Final Lab Report

Ryan French
ryanfrench3@montana.edu
Graduate Student, Department of Physics
Montana State Physics University - May 2, 2020

1. Introduction

This lab is designed to demonstrate all knowledge attained during an undergraduate program in electrical engineering, including communication protocols, circuit design, and internal microcontroller data methods.

In the final setup, a master microcontroller is used to communicate with multiple systems:

- LED bar graph
  - Controlled through a slave MCU. Communication is handled by I^2C communication protocol.
- 4x4 membrane keypad
  - Input from the keypad are handled through the master MCU. Readings are triggered by a physical interrupt.
- 32-character LCD
  - Conrolled through a slave MCU. Communication is handled by I^2C communication protocol.
- LM92 digital temperature sensor
  - Readings are handled through the master MCU using I^2C communication protocol.
- LM19 analog temperature sensor
  - Readings are handled through the master MCU using analog readings.
- DHT22 temperature and humidity sensor
  - Readings are handled through the master MCU using a single-wire digital protocol.
- DS1337 real time clock
  - Readings are handled through the master MCU using I^2C communication protocol.
- Thermoelectric heat pump (TEC)
  - Controlled by the master MCU, using PID to maintain consistent output temperature.

The ultimate goal is to control the TEC, using all peripheral devices to ensure proper functionality.

Note: In this report, all references to master and slave MCUs correspond to MSP4302355 and MSP430FR2310, respectively.

2. Purpose & Theory

2.1. LED Bar Graph

A slave controls a series of 8 LEDs, contained in bar graph module. In the three states of the TEC the bar graph will display different patterns. While heating, it displays a rightward-moving graph; while cooling, it displays a leftward-moving graph; and while off, it displays a blank graph. These graphs are triggered by

the the master: upon a state change, the new state is sent to the slave. These states are represented by an 8-bit unsigned integer (0=off, 1=cooling, 2=heating).

2.2. 4x4 Membrane Keypad

This membrane keypad is a generic input device, and are available from many online vendors. They're usually listed as keypads for Arduinos. Their operation is simple: 8 lines are connected to rows/columns (one for each row and one for each column). The wires at crossing of rows/columns are connected through a pushbutton. On the 4x4 keypad, there are buttons for digits 0-9, *, #, and letters A-D. When a button is pushed, it connects the column and row wires.

Reading input from the keypad is simple: the row wires are held high and the column wires are held low through a pull-down resistor. When a button is pressed, it pulls one of the column pins high. There are two main methods to detect a button press:

1. Continuously all four column pins for a logical high.
2. Create a physical interrupt and poll the interrupt pin.

Obviously, method 2 is much easier. To achieve this, the four column pins are connected to a series of OR-gates whose output is tied to an input pin on the master. This pin can either be set up as a physical interrupt or can be polled over a certain interval (the latter method was chosen for this lab, for reasons explained in section TODO)

2.3. LCD

The LCD used in this project is a dot matrix module controlled with a 3-state I/O data bus. A slave controls the write operations after receiving data from the master through I^2C. Specific implementation can be found in section 3.3.

2.4. LM92 Temperature Sensor

The LM92 is a 12-bit temperature sensor integrated circuit with an accuracy of ± 0.50°C (± 0.9°F) in the range of 10°C to 50°C and a resolution of 0.0625°C. Its power-up state defaults to a range of 10°C to 64°C and its internal pointer points to the temperature register; these settings work for this lab's purposes, so no initial setup is required. Temperature readings are requested by the master via I^2C. The LM92 has a variable I^2C address, which is controlled digitally by its two address-input pins. For this lab, both of these pins are tied to ground.

Temperature readings from the LM92 account for hysteresis, so there is no need to calculate hysteresis coefficients like in the case of an RTD (coefficients are determined by increasing and decreasing temperature and recording the varying R, $\frac{\partial R}{\partial T}$, and $\frac{\partial^2 R}{\partial T^2}$ values, where R is the raw reading).

Readings from the LM92 must be multiplied by its resolution (0.0625) to record its temperature reading in degrees Celcius.

2.5. LM19 Temperature Sensor

The LM19 is a precision analog output integrated-circuit temperature sensor with an operating temperature range of -55°C to 130°C. While its transfer function is predominately linear, it has a slight parabolic curvature. Its parabolic transfer function is defined as:

$$T = -1481.96 + \sqrt{2.1962 \times 10^6 + \frac{(1.8639 - V_O)}{3.88 \times 10^{-6}}} \tag{1}$$

where V_O is the output voltage. When using equation 1, the accuracy is ± 2.5°C (4.5°F).

