# Sukup RTD Circuit Tester

Sddec24-04
Tony Haberkorn, Justin Garden, Michael Hurley, Sam Estrada
Advisor: Dr. Neihart
Client: Dana Conrad (Sukup)

# Introduction

- Users
  - Sukup Electrical Engineer
  - Sukup Technician
- Problem
  - Test PCBs after production
  - Test newly developed PCBs
- Importance
  - Ensure faulty PCBs do not get sold
  - Streamline future design processes

# User Needs

- Need kit to test RTD circuits
- Test circuit boards at end of production line
- Test newly developed circuit boards
- Create new testbenches
- Test boards on pass/fail basis, also include description of failures

# Requirements

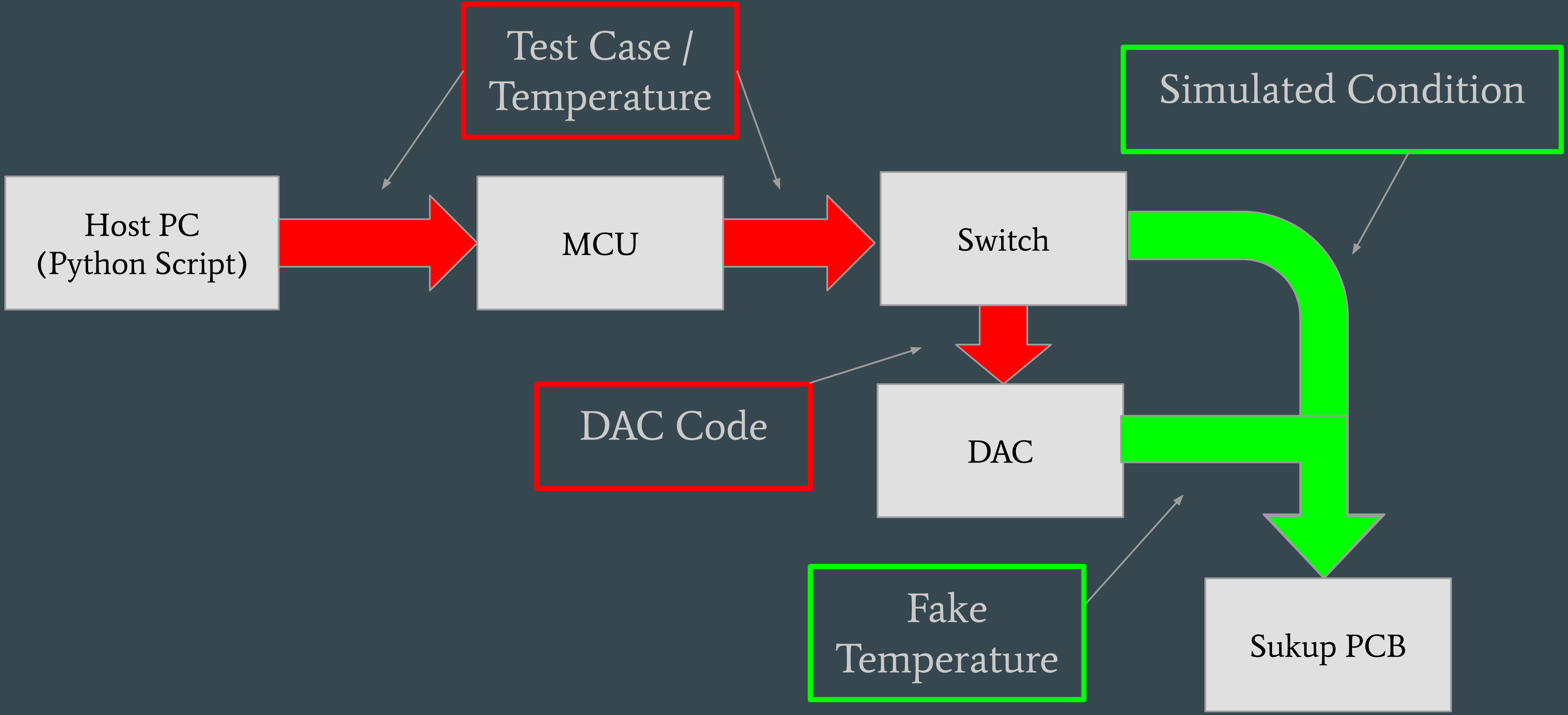
- Purpose - Our design simulates operational & fault conditions for testing
- Key Requirements
  - Uses standard USB
  - Simulates temperature values
    - Measure accuracy of temperature measurement chip (MAX31865)
  - Test open, short and over-voltage conditions in RTD
  - Test Modbus communication
  - Test surge protection

# Design

# Design Overview Description

- Host PC
    - Runs python script that allows user to define test case / temperature and sends to our PCB
    - Waits for our PCB to run tests and displays results
- MCU
    - Reads data from host PC
    - Controls the DAC and switches
- DAC
    - Generates the test conditions by outputting a simulated voltage
- Switch
    - Selects between test cases (4 cases)

