4-20-2021

EDMMS Temperature Controller

Anthony Kirkland
Western Michigan University, anthony.e.j.kirkland@gmail.com

Presentation

Group participants:

Macallister Armstrong
Jeremy Evans
Anthony Kirkland
Lorand Mezei

Course:

CS 4910 - Software and System Design II: Implementation and Testing

Professor:

Jason Johnson

Assignment:

Final Report

Due date:

05/01/2021

2. Abstract

Temperature control systems in consumer appliances like that of a thermostat interfacing with HVAC systems, refrigerators and ovens are oscillatory in nature. There is a temperature at which the machine that causes the change in the system comes on and a different temperature at which it comes off. While sufficient for humans, welding, metal casting, and other metallurgical processes require precise temperature control, more precise than the hysteresis of a consumer system.

Proportional integral derivative (PID) provides a better way of monitoring the way temperature changes when the entity that changes the environment comes on and renders changes in system temperature more precise without as much oscillation as that of a consumer system. The temperature controllers that do this, however, tend to cost around $200 to $300 at least. Allin, Machine Shop Spec in the Department of Engineering Design, Manufacturing and Management Systems (EDMMS) at WMU, seeks to construct a PID temperature controller using an MSP430 to mitigate that expense (the MSP430 Allin intends to use for this project is inexpensive: around $2 for the chip itself and $10-$15 for the launchpad).

3. Problem Statement

3.1. Need

The purpose for this project was to create a programmable interface for controlling an industrial oven, refrigerator, or freezer. Industrial applications often require a set of steps where, for example, the temperature rises from room temperature to a given temperature, remains at that temperature for a given time interval, rises to another temperature, remains at that temperature for another time interval, and ultimately powers down. Microcontrollers with preloaded software already exist for these applications, but run in the range of several hundred dollars. Using the MSP430 with custom built temperature controller software greatly reduces the cost, and can be used again and again by the Engineering Design, Manufacturing, and Management Systems (EDMMS) department at WMU.

3.2. Objective(s)

To be delivered will be the C code for the MSP430 that will:

- Use PID to analyze a voltage representing a temperature reading continuously sent from ADC connected to a thermocouple.
- Use the resulting values to determine when to toggle the state of the active GPIO pins, which will effectively toggle a switch (to open/close a valve or toggle a heating element).
- Provide a CLI that will interface with a spectrometer via UART that will enable us to set the parameters with which to toggle the GPIO pins.

- A validation framework to verify the code's ability to perform the aforementioned tasks.

The code to be developed will integrate with the hardware Allin is developing separately for the project. We will validate the code by deploying it on a test bed Allin will develop that will simulate the behavior of an oven.

3.3. Terms, Acronyms, Glossary

ADC: Shorthand for analog-to-digital converter; it is a system that converts an analog signal (i.e., sound picked up from a microphone or light entering a digital camera) into a digital signal. For the purposes of this project, it is a system that receives a voltage as input and converts it to a digital number.

GPIO: General-purpose input/output; it is an uncommitted digital signal pin on an integrated circuit or electronic circuit board which may be used as input, output, or both, and is controllable by the user at runtime.

Hysteresis: The dependence of the state of a system on its history. It is the phenomenon in which the value of a physical property lags behind changes in the effect causing it by a certain amount. One example of this is the interval of time between setting the desired temperature in an oven and the actual temperature in the oven after the preheating process.

Metallurgy: A domain of materials science and engineering concerned with the physical and chemical behavior of metallic elements. It is also concerned with the production of metals and the engineering of metal components used in industrial and consumer products.

MSP430: A family of general-purpose MCUs (microcontroller units) produced by Texas Instruments (TI). For our purposes, we are using a low-powered MSP430 with a 10-bit ADC.

PID: A control loop mechanism that automatically adjusts a control output based on the difference between a set point (SP) and a measured process variable (PV), called the *error value* e(t). The outputted value u(t) is transferred as the system input.

Potentiometer: A three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. They are commonly used to control electrical devices that operate off variable voltage such as volume controls on audio equipment.

Spectrometer: An instrument used to separate and measure spectral components of a physical phenomenon.

SPI: Short for serial peripheral interface; it is a synchronous serial communication interface used for short-distance communication among two or more connected devices in embedded systems. It is a full-duplex interface, meaning all connected parties can communicate with one another in both directions.

Thermocouple: An electrical device that produces a temperature-dependent voltage, which is often used to measure temperature. Industrial standards vary by cost, availability, convenience, melting point, chemical properties, stability, and output. We are using a Type K thermocouple for this project.

