University of British Columbia  
Electrical and Computer Engineering  
ELEC291/ELEC292 Winter 2025  
Instructor: Dr. Jesus Calvino-Fraga

Section 201

***Project 1 – Oven Reflow***

***Controller***

Group A13

|  |  |  |  |
| --- | --- | --- | --- |
| ***Student #*** | ***Student Name*** | ***% Points*** | ***Signature*** |
| 31668908 | Yassin Abulnaga | 100 |  |
| 19097781 | Faris Alshouani | 100 |  |
| 72193626 | Ali Danesh | 100 |  |
| 44837953 | Ronald Feng | 100 |  |
| 77438398 | Drédyn Fontana | 100 |  |
| 26570291 | Nick Unruh | 100 |  |

Date of Submission: February 28, 2025

**TABLE OF CONTENTS**

1. **Introduction**
2. **Investigation**
   1. *Idea Generation*

*2.2. Investigation*

* 1. *Design Data*
  2. *Collection Data*
  3. *Synthesis*
  4. *Analysis of Results*

1. **Design**
   1. *Use of Process*
   2. *Need and Constraint Identification*
   3. *Problem Specification*
   4. *Solution Generation*
   5. *Solution Evaluation*
   6. *Detailed Design*
   7. *Solution Assessment*
2. **Life-Long Learning**
3. **Conclusions**
4. **References**
5. **Bibliography**
6. **Appendices**

**1. Introduction**

**Objective**

Our objective was to design, build, program, implement, and test a functioning reflow oven controller used to attach surface-mount components (SMT) to a printed circuit board (PCB). The controller must regulate the heating process while returning accurate temperature readings. Additionally, the reflow oven controller should feature an intuitive and efficient user interface. The reflow soldering profile will be analyzed using Python to evaluate temperature trends and heating cycles.

**Specifications**

The project comprised both hardware and software elements. Table 1 presents a detailed breakdown of the hardware utilized in circuit assembly, while Table 2 delineates the corresponding software components.

|  |  |
| --- | --- |
| **Main Chipset** | **Op Amp / Temperature Circuit** |
| N76E003AS20 Microcontroller | K-type Thermocouple |
| **Associated Parts for Chipset** | 2x 10MΩ Resistors |
| 0.1uF Capacitor | 2x 33kΩ Resistors |
| 330Ω Resistor | LMC7660IN Voltage Converter |
| LED 5MM GREEN | 2x 10μF Capacitors |
| 20-SOP to 20-DIP adapter | PV36W103C01B00 Potentiometer |
| BO230XS USB adapter | OP07 Precision Operational Amplifier |
| 2x Push Buttons | LM4040 Voltage Reference (4.096V) |
| **LCD Display Circuit** | **Push Button Circuit** |
| LCM-S01602DTR/M LCD Display | 6x Push Buttons |
| 2kΩ Resistor | 5x 1N4148 Diodes |
| **Oven Control Circuit** | 10kΩ Resistor |
| FQU13N06LS MOSFET |  |

Table 1: Hardware parts used to assemble the system.

|  |  |
| --- | --- |
| **Assembly Functions** | **Processing Functions** |
| Read ADC | Read Serial Port |
| Calculate Temperature | Generates Moving Average |
| Count Seconds | Generates Temperature Strip Chart |
| Finite State Machine (FSM) |  |
| Read Push Buttons |  |
| Passcode |  |
| Update Variables |  |
| Display LCD Information |  |

Table 2: Software functions used for the project

A diagram of a computer

AI-generated content may be incorrect.Figure 1: A block diagram of the integration between software and hardware components.

**2. Investigation**

**2.1. Idea Generation**

Our group created a Gantt chart to systematically outline the requirements of the project and establish a structured timeline which included each group members responsibilities. In addition, our team created a table for additional features that could add to the efficiency and user-friendliness of the reflow oven controller. With our requirements outlined and our ambitions of including additional features, we generated many different solution ideas, design approaches, and implementations, eventually leading us to the one that we pursued.

**2.2. Investigation Design**

Our team adopted a structured approach, which integrated information gathering, data analysis and experimentation to ensure a comprehensive understanding of our system. The investigation process was carried out in the following key stages:

*Information & Data Gathering:*

* Conducted research on thermocouple characteristics and reflow oven thermal profiles.
* Collected reference temperature data for calibration and accuracy validation of thermocouple.

*Circuit Design:*

* Designed the hardware schematic, integrating the thermocouple, ADC microcontroller and LCD display.

*Software Development & Implementation:*

* Developed firmware in assembly to handle ADC readings, PWM control, and LCD updates.
* Implemented a finite state machine to regulate the reflow process, while data logging and visualizing the temperature profile using a python script.