# Design Overview Visualization



# Software

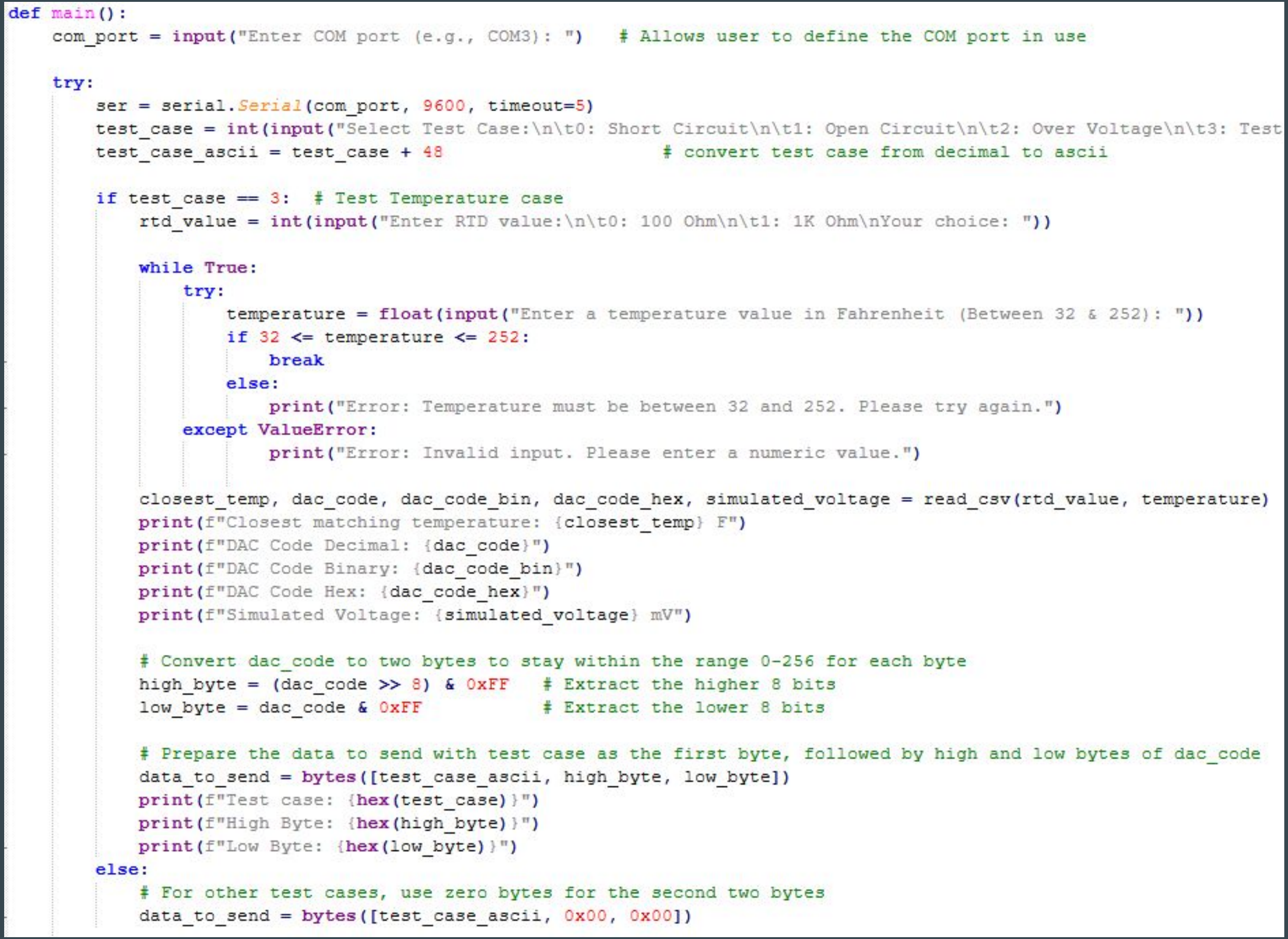
# Host PC

- User Interface
  - Command terminal
- Python script
  - Easy configuration
  - Simple interface
  - Same as Sukup code
- Sends data to MCU
  - Three bytes