2.6. DHT22 Temperature & Humidity Sensor

The DHT22 is a dual temperature and humidity sensor module manufactured by Adafruit. It contains a capacitive humidity sensor, a thermistor, and a couple of ICs which handle analog-to-digital conversion and digital output. It is a very basic and slow, though it has a slightly better accuracy than the LM19 sensor (± 2°C; ± 3.6°F). The range of temperatures this module is good for is 0°C to 50°C. The humidity sensor best operates in a range of 20-80%, with an accuracy of ± 5%.

The DHT22's digital output is a special protocol which is best handled by a library (it involves waiting for a low signal, then a succession of awaits, where readings must be made at a certain frequency).

A significant downfall of this module is its sampling rate: queries can only be made every 1 to 2 seconds.

2.7. DS1337 Real Time Clock

The DS1337 is a serial real-time clock integrated circuit controlled through I^2C protocol. It maintains time through an external 32.768kHz quartz crystal oscillator. While the physical setup contains more information, a DS3231 module was used instead due to physical damage to the DS1337 chip.

This DS3231 module is also controlled by a 32.768kHz quartz crystal oscillator and is interacted with through I^2C protocol. Because of its battery backup, it is not necessary to initialize the time every time the circuit is booted. A library is used to simplify communication with the RTC.

The main function of this module is to track the time passed in a given TEC state (i.e., upon a state change, the time passed, Δt, is set to 0. Then with every write to the LCD, the new time is read and Δt is updated).

2.8. Thermoelectric Heat Pump

Thermoelectric heat pumps take advantage of the Peltier effect to create a heat differential across an interface. As far as implemention is concerned for this project, a TEC driver was created based on a very efficient design found in [1]. See the attached schematic for its setup.

To control the temperature, PID functionality must be implemented, which means calculating the PID constants for this setup. An attached PDF contains work I've done previously on PID theory and the methods of calculating these constants.

If you're interested, I have added information on Peltier and Seebek theory in appendix A. This theory is a portion of my specialization, and I find it very interesting. I think it's worth a read.

3. Microntroller Implementation

3.1. Master

The master obviously handles most of the work. After preparing all necessary data containers, the master initializes the RTC and the LCD. The main loop is simple: it only runs the protothreads.

Protothreading is a method of simulating multithreading on a single-thread processor. This is done by assigning a time interval to each process you want the processor to run. To accomplish this on the master, a custom class was created. When instantiated, the objects contain a time interval and pointer to a function. Three objects are created on the master, and the main loop does a check for each thread: if thread's object's time has passed, it runs the function pointed to by the object. The object then resets its timer.

The three protothreads created for the master correspond to methods to handle LCD output, keypad checking, and temperature reading with thread delays of 1000 ms, 100 ms, and 600 ms, respectively. All three functions have void signatures.

The temperature thread function takes averaged reads from all three temperature sensors and saves them to their data container (array of floats). The keypad thread function checks for valid input from the keypad. First, it checks if the physical interrupt is triggered. If it is, it reads the input. The input is run through several checks: if it's a new input, continue else break; if it's a state change, set the state flag accordingly, reset the run time, and send the state data (integer) to the LED slave.

Finally, the LCD thread does two things. First, it converts all numerical data to character data. This is very easy for integers, but for the float data, an algorithm was developed that takes advantage of C type conversion and modulus math. Once the character data array is full, an exclamation point is appended. For an explanation for this, see the section below on the LCD slave.

The decision graph guiding the master's process can be found in appendix B.

3.2. LED Slave

The LED slave is incredibly simple. Upon receiving the state data over I^2C, the slave begins lighting up the LED bar with a pattern using a flip-flop.

3.3. LCD Slave

The LCD slave begins by setting up a data container for incoming characters and initializing the LCD with the main information, with blanks where the data will be contained. The main loop simply polls for I^2C data. When data arrives, the byte is stored in the data arrray and the array's pointer is incremented. If the incoming byte is a '!', then the slave knows that it has received all data. It resets the data pointer, calls an update function, then clears the data array.

The update function transfers the data to the screen. Since the incoming data was already translated into characters by the master, the slave doesn't have to worry about any type conversions. The screen's cursor is set to areas where each data point will be printed. The data is then written to the LCD by overwriting the previous data. This is done to avoid clearing the screen during each write process, which can make the screen appear to flicker.

4. Discussion

This lab's implemention follows so naturally from previous labs that there weren't any large issues in getting everything to work. An issue that did plague the lab was mechanical, and it was caused by excessive noise on the power line. This was fixed by adding appropriate bypass capacitors on the power strips.