*Experimental Testing & Validation:*

* Conducted bench tests to verify ADC readings, LCD output and PWM functionality.
* Performed temperature validation experiments by comparing thermocouple readings with a calibrated reference.

**2.3. Data Collection**

Our group used data to verify that our circuit was functioning which in turn aided in debugging. The Fluke 45 Digital Multimeter was our primary tool. We used the multimeter to measure the resistances of the resistors in the op-amp circuit to improve our temperature calculations. The Fluke 45 Multimeter was also used to verify the correct voltage gain of the op-amp circuit. With the TTI EX354T Power Supply, we measured the input and output voltages of the op-amp to ensure our voltage gain was correct. Additionally, we used the multimeter to measure the thermocouple’s voltage. A Python program was also used to convert the voltage into the thermocouple’s temperature, which we could then use to compare our microcontroller’s temperature calculations with it. Further details are in Data Synthesis.

**2.4. Data Synthesis**

After using the tools shown above to collect the data, we used a Python program that printed the multimeter’s temperature reading, the microcontroller’s temperature reading, and their difference to the console. We could also accomplish further data synthesis by compiling the data into a Google Sheets document, where we could sort and visualize the data with graphs. We were also able to test our FSM assembly code by uploading it to our microcontroller and observing the LCD display to check if the states were changing at the correct times and temperatures we set.

**2.5. Analysis of Results**

Our group assessed the validity of our conclusions by repeatedly performing tests to compare the multimeter’s thermocouple temperature and the microcontroller’s thermocouple temperature. We expected noise in the measurements of microcontroller ADC, so we deemed it acceptable if our measurements had an error of ±3°C. If our temperature readings exceeded the ±3°C error, then we adjusted our circuit and our calculations until our measurements were in the acceptable range. We also confirmed our assembly code was working as intended by confirming that the states of the FSM changed at the correct times and temperatures.

**3. Design**

**3.1. Use of Process**

Our team successfully developed the reflow oven controller by adhering to a structured engineering design process. We followed these steps: problem definition, idea creation, design implementation, and testing and verification.

* We established the basic functionalities and identified constraints such as time, components, and hardware restrictions.
* We broke down functionalities into smaller subsections, brainstormed extra functionalities, and categorized ideas based on feasibility and impressiveness within constraints.
* Software was developed in small subsections and integrated into a project file for debugging and organization. We also created new versions of the project file over time. Hardware design was based on previous labs and lecture notes.
* We conducted thorough and repeated testing to identify and debug any issues in our system. Once we resolved the bugs, we gathered verification data to ensure the system's accuracy.

**3.2. Need and Constraint Identification**

Our team identified customer, user, and enterprise needs by adopting the perspective of the end user and considering key factors such as customizability, clarity, safety, and accuracy. Constraints of achieving these needs include our limited project time, resources, and people working on the project. It is essential to focus on the priority needs of the consumer before adding extra features and functionality. Enterprise needs include cost-effectiveness, scalability, and ease of integration with existing systems to ensure long-term sustainability. Additionally, maintaining regulatory compliance and minimizing operational risks are crucial for aligning the project with enterprise objectives.

**3.3. Problem Specification**

We specified additional design requirements by generating ideas based on the needs presented above and then rejecting those that were infeasible given our constraints. This process generated several additional design requirements. For example, we wanted the runtimes and temperatures for various stages of the reflow process to be customizable. For clarity, we wanted the LCD to display important information such as the runtime and the temperature readings. To ensure the safety of our project, we wanted features that can stop the process immediately and can check for unsafe conditions. For accuracy, we also aimed to have the measured temperature and the actual temperature within an acceptable error range.

**3.4. Solution Generation**

Several design solutions were considered to meet the functional specifications of the reflow oven controller, which included successful implementations and discarded approaches:

* **User Input Mechanism:** We explored using push buttons or a joystick to allow users to interact with the system. Ultimately, push buttons were chosen for their simplicity and reliability.
* **Finite State Machine (FSM) in Assembly**: To regulate the reflow process, we implemented an FSM in assembly. This approach ensured precise control over the different states while efficiently managing system resources.
* **LCD Integration:** We designed the LCD to interact with the push buttons, which enabled user input and navigation through the system functions, which was crucial for implementing features such as the passcode entry stage.
* **Temperature Strip Chart Generation:** For data logging and visualization, we evaluated Python and MATLAB as potential tools for generating a strip chart. While both were viable solutions, we ultimately settled on Python due to its ease of integration with serial data logging and flexibility in real-time visualization.

**3.5. Solution Evaluation**

To determine the best solution to meet the project criteria, we compared each solution’s cost of implementation and contribution towards the final circuit.