# Python Script

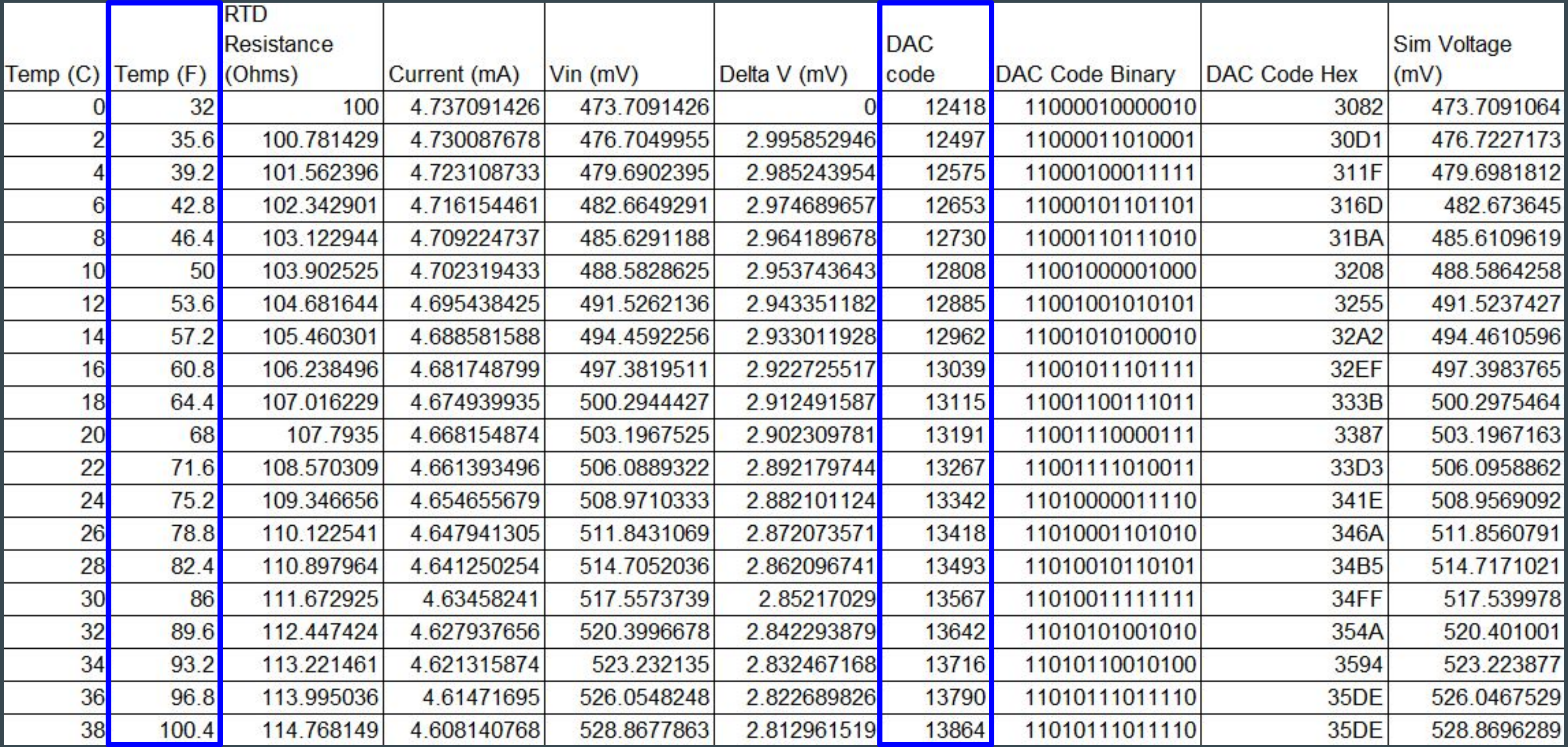
- Configure communications
    - COM port
- Test case
    - Short circuit
    - Open circuit
    - Over voltage
    - Temperature
        - RTD value
        - Test temperature
            - CSV look-up table
- 3 bytes sent
    - First byte - Test case
    - Second and third byte - DAC code

