Johns Hopkins University

Project 2

Serial Transmit of Temperature

Miles Gapcynski

EN.605.715.81.FA19 - Software Development for Real-Time Systems

Professor Doug Ferguson

09/22/2019

Contents

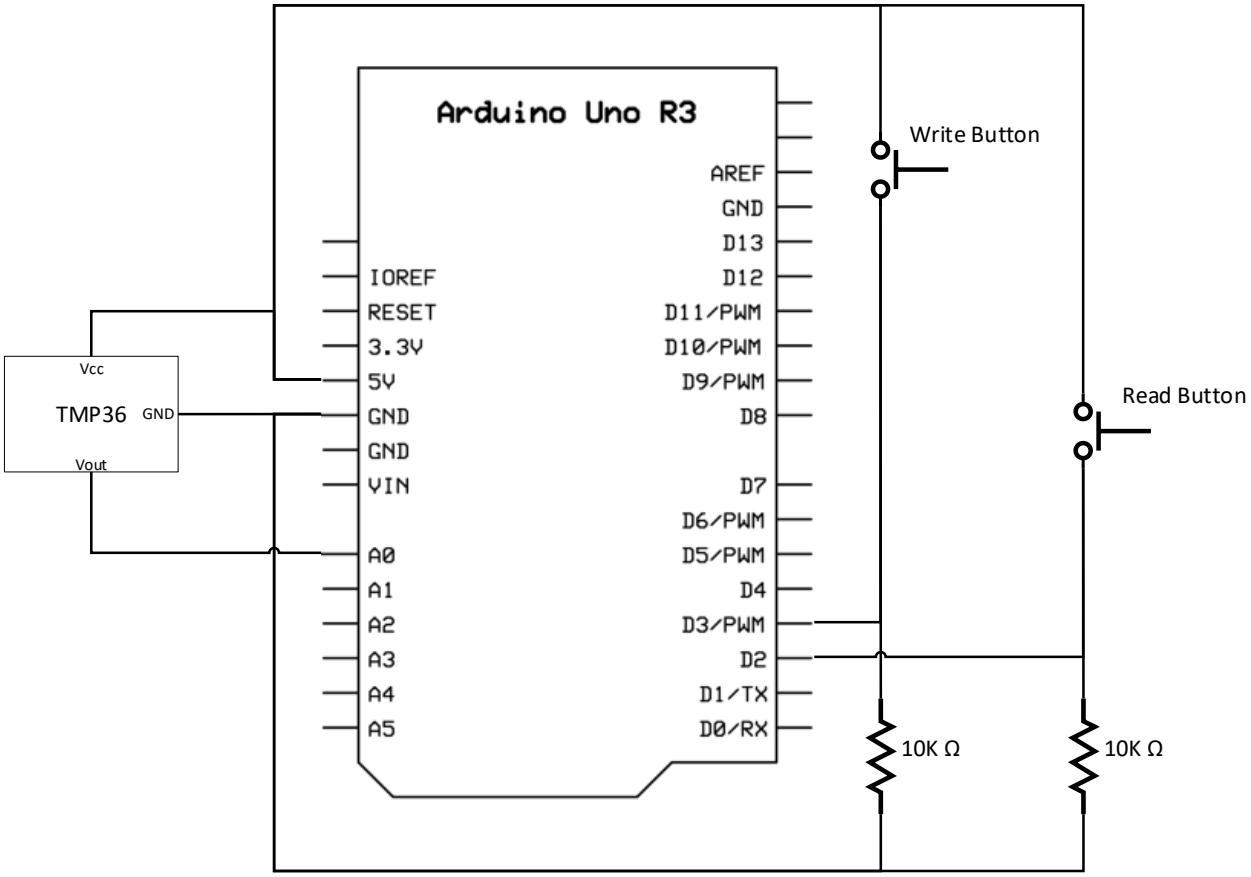
Derived Requirements

The following requirements were derived from the Project 2 Serial Transmit of Temperature document:

- The system shall respond to user input that dictates what mode it's running in, and the system shall support the following modes:
    - Store temperatures
    - Transmit temperatures
- The system shall capture the temperature at a periodic rate of 10 seconds, which shall be driven by an interrupt.
- The system shall convert the temperature to Fahrenheit and persist the data so that it can be read at a later time.
- The system shall persist at least 10 minutes worth of temperature recordings.
- The system shall transmit the recorded time and temperature across Serial (USB) to a host machine.
- The system shall transmit the recoded time and temperature as comma separated values.