* Using a joystick instead of push buttons could have enhanced the user experience, but ultimately the time and work required to implement it outweighed its benefits.
* Instead of connecting push buttons directly to the microcontroller, we used the LCD to read five of the push buttons. Doing this allowed more pins to be used for implementing other features.
* As mentioned above, we chose to use Python for the strip chart due to Python’s ease of use along with the fact that our team had more experience with Python data logging than MATLAB.

**3.6. Detailed Design**

**LCD Push Buttons**

We referenced existing documentation to build the LCD push button circuit and to code the program that reads each push button [1], [2]. The circuit diagram is shown below in Figure 2.

A diagram of a circuit

AI-generated content may be incorrect.

Figure 2: Circuit diagram of the push buttons connected to the LCD.

Each diode and push button are connected to a certain pin of the microcontroller. All the push buttons are connected to pin 10, which is also connected to a resistor that goes to a voltage of 5V. The source code for the push buttons is in the appendix. First, the program checks if any push button has been pressed by setting the microcontroller’s pins 11, 16, 17, 18, and 19 to a low voltage. If any push button has been pressed, then pin 10 will have a low voltage, which we can measure. If the program finds that a push button has been pressed, then it will check each button individually. For example, to check the left-most push button, it will set pin 11 to a low voltage and the rest of the pins to a high voltage. We can then measure if pin 10 has a low or high voltage, which would then tell us if the left-most push button has been pressed or not respectively. This process is then done for each push button.

**Op-Amp Voltage Amplifier Circuit**

We referenced existing documentation to build the op-amp voltage amplifier circuit [3]. The circuit diagram is shown below in Figure 3.

**A diagram of a circuit

AI-generated content may be incorrect.**

Figure 3: Circuit diagram of the op-amp circuit.

The OP07 op-amp requires -5V at its pin 4. To acquire this voltage, we used a LMC7660IN Voltage Converter. We built the LMC7660 circuit as according to the reference and verified that it correctly produces the correct voltage at pin 5. For the op-amp, we built the standard voltage subtractor circuit. This creates a voltage gain proportional to the resistor values in the circuit, as shown by the equation below:

In our final circuit, we chose resistor values 10MΩ and 33kΩ, which results in a gain of approximately 303 V/V. The potentiometer at pin 1 and pin 8 is an optional component to null the offset voltage of the OP07 op-amp.

**Finite State Machine**

We referenced existing documentation to help create the code for our FSM [4]. Additionally, our experience with FSMs in CPEN 211 gave us the conceptual understanding to add modifications to the FSM. The source code for the finite state machine is in the appendix. Figure 4 below is a visualization of the FSM in our code.

A diagram of a computer program

AI-generated content may be incorrect.

Figure 4: Block diagram of the finite state machine.

To design the FSM, we first needed to identify which states the system needs. Six of the states (Rest, Ramp to Soak, Soak, Ramp to Reflow, Reflow, Cool Down) are necessary for the reflow oven process. We decided to add two more states (Error, Done) with the intent of displaying messages to the LCD to improve the clarity of our project. The Passcode state was added as an additional feature to our project. Next, we needed to identify how the states connect to each other and how you go from one state to the other. The reflow oven states should be sequential, and moving from one state to the other depends on the time, the temperature, or the state of the push buttons. The Done state occurs after we finish the reflow process, and the Error state is for when we encounter an error. The Passcode state should be the starting state, and you exit it when the correct passcode is entered. After that, we needed to identify the output of each state, which was controlling the PWM and displaying messages on the LCD.

To determine the current state of the FSM, we used a variable “FSM1\_state” that stores the current state as a number. Every time the FSM code is run, “FSM1\_state” is checked to determine which state’s code should be run. Next is determining how to change states. We have more variables “temp” and “sec” that store the thermocouple’s temperature and the state’s runtime respectively. We usually compare the values stored in the variables with other values to determine whether we update “FSM1\_state” to a new number. Other times we check the states of the push buttons to determine the value stored in “FSM1\_state”. Last is determining the outputs of each state. Each state controls how much power is supplied to the SSR, which is done by setting the variable “pwm” to a number from 0 to 100. Additionally, each state had to display a certain message on the LCD. For example, the resting state displayed messages on the LCD by doing “lcall Display\_Setup\_Info”. We had additional variables, “state\_0\_flag”, “active\_flag”, “error\_flag”, and “done\_flag” that depend on if it is the first time we have encountered a specific state. These variables were used to avoid constantly writing messages to the LCD, which was causing flickering. Additional details of the FSM can be found in the source code in the appendix.

**Op-Amp Voltage to Temperature Calculations**

The source code for the op-amp voltage to temperature calculations is in the appendix. The process of designing the code for calculating the temperature of the thermocouple was done using a mathematical analysis of the op-amp circuit [5]. First, we found the output voltage of the op-amp by using a 4.096V voltage references and the ADC readings of the microcontroller.