Voltage Divider: A circuit that turns a large input voltage into a smaller one using two series resistors. It is necessary to, for instance, compress the range of an input voltage down to that which a receiver can accept.

4. Problem Analysis and Research

We ran into quite a few problems through the duration of this project. We were rusty with MSP430 development. We've had experience with MSP430 development from CS 2230 Computer Organization and Assembly Language, but needed to be brought up to speed. We were unable to do any meaningful testing or development other than the shell without test beds. It took our client a few months to order the parts, perform soldering, and create the test bed prototype. After creating the first test bed prototype, our client realized that it could heat up to 400 degrees Fahrenheit. The client had to create his own thermal paste to prevent a potential fire hazard. When we received the remaining test beds, we had about a week to attempt to implement SPI communication, incorporate our PID algorithm, and test the whole process. These were things we did research on, but found difficult to implement in code. We ultimately implemented SPI communication, but were unable to incorporate the PID algorithm due to time constraints.

5. Requirements

5.1 Spectrometer CLI

5.2 Levels of Interactivity

The CLI should support two levels of interactivity:

Normal: The interface simply returns a signal indicating whether a command was successfully sent (ACK, “acknowledgement;” ASCII 0x06) or unsuccessfully sent (NAK, “negative acknowledgement;” ASCII 0x05).

Verbose: the interface responds to any command by describing in text what has been done (or explaining that the command is not understood).

5.3 Commands

The CLI should support commands that are generally of the form of one to two letters followed by a number. Below is a list of some of the commands needed to be supported:

Command	Description
help	List available commands
PID a b c	Set 3 values for PID, respectively. P = a, I = b, D = c.
args	Prints back given arguments
+	Run the currently selected program from the currently selected step.
|	Pause execution and maintain the current setpoint.
-	Halt execution and turn the output pin off
p N	Load program N, step 0.
s N	Load step N of the current program.
. N	Set the target setpoint of the current step to N kelvins (only allowed when execution is halted).
m N	Set the interval of the current step in N minutes (only allowed when execution is halted).
v [01]	Reset/set verbose mode.
r N	Set the reporting interval to N seconds.
show	Display the contents of the PID object.

Interaction with ramping programs should be straightforward. For example, inputting “p 1” would load the program in slot 1 to be run or modified. If “s 3” is then entered, the third step of the program in slot 1 should be loaded. The command “p 0” should be a stepless program that just has a setpoint so that the device can be turned on, set to one temperature, and left.

Step 0 of any program does not have an interval; it is just a start point so that a ramp in step 1 knows where it is starting from (it would be cool if setting to 0 caused step 1 to sample the current temp and start from there, but that is extra).

5.4 Reporting

The CLI should command and report temperatures in Kelvins as integers (though it may need to store fractional representations of temperatures internally). Setpoint and actual measured temperature should be reported at user-defined intervals down to whatever the debounced sampling interval is (e.g., .800 k796).

5.5 Temperature Readings

Temperatures should be represented as 16-bit unsigned integers mapped to Kelvins (not Celsius or Fahrenheit. Temperature readings should be debounced, the parameters of which should be easily adjusted (at minimum by recompiling/reflashing). Floating-point data should be easily avoided whenever possible (if unavoidable, fixed-point binary values are preferable). If the interval is set for its maximum value, the software should recognize that as a "hold" command and stay at that temperature until given other instructions.

When ramping, an ideal implementation would include the ability to change the setpoint over time automatically. This is typically handled with a series of steps made up of an interval and a target temperature. If the target temperature differs from the target of the previous step, the setpoint is linearly altered over the interval; if the target temperature is the same, the controller should "soak" at that temperature for the given interval. It is fine if there are a fixed number of "program" slots for ramping profiles (4-8 ought to be sufficient) with each program made up of a fixed number of intervals (8 or so should be sufficient). These programs should be stored in the MSP's flash memory so that they can be recovered after an interruption in power and are not lost.

5.6 System Requirements Specification

The Temperature Controller will be programmed on an MSP430 Launch Pad provided by Texas Instruments, which will be connected to a breadboard with wires, temperature control hardware (provided by the WMU Engineering Department), Thermocouple, MAX31855, 40A HOYMK Solid State Relay. The program will be written in C language. Because this program will need to process real time data from a constant external source, Calculus algorithms will have to be implemented to record data over time and apply mathematical functions to that data. Data that will be processed will be the input data from the temperature control hardware, potentiometer, etc. A history of this data over time will be analyzed and integrated over time, which will require the implementation of a Proportional, Integral, Derivative (PID) algorithm. There are specific events that occur in the temperature control hardware that we need to be aware of. The voltage from the hardware will reverse sign at 0 degrees celsius. There will be a second pin on the MSP430 that will handle that. Noise filtering from the external data will need to be done by debouncing the input. 3 bits will be unstable because of temperature change. The temperature control device is usable with either a relay controlling power through a heating element, or by controlling a valve to direct the flow of liquid nitrogen into a container. The output will actuate a relay that responds to the voltages at which the MSP430 operates. To test the functionality in a real system, the program will potentially be hooked up to a toaster or something similar to test sensing and controlling temperature.

