5780_Mini Project: A Simple Cooling System

Freddie Rice, Shem Snow, Edison Yang

April 25, 2024

Abstract

The development of a cooling system for a fluid mixer. It utilizes a USART interface with a keyboard to allow concentration control of the mixture by activating digitally controlled solenoid valves. Temperature regulation is achieved with embedded software that utilizes a real-time sensor, and pump, to adjust flow rate through a heat exchanger as the temperature readings change. This strategy enables rapid cooling in response to elevated temperature.

Purpose

This project is a cooling system that uses various concepts learned throughout the semester, including the USART communication protocol, timers, interrupts, PID control, and analog-to-digital conversion (ADC). Its design is shown in Figure 1 and our implementation of it is shown in figure 2.

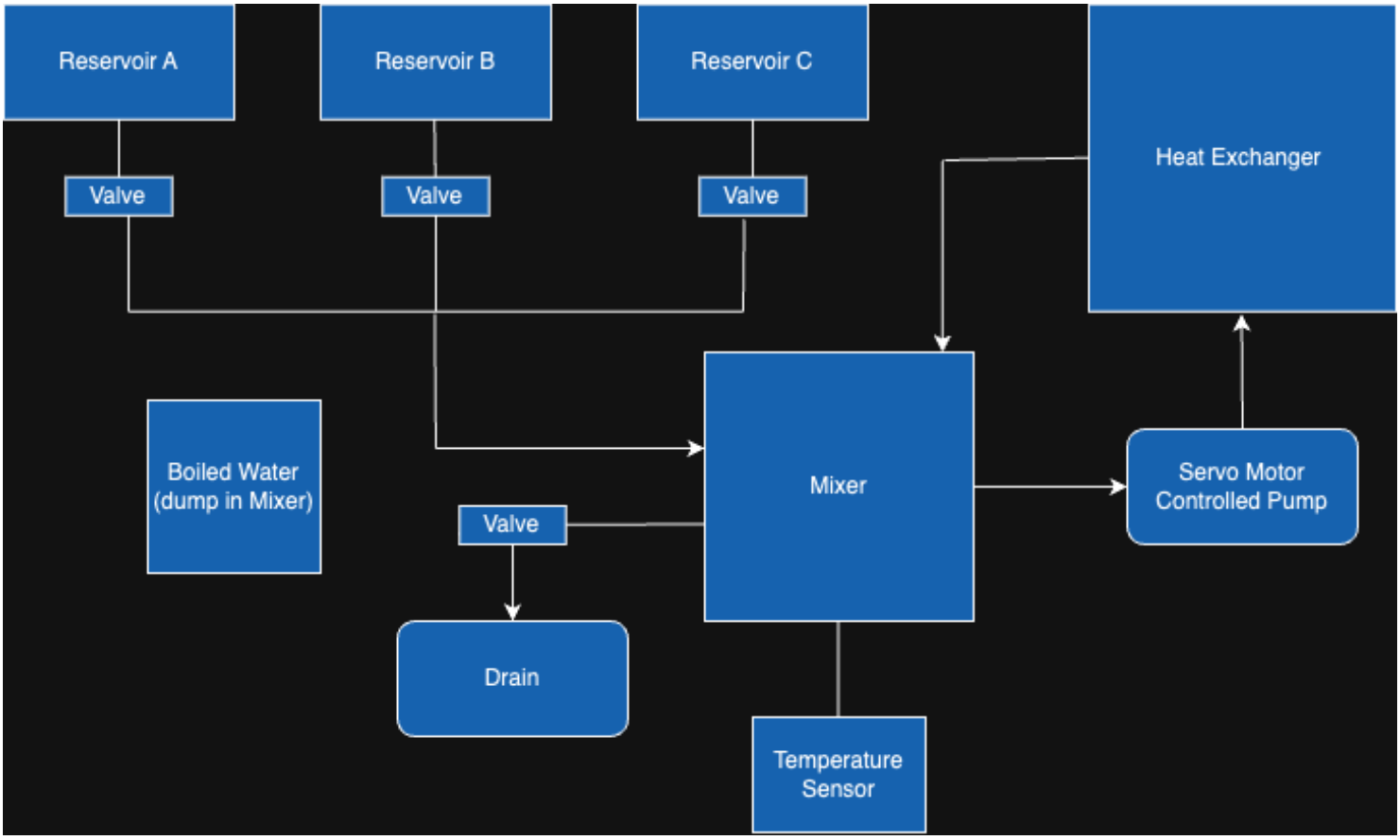

Figure 1: Functional Diagram of the Cooling System: A computer interface which communicates via USART is used to provide users digital control of solenoid valves that connect reservoirs of liquid to the main mixer tank. Temperature regulation is managed through a DS18B20 sensor and a submersible water pump that is powered when temperature thresholds are exceeded.