With the output voltage and the resistor values, we can calculate the voltage of the thermocouple, as shown in the equations below.

As we know the thermocouple generates a voltage of 41μV per °C, we can then calculate the temperature difference of the thermocouple.

As this is the temperature difference between the hot and cold junctions of the thermocouple, we then add the temperature of the cold junction to acquire the temperature of the hot junction.

This is our final equation. The temperature calculation code uses a rearranged version of this equation to calculate the temperature, shown below.

This was done to simplify calculations and to avoid losing precision due to integer division.

First, we pre-calculated and stored its value o a symbol called “CONSTANT”. We also stored a set value to “TEMP\_COLD”. We then read the ADC to acquire the value from the LM4040 and stored that value in a variable “VAL\_LM4040”. Next, we read the ADC to acquire the value from the OP07. We multiplied that value by “CONSTANT”, divided it by “VAL\_LM4040”, then added “TEMP\_COLD” to achieve our final value.

**Passcode Add-on**

Our passcode feature provides extra security to the system, only allowing authorized users to access the reflow over controller. It works as an initial state when the system is first turned on, and the user is prompted with a message on the LCD display asking them to enter a passcode. The user can then use the 4 pushbuttons to enter the correct combination, and the fifth pushbutton to hit enter and check if the passcode is correct or not. If the passcode entered is incorrect, the LCD display displays an error message letting the user know the wrong passcode was entered and returns to the initial display allowing the user to re-enter the correct passcode. When the correct passcode is entered, the system goes into resting state, and functions as normal. The system will prompt for the passcode to be entered for two cases, firstly when the system is booted up, and secondly after the reset button is pressed when the reflow process is complete. Each pushbutton is associated with a certain string, and when the button is pressed the string is stored in memory (Appendix lV.2). After four strings have been stored the code will not accept any more strings to be stored, and upon pressing pushbutton 5, the system taken the string and uses a check passcode function to verify the validity of the passcode. If the passcode is valid, the current state will be changes in the FSM to the rest state (Appendix lV.2). If the passcode is not valid, the current state will remain in the passcode state, prompting the user back to the enter passcode stage.

**Python Display**

To enhance the usability of the reflow oven controller, we developed a **Python-based display interface** that provides real-time visualization of the temperature profile, process progress, and system state transitions. This add-on communicates with the microcontroller via a serial connection, capturing temperature data and user interactions to dynamically update the display.

#### **Temperature Monitoring**

The system continuously reads temperature data from the oven’s thermocouple through a serial connection. The temperature values are plotted in real-time, allowing users to monitor the reflow process. The plot dynamically adjusts its x-axis to accommodate the latest data, ensuring a smooth, continuous display of temperature trends.

#### **Process State Tracking**

To help visualize the finite state machine (FSM) governing the reflow process, the interface includes two progress bars:

* **Overall Process Progress**: Displays the percentage of total process time elapsed.
* **Current State Progress**: Tracks the duration within each reflow stage (e.g., ramp-up, soak, reflow, cooling).  
  Each state transition is automatically detected based on predefined time thresholds, ensuring real-time updates.

#### **Interactive Bouncing Ball Visualization**

To provide an engaging, intuitive representation of the temperature, a **bouncing ball simulation** is incorporated. The ball’s movement speed and background color change dynamically in response to temperature variations:

* **Higher temperatures** accelerate the ball’s movement.
* **The background transitions from cool blue to red, visually reflecting heat levels.**  
  Additionally, a trail effect provides a visual cue of the ball's recent motion, enhancing user feedback.

#### **User Interaction and Control**

Users can interact with the system via keyboard shortcuts:

* Pressing **Spacebar** toggles pause/play for the animation.
* Closing the window properly shuts down the serial connection to prevent data loss.

This add-on provides a **clear, interactive, and user-friendly** way to monitor the reflow process, ensuring accurate tracking of temperature profiles and state transitions in real time.

**Music**

We integrated a speaker to enhance user feedback during the reflow process. Specifically, during the cooling phase, the system plays the Supermario theme song as an indicator that the process is near completion. To achieve this, we programmed converted musical notes into corresponding frequency values, suitable for the microcontroller’s PWM generation. Each note was defined using hexadecimal frequency values, for example: the note E5 (≈659 Hz), DB 0xCE, 0x89, 150 .

**Custom Characters**

During each stage leading up to the cooling state in our oven reflow controller we displayed a flame to indicated ‘heating’ or ‘hot’. Once our oven had finished and our controller began the cooling process, a snowflake to represent ‘cooling’ was displayed. The code for this add-on can be found in Appendix IV. The Create\_Snowflake and Create\_Flame function defines the pixel pattern by writing a sequence of bytes to the Character Generator RAM of the LCD. If the system is in a cooling state, the Display\_Snowflake function is called to put a snowflake icon at a specified cursor position which isn’t already in use; otherwise, the Display\_Flame function is called to put a flame on the LCD.