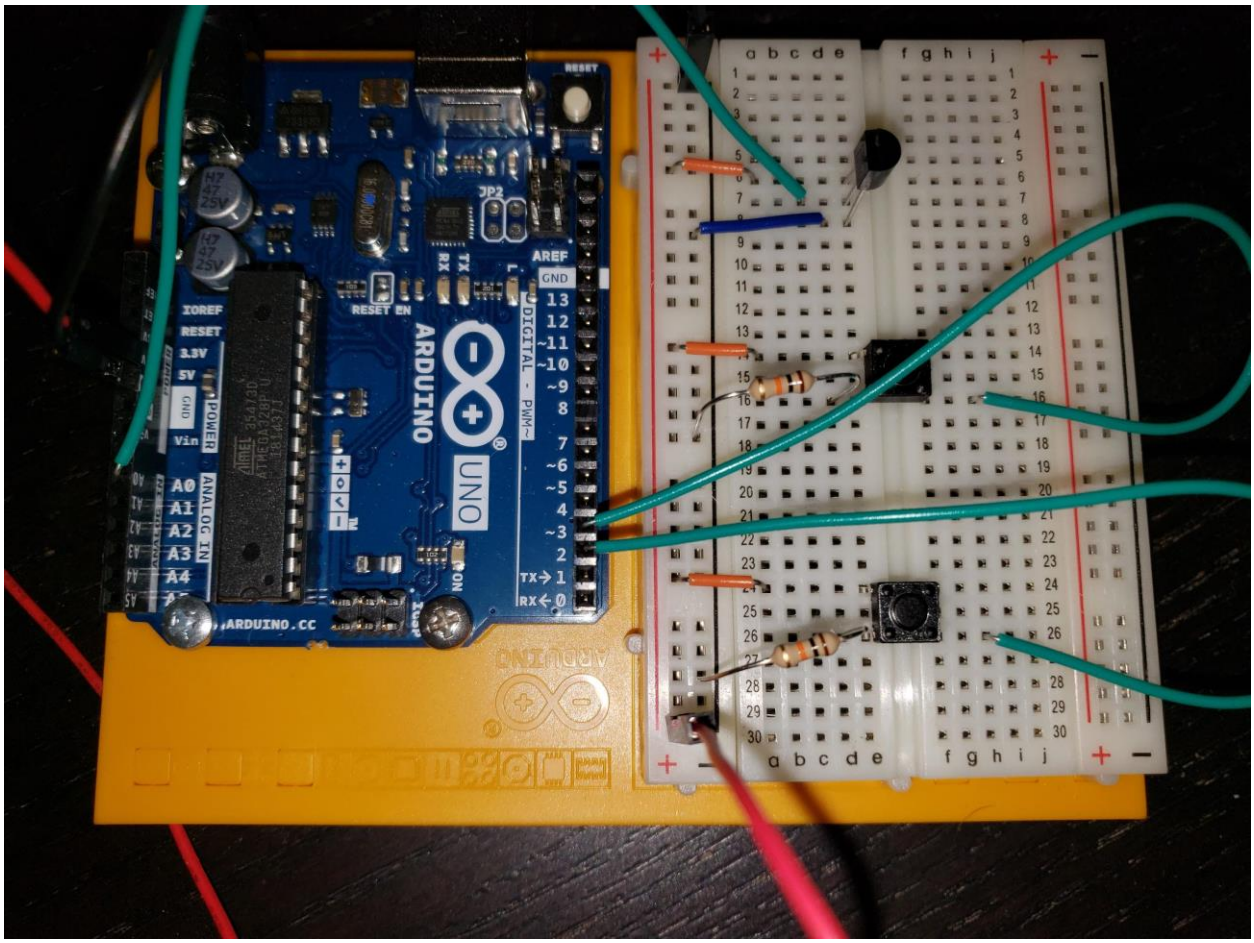
Hardware Design

The following diagram is a schematic of the circuit connected to an Arduino Uno (rev. 3) that will capture temperature data from a sensor:



Board Layout

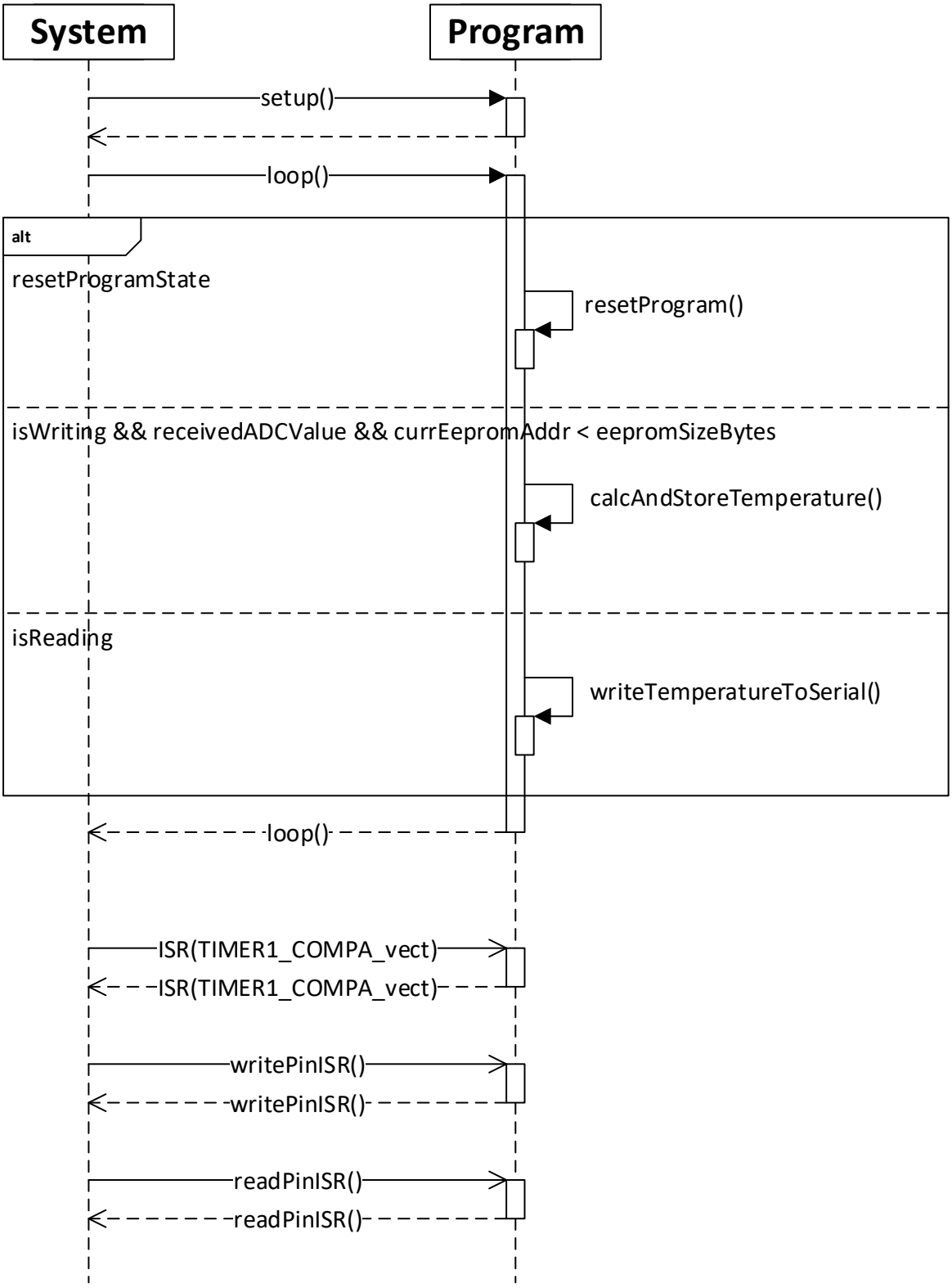
The following picture showcases how the hardware design was implemented using an Arduino Uno (rev. 3) and breadboard:



Software Design and Implementation

Sequence Diagrams

The following diagram is a sequence diagram of the program that reads, stores, and transmits the temperature data:



Temperature Sensor

I used the TMP36 temperature sensor made by Analog Devices to read the temperature. According to the data sheet (https://www.analog.com/en/products/tmp36.html#product-overview), this sensor does not require calibration in order to provide typical accuracies of ±1°C at +25°C and ±2°C over the -40°C to +125°C temperature range. In addition, the sensor has been calibrated directly in Celsius to output voltage linearly proportional to 10mV/°C. The TMP36 Vout is connected to the Arduino Uno's A0 pin to read the voltage value. All of the Uno's analog pins are 10 bits, which means the ADC outputs a value between 0 and 1023. The following formulas are used to convert the ADC value to °F:

$$\boldsymbol{temperatureV} = (\boldsymbol{adcValue} / \mathbf{1024.0}) + \mathbf{5.0}$$

- The 5.0 value is the reference voltage of 5V

$$\boldsymbol{temperatureC} = (\boldsymbol{temperatureV} - \mathbf{0.5}) * \mathbf{100.0}$$

- The TMP36 sensor has a resolution of 10 mV/°C and uses a 500mV offset to allow for negative temperatures. The actual formula is $C = (mV - 500) / 10$, which can be converted to voltage (as seen above).

$$\boldsymbol{temperatureF} = (\boldsymbol{temperature} * \mathbf{9.0} / \mathbf{5.0}) + \mathbf{32.0}$$