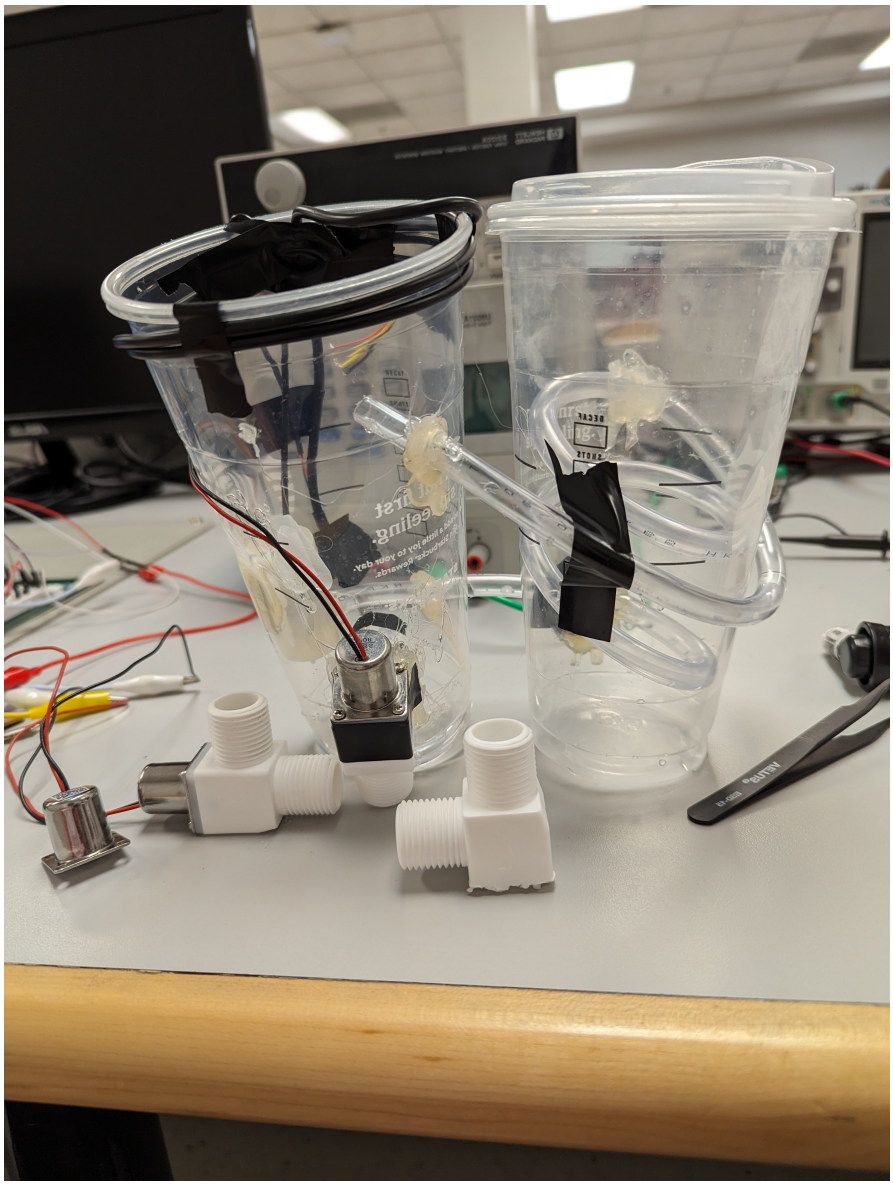

Figure 2: Left cup is the mixer. Right cup is the heat exchanger. The three valves for reservoirs A, B, and C, are on the table in front of the two cups. The valve for the drain is glued into the mixer. The temperature sensor is taped to the inside of the mixer. The pump is stuck in the mixer with tubing to transfer liquid out of it and through the heat exchanger then back into the mixer.

