EE 347

PROJECT REPORT

Project No. 1

Microcontroller-Based Tea Kettle

Written by: Ian Berdan Date Submitted: March 27th, 2021

Written by: Mason Rakowicz Lab Section: 02, Thursday, 1400

Instructor: Dr. Robert Fourney

Introduction

The objective of this project was to implement the functionality of heating a tea kettle to a user-defined temperature (in Celsius) by means of a TM4C123GH6PM microcontroller [1]. Two programing languages were used to accomplish this goal, C and Assembly (ASM), with each program independent of one another. The C version allowed the user to input a desired temperature on a keypad in which the tea kettle would heat up to, settle at, and maintain the desired temperature. Conversely, the ASM allowed for the user to select between three predetermined temperature modes for the kettle to heat up to, settle at, and maintain.

Procedure

Both design versions followed a similar flow of information, Fig.1.



Figure 1: Tea Kettle Layout

Either program design set up the microcontroller to perform actions based on inputs received from the circuit keypad and the thermistor which controls the tea kettle and respective LEDs. The procedure for each version is shown below.

1. Design in C

The C design was based around the intention for the user to enter a temperature on a keypad, 0-99°C, as requested by a terminal emulator (Tera Term). Once a value was entered, the tea kettle would heat to the desired temperature, updating the user on the current temperature every few seconds. Among its other main functions, an active high master switch was also included with an LED to display if the system was functional.

Main functions the program navigated through included an A2D initialization, keypad initialization, keypad scan, rsfPutChar, and temperature computation. A 16-pin keypad was decoded such that the columns (pins PB0-3) functioned as outputs and the rows (pins PC4-7) functioned as inputs with 1.8kΩ pull down resistors. The program communicated by printing the current temperature reading and the desired temperature to Tera Term via rsfPutChar. Values input by the user were stored and compared with the current temperature reading where the microcontroller would then turn on the tea kettle if deemed too cold or shut off the kettle if deemed too hot or just right. A flowchart of the overall C functionality is shown below, Fig. 2.



Figure 2: C Program Flowchart

The C design began with an analog to digital (A2D) conversion calibration with an MCP9701 thermistor [2] and an A2D C program. HEX values produced by the A2D program were recorded and associated with a specific temperature, where a line of best fit was produced. This line’s equation was implemented in the C main program so it could determine when to progress to different areas of code. Through additional testing with both a reliable external thermometer and the C code, the equation was fine-tuned to allow more precise temperature readings. The circuit was then assembled with the over all tea kettle layout in mind (Fig. 1) with a block diagram of the circuit shown below, Fig. 3.



Figure 3: C Program Circuit Block Diagram

Heating floors and ceilings were fine-tuned and hard coded into the program to allow for room for error. Maximum allowable temperature was set as two degrees warmer than the desired, and minimum allowable temperature was set as two degrees cooler than the desired. The former was chosen because of an initial project constraint to warn the user the temperature is too hot, and the latter was chosen to allow for the residual coil heat to transfer to the water once turned off. This also provided a reasonable range to take A2D samples to produce more consistent results.

A2D continuously took temperature readings throughout the program to ensure the tea kettle reacted in the appropriate manner. If the program deemed the measured temperature was less than the two-degree threshold, the relay switched on, as well as an associated amber light, to turn on the kettle. Once an A2D reading was taken that was larger than the lower threshold, the relay turned off, as well as the amber light, and a green light turned on to indicate the water was of desired temperature. If at any point an A2D reading was taken that was larger than the higher threshold, the green light turned off and a red light flashed to alert the user that the water was hotter than the desired temperature.