# Python Script



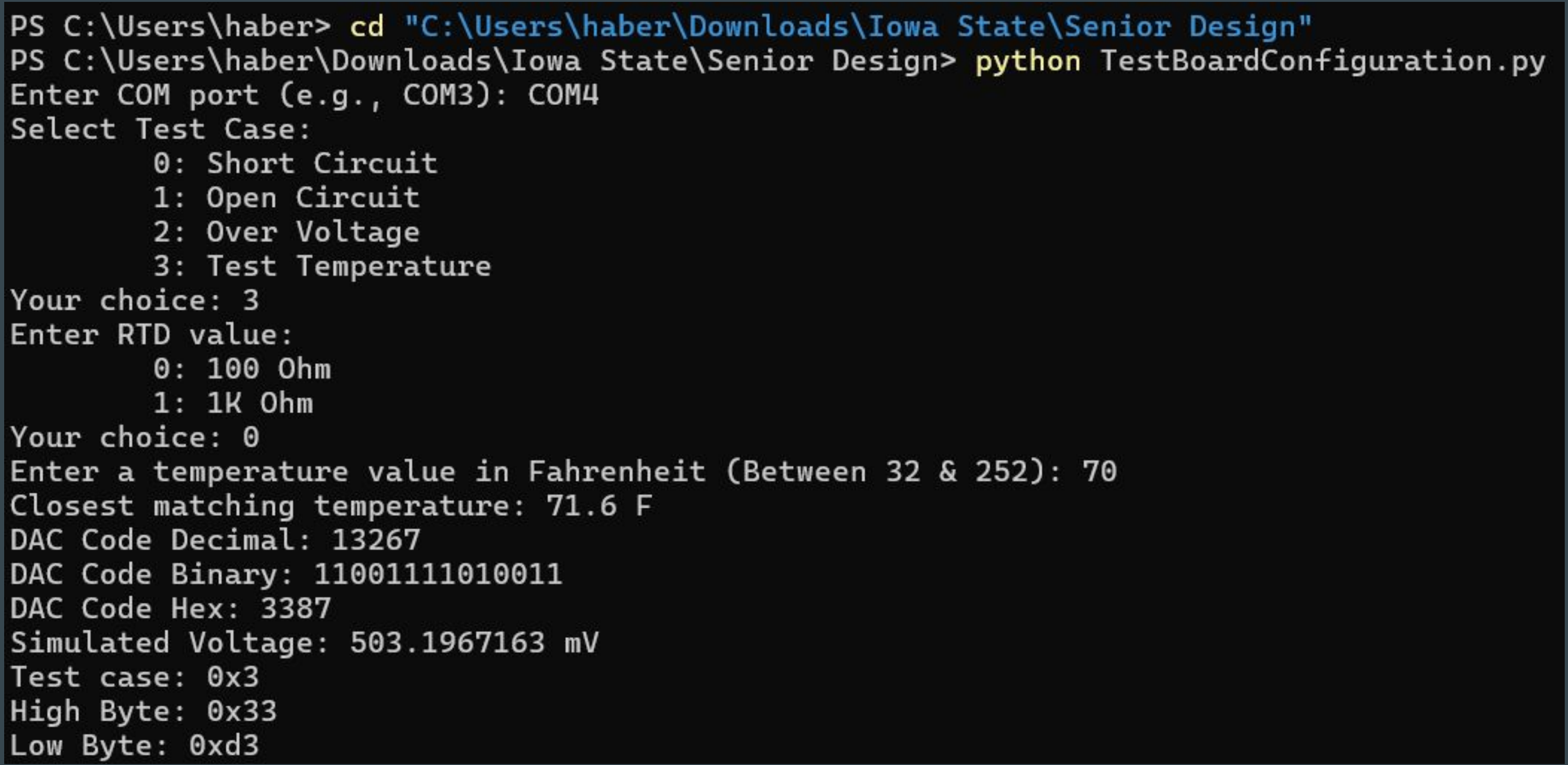
```python
def main():
    com_port = input("Enter COM port (e.g., COM3): ")   # Allows user to define the COM port in use

    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])
```

# CSV File



| Temp (C) | Temp (F) | RTD Resistance (Ohms) | Current (mA) | Vin (mV) | Delta V (mV) | DAC code | DAC Code Binary | DAC Code Hex | Sim Voltage (mV) |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 32 | 100 | 4.737091426 | 473.7091426 | 0 | 12418 | 11000010000010 | 3082 | 473.7091064 |
| 2 | 35.6 | 100.781429 | 4.730087678 | 476.7049955 | 2.995852946 | 12497 | 11000011010001 | 30D1 | 476.7227173 |
| 4 | 39.2 | 101.562396 | 4.723108733 | 479.6902395 | 2.985243954 | 12575 | 11000100011111 | 311F | 479.6981812 |
| 6 | 42.8 | 102.342901 | 4.716154461 | 482.6649291 | 2.974689657 | 12653 | 11000101101101 | 316D | 482.673645 |
| 8 | 46.4 | 103.122944 | 4.709224737 | 485.6291188 | 2.964189678 | 12730 | 11000110111010 | 31BA | 485.6109619 |
| 10 | 50 | 103.902525 | 4.702319433 | 488.5828625 | 2.953743643 | 12808 | 11001000001000 | 3208 | 488.5864258 |
| 12 | 53.6 | 104.681644 | 4.695438425 | 491.5262136 | 2.943351182 | 12885 | 11001001010101 | 3255 | 491.5237427 |
| 14 | 57.2 | 105.460301 | 4.688581588 | 494.4592256 | 2.933011928 | 12962 | 11001010100010 | 32A2 | 494.4610596 |
| 16 | 60.8 | 106.238496 | 4.681748799 | 497.3819511 | 2.922725517 | 13039 | 11001011101111 | 32EF | 497.3983765 |
| 18 | 64.4 | 107.016229 | 4.674939935 | 500.2944427 | 2.912491587 | 13115 | 11001100111011 | 333B | 500.2975464 |
| 20 | 68 | 107.7935 | 4.668154874 | 503.1967525 | 2.902309781 | 13191 | 11001110000111 | 3387 | 503.1967163 |
| 22 | 71.6 | 108.570309 | 4.661393496 | 506.0889322 | 2.892179744 | 13267 | 11001111010011 | 33D3 | 506.0958862 |
| 24 | 75.2 | 109.346656 | 4.654655679 | 508.9710333 | 2.882101124 | 13342 | 11010000011110 | 341E | 508.9569092 |
| 26 | 78.8 | 110.122541 | 4.647941305 | 511.8431069 | 2.872073571 | 13418 | 11010001101010 | 346A | 511.8560791 |
| 28 | 82.4 | 110.897964 | 4.641250254 | 514.7052036 | 2.862096741 | 13493 | 11010010110101 | 34B5 | 514.7171021 |
| 30 | 86 | 111.672925 | 4.63458241 | 517.5573739 | 2.85217029 | 13567 | 11010011111111 | 34FF | 517.539978 |
| 32 | 89.6 | 112.447424 | 4.627937656 | 520.3996678 | 2.842293879 | 13642 | 11010101001010 | 354A | 520.401001 |
| 34 | 93.2 | 113.221461 | 4.621315874 | 523.232135 | 2.832467168 | 13716 | 11010110010100 | 3594 | 523.223877 |
| 36 | 96.8 | 113.995036 | 4.61471695 | 526.0548248 | 2.822689826 | 13790 | 11010111011110 | 35DE | 526.0467529 |
| 38 | 100.4 | 114.768149 | 4.608140768 | 528.8677863 | 2.812961519 | 13864 | 11010111011110 | 35DE | 528.8696289 |

