Precision-controlled Temperature Controller Module Proposal

Ryan French, Montana State University May 4, 2020

1 Purpose

This proposal offers a preliminary design for a modular, programmable temperature controller that can be easily integrated into existing microcontroller projects. The ultimate goal of this project is to achieve temperature control with an external Peltier thermal pump. The current state of the design will be detailed in the following sections.

2 Setup and Descriptions

A full schematic is found in appendix A.

The proposed circuit currently contains 4 major integrated circuits. The microcontroller which handles all data is a 16-pin MSP430FR2311, which will be referred to as "master" in this document. The master handles the other three ICs: a real-time clock, an 8-line LCD display, and a thermoelectric pump driver.

The thermoelectric pump driver is a MAX1968, a high-efficiency switch-mode driver. This IC takes an analog input which sets the thermoelectric cooler (TEC) current. This input is centered around 1.5V, so that inputs above 1.5V are considered "cooling" and inputs below 1.5V are "heating." Built into the MAX1968 is a status output: an analog signal which corresponds to the current through the TEC. This signal varies from 1.5V (V_{ref}) to 3.3V, with 1.5V corresponding to no current.

The real-time clock is a Maxim Integrated DS1337 I^2C serial real-time clock, an 8-pin, low-power clock / calendar with programmable time-of-day. The current design incorporates a cell battery for backup. A 32.768 kHz quartz crystal is required for proper functionality, though it should be noted for future revisions that there exist packages with an integrated crystal to simplify the design work.

The current LCD part is listed as DOGXL160-7, which is a high-contrast LCD with I^2C interface. This LCD is subject to change in the future, however, as a goal of this project is to minimize size of the module.

A 1x5 Molex connector was chosen for a connection to a 5-button keypad. This keypad will have "up", "down", "left", "right", and "select" buttons for navigating around the LCD screen to change values.

Two other important features of this module are the on-board reset switch for rebooting the MCU and a 1x2 Molex microconnector set up to act as a programming port for the on-board master. Early revisions will most likely also include a 1x2 generic header to make attaching wires easier. While the ultimate goal of this module is to work "out of the box," including a proper programming port is a great feature for making modifications to the system when part of a larger project. The ideal way to do this is with a compilation toolchain on a microprocessing unit that manages the project's subsystems (i.e., a Raspberry Pi). The programming port of this module could be directly connected to processor pins so the user will only have to manage code on the microprocessor and offload it to the master MCU.

3 Firmware Methods

As it stands, the master MCU must communicate simultaenously with four processes. In such cases, it is not always wise to run a sequential loop over process methods, as this can waste a lot of clock cycles on unncessary data requests. This is why the proposed approach to maintaining smooth operation is protothreading.

3.1 Protothreading: Brief Overview

Most microcontrollers only have one thread to execute code on, and therefore can't do parallel tasks. A usual way of mitigating this is to add extra MCUs, so each does one or two semi-laborious tasks. Simulating threads can be done, though, by exploiting some features of C (thanks to Adam Dunkels for discovering this!) Protothreads provide sequential flow without state machines or full-threading. Originally developed for memory-constrained embedded systems (each protothread only has a 2 byte overhead!), this method will be perfect for freeing memory for some of the more heavy data containers that the master must allocate.

3.2 Protothread 1: Temperature

When this thread executes, it must do several things: read temperature, compare the temperature to the set temperature, check current trend in temperature (we'll come back to this), adjust analog output to the TEC driver, and verify that the TEC current is at a safe value. In more detail:

- Read the temperature, T, from the master's internal temperature sensor.
  - In the future, swap for RTD and do some sort of averaging.
- Compare T to the previous temperature readings. Using this along with the intended temperature, adjust the analog output that controls the TEC's current. This analysis will be done with a PID algorithm.
  - Analog output is handled by PWM, of course. One could write a library to handle the duty cycle of the output, but there already exist several great analog libraries for this particular MCU. These libraries use the master's internal clock to time the digital pulses.
  - There are better ways of sending this signal to the driver, though. That will be addressed in section 4.2.
- Check for over-current on TEC by reading the analog output of the TEC driver's status bus. If current is higher than safety limit, shut off the TEC.