1. Design in ASM

The ASM design was based around the ability for the user to press one of three buttons on the keypad associated with one of three temperature settings: soup, simmer, and boil. Also included was a master switch; active high to enable the program verified by its own LED. If soup was selected, a green LED illuminated, and the water was heated to approximately 80°C where temperature was maintained. Similarly, if simmer or boil was selected, either a yellow or red LED illuminated, and the water was heated to either 92°C or 99°C, respectively. Additionally, the temperature LEDs used in the C program were implemented in the same way to alert the user when the water temperature was desired, heating up, or too hot. A flowchart of the overall ASM functionality is shown below, Fig. 4.



Figure 4: ASM Program Flowchart

The program monitored for button presses where it would branch to its respective temperature subfunction to run. Though the whole keypad was decoded, only the top three buttons in row 1 were used. Columns 1, 2, and 3 were configured as inputs (pins PB0-2) with 1.8kΩ pull down resistors and row 1 was configured as an output (pin PC4). Based on which button was pressed, an internal flag was set so the program knew where to branch to if a different button was pressed. An A2D value was measured and, based on the button press, would compare with the set numbers, and decide if the water needed to warm up, cool off, or was within range. Set numbers were interpolated from the previously derived equation in the C design. Once more, the physical circuit was assembled with a block diagram representation shown below, Fig. 5.



Figure 5: ASM Program Circuit Block Diagram

Like the previous C design, the ASM program took A2D samples as it cycled through main functions. Functions included main beginning, keypad scan, A2D sampling, soup, simmer, boil, and initializations. Within each of the soup, simmer, and boil functions, the heat ceiling was increased compared to the C program. Rather than two degrees higher, the ceiling was set at six degrees higher to allow for more consistent A2D readings before the temperature became too hot. The heat floor, on the other hand, was set to the desired value through fine tuning and experimentation.

Results and Analysis

Results can be broken up into the same two parts.

1. Design in C

The C version of the project worked as anticipated. The algorithm behaved as desired without the need for it to be significantly optimized. However, various design implementations were added from the initial design to ensure enough speed throughout the program. For any given delay, the minimum value used was 500ms or less before testing if the reset button or master switch was pressed. This was chosen because of the keypad input with the assumption that the user would type slower than a typical keyboard. This also aided in debouncing the mechanical buttons.

The A2D code was initially data mapped with a potentiometer to vary the voltage and compare the given digital value. The plotted data is shown below, Fig. 6, where the slope was calculated to be 0.0398 through linear interpolation. Note that the temperature of the water was never constant which gave accuracy limitations as shown in the nonlinear graph.

Figure 6: Thermistor Calibration

From the given slope, an offset was found through trial and error such that the slope of the graph began at 0°C. The final equation was created (1).

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

When the thermistor was implemented in code, an error of ±2°C was present. This was due to a slope that was not perfectly linear. Furthermore, the temperature sensor used for calibration utilized a metal bulb to heat up with a larger thermal transfer rate compared to the thermistor used. Therefore, when heating the water, the thermistor would lag slightly behind. Depending on the tea kettle used, the overshoot would vary slightly, in that it heats up faster or slower than the kettle used for calibration. Another significant variable watched was the amount of water. In general, the project was created using about 1.75 liters of water. If the water level changed, the overshoot compensation would need to be adjusted as the thermal capacity increased or decreased.

From trial and error, the minimum temperature difference as measured by the thermistor was ultimately found to be -1°C with the design specification of a maximum measured +2°C. The lower bound was chosen while assuming an underdamped response in the water temperature. Therefore, when heating up, the tea kettle will turn off one degree before expecting the temperature to steadily rise after the heat source was cut off.

After considering the thermistor tolerance of ±2°C according to the datasheet and the measured overshoot tolerance, the maximum error was -3°C to +4°C. It should be noted that as the input temperature increases, the percent error decreases overall. Thus, the overall percent error of the system at specified temperatures are shown in Table 1. Theoretically, average users are expected to input a temperature above room temperature, specifically, in the range of 70-100°C. Therefore, the worst-case maximum error an average user will see is 5.71%.

