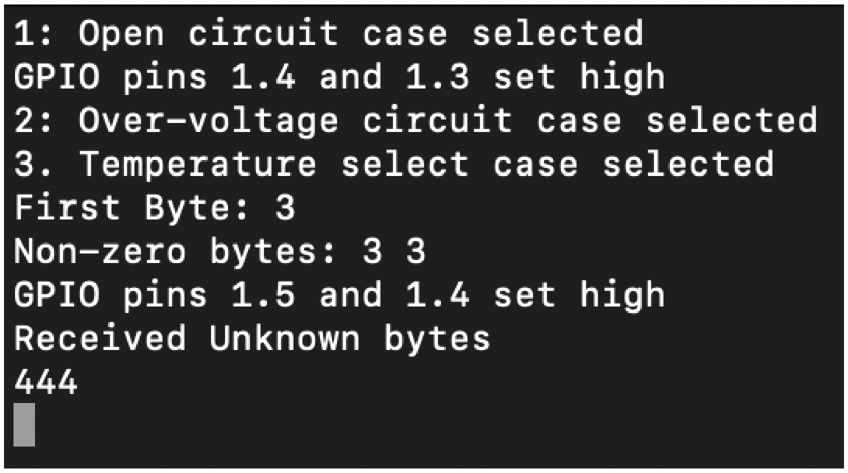
In testing the MCU code it was crucial to break down modules into more manageable blocks. The first part of functionality to verify was that there is communication through UART. We were able to verify this by using the command terminal and establishing a serial connection. To test this the simplest code to write was using the driver library example for a character echo. This code receives a character from the host PC and echos it back to the terminal, writing that specific character. With this code verified, we can move onto using the characters to perform various functions.



```
    __enable_interrupt();

    while (1)
    {

        __no_operation();

        __bis_SR_register( LPM0_bits );

        // Do something with RX data
        // send OK

        // Process data

    }
}



//******************************************************************************
//
//This is the USCI_A0 interrupt vector service routine.
//
//******************************************************************************
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCI_A1_VECTOR
__interrupt
#elif defined(__GNUC__)
__attribute__((interrupt(USCI_A1_VECTOR)))
#endif
void USCI_A1_ISR (void)
{
    switch (__even_in_range(UCA1IV,4)){
        //Vector 2 - RXIFG
        case 2:
            receivedData = USCI_A_UART_receiveData(USCI_A1_BASE);
            //receivedData = USCA1RXBUF;

            //USCA1TXBUF = receivedData;

            USCI_A_UART_transmitData( USCI_A1_BASE, receivedData );

            __bic_SR_on_exit( LPM0_bits );

            break;
        default: break;
    }
}
```

Figure 14 UART Communication initialization

The following block takes the UART character received and sets GPIO pins high or low, this setting helps us set the switches present on the board for the correct output. We have four different settings for this and began testing without the physical board constructed. To do this, we used a multimeter to measure the output voltage at the corresponding GPIO pin. If the pin read 3.3V it was set high and would close the switch, if it was set to 0 the pin is low.



```
1: Open circuit case selected
GPIO pins 1.4 and 1.3 set high
2: Over-voltage circuit case selected
3. Temperature select case selected
First Byte: 3
Non-zero bytes: 3 3
GPIO pins 1.5 and 1.4 set high
Received Unknown bytes
444
```

Figure 15 MCU Test code Example

Also included in this for debug LEDs would be turn on that would represent the testcase selected. These LEDs helped our python code verify that it was transmitting the correct bytes and being processed correctly by the MCU. In addition to this the MCU code was tested to verify that it would wait for three full bytes to be sent before performing any pin setting, this helped ensure that our MCU would not process data that was not ready to be tested.

When testing the PCB, the components were checked for proper continuity using the layout done in KiCAD as reference. This included checks for proper soldering joints as well under the lab scope. Next was ensuring the switches and DAC operated as intended by running our code and verifying the expected and measured continuity and value measurements matched.

To test the DAC, we used an oscilloscope to measure the waveforms of the SPI clock, chip select, and transmission line. We started by changing our DAC initialization value from zero to something easy to see like all hexadecimal A's, or alternating 1's and 0's in binary.