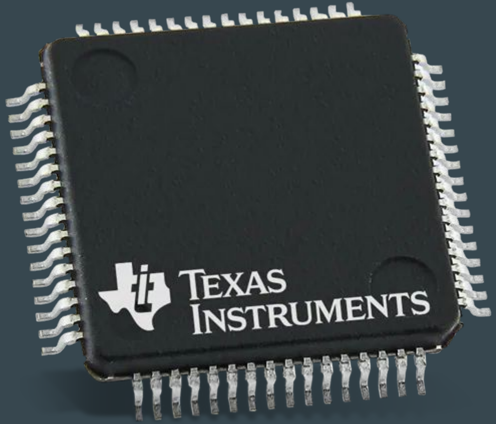
# Example Test



```
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
```

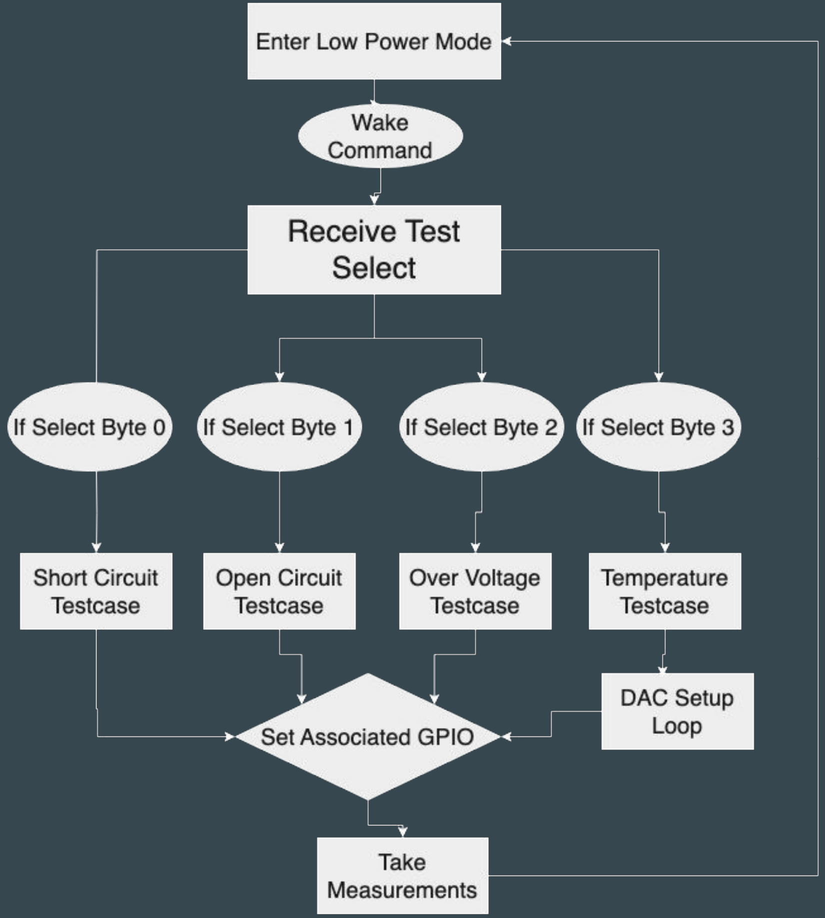
# Host to MCU Requirements

- MCU code needs to handle data sent by Host
- MSP430 microcontroller in C
- Wait for Host to send complete package
- Confirm to user test case was set
- Set test conditions