6. Standards and Constraints

6.1. Applicable Standards

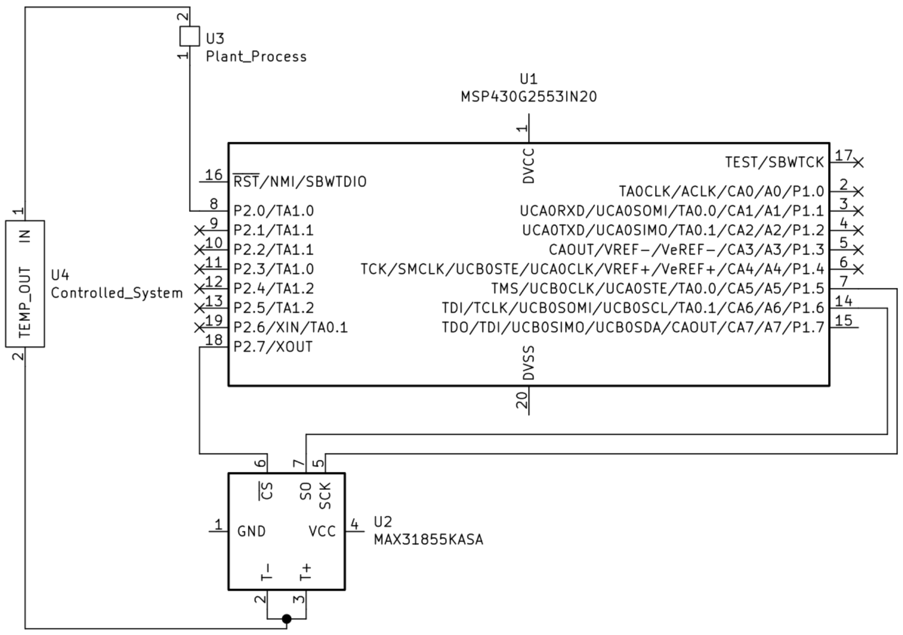
We had to adhere to the GNU/LINUX, C Language, and MSP430G2553 development standards.

6.2. Constraints

We were constricted by time and hardware. We took on a 2 semester project that was to be completed by April 21, 2021. Our code had to be runnable on Linux machines using the MSP430G2553. We were given three operational test beds. We had to cooperate with limited access to hardware. There were no ethical concerns as completing this project would benefit everyone. There were no concerns about cost of production or maintenance as this would be handled internally. There are also no concerns about health and safety as we are only responsible for the software, which handles non-personable or sensitive data.

7. System Design

7.1. Hardware Components



U3
Plant_Process
U1
MSP430G2553IN20
DVCC
RST/NMI/SBWTDIO
P2.0/TA1.0
P2.1/TA1.1
P2.2/TA1.1
P2.3/TA1.0
P2.4/TA1.2
P2.5/TA1.2
P2.6/XIN/TA0.1
P2.7/XOUT
TEST/SBWTCK
TA0CLK/ACLK/CA0/A0/P1.0
UCA0RXD/UCA0SOMI/TA0.0/CA1/A1/P1.1
UCA0TXD/UCA0SIMO/TA0.1/CA2/A2/P1.2
CAOUT/VREF-/VeREF-/CA3/A3/P1.3
TCK/SMCLK/UCB0STE/UCA0CLK/VREF+/VeREF+/CA4/A4/P1.4
TMS/UCB0CLK/UCA0STE/TA0.0/CA5/A5/P1.5
TDI/TCLK/UCB0SOMI/UCB0SCL/TA0.1/CA6/A6/P1.6
TDO/TDI/UCB0SIMO/UCB0SDA/CAOUT/CA7/A7/P1.7
DVSS
U4
Controlled_System
IN
TEMP_OUT
U2
MAX31855KASA
CS
SO
SCK
GND
VCC
T-
T+

Displayed above is a schematic of the hardware used for our application. For our design, the MSP430 is connected to a MAX31855 in a 3-wire SPI configuration, where data flow goes from SO on the MAX31855 to MISO on the MSP430G2553 (through which to receive temperature values), from SCK (P1.5) on the MSP430 to SCK on the MAX31855 (to synchronize system clocks on both devices, dictated by the MSP430G2553), and from P2.7 on the MSP430G2553 to CS on the MAX31855 (setting CS to low "selects" the MAX31855, enabling it to send data to the MSP430G2553).

