Software Requirements Specification

EDMMS Temperature Controller

Version 1 • Dec 17, 2020

Macallister Armstrong
Jeremy Evans
Anthony Kirkland
Lorand Mezei

Table of Contents

1: Preface

1.1: Purpose

This is the Software Requirements Specification for the EDMMS Temperature Controller developed for Allin Kahrl in fulfillment of the senior design requirements for 2020-2021. The main purposes of this document are to illustrate the user and systems requirements for the EDMMS Temperature Controller.

There are software and hardware requirements associated with this project; written in this article are the software requirements and how the deliverable integrates with the hardware Allin develops.

1.2: Versioning Rationale

The initial version of this document was created after eliciting user requirements and constraints from Allin. Successive versions will be created when further requirements are known and assumptions about existing requirements are corrected. Its intended readership are the project's developers to keep in line with what is expected to be delivered, Allin for requirements validation, and Jason Johnson in satisfaction of Senior Design 2020-2021.

1.3: Version History

Name	Date	Reason(s) for Changes	Version
Jeremy Evans Anthony Kirkland Lorand Mezei	Dec 17, 2020	Initial SRS. Content is taken from the initial meetings we had with Allin and the desired list of features Allin sent.	1

2: Introduction

2.1: Background

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).

2.2: Deliverable

To be delivered will be the C and assembly 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: Glossary

3.1: 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.

3.2: 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.

3.3: 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.

3.4: 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.

3.5: 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.

3.6: 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.

3.7: 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.

3.8: Spectrometer

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

3.9: 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.

3.10: 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: User Requirements Definition

Statements of what services (plus diagrams) the system is expected to provide to users and the constraints under which it must operate. Non-functional requirements should be included here. Product & process standards should be here.

4.1: Spectrometer CLI

4.1.1: 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).

4.1.2: 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
+	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 (maybe this should ask for confirmation).
pN	Load program N, step 0.
sN	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).
mN	Set the interval of the current step in N minutes (only allowed when execution is halted).
v[01]	Reset/set verbose mode.
rN	Set reporting interval to N seconds.

Interaction with ramping programs should be straightforward. For example, inputting “p1” would load the program in slot 1 to be run or modified. If “s3” is then entered, the third step of the program in slot 1 should be loaded. The command “p0” 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).

4.1.3: 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).

4.2: 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 preferrable). 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: 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, LEDS, temperature control hardware (provided by the WMU Engineering Department), Thermocouple, Voltage Divider Circuit, 7 segment display, and a Potentiometer. The program will be written in Assembly Language and C language and will use Energia IDE in Linux. 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 celcius. 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.