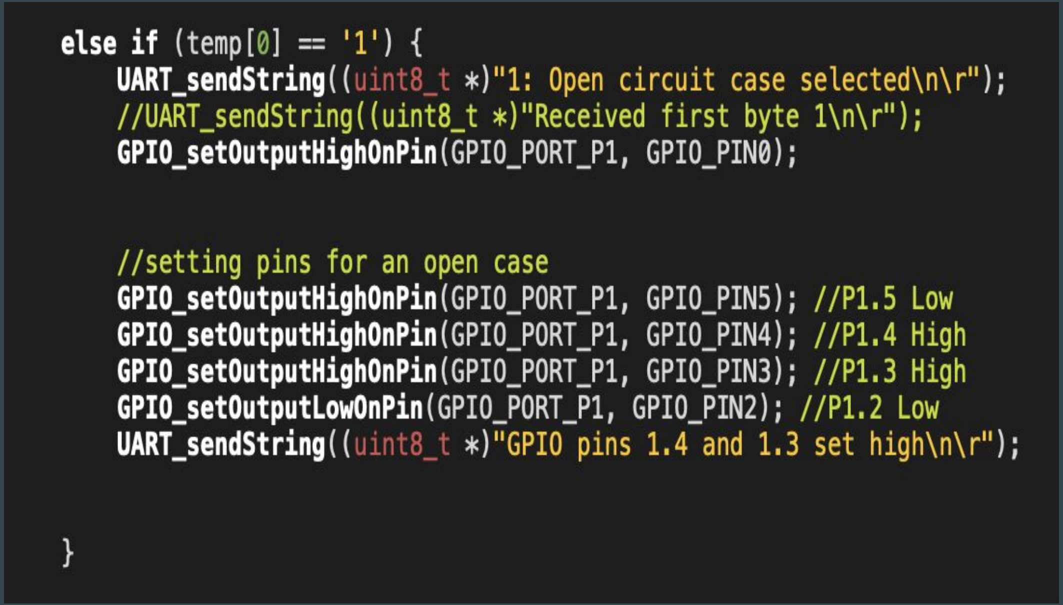
# MCU Code Block Diagram

- MCU is in low power mode
- Wakes and begins processing
- Checks first byte
- Goes into test case loop
- Sets GPIO pins for condition



# Example Loop Breakdown

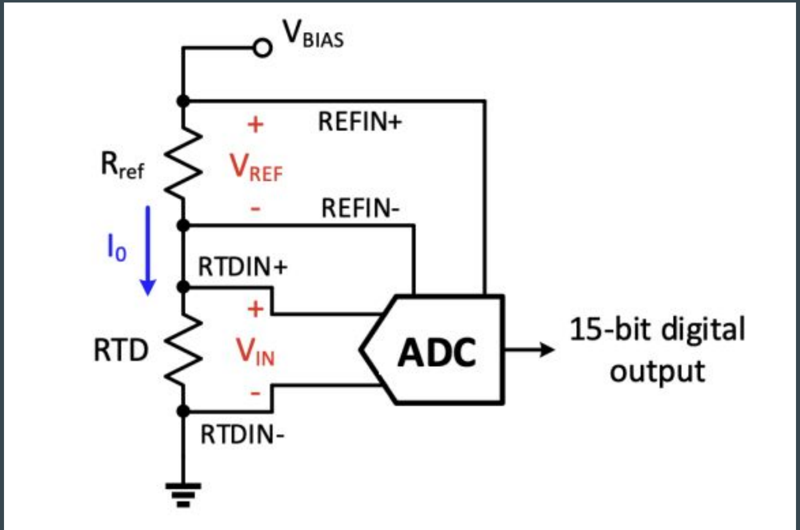
- Example of Open test case
- Waits for 3 bytes
  - Processes first byte
- Sets associated GPIO pins
- Send confirmation string for debug
- Sets two LEDs for user confirmation

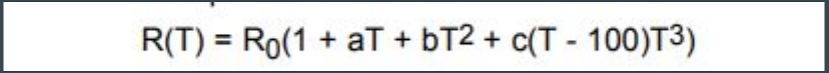


```
else if (temp[0] == '1') {
    UART_sendString((uint8_t *)"1: Open circuit case selected\n\r");
    //UART_sendString((uint8_t *)"Received first byte 1\n\r");
    GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);


    //setting pins for an open case
    GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN5); //P1.5 Low
    GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN4); //P1.4 High
    GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN3); //P1.3 High
    GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN2); //P1.2 Low
    UART_sendString((uint8_t *)"GPIO pins 1.4 and 1.3 set high\n\r");


}
```