3.3 Protothread 2: Keyboard

When this thread executes, it scans all the digital input lines from the keypad. If a press occurs, store the press data and alter the LCD accordingly. The master then creates a protothread "wait until" (PT_WAIT_UNTIL) which essentially pauses the execution of the current thread until a given condition is met. The condition in this case will be a timeout (user hasn't pushed a key in a long time).

3.4 Protothread 3: Time

When this thread executes, the master addresses the RTC over the I^2C bus to retrieve all datetime data. The master will then analyze the data and store it. Not much more to it.

3.5 Protothread 4: LCD

When this thread executes, all *read* data will be (re)displayed on the LCD. In this case, "read data" refers to all data that isn't static or set by the user. The main screen of the LCD shows the current date, time, temperature, TEC current, set temperature point, and an arrow representing heating/cooling. A settings icon is found in the corner. The user can click here to change values for the temperature module. In the following images, I give an example of a main screen and a settings screen. With the new values from settings, the controller is now in an unsafe current state. The third image shows how the LCD would inform the user.

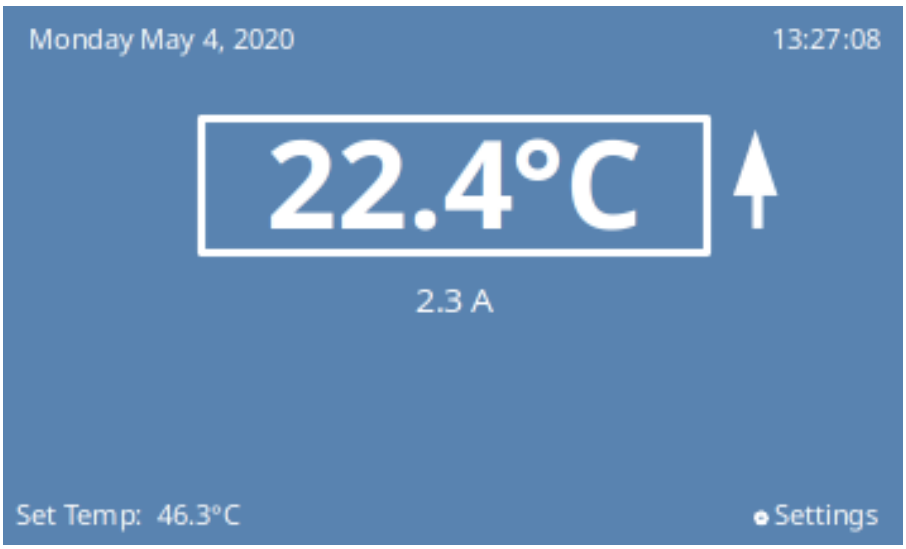

Figure 1: The LCD screen. Date and time are displayed on top, the current temperature is in large font, with the TEC's current displayed underneath. The arrow to the right indicates that the system is heating up. In the bottom left is the user-set final temperature. The user can use the buttons to navigate to the bottom right to enter the settings screen.