**Automatic Door-Opening Function**

In order to further automate the reflow oven process, we integrated an XD-3420 DC 12V motor, capable of a maximum speed of 3500 RPM. While the motor's full power was unnecessary for opening the oven door, knowing it had ample torque provided confidence in its reliability. To ensure adequate power supply, we connected the motor to a Xantrex XPH 35-4T Triple DC Power Supply, as well as a switch, which was to be controlled by the reflow oven controller. Through testing, we determined that 4.5V was optimal—providing sufficient force to open the door smoothly without exerting excessive strain that could damage the mechanism.

**3.7. Solution Assessment**

Many parts of our system can be assessed simply with observation because we designed our project with clarity in mind. For example, for us to assess our FSM, we decided to display the current state, temperature, and time on the LCD. This enables us to confirm our FSM was working correctly as we can observe the LCD to confirm the FSM states were changing at the correct times and temperatures. Additionally, we were able to confirm that the FSM was controlling the PWM correctly, as the lights on the oven indicated how much power was being sent to the SSR. These tests allowed us to confirm that our current implementation of the FSM meets the requirements. For the LCD push buttons, we were also able to assess the functionality of our circuit by using the sample code from the references [2]. We also confirmed that the LCD push buttons were compatible with our assembly code as the push buttons were able to be used to customize the reflow oven parameters. For the op-amp circuit, we were able to assess the circuit’s ability to amplify the thermocouple voltage by using the Fluke 45 multimeter to measure the input and output voltages.

As shown above in Investigation, we tested our temperature calculation software by comparing it to the multimeter’s measurements. With a Python program, we could see the difference between them and immediately judge if adjustments were necessary. We confirmed our temperature measurements were accurate to ±3°C after multiple tests. Figure 5 below shows the data collected from testing our final temperature calculations.

A graph showing a graph of a temperature

AI-generated content may be incorrect.

Figure 5: Plot of the difference between the measured multimeter temperature and the calculated microcontroller temperature from temperatures 25°C to 240°C.

Figure 5 suggests that the microcontroller’s ADC measurements had considerable noise, despite attempts in software to reduce it. We suspect that a RC high-pass filter may further reduce noise and thus improve our temperature calculations.

**4. Life-Long Learning**

For this project, we used our experiences from previous courses to help us with programming assembly code and building hardware. For example, courses such as ELEC 201 and CPEN 211 were helpful towards understanding how to amplify the thermocouple voltage using an op-amp and how to design and modify a finite state machine. Furthermore, our experiences with this project reinforced the importance of time management. With better time management, the completion of our project could have gone more smoothly. This project gave us valuable experience with teamwork, project management, software design, and circuit design.

**5. Conclusion**

Our design is a reflow oven controller that is used to attach surface-mount components to a printed circuit board. It allows for users to set parameters and start and stop the oven through push buttons. Additionally, the oven temperature and the runtime are presented by an LCD screen and a Python strip chart. Our design has extra functionality, such as a passcode. We encountered delays debugging the assembly code since only one person could use the physical circuit at a time. We also encountered issues with the circuit seemingly outputting incorrect voltages. However, we were able to demonstrate a completed project in the end. We estimate that the project took 24 hours of work in total.

**6. References**

[1] J. Calviño-Fraga, “Adding Multiplexed Push Buttons to the N76E003 + LCD”, University of British Columbia, Electrical and Computer Engineering, ELEC291/ELEC292, Accessed Feb. 7, 2025

[2] J. Calviño-Fraga, “N76E003 LCD\_Pushbuttons.asm”, University of British Columbia, Electrical and Computer Engineering, ELEC291/ELEC292, Accessed Feb. 7, 2025

[3] J. Calviño-Fraga, “Project 1 – Reflow Oven Controller”, University of British Columbia, Electrical and Computer Engineering, ELEC291/ELEC292, Feb. 2, 2024

[4] J. Calviño-Fraga, “Project 1 – EFM8 board, FSM, NVMEM, and tips”, University of British Columbia, Electrical and Computer Engineering, ELEC291/ELEC292, Feb. 9, 2024

[5] J. Calviño-Fraga, “Project 1 – Reflow Oven Calculations”, University of British Columbia, Electrical and Computer Engineering, ELEC291/ELEC292, Feb. 7, 2025

**7. Bibliography**

Texas Instruments, “LM4040 Precision Micropower Shunt Voltage Reference”, Jun. 2024, <https://www.ti.com/lit/ds/symlink/lm4040.pdf?ts=1739782571374>