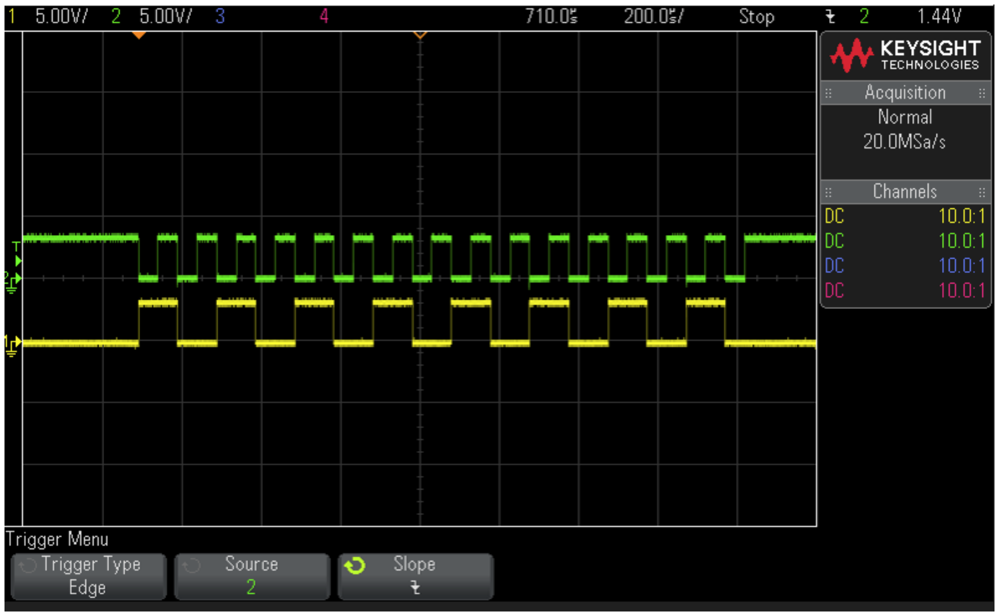


Figure 16 Data transmission with SPI

After confirming that we were able to transmit data using the SPI, we wanted to test if it would send the correct data when choosing a temperature. When doing this test, we were also testing our Python code to make sure the correct data was being sent to the MCU from our temperature table. We tested at 70 degrees Fahrenheit, and the expected binary value for this can be seen above in figure 4. Our measured results for this can be seen below. In the figure, the blue line is our chip select, the green is the clock, and the yellow is transmitted data. It can be seen that our transmitted data matches what we expected to see. The chip select is also low during the time that data is being sent, so when it goes high again the DAC should output the corresponding voltage.

However, when we measured the output of the DAC, we did not get the expected output of 503 mV. Instead, we got an output of 1.2 V. We tested multiple different values for the DAC and always got and output of 1.2V. Because we know that the data being transmitted is correct and the chip select and clock lines work how they are expected to, we determined that it is a hardware issue. To fix this we assembled a new PCB and did testing with that instead. Even after switching PCB's, though, we continued to get incorrect DAC outputs. We double checked our PCB layout and everything appeared to be correct, so we could not figure out where the issue with the hardware was.

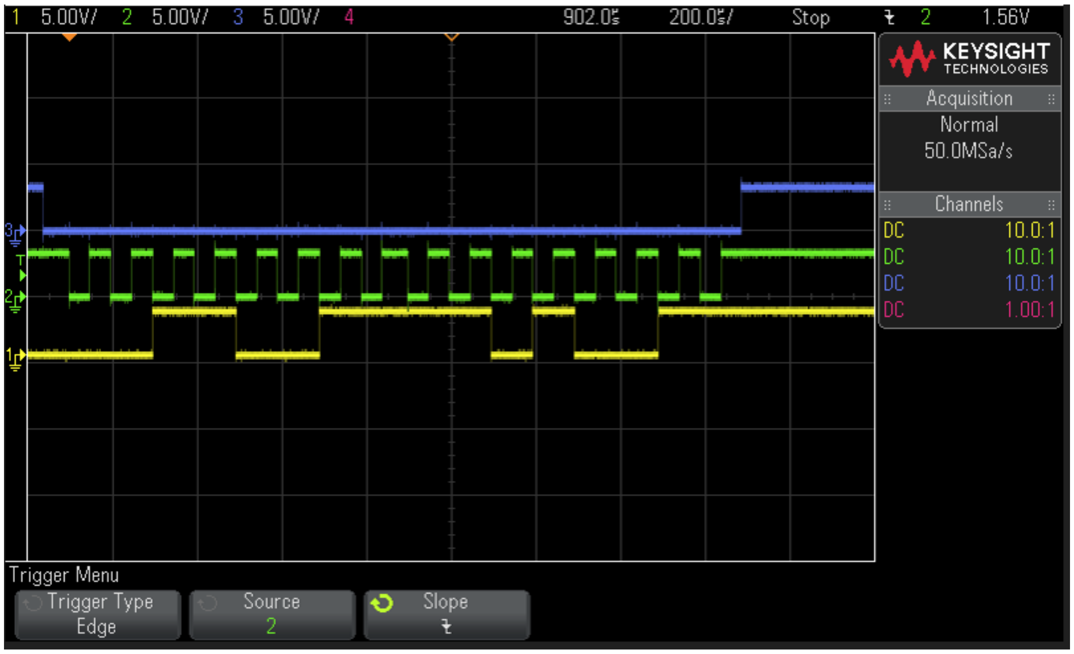


Figure 17 Data transmission for 70-degree test case

5.2 Interface Testing

When working through our design we decided to test our user interface first through the command terminal being used directly with the user. The user interacts the most with this interface by locating the file path in which the python script is saved. Once there the user can simply run the python run command and start the code flow. The code prompts the user for input such as test case desired and temperature simulation. The user can also define different COM ports if more than one device is connected. This interface was tested to make sure that we could handle any input that the user may choose. This includes error messages if the user does not input a command that is recognized by the system and prompts for a correct input.