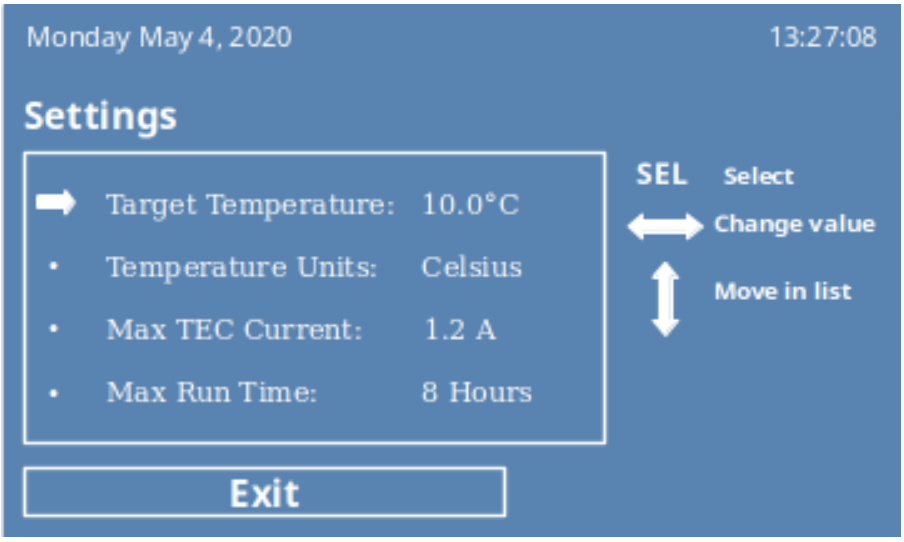

Figure 2: The settings screen, where users can very easily choose temperatures/currents/units.

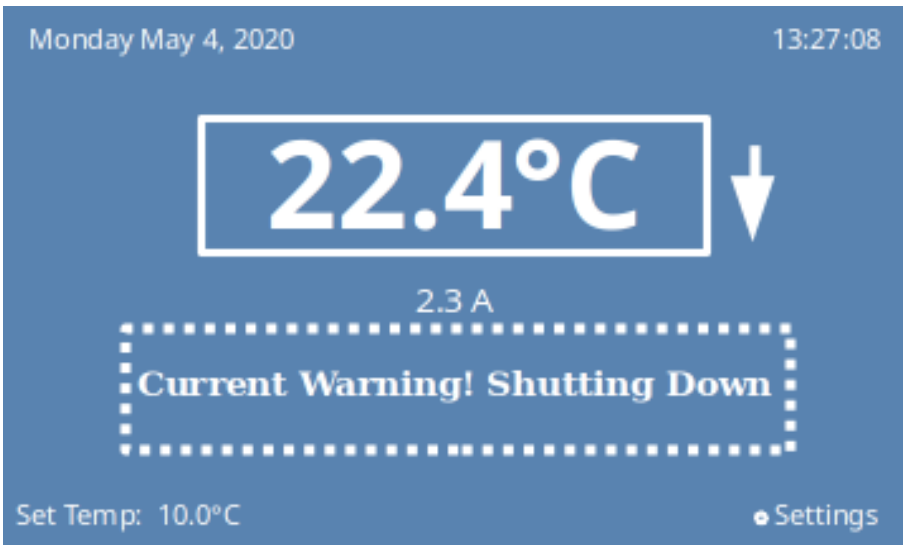

Figure 3: Now that TEC current is higher than the user-specified max, the display warns the user and the MCU starts a shutdown routine.

The bulk of the work in writing to the LCD is alignment. This initial proposal would instead begin with a very basic UI. Moving forward, this UI may necessitate further changes to the board (see section 4.3).

4 Additional Notes

4.1 General Improvements

While initial implementations of this temperature controller may be geared toward hobbyists and tinkerers, there will need to be major alterations to make this module feasible for any professional work. The first thing that should be addressed after initial development is the position of the Peltier device. A standard wire-to-board connector should placed near an edge of the PCB to extend the module's use to a variety of Peltier devices (products in the Molex "-fit" line hold securely and are rated for the required high current loads).

Of course, if the Peltier device is away from the board, that means that the temperature measurements must be made in a different way. For accurate temperature measurements, an RTD should be considered. If that is the route that the project should take, it will need to be determined if the master's ADC resolution is sufficient or if an external ADC will need to be brought on the board.

Finally, if this project is to be seriously considered for modularity, it will be necessary to add an external communication port. However, this will require some careful thought, because if a connection is added to the existing I^2C bus, this could lead to address-space collision.

4.2 Increasing Efficiency of Peltier

It is necessary to drive a TEC with an analog signal, and that's what the TEC driver does for this project. However, the signal originating from the master is not analog. It would be great to see some buffering between the PWM signal and the TEC driver input. This will also decrease the power needed to run the module.