Additionally, the MSP430G2553 is connected to a device that controls the temperature in a system (designated here generically as the Plant Process) through a connection from P2.0 (configured as an output pin) to the plant process. The MSP430G2553 activates this device by sending a signal through P2.0 and deactivates it by clearing the signal from P2.0.

8. Testing

Testing was one of the more difficult aspects of the project. Because our implementation was directly related to the output of our test beds, there was virtually no way to test the process without them. In the week when we had working test beds, the team was focused on implementing SPI communication to the Thermocouple-to-Digital Converter.

For this reason, testing was relegated to input validation. Even this was trivial, as all of the inputs were non-negative integers, with the verbose flag being a zero or a one. Testing would have been done on the test beds as a live system had the team gotten that far.

9. Results

We have created a working shell with input validation. This shell updates the variables of the PID controller data structure. We have implemented SPI communication. We have studied and documented the PID algorithm. We produced a 10 page document on the algorithm. Unfortunately. We were unable to incorporate and test the PID algorithm due to time constraints.

Future Recommendations:

- Incorporate PID algorithm
- Connect data structure elements to required functionality
- Create usage documentation

9.1. Realization of requirements

Realizing the requirements would have meant having functionality for all of the commands, including programs, steps, a working PID algorithm, etc. The majority of those requirements were not met because of the lack of a testing environment (our test beds) for the majority of the year. The client revised those specifications after the late hand off of the test beds, asking for either a working UI or a working PID algorithm. In theory, both may have been produced. The CLI has been thoroughly tested, so that requirement was certainly met. There was not enough time to integrate the PID algorithm, so although we developed code for that, it is untested.

9.2. Realization of Standards and Constraints

The standards and constraints were all realized. We were given a specific microcontroller and Thermocouple-to-Digital Converter and asked to create C code. The client did not specify a specific C standard or MSP430 toolchain, but the MSP430 toolchain used and C99 are the same tools used in WMU's CS 2230, so surely this is acceptable to the client. We have met all time constraints as far as development and hand off.

9.3. Testing results

As mentioned above, the only meaningful testing we were able to perform was input validation, which was trivial.

10. Future Work

Future developers would need to implement the functionality for nearly all of the commands. There was not time to implement actual programs or steps, so there is plenty of room for improvement upon the basic design. They would need to logically connect the inputs from the shell and Thermocouple-to-Digital Converter and the output to the heating or cooling device with the PID algorithm. They would also need to test the efficacy or the algorithm to make sure that the set point does not stray from the internal device temperature, and that ramping happens in a methodical way, to prevent the device from flickering on and off.

11. Conclusion

In this project, we were able to complete a working CLI in Linux that was outfitted with input validation. The values that were used in this shell are stored in a shell updates controller data structure. SPI (Serial Peripheral Interface) communication between the MSP430 and the test bed was implemented. The theory of a PID (Proportional Integral Derivative) controller was thoroughly studied and documented in a 10 page report.

Code for the PID algorithm was written, however, we were unable to incorporate the PID algorithm into the MSP430 to be tested alongside the test bed.

Testing was not able to be done because our client shipped us the test beds months late, resulting in our team having only a week to complete the project. We could not implement the PID algorithm on time, so we were not able to perform software or hardware testing.

This project is very much a work in progress. Recommendations for future development on this project are to incorporate the PID algorithm to use with the MSP430 and Testbed, connect data structure elements to required functionality, and create usage documentation.

Appendices:

A. Project Management Plan

Our plan was largely driven by access to the test beds. We first created test programs such as an LED dimmer to brush up on timer interrupts and MSP430 development. We then developed, validated, and tested the CLI. We simultaneously researched and developed the PID algorithm without the relevant hardware. After receiving the test beds, we implemented SPI communication to the Thermocouple-to-Digital converter. We made use of projects and milestones for each of these stages.

B. Progress Reports

Our progress reports are being submitted alongside this document.

C. Development Costs

I. Institution Costs

The institution purchased 4 MSP430G2553's and test bed's, which included a thermocouple, a MAX31855 Thermocouple to digital converter and a solid state relay. We approximate that this cost $164.

II. Sponsor Costs

There were no expenses for our sponsor.

III. Team Costs

There were no expenses for the team.