The purpose of our system is to maintain the fluid in the main tank (Called "Mixer" in the diagram) below the threshold temperature of 20° C. It does this by pumping water from the mixer through a heat exchange where it will be cooled.

This works according to the "specific heat transfer rate" equation:

$$\dot{Q} = \dot{m} \cdot c \cdot \Delta T \tag{1}$$

that says the rate of heat transfer ($\dot{Q}$) is proportional to the mass flow rate ($\dot{m}$). The other two values in the equation (specific heat and temperature gradient across the two systems) are relatively constant and can be ignored for our purposes. In conclusion, **the more mass that flows across a space with a temperature gradient, the faster that mass will cool**.

A notable fact about our project is that when we measure a temperature, that temperature can be classified and the pump can react accordingly:

1. Low Temperature: the pump can stay off because the mixer is at an optimal temperature.
2. Medium Temperature: the pump turns on to a low speed.
3. High Temperature: the pump turns on to a medium speed.

4. Even Higher temperature: the pump is turned to a high speed by applying its maximum rated power supply.

Functionality

We Separated the concerns of our project into four different .c and .h files.

main:

Initializes all the peripherals we use (timers, GPIO A, B, C for LEDs and pins for valve control, pump driving, temperature sensing, and ADC operation) and configures their pins for our purposes. Then starts running timer 2 which, when it expires, triggers the temperature sensing code that interrupts the main loop which is constantly reading the USART communication lines to control the valves.

motor:

Calls the TempSense() method and compares the result with the desired 20° C and based on the error, uses a Proportional Integral (PI) to control loop to drive an output according to the equation:

$$output = (K_p * error) + K_i * error\_integral \tag{2}$$

The output is fed into the pump as the source voltage. The results is that when the error is large (the observed temperature is far from the desired temperature) the pump will be turned onto high speed. When the error is low then the pump will be on low speed or even off.

temp:

The temperature sensor uses the "one-wire" communication protocol and it is triggered by a timer that interrupts the "Valve_Control()" method. The one-wire protocol has only a single data line connected to a pull-put resistor. A master begins broadcasting messages on that line by pull its voltage low for a brief period then allow it to rise again due to the pull up resistor. The first part to any message is to specify the slave address that should receive the message. We only connected one device so we can save time by not specifying any slaves and instead broadcasting a "skip ROM" command that says "this message is for all slaves". After that the temperature sensor will automatically begin broadcasting the temperature. It takes time however so a delay is enforced to allow the temperature sensor sufficient time to report.

valves:

Valve control is achieved using the USART communication protocol with the keyboard on a laptop. We used the puTTY program communicating on serial line 3 with a speed/baud rate of 115200 to detect key presses that would open and close our valves. A prompt is broadcasted from the flashed program then the protocol is initiated to listen for key presses that correspond to each valve. Unexpected key presses have no affect but the proper key presses progress the program to where it listens for the 'open' or 'close' command which will trigger two GPIO pins connected to an H-bridge to open or close the specified valve.

Set Up

- One H-bridge is needed for each valve and the pump. Their connections are shown in figures 3 and 4.
- Common lines for ground and 5 V should be used from the STM board.
- Two pins are needed to control the valves since opening and closing them requires current flow in both directions. We are using the LED pins (PC 6, 8, and 9) to indicate when valves are open and closed. The second control comes from pins PB11, 12, 13. Pairs are PC6-PB11, PC8-PB12, and PC9-PB13. These two pins go into the in1 and in2 pins of the H-bridge.
- The positive and negative valve terminals connect to out 1 and out 2 on the H-bridge.
- PC4 is the USART transmitter pin and should be connected to the USART chip's receiver pin. PC5 is the USART receiver pin and should be connected to the USART chip's transmitter pin.
- The temperature sensor has three terminals. Power (red), Data (yellow), and Ground (black). Power goes to 5v. A 4.7 Ω resistor connects 5 V to Data. Ground goes to ground.

- The temperature sensor also requires that PA8 and PA9 are connected to the data line because messages in the "one-wire" protocol are acknowledged by pulling the data line low.