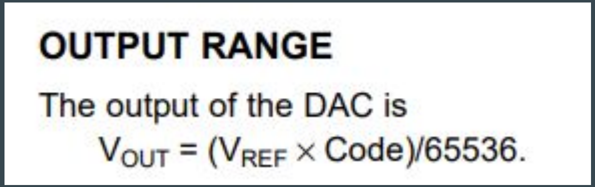
# Temperature Simulation

- Find voltage drop across RTD
- Calculate DAC code needed to produce voltage
- Store DAC code as two bytes
- Load code into DAC one byte at a time





$$R(T) = R_0(1 + aT + bT^2 + c(T - 100)T^3)$$



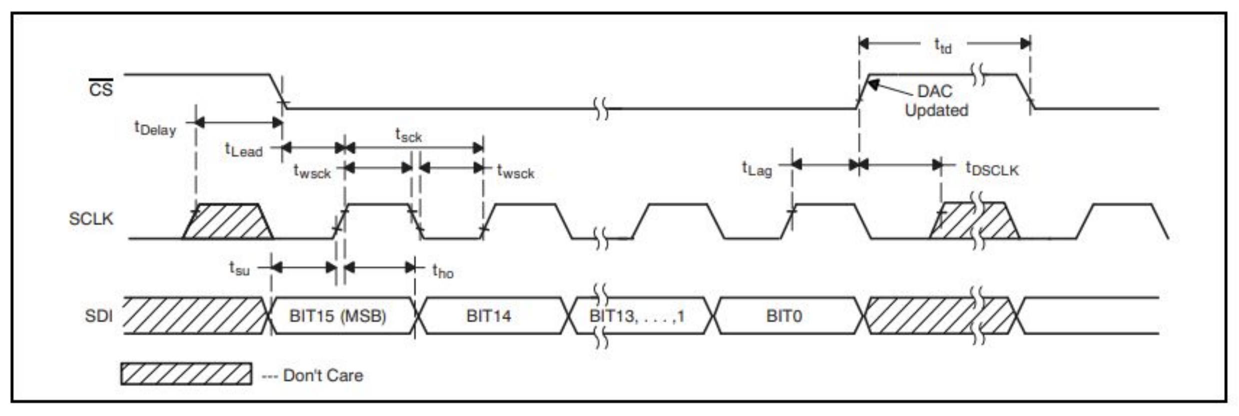
**OUTPUT RANGE**

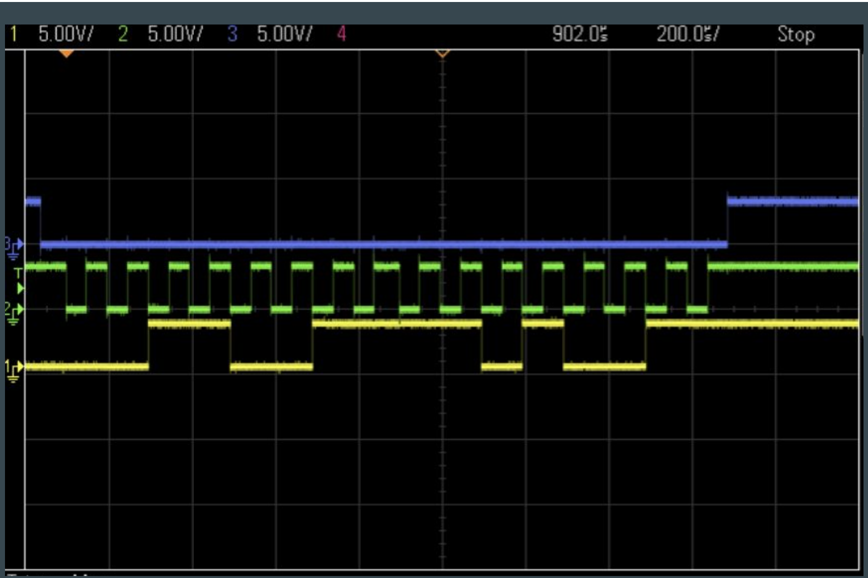
The output of the DAC is

$$V_{OUT} = (V_{REF} \times \text{Code})/65536.$$

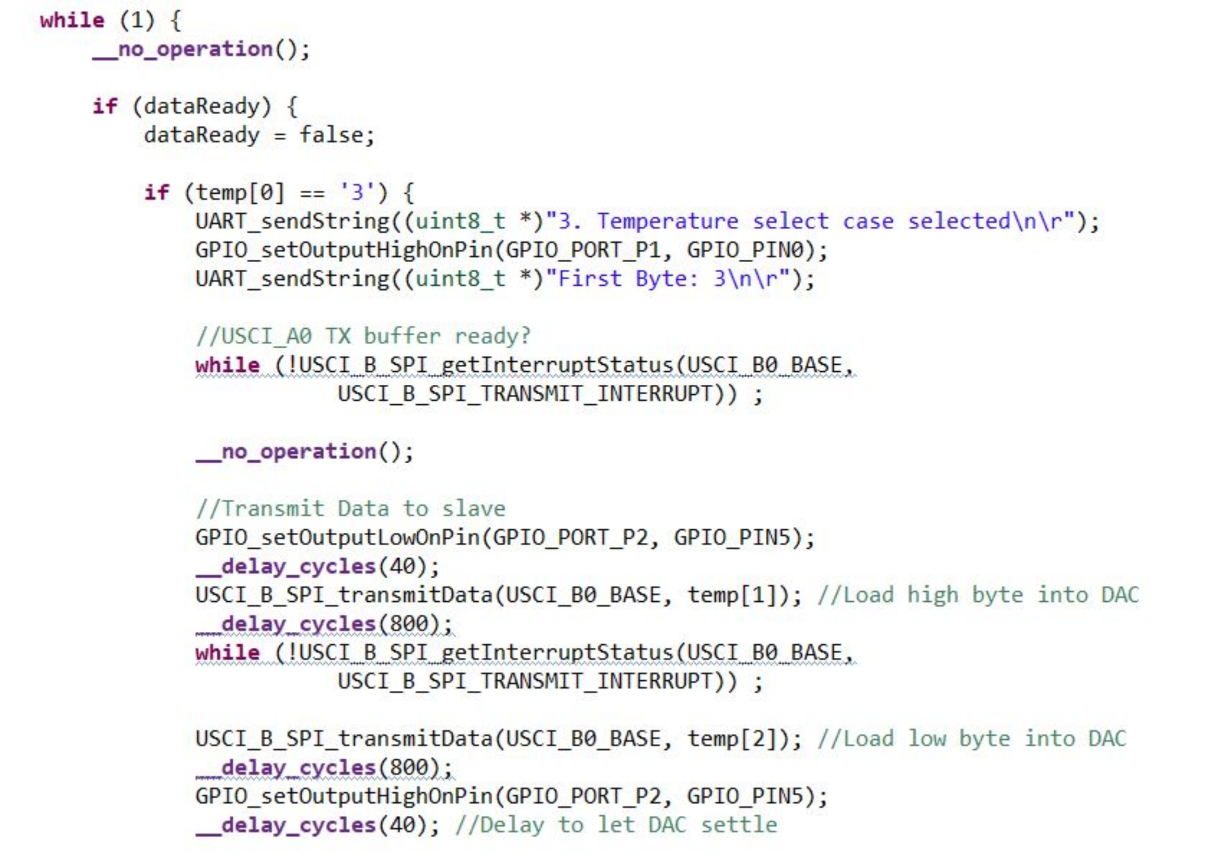
# Loading the DAC

- Set chip select low
- Send data on falling edge
- Read data on rising edge
- Latch data when chip select is set high





# DAC Output Script

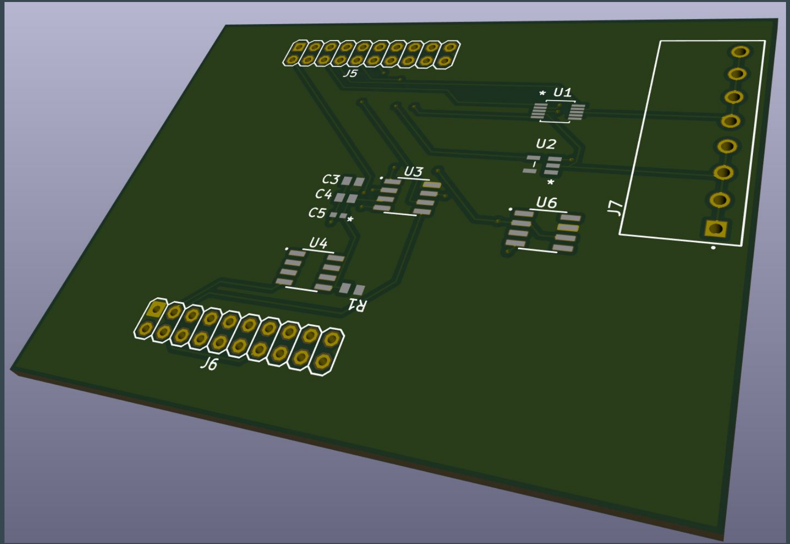


```
while (1) {
    __no_operation();

    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
```

# Hardware

# PCB Design Requirements

- Data lines need to be stable
- Power into sensitive components needs to be clean
- Outputs easily accessible and usable
- Able to be connected to MSP430 Launchpad
- Avoid backfeeding during short and open circuit testing



# Results

# Progress / Outcomes

- Software and hardware have been developed
- Requirements met
  - Uses standard USB
  - Test open, short and over-voltage conditions in RTD
- Requirements partially met
  - Test Modbus communication
- Requirements not met
  - Simulates temperature value
    - Measure accuracy of MAX chip
  - Test surge protection

# Future Work

- Need to be able to sweep through all tests
- Test Surge protection
- Test Modbus communication protocol
- Incorporate faster connection changes for testing

# Questions?