Texas Instruments, “LMC7660 Switched Capacitor Voltage Converter”, Apr. 2013, <https://www.ti.com/lit/ds/symlink/lmc7660.pdf>

Texas Instruments, “OP07x Precision Operational Amplifiers”, Mar. 2023, <https://www.ti.com/product/OP07>

**8. Appendices**

**APPENDIX I: LCD\_PB Function**

Reads the Push Buttons connected to the LCD

**LCD\_PB:**

**setb PB0**

**setb PB1**

**setb PB2**

**setb PB3**

**setb PB4**

**setb P1.5**

**clr P0.0**

**clr P0.1**

**clr P0.2**

**clr P0.3**

**clr P1.3**

**jb P1.5, LCD\_PB\_Done**

**Wait\_Milli\_Seconds(#50)**

**jb P1.5, LCD\_PB\_Done**

**setb P0.0**

**setb P0.1**

**setb P0.2**

**setb P0.3**

**setb P1.3**

**clr P1.3**

**mov c, P1.5**

**mov PB4, c**

**setb P1.3**

**clr P0.0**

**mov c, P1.5**

**mov PB3, c**

**setb P0.0**

**clr P0.1**

**mov c, P1.5**

**mov PB2, c**

**setb P0.1**

**clr P0.2**

**mov c, P1.5**

**mov PB1, c**

**setb P0.2**

**clr P0.3**

**mov c, P1.5**

**mov PB0, c**

**setb P0.3**

**LCD\_PB\_Done:**

**ret**

**APPENDIX II: Finite State Machine Source Code**

Source code for the finite state machine.