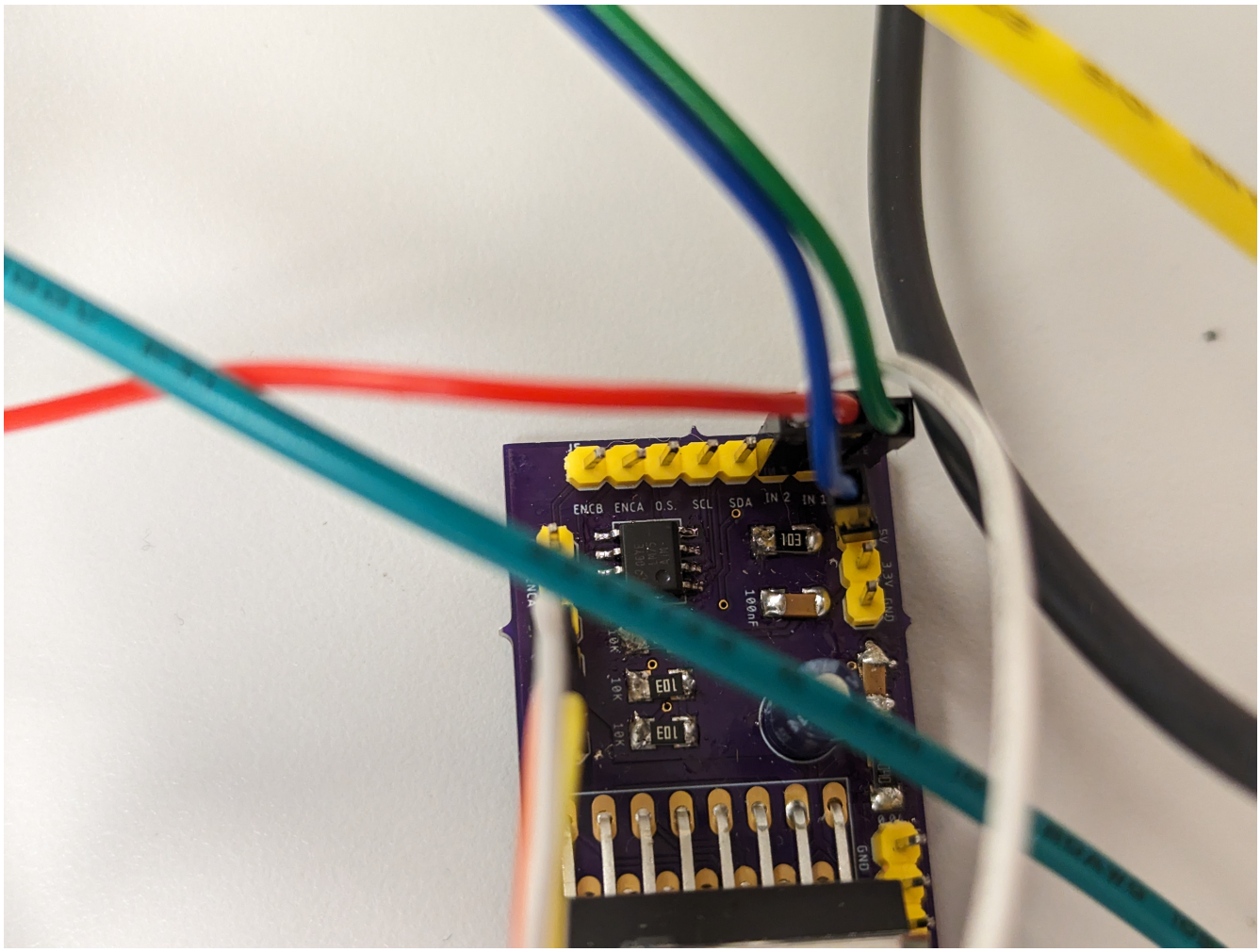

Figure 3: In1 and In2 connect to the two voltage sources for logic on the board. The Enable pint (green wire) is connected to 5V to activate the H-bridge