If a thermoelectric heat pump was added, this lab would have to be slightly adjusted. In the temperature thread, an extra step would have to be added in which the current temperature would have to compared to the previous temperature. This, along with a PID equation, would be used together to maintain a certain temperature. Another modification that would be made is in reading the keypad. Instead of simply using it to heat and cool, it would be used as an input for a target temperature. This would require a lot more work on the master, as this input could take several seconds. That means that checks would have to be made on this input while also maintaining the other threads as expected. A general idea of how this could be implemented is by creating another protothread that acts as a watchdog. After a given number of cycles for this thread, if another relevant key isn't pressed, then the input data would be cleared.

This lab is easily extensible to real-world applications. For example, all of these methods are important in machines that must maintain multiple subsystems at once. In my previous work, it was necessary to control the temperature of a stator while also monitoring external errors and a motor's position. I used one microcontroller to do this by using protothreads like were used in this lab: one thread read the RTD, compared the temperature through a PID function, and adjusted the heater; another thread polled an I^2C line for incoming data and/or error codes from a master multiprocessor; and one thread that requested the position of a stepper motor through an encoder, which was then communicated to a specialized subsystem that ran that motor.

A. Peltier Effect Theory

The following discussion is derived from [2] and [3].

A.1. Thermoelectric equation

Starting with transport equations, we can derive a law describing temperature distribution in a thermoelectric heat pump. Thermoelectricity is a theory whose full explanation requires quantum mechanics and fermionic statistical mechanics, however, a general law can be derived using some approximations. First, because fermions in a lattice respond primarily to local potentials, it can be assumed that macroscopic potential gradients vary linearly on the scale of the lattice. Therefore, we can write conservation laws for current fluxes in conductors/semiconductors as:

$$J_e = L_{11}\left(-\nabla \Phi\right) + L_{12}\left(-\nabla T\right) \tag{1}$$
$$J_q = L_{21}\left(-\nabla \Phi\right) + L_{22}\left(-\nabla T\right) \tag{2}$$

where J_e is the charge current flux and J_q is the heat current flux. These fluxes are coupled through the L_{ij} tensor. Φ is the electromotive force, i.e., the electrochemical potential divided by unit charge.

Assuming an isothermal conductor, the heat current,

$$J_q = \Pi J_e \tag{3}$$

is proportional to electrical current via the Peltier coefficient, Π. When two materials with different Peltier coefficients are brought into contact, it can be shown that the heating, Q, at the junction is equal to

$$Q = \left(\Pi_2 - \Pi_1\right) I \tag{4}$$

where I is the electrical current. We can eliminate Φ in equation 1 and express the heat current as

$$J_q = \Pi\, J_e - \kappa_e \nabla T \tag{5}$$

where the electrical contribution to thermal conductivity is

$$\kappa_e = L_{22} - \frac{L_{12}\, L_{21}}{L_{11}} \tag{6}$$

Combining equation 5 with the first law of thermodynamics, we get the equation determining the temperature distribution in a thermoelectric heat pump:

$$C_P \frac{\mathrm{d}T}{\mathrm{d}t} = \kappa \left(\nabla^2 T\right) - \mathcal{K}\, \vec{J}_e \cdot \nabla T + \frac{J_e^2}{\sigma} \tag{7}$$

where σ is the electrical conductivity of the material, C_P is the specific heat at constant pressure, and $\mathcal{K}$ is the Kelvin (Thomson) coefficient. There are two interesting limits to take with this equation. In the first limit, assume that the spatial variation of temperature is zero, i.e., the conductor is held at constant temperature. All spatial derivatives go to zero and we arrive at

$$\begin{aligned} C_P \frac{\mathrm{d}T}{\mathrm{d}t} &= \frac{J_e^2}{\sigma} \\ P &= \frac{1}{\sigma} J_e^2 \\ \boxed{P = I^2 R} & \end{aligned} \tag{8}$$

which is simply Joule heating, as we would expect. In the second limit, assume that the temperature is time-independent and varies linearly. The second spatial derivative and the time derivative get thrown out. The solution, $T(x)$, is

$$\begin{aligned} \mathcal{K}\,\vec{J}_e \cdot \nabla T &= \frac{J_e^2}{\sigma} \\ \nabla T &= Sgn(\vec{J}_e \cdot \nabla T)\,\frac{J_e}{\sigma}\,\frac{1}{\mathcal{K}} \\ \boxed{T(x) &= \frac{1}{\mathcal{K}}\,Sgn(\vec{J}_e \cdot \nabla T)\,I\,(x - x_0) + T_0(x)} \end{aligned} \tag{9}$$

where $Sgn(x)$ is the signum function (returns the sign of argument) and $T_0(x)$ is the initial temperature distribution. As expected, in the limit that current goes to zero, the initial and final temperature distributions are the same. It should also be noted that the temperature difference increases linearly with current. As a final insight, consider the direction of current: if current flows in the direction of increasing temperature, the signum function equals 1, which means that temperature difference will increase. Similarly, for current opposing the temperature gradient, the temperature difference will decrease.