**FSM1:**

**mov A, FSM1\_state**

**cjne A, #8, FSM1\_not\_passcode**

**ljmp FSM1\_state\_passcode**

**FSM1\_not\_passcode:**

**cjne A, #0, FSM1\_state1**

**;--- RESTING STATE (State 0) ---**

**FSM1\_state0:**

**mov pwm, #0**

**mov sec, #0**

**mov runtime\_sec, #0**

**mov runtime\_min, #0**

**lcall Update\_Variables**

**jb state\_0\_flag, Not\_First\_Time**

**lcall Display\_Setup\_Info**

**setb state\_0\_flag**

**Not\_First\_Time:**

**lcall Display\_Setup\_Info2**

**jb START\_STOP\_BUTTON, FSM1\_state0\_done**

**Wait\_Milli\_Seconds(#50)**

**jb START\_STOP\_BUTTON, FSM1\_state0\_done**

**check\_release0:**

**lcall LCD\_PB**

**jnb START\_STOP\_BUTTON, check\_release0**

**mov FSM1\_state, #1**

**lcall Display\_Active\_Info**

**mov sec, #0**

**FSM1\_state0\_done:**

**ljmp FSM2**

**;--- RAMP TO SOAK (State 1) ---**

**FSM1\_state1:**

**cjne A, #1, FSM1\_state2**

**mov pwm, #100**

**mov A, #TIME\_ERROR**

**clr C**

**subb A, runtime\_sec**

**jnc FSM1\_error\_checked**

**mov A, #TEMP\_ERROR**

**clr C**

**subb A, temp**

**jc FSM1\_error\_checked**

**mov FSM1\_state, #6**

**ljmp FSM2**

**FSM1\_error\_checked:**

**mov A, temp\_soak**

**clr C**

**subb A, temp**

**jnc FSM1\_state1\_done**

**mov FSM1\_state, #2**

**mov sec, #0**

**FSM1\_state1\_done:**

**ljmp FSM2**

**;--- SOAK (State 2) ---**

**FSM1\_state2:**

**cjne A, #2, FSM1\_state3**

**mov pwm, #20**

**mov A, time\_soak**

**clr C**

**subb A, sec**

**jnc FSM1\_state2\_done**

**mov FSM1\_state, #3**

**mov sec, #0**

**FSM1\_state2\_done:**

**ljmp FSM2**

**;--- RAMP TO REFLOW (State 3) ---**

**FSM1\_state3:**

**cjne A, #3, FSM1\_state4**

**mov pwm, #100**

**mov A, temp\_refl**

**clr C**

**subb A, temp**

**jnc FSM1\_state3\_done**

**mov FSM1\_state, #4**

**mov sec, #0**

**FSM1\_state3\_done:**

**ljmp FSM2**

**;--- REFLOW (State 4) ---**

**FSM1\_state4:**

**cjne A, #4, FSM1\_state5**

**mov pwm, #20**

**mov A, time\_refl**

**clr C**

**subb A, sec**

**jnc FSM1\_state4\_done**

**mov FSM1\_state, #5**

**mov sec, #0**

**FSM1\_state4\_done:**

**ljmp FSM2**

**;--- COOL DOWN (State 5) ---**

**FSM1\_state5:**

**cjne A, #5, FSM1\_state6**

**mov pwm, #0**

**mov A, temp**

**clr C**

**subb A, #60**

**jnc FSM1\_state5\_done**

**mov FSM1\_state, #7**

**FSM1\_state5\_done:**

**ljmp FSM2**

**;--- ERROR (State 6) ---**

**FSM1\_state6:**

**cjne A, #6, FSM1\_state7**

**mov pwm, #0**

**jb START\_STOP\_BUTTON, FSM1\_state6\_done**

**Wait\_Milli\_Seconds(#50)**

**jb START\_STOP\_BUTTON, FSM1\_state6\_done**

**check\_release:**

**lcall LCD\_PB**

**jnb START\_STOP\_BUTTON, check\_release**

**mov FSM1\_state, #0**

**clr state\_0\_flag**

**clr active\_flag**

**clr error\_flag**

**clr done\_flag**

**FSM1\_state6\_done:**

**ljmp FSM2**

**;--- DONE (State 7) ---**

**FSM1\_state7:**

**mov pwm, #0**

**jb START\_STOP\_BUTTON, FSM1\_state7\_done**

**Wait\_Milli\_Seconds(#50)**

**jb START\_STOP\_BUTTON, FSM1\_state7\_done**

**check\_release1:**

**lcall LCD\_PB**

**jnb START\_STOP\_BUTTON, check\_release1**

**mov FSM1\_state, #0**

**clr state\_0\_flag**

**clr active\_flag**

**clr error\_flag**

**clr done\_flag**

**FSM1\_state7\_done:**

**ljmp FSM2**

**;--- Common Post-State Code (FSM2) ---**

**FSM2:**

**mov A, FSM1\_state**

**cjne A, #0, FSM2\_not\_state0**

**ljmp FSM2\_done**

**FSM2\_not\_state0:**

**cjne A, #6, FSM2\_no\_error**

**jb error\_flag, Not\_First\_Time1**

**lcall Display\_Error\_Info**

**setb error\_flag**

**Not\_First\_Time1:**

**lcall Display\_Error\_Info2**

**ljmp FSM2\_done**

**FSM2\_no\_error:**

**cjne A, #7, FSM2\_Not\_Done**

**jb done\_flag, Not\_First\_Time2**

**lcall Display\_Done\_Info**

**setb done\_flag**

**;---------------------------------------------------------**

**; ADDED HERE: Once we first enter the DONE state,**

**; call the Mario tune subroutine to play the song \*once\*.**

**;---------------------------------------------------------**

**Not\_First\_Time2:**

**ljmp FSM2\_done**

**FSM2\_Not\_Done:**

**jb active\_flag, Not\_First\_Time3**

**lcall Display\_Active\_Info**

**setb active\_flag**

**lcall Play\_Mario\_Once**

**Not\_First\_Time3:**

**lcall Display\_Active\_Info2**

**jb START\_STOP\_BUTTON, FSM2\_done**

**Wait\_Milli\_Seconds(#50)**

**jb START\_STOP\_BUTTON, FSM2\_done**

**check\_release2:**

**lcall LCD\_PB**

**jnb START\_STOP\_BUTTON, check\_release2**

**mov FSM1\_state, #0**

**clr state\_0\_flag**

**clr active\_flag**

**clr error\_flag**

**clr done\_flag**

**FSM2\_done:**

**ret**

**APPENDIX III: Read\_Temperature Function**

Reads the ADC and calculates the thermocouple temperature.

**Read\_Temperature:**

**anl ADCCON0, #0xF0**

**orl ADCCON0, #0x00 ; Channel 0**

**lcall Average\_ADC**

**mov VAL\_LM4040+0, R0**

**mov VAL\_LM4040+1, R1**

**anl ADCCON0, #0xF0**

**orl ADCCON0, #0x07 ; Channel 7**

**lcall Average\_ADC**

**mov x+0, R0**

**mov x+1, R1**

**mov x+2, #0**

**mov x+3, #0**

**Load\_y(CONSTANT)**

**lcall mul32**

**mov y+0, VAL\_LM4040+0**

**mov y+1, VAL\_LM4040+1**

**mov y+2, #0**

**mov y+3, #0**

**lcall div32**

**Load\_y(COLD\_TEMP)**

**lcall add32**

**mov temp, x+0**

**ret**

**APPENDIX IV**

**Appendix lV.1:**

**Create\_Snowflake:**  
 **WriteCommand(#0x40) ; Memory location for first byte of custom character 0**  
 **WriteData(#0x0A) ; \* \***   
 **WriteData(#0x04) ; \***   
 **WriteData(#0x15) ;\* \* \***   
 **WriteData(#0x0E) ; \*\*\***   
 **WriteData(#0x15) ;\* \* \***   
 **WriteData(#0x04) ; \***   
 **WriteData(#0x0A) ; \* \***    
 **WriteData(#0x00) ;**   
 **ret**  
  