Another unit that was tested was the MCU interface for debugging. In this we included functions that output messages if the user decides to debug from the MCU directly. This interface gives the user information on what testcases are being ran and GPIO pins being set through UART messages.

5.3 Integration Testing

The critical integration path in our design was most prevalent in the communication between the host PC, MCU, DAC, and finally the PCB. This path begins with the users input into the host PC and gathers three bytes pertaining to testcase and two temperature bytes. The python code then ships this data to the MCU to be used. We tested this by flashing the code to the MCU and running the python code at the same time.

We were able to see that the python code was able to send the correct data since the LEDs would turn on to the expected testcase as well as echo back the correct control byte. To also check the integration with the PCB we were able to see that the switches would correctly switch to the desired state based on the python codes input.

5.4 System Testing

In the overall system we wanted to see each of our testcases behavior when connected to Sukup's board. The first testcase we chose was a short circuit, in this we expected to provide Sukup's board a physical short across the RTD lines and see a very low temperature reading. This was indeed the case as it showed a value of –395.5 F°. Following this we wanted to test the open circuit case by having the user input the command, and making our board emulate an open circuit. This test case also worked, causing the Sukup board to read a value of 3966 F°. The next test case on the list was over-voltage, and this also worked, causing the Sukup board to read 3966 F° once again. The last test case, the temperature test case was the only one that had problems. When we tried to simulate 71 F° we saw the Sukup board measure 183.5 F°. When simulating 200 F° it measured 179.3 F°. The only reason these provided different temperature values was because of some slight noise in the system.

5.5 Regression Testing

We were able to ensure that new additions did not break the previous functionality by only making small improvements and testing often. An example of this during MCU testing would be when testing different GPIO pin configurations, a function that would set all of them low on boot or in-between testcases was implemented. This helped us ensure that a switch would not try to close at the same time another pin was set high causing a short.

5.6 Acceptance Testing

We were able to ensure that design requirements were met by testing the overall system functionality in the environment in which it will find itself in. When reflecting on our list of requirements we can see that we were able to test fault conditions of short and open circuits. We were also able to ensure that RS485 communication as well as Modbus was functional through our interface between the host PC and Sukup's board. We were also able to test overvoltage cases as well as that data transmission was being accurately represented between the host PC and our board.

As for the physical requirements we were able to communicate and power the device using USB. We were as well able to display and process the results through the command terminal in a format familiar to users. Finally, the final PCB was produced in a small form factor that could be easily transported and powered.

5.7 User Testing

In testing for our users, we wanted to implement a method in which we gave a peer the device and the instruction guide and see how they interacted with the testing methodology. In giving this to a peer we wanted to simulate giving the device to a Sukup technician that was unfamiliar with the device. In this we found that we are able to run the code easily when the files were located in the correct workspace. We noticed that the issues we faced in new users was that there must be some knowledge running python code and using the command terminal to do so.

We also found that setting up the entire test connection could be a bit challenging so a resolution we suggested was to change the connection terminals to some form of quick connect rather than screw terminals. These screw terminals were the lengthiest in the process as they are removed for four different terminals.

5.9 Results

The results of our testing proved that using our device the user will successfully be able to test the open circuit, short circuit, and overvoltage fault conditions across the RTDs on Sukup's board. We were not able to get the DAC to output the correct voltage for testing the accuracy of the MAX chip Sukup is using, but we did confirm that the correct data is being sent to the DAC. Because we know that data is correctly being sent to the DAC, we believe the reason it is not outputting the correct voltage is a hardware issue. We double checked our schematic and PCB layout and could not find where the issue would be, however.

6 Implementation

6.1 Design Analysis

Our design functions mostly how it is supposed to. It is easy for the user to communicate with our device by using a command terminal, and our program walks the user through the steps they have to do to run each individual test. The user is able to easily test the open circuit, short circuit, and overvoltage fault conditions across the RTDs on the Sukup board using our device. Our temperature accuracy test does not work how we expected it to, however. No matter what temperature we want to test for, the DAC on our device always outputs the same voltage.

We believe this is a hardware problem but after double checking our schematic and PCB layout, we were unable to find what might be causing the problem. We did find some errors in our PCB layout earlier in the project, so if we had checked our layout more carefully before we had the PCBs manufactured, we may have been able to catch the issue causing the DAC to not function how we expected.