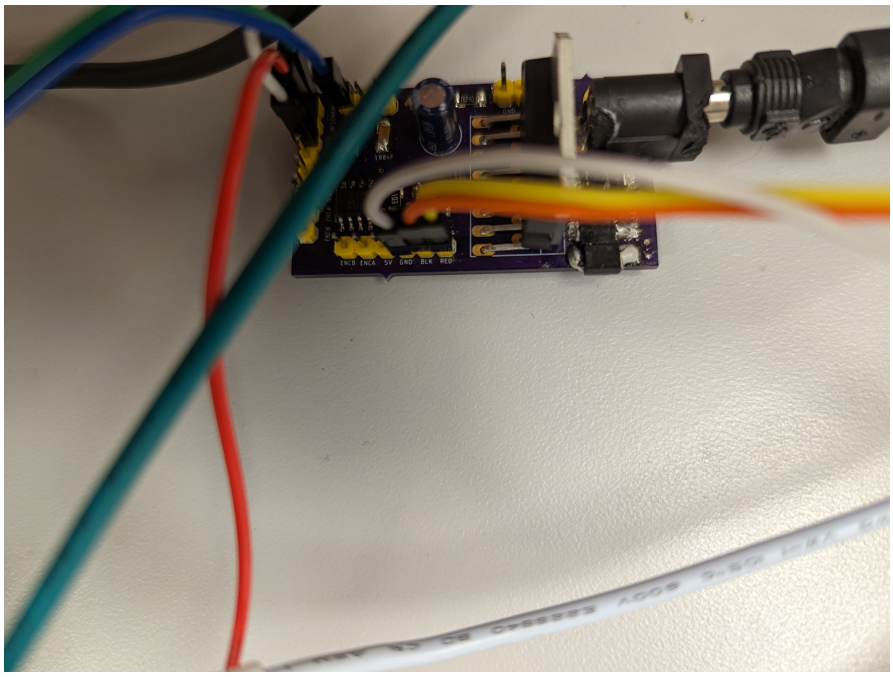

Figure 4: The black and red pins (orange and yellow) connect to the GPIO pins that control the direction or current for the valve or pump. The grey wire is for ground.

Operating Instructions

To start off, connect all the wires according to the "Set Up" section and power each H-bridge. Then flash the program and run puTTY.

The three valves correspond to key presses 1, 2, and 3. You first need to press one of those numbers to specify the valve you operate. Once the program recognized you've specified a valve, it will ask you which operation you want to perform. There are only two operations. Open, which is triggered by pressing 'O', and close which is triggered by pressing 'C'. The LED lights on the STM board will indicate your success. Red is valve 1. Green is valve 2, and Blue is valve 3.

The pump's source of power is automatically adjusted as the temperature changes. To watch is work, you simply need to manually change the temperature and watch the pump react. The higher the temperature, the faster the pump. If you want to read the current temperature, you will need to start a debugging session in the Kiel program. Open a watch window and find the Temperature variable (room temperature is around 24°). Manually change the temperature and watch the pump react.

Alternative Temperature sensing Method

Our first implementation of this system **used an ADC** to read the voltage across a thermocouple to trigger three different pump speeds. We got the code to a point that we are able to control the pump speed by applying different voltages across the thermocouple however the thermocouple we bought was a stupid piece of crap that didn't even work because its resistance did not change according to the environment around it. We literally set fire to it and measured the voltage across it while receiving a variety of currents and no change in voltage when the thermocouple was then put into an ice bath.

If someone were to obtain a thermocouple that actually worked. This project could be done using an ADC instead of the "one wire" communication protocol our digital temperature sensors use. Simply follow these instructions:

1. Open the "main.c" file.
2. Locate the "void TIM2_IRQHandler(void)" method.
3. Inside it, replace the line "PI_update();" with "Sense_Temperature();".

Common Pitfalls

- The temperature sensor is triggered by timer 2 but also triggers timer 6 to incorporate a delay because the one-wire protocol requires time delays to send messages.

If timer 2 has the same priority as timer 6 or the systick timer, which is implicitly triggered by the HAL_delay function, then the program will be deadlocked into a state where the systick is waiting for the temperature sensor to report the current temperature but it cannot be reported because systick has not indicated that the temperature has been read by the sensor.

Therefor the **systick timer should have a higher priority than timer 2** so that the temperature readings can be acknowledged.

- Duplicate variable instantiations should be avoided by using #include statements in main for the .h files.