Table 1: Theoretical Temperature Percent Error

|  |  |
| --- | --- |
| Example Temperature (°C) | Percent Error (%) |
| 10 | 40 |
| 20 | 20 |
| 30 | 13.33 |
| 40 | 10 |
| 50 | 8 |
| 60 | 6.67 |
| 70 | 5.71 |
| 80 | 5 |
| 90 | 4.44 |
| 100 | 4 |

Theoretically, the percent error could be reduced further by designing to code to cut off prematurely as to account for overshoot, and then update the cutoff temperature after the initial rise to maintain a ±0.5°C tolerance due to rounding limitations. Using this method, the average user should see a maximum error of 3.57%.

Through additional testing, the actual percent error for the C version was found by heating up the tea kettle to specified temperatures and letting it settle. The temperature was read from both Tera Term and the temperature probe where this data was tabulated in Table 2. Note that this test used 1.5 liters of water, approximately the average volume of water used throughout the project. Also, a portion of the error is found due to the truncated float temperature to an integer within the program.

Table 2: Measured Percent Error in C

|  |  |  |  |
| --- | --- | --- | --- |
| Desired Temp (°C) | Thermistor Temp (°C) | Temperature Probe (°C) | Percent Error (%) |
| 50 | 50 | 50.6 | 1.19 |
| 60 | 61 | 61.6 | 0.97 |
| 70 | 70 | 70.7 | 0.99 |
| 80 | 80 | 80.5 | 0.62 |
| 90 | 89 | 89.4 | 0.45 |
| 96 | 96 | 95.4 | 0.63 |

From the measured data, it can be seen the maximum percent error was 1.19% (at the lowest temperature) with the average error at 0.81%. At most, the temperature was off by 1°C. When considering the theoretical error with the measured error, the constructed project performed significantly better.

1. Design in Assembly

The ASM version of the project also worked as anticipated, with the algorithm needing no significant remodel. It followed a more linear approach to achieve its project goals, as it had more limited functionality compared to its C counterpart. However, like the C version, the minimum time to check a button was 500ms for the same reasons listed above: the user will input slower than a standard keyboard, and to help debouncing issues.

The project in assembly utilized the same calibration techniques as the project in C, with (1) used to calculate the designated hexadecimal values associated with each desired temperature. Instead of actively testing, the temperatures were hard coded in. Similarly, the maximum percent error analysis remains the same. Though the heat ceiling limits were larger than in the C version, the ASM version met the project requirements of remaining within ±2°C of the desired temperature. If not within the threshold, the user would be alerted via LEDs and the program would adjust accordingly. Performing the same analysis for measuring the settled temperature and comparing to the temperature probe, the results were shown in Table 3.

Table 3: Measured Percent Error in ASM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mode | Desired Temp (°C) | Thermistor Temp (°C) | Temperature Probe (°C) | Percent Error (%) |
| Soup | 80 | 80 | 81.1 | 1.36 |
| Simmer | 90 | 90 | 90.6 | 0.66 |
| Boil | 99 | 98 | 97.3 | 0.72 |

A similar error was found with the maximum being 1.36% and the average at 0.91%. This is to be expected as it shows both the programming languages are of equal standing. Neither version of the project tended to be more accurate than the other at reasonable water temperature and volume. Overall, the ASM version was as accurate as the C version.

Conclusions

By use of a microcontroller and some external circuitry, a tea kettle’s water temperature was able to be reliably controlled and maintained. Though both versions of code differed from one another, in both functionality and language, they were both able to meet design specifications for their respective goals. This was despite the variability in temperature readings that necessarily led to an error analysis. Ultimately, however, implementation of a microcontroller paired with both C and ASM programs proved to be effective and capable of heating water to desired temperatures with tolerable error.

References

[1] Texas Instruments, “Tiva TM4C123GH6PM Microcontroller” TM4C123GH6PM datasheet, Jun. 2014

[2] Microchip, “Low-Power Linear Active Thermistor ICs” MCP9701 datasheet, 2009