7 Professional Responsibility

7.1 Areas of Responsibility

Area of Responsibility	Definition	NSPE Canon	IEE Code of Ethics 9 (Avoid injury by false action)	Sddec24-04's words
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence	Perform services only in areas of their competence; Avoid deceptive acts	Avoid intentionally making decisions that could cause harm	Only agree to work if you know how to do the work correctly
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs	Act for each employer or client as faithful agents or trustees	Act in a way as to not deceive clients	Set reasonable prices and try to stick a closely to the budget as possible
Communication Honesty	Report works truthfully, without deception, and understandable to stakeholders	Issue public statements only in an objective and truthful manner; Avoid deceptive acts	Report issues truthfully and without deceit	Do not lie to clients or coworkers, make issues known and own mistakes
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders	Hold paramount the safety, health, and welfare of the public	Avoid injury to others by false or malicious actions	Do not cause injury to yourself or others
Property Ownership	Respect property, ideas, and information of clients and others	Act for each employer or client as faithful agents or trustees	Act in a way as to not intentionally harm another's property	Respect other people's physical property, do not steal intellectual property
Sustainability	Protect environment and natural resources locally and globally		Protect the environment as to not cause injury to others	Do not harm the environment

Social Responsibility	Produce products and services that benefit society and communities	Conduct themselves honorable, respectively, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession	Report issues that may cause harm to society	Use your skills for the good of humanity

7.2 Project Specific Professional Responsibility Areas

Because our project is so specific to a single need, not many of the professional responsibility areas apply directly.

- Work Competence
    - Our team must work to design and produce a quality product that will solve the issue we have been presented we believe this was produced in most cases of our requirements except for simulated voltage case
- Financial Responsibility
    - Our project is relatively small and will have no issues staying under the budget provided we believe we accomplished this to a high level staying under 100 dollars
- Communication Honesty
    - Our team is constantly staying in touch with our client as well as each other and our advisor, we believe we communicated plans effectively during the design
- Health, Safety, Well-being
    - Our product will help minimize the risk of grain becoming too hot and combusting. We would believe that our solution does aid in accomplishing these goals
- Property Ownership
    - Our team is designing our product from scratch and not plagiarizing similar ideas that may be available
- Sustainability
    - Our product will help reduce fires and save grain production
- Social Responsibility
    - Our product will help prevent grain from going to waste

7.3 Most Applicable Professional Responsibility Area

The most applicable responsibility area to our project is work competence or communication honesty. We must work hard to produce a quality product that effectively accomplishes the required tasks. To do this we must also communicate often and effectively with not only ourselves but with others that are not as involved with the design and production processes.

8 Conclusions

8.1 Summary of Progress

The purpose of our project was to make a device that could be used to test and help further develop the temperature sensor being produced by Sukup that will be used in grain silos. We needed to make a device to measure the accuracy of the MAX chip Sukup is using for calculating the temperature from the RTDs being used. Our device also needed to test the open circuit, short circuit, and overvoltage fault conditions across the RTDs on Sukup's device. We were able to make a device that does the majority of this. The only test case that does not work as expected is our temperature accuracy test.

8.2 Value Provided

Even without being able to test for the accuracy of the MAX chip used by Sukup, our device can still provide valuable information through the fault condition tests. By using our device to test the fault conditions the user will still be able to see if there is an issue with the MAX chip. Our device can also help diagnose problems and help further develop Sukup's device. For example, if the user is running the open circuit condition test, but the results show a short circuit or something other than the expected result, the user knows that something is either wrong with that specific device or with the design.

8.3 Next Steps

The next step for this project is most likely redesigning the PCB to eliminate hardware issues. Other than the accuracy test not working, our device works as intended. Because of this it is most important to fix the accuracy test which we believe can be done by redesigning the PCB. Something that we wanted to implement but couldn't due to design changes was a function to run all tests back-to-back without prompting the user to do anything other than start the function. This would most likely involve combining the Python code we developed and the Python code Sukup provided to us. This would allow the user to save a little bit of time when running the tests.

9 References

"Temperature vs. Resistance for PT1000 Sensors (PT1000 Resistance Table)." *Pt1000 Resistance Table*, www.sterlingsensors.co.uk/pt1000-resistance-table. Accessed 9 Dec. 2024.

RS-485 Basics Series, www.ti.com/lit/wp/slla545/slla545.pdf. Accessed 9 Dec. 2024.

"RS-485 Serial Interface Explained." *Same Sky*, 14 Aug. 2020, www.sameskydevices.com/blog/rs-485-serial-interface-explained.

(RS485 to USB)

https://www.mouser.com/ProductDetail/DFRobot/FIT0737?qs=pUKx8fyJudCKqKUv6GJ5fA%3D%3D

(Max31865)
https://www.mouser.com/datasheet/2/609/MAX31865-3128729.pdf

(MAX483EESA+T)

https://www.mouser.com/datasheet/2/609/MAX1487E_MAX491E-3129474.pdf

https://modbus.org/
(Modbus basics)

https://ieeexplore.ieee.org/document/27744
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4483758
https://ieeexplore.ieee.org/document/4483758

DAC
https://www.ti.com/lit/ds/symlink/dac8830.pdf?ts=1713482408179&ref_url=https%253A%252F%252Fwww.ti.com%252Fsitesearch%252Fen-us%252Fdocs%252Funiversalsearch.tsp%253FlangPref%253Den-US
https://www.digikey.com/en/products/detail/texas-instruments/DAC8830ICD/1628957

MCU
https://www.ti.com/tool/MSP-EXP430F5529LP/

MSP datasheet (register tables pages 71-82)
https://www.ti.com/lit/ds/symlink/msp430f5529.pdf?ts=1729471751669&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FMSP430F5529%253Futm_source%253Dgoogle%2526utm_medium%253Dcpc%2526utm_campaign%253Depd-null-null-GPN_EN-cpc-pf-google-wwe%2526utm_content%253DMSP430F5529%2526ds_k%253D%257B_dssearchterm