4.3 Evolving the User Interface

With time, it isn't difficult to create a nice-looking UI on a dot LCD. However, a lot of memory is used in creating graphics. Some LCD drivers come with additional RAM space to store these graphics, but it would make more sense to store any UI graphics on-board. With that in mind, it's probably a good idea to add memory in the future. A simple EEPROM chip to hold this data would probably suffice.

A Proposal Schematic

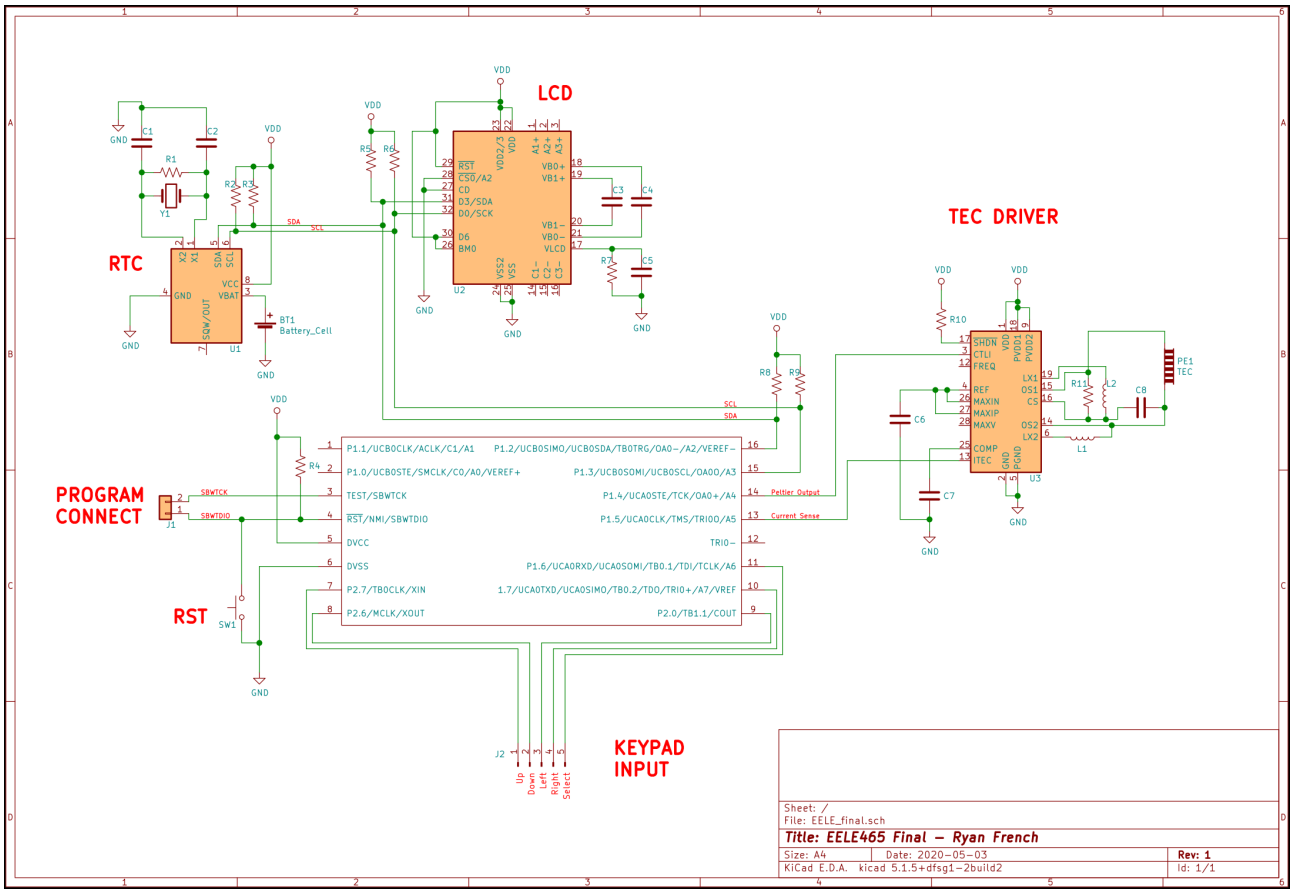

Figure 4: Proposal schematic