Reading Temperature

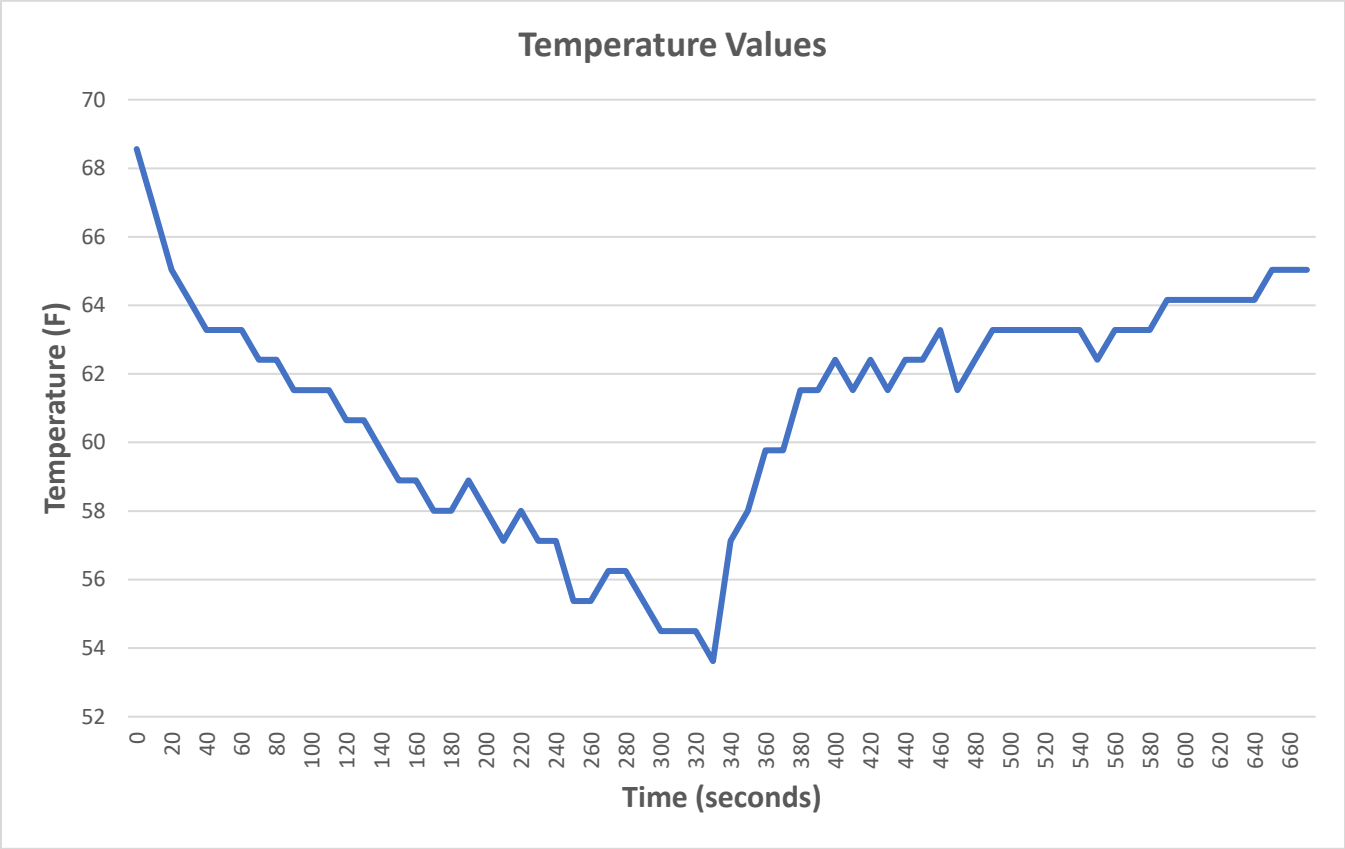
I utilized a timer that triggers a periodic interrupt in order to read the temperature every 10 seconds. The Arduino Uno has 3 timers that increment a count at the rate they have been configured to run at. Timer0 and Timer2 can hold 8 bit values, and Timer1 can hold 16 bit values. Regardless of the size, all of the timers overflow their count very quickly if the count is incremented at the Uno's clock rate of 12 MHz, so a prescale value must be applied to the timer to reduce the rate it increments it's count at. A 10 second timer is not possible on the Uno, so I chose to configure a timer to operate at 1 Hz. The ISR for this timer increments a separate count and reads the ADC value from the analog pin connected to the TMP36 once every 10 seconds.

Operating Modes

My development environment involves using a desktop, so I cannot transmit the temperature to the host over Serial from within the refrigerator. As such, I designed my hardware and program to store the temperature data in the EEPROM to persist the data and transmit it to the host machine at a later time. I utilized two pushbutton switches to control what mode the program was running in, and both switches are connected to digital pins that trigger interrupts on a rising edge. The interrupts set flags that the main loop checks to determine what it should be doing. One thing I found out while doing this is the pushbuttons are kind of noisy and sometimes triggered multiple interrupts, so I added a short debounce in the pushbutton ISRs to prevent the ISRs from being executed multiple times.

Results

The following plot was generated from the CSV values output from the Arduino to the host machine over Serial (USB). From this plot you can see that the Arduino was taken out of the refrigerator around the 340 second mark.



Video Demonstration

A video demonstration of the software and Arduino running can be found at the following link:

https://www.dropbox.com/s/ndb0ihriytqkw16/Miles_Gapcynski_EN_605_715_81_Project_2.mp4