%257D%2526DCM%253Dyes%2526gad_source%253D1%2526gclid%253DCjwKCAjw1NK4BhAwEiw
AVUHPUFk91Sqqm26xoWrDc3458vLpBpC21p2vVtlSKCTfo79EqLVMmpo5aRoCaFEQAvD_BwE%2526gclsrc%253Daw.ds

Op Amp
https://www.ti.com/lit/ds/symlink/opa277.pdf?ts=1727040347985

2.5 voltage reference regulator
https://www.ti.com/lit/ds/symlink/lm336-2.5.pdf?ts=1727128810245&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLM336-2.5

10 Appendices

Appendix 1 – Operation Manual

1) Wiring
    a) Sukup board
        i) Wire the Sukup board in the same fashion as normal, but do not connect any RTDs (still include the jumpers from the RTD inputs to the Force inputs as if using 2-wire RTDs)
    b) Test kit
        i) Connect USB-C on test kit to computer
        ii) Connect jumpers from J7 to the RTD inputs
            (1) The 4 terminals on the right (when looking at the test kit from the side that has J7) are the RTD+ jumpers
            (2) The 4 terminals on the left (when looking at the test kit from the side that has J7) are the RTD- jumpers

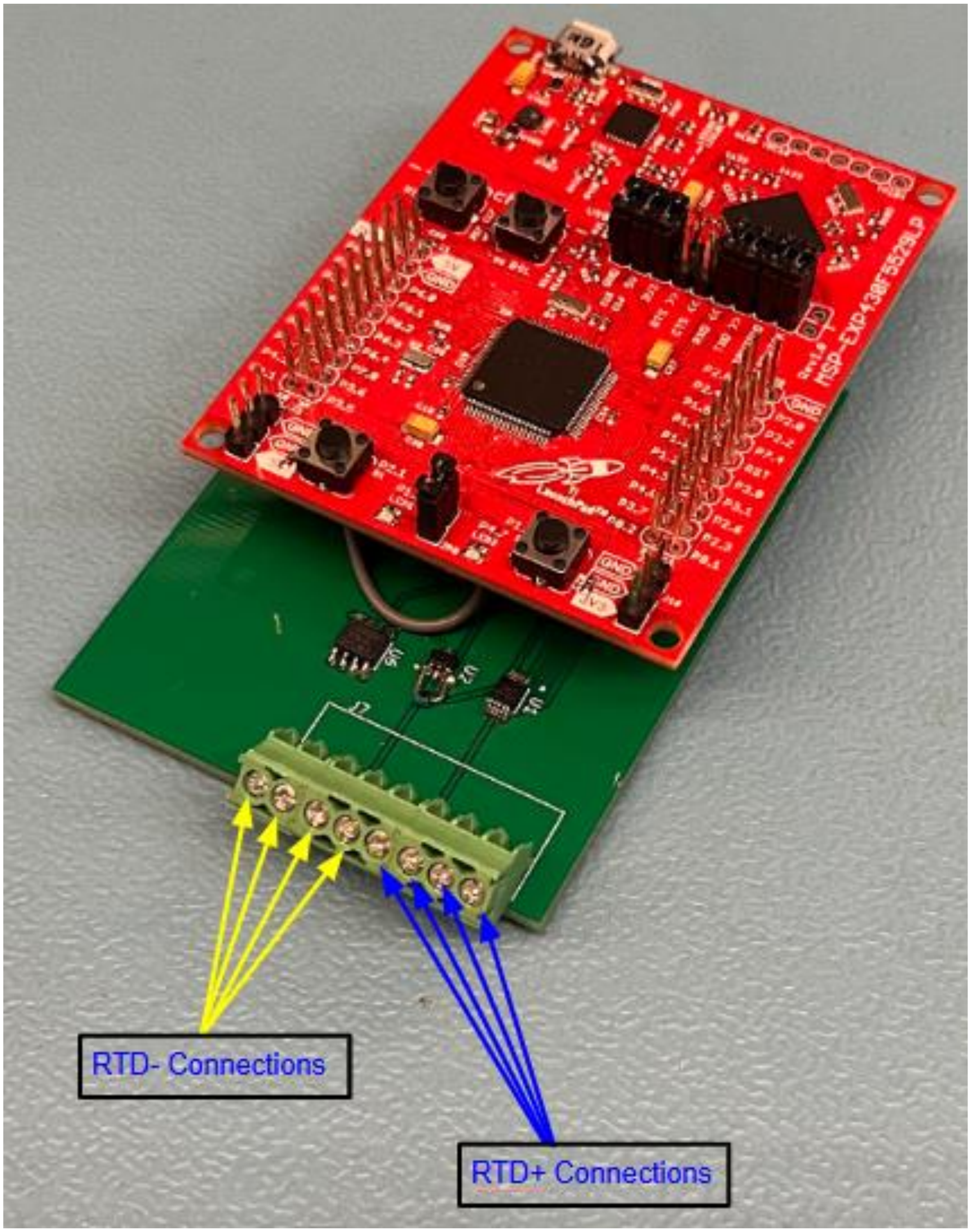


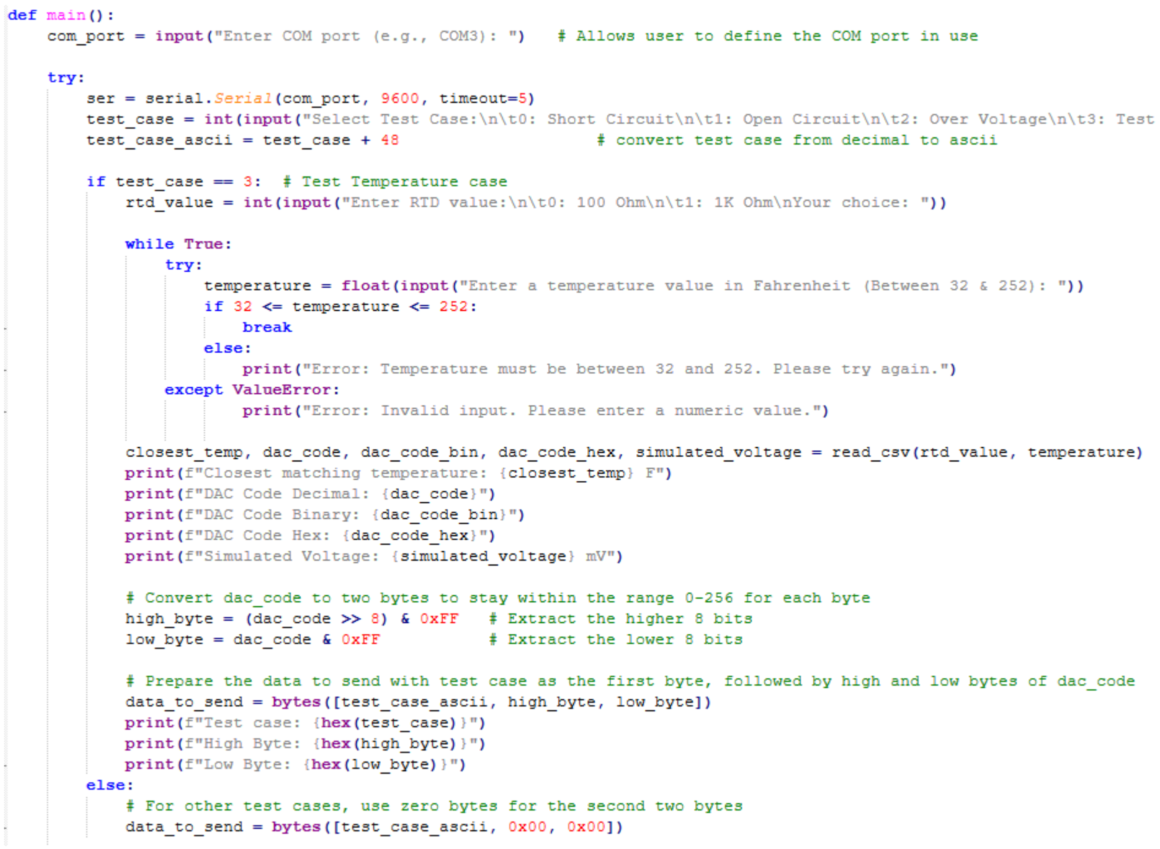
Figure 18 Connection terminals on test kit board

2) Pre-setup
    a) Open the TestBoardConfiguration.py script
    b) Edit line 6 to be the file path to the save location of the CSV files
    c) Edit line 7 if you changed the names of the CSV files

        d) Save script
3) Setup
    a) Open Power Shell (command terminal)
    b) Open the file path to the script by typing cd "C:\example\file\path"
        i) Include quotation marks
    c) Hit enter
4) Using the script
    a) First question will be to enter the COM port
        i) Enter the name of the COM port that the test board is plugged into, then hit enter
    b) Second question will be to enter the test case you would like to run