B. Master Flowchart

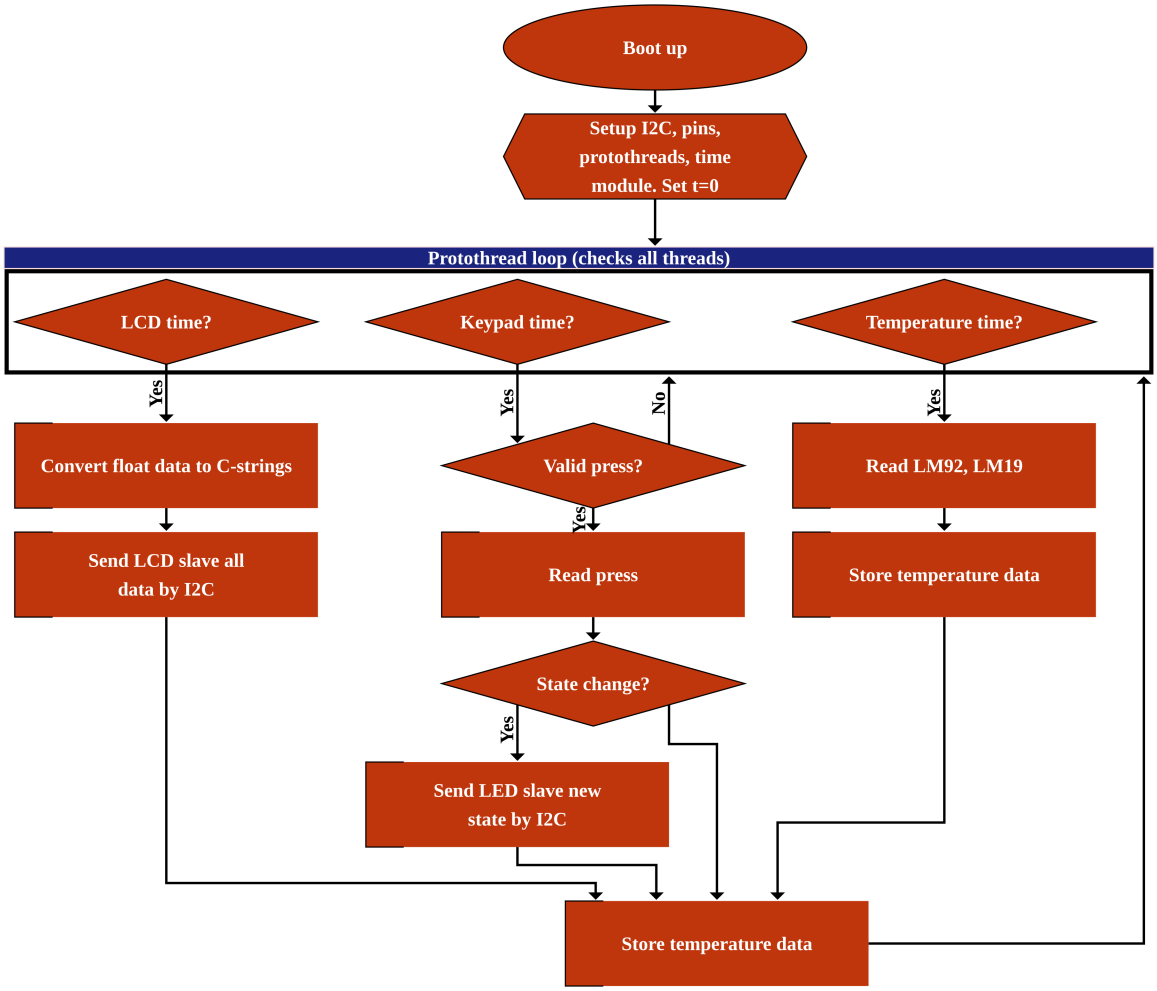

Figure 1: *MSP430FR2355 Flowchart*

References

[1] N. Albaugh, “Optoelectronics circuit collection,” 2001.

[2] *Solid State Physics*. Harcourt College Publishers, 1976.

[3] “Seebeck and Peltier Effects.”