        i) Enter the number that appears next to the test case that you want to run, then hit enter
    c) If the temperature test case is selected, the next question will be to enter the RTD resistance value
        i) Set the switches on the Sukup board to the desired resistance
        ii) Enter the number that appears next to the RTD resistance value chosen, then hit enter
    d) The last question will be to enter the temperature you would like to select
        i) Enter the desired temperature (in Fahrenheit), then hit enter
        ii) The script will select the closest possible temperature value
        iii) The script will then display various values (such as the temperature it is testing, the DAC code in binary, etc.) that can be commented out in the script if desired, so they do not get displayed
5) Timeout error
    a) The MCU code has a section that will echo back to the host PC whatever code it receives after the test case has been sent. That section has been commented out. Because of this, the command terminal will display a timeout error 5 seconds after it sends the data, but the data has already been sent and the test kit is ready to go. To stop this error, either uncomment the echo code on the MCU (the command terminal will then display a message saying echo received and display some more data). The other method is to comment out the echo code on the python script so that it does not wait for the echo.
6) Errors
    a) If the echo code in the MCU is uncommented and the echo is in effect
        i) If the script displays a timeout error, unplug the test board from the computer and plug back in
    b) If the script displays a COM port not found error, check that the correct COM port was entered, try unplugging the MCU and plugging back in, or try entering a different COM port
7) CTRL + C will end the program and allow the user to start a new test

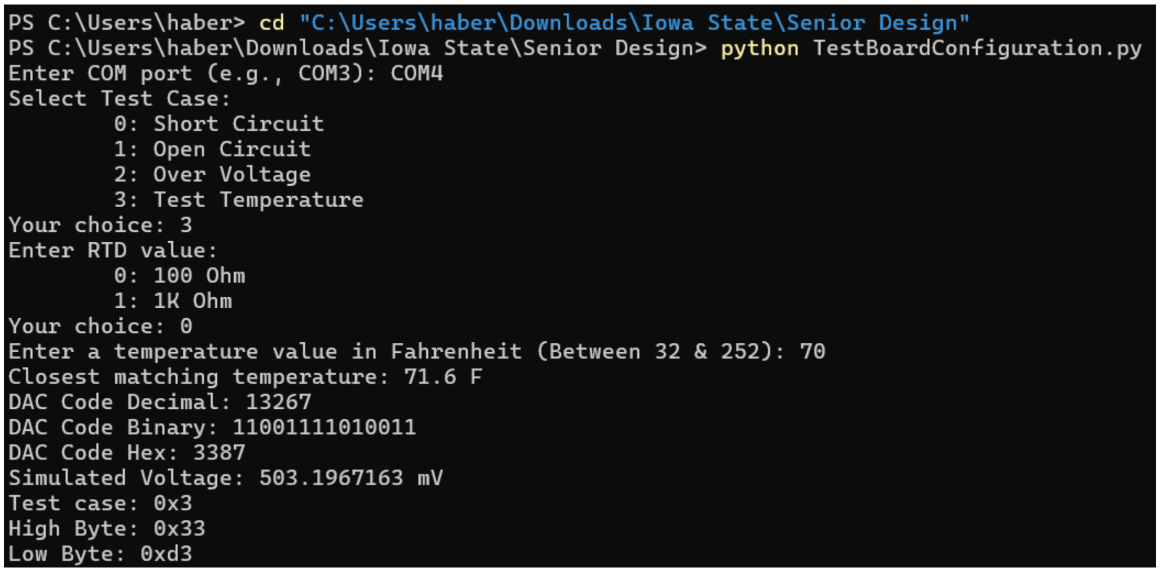
Appendix 2 – Code

- https://iastate.box.com/s/ov3fcsuwh3bbaqqku1boqzdrch35384a



```python
def main():
    com_port = input("Enter COM port (e.g., COM3): ")   # Allows user to define the COM port in use

    try:
        ser = serial.Serial(com_port, 9600, timeout=5)
        test_case = int(input("Select Test Case:\n\t0: Short Circuit\n\t1: Open Circuit\n\t2: Over Voltage\n\t3: Test
        test_case_ascii = test_case + 48                    # convert test case from decimal to ascii

        if test_case == 3:  # Test Temperature case
            rtd_value = int(input("Enter RTD value:\n\t0: 100 Ohm\n\t1: 1K Ohm\nYour choice: "))

            while True:
                try:
                    temperature = float(input("Enter a temperature value in Fahrenheit (Between 32 & 252): "))
                    if 32 <= temperature <= 252:
                        break
                    else:
                        print("Error: Temperature must be between 32 and 252. Please try again.")
                except ValueError:
                        print("Error: Invalid input. Please enter a numeric value.")

            closest_temp, dac_code, dac_code_bin, dac_code_hex, simulated_voltage = read_csv(rtd_value, temperature)
            print(f"Closest matching temperature: {closest_temp} F")
            print(f"DAC Code Decimal: {dac_code}")
            print(f"DAC Code Binary: {dac_code_bin}")
            print(f"DAC Code Hex: {dac_code_hex}")
            print(f"Simulated Voltage: {simulated_voltage} mV")

            # Convert dac_code to two bytes to stay within the range 0-256 for each byte
            high_byte = (dac_code >> 8) & 0xFF   # Extract the higher 8 bits
            low_byte = dac_code & 0xFF           # Extract the lower 8 bits

            # Prepare the data to send with test case as the first byte, followed by high and low bytes of dac_code
            data_to_send = bytes([test_case_ascii, high_byte, low_byte])
            print(f"Test case: {hex(test_case)}")
            print(f"High Byte: {hex(high_byte)}")
            print(f"Low Byte: {hex(low_byte)}")
        else:
            # For other test cases, use zero bytes for the second two bytes
            data_to_send = bytes([test_case_ascii, 0x00, 0x00])
```

Figure 19 Snippet of Host PC Python script



```
PS C:\Users\haber> cd "C:\Users\haber\Downloads\Iowa State\Senior Design"
PS C:\Users\haber\Downloads\Iowa State\Senior Design> python TestBoardConfiguration.py
Enter COM port (e.g., COM3): COM4
Select Test Case:
        0: Short Circuit
        1: Open Circuit
        2: Over Voltage
        3: Test Temperature
Your choice: 3
Enter RTD value:
        0: 100 Ohm
        1: 1K Ohm
Your choice: 0
Enter a temperature value in Fahrenheit (Between 32 & 252): 70
Closest matching temperature: 71.6 F
DAC Code Decimal: 13267
DAC Code Binary: 11001111010011
DAC Code Hex: 3387
Simulated Voltage: 503.1967163 mV
Test case: 0x3
High Byte: 0x33
Low Byte: 0xd3
```

Figure 20 Example of running the python script

Appendix 3 - Team

Team Members

Justin Garden

Samuel Estrada

Tony Haberkorn

Michael Hurley

Required Skill Sets for Your Project

The design requires knowledge of power supplies, usage of microcontrollers, and the functionality of different voltage monitors. We are also using C and python to help run testcases as well as return information about our system. In addition, device interconnections have to communicate with each other using different communication protocols such as UART and SPI for the DAC in particular. We will also need the knowledge of PCB design and common practices used to produce a functioning board as well as the software tool KiCad to produce said board.

Skill Sets covered by the Team

● Samuel Estrada – Skill set includes experience coding test cases in python, hardware development, PCB design, and analog part testing.

● Justin Garden - Skill set includes PCB design, circuit design, and microcontroller programming.

● Tony Haberkorn - Skill sets include circuit design, RTD knowledge and block schematic diagrams.

● Michael Hurley - Skills include PCB design and assembly, Altium experience and circuit testing and troubleshooting.

Project Management Style Adopted by the team

The project management style we have chosen is agile. A major reason for this is because as we have learned more about our client's wants and needs from the project, our requirements and design choices have changed. An agile management style also allows us to focus on short term sprint goals that will allow us to better stay on track for finishing the project on time. A downside to this is that we don't have many hard deadlines which could allow us to become distracted and put too much focus on an area that doesn't require it.

Individual Project Management Roles

Justin - Handled client relations and DAC conversion charts

Sam - Handling PCB design review methods and C script coding for MCU testcase selections

Tony - Handled hardware and power management as well python script for host PC

Michael - Handling the Altium PCB designer workspace

Team Contract

Team Members:

1) Anthony Haberkorn 2) Samuel Estrada

3) Michael Hurley 4) Justin Garden

Team Procedures

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:
   -Face-to-face meetings on Monday on a weekly basis (9:45am) located at TLA/Library
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:
   -Be able to produce and explain design choices at each meeting to catch members up on progress
3. Expected level of communication with other team members:
   -Phone and face-to-face meetings. Make sure all issues are addressed during meetings that may impede others work.
4. Expected level of commitment to team decisions and tasks:
   -Be able to put forth enough time outside of scheduled meetings to keep up on tasks and push the project forward.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):
   Tony Haberkorn - Organization Michael Hurley - Individual component design Samuel Estrada - Testing Justin Garden - Client interaction
2. Strategies for supporting and guiding the work of all team members:
   -Individual updates at each weekly meeting
   -Work reviews with questions/comments from team members
3. Strategies for recognizing the contributions of all team members:
   -Individual updates at each weekly meeting

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.
   Samuel Estrada - Experience with Altium, kicad, part selection, communication protocols, C, python, and PCB design
   Tony Haberkorn - RTDs, industrial applications, Modelsim, Quartus, CNC cutter
   Michael Hurley- Altium pcb design and manufacturing, digikey part research and choice, matlab/ c coding.
   Justin Garden - Circuit/PCB design, C programming, Component research
2. Strategies for encouraging and support contributions and ideas from all team members:
   Give opportunities for team/client feedback during meeting times

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)
   - Talk during meetings and bring up issues. Can also talk to advisor
   - Communicate issues as they come up in a timely manner

Goal-Setting, Planning, and Execution

1. Team goals for this semester:
   Begin designing solution for Sukup's test board. We plan to have a prototype that can test open, short, overvoltage, and temperature simulations.
2. Strategies for planning and assigning individual and team work:
   Break project into manageable small tasks, tasks too large for one individual will be set aside for meeting/ workdays
3. Strategies for keeping on task:
   -Individual updates at each weekly meeting
   -Work reviews with questions/comments from team members
   -Goals to accomplish during each meeting will be reviewed and tracked

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?
   The offender will be reminded of their responsibilities frequently until the next group meeting.
2. What will your team do if the infractions continue?
   It will be brought up to the instructors to determine the best course of action from that point onward.

****

a. I participated in formulating the standards, roles, and procedures as stated in this contract.
b. I understand that I am obligated to abide by these terms and conditions.
c. I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

1) ________________Justin Garden______________________________ DATE ___01/30/24 __________

2) ________________Samuel Estrada____________________________ DATE ___01/30/24 __________

3) _______________ Tony Haberkorn ___________________________ DATE ___01/30/24 __________

4) ____________ Michael Hurley ______________________________ DATE ___01/30/24 __________