**;===========================================================================**  
 **; LCD Icon Display Routines**  
 **;===========================================================================**  
 **; These routines display a flame when in the reflow state (FSM1\_state=4)**  
 **; and a snowflake in every other state.**  
  
**Display\_Temperature\_Icon:**  
 **mov A, FSM1\_state**  
 **cjne A, #5, Display\_Flame ; If state ≠ 4 (not reflow), show Snowflake**  
 **ljmp Display\_Snowflake ; Otherwise, show Flame**  
  
**Display\_Snowflake:**  
 **Set\_Cursor(1, 9) ; Set cursor to desired position (adjust as needed)**  
 **Display\_char (#0) ; Display custom character 0 (snowflake)**  
 **ret**  
  
**Display\_Flame:**  
 **Set\_Cursor(1, 9) ; Set cursor to desired position (adjust as needed)**  
 **Display\_char (#1) ; Display custom character 1 (flame)**  
 **ret**

**Appendix lV.2: Passcode Add-On Code**

**Passcode LCD Code**

**passcode\_prompt: db 'Enter Passcode: ',0   
passcode\_fail: db 'Wrong Passcode',0  
 passcode\_fail2: db 'Try Again',0  
 blank\_line: db ' ',0   
  
LCD\_Clear:  
 mov A, #0x01   
 lcall LCD\_SendCommand  
 Wait\_Milli\_Seconds(#2)   
 ret  
   
Display\_Passcode\_Info:  
 lcall LCD\_Clear  
 Set\_Cursor(1,1)  
 Send\_Constant\_String(#passcode\_prompt)   
 Set\_Cursor(2,1)  
 Send\_Constant\_String(#blank\_line)   
 ret  
   
FSM1\_state\_passcode:  
 mov pwm, #0  
 lcall Display\_Passcode\_Info   
Passcode\_Wait:  
 lcall LCD\_PB   
 jnb PB4, Passcode\_Button1   
 jnb PB3, Passcode\_Button2   
 jnb PB2, Passcode\_Button3   
 jnb PB1, Passcode\_Button4   
 jnb PB0, Passcode\_Return   
 sjmp Passcode\_Wait   
  
Passcode\_Button1:  
 mov A, #'1'  
 lcall Save\_Passcode\_Digit  
 sjmp Passcode\_Wait  
   
Passcode\_Button2:  
 mov A, #'2'  
 lcall Save\_Passcode\_Digit  
 sjmp Passcode\_Wait  
   
Passcode\_Button3:  
 mov A, #'3'  
 lcall Save\_Passcode\_Digit  
 sjmp Passcode\_Wait  
   
Passcode\_Button4:  
 mov A, #'4'  
 lcall Save\_Passcode\_Digit  
 sjmp Passcode\_Wait  
   
Passcode\_Return:  
 Wait\_Milli\_Seconds(#200)   
 lcall Check\_Passcode   
 mov A, FSM1\_state  
 cjne A, #8, Exit\_Passcode\_Return   
 sjmp FSM1\_state\_passcode   
Exit\_Passcode\_Return:  
 ret  
   
Save\_Passcode\_Digit:  
 Wait\_Milli\_Seconds(#200)   
 push ACC   
 mov A, passcode\_index  
 cjne A, #PASSCODE\_LENGTH, Save\_Digit\_OK  
 pop ACC   
 ret  
 Save\_Digit\_OK:  
 pop ACC   
 mov R0, passcode\_ptr   
 mov @R0, A  
 inc passcode\_ptr  
 inc passcode\_index  
 lcall Update\_Passcode\_Display   
 ret  
   
Update\_Passcode\_Display:  
 Set\_Cursor(2,1)  
 Send\_Constant\_String(#blank\_line)   
 Set\_Cursor(2,1)  
 mov R6, passcode\_index  
 Update\_Passcode\_Display\_Loop:  
 cjne R6, #0, Display\_Asterisk  
 ret  
 Display\_Asterisk:  
 mov A, #'\*'  
 lcall ?WriteData   
 djnz R6, Update\_Passcode\_Display\_Loop  
 ret  
   
Check\_Passcode:  
 mov A, passcode\_index  
 cjne A, #PASSCODE\_LENGTH, Passcode\_Failed   
 mov R0, #passcode\_buffer   
 mov DPTR, #correct\_passcode   
 mov R2, #PASSCODE\_LENGTH   
 Check\_Loop:  
 clr A   
 movc A, @A+DPTR   
 mov B, @R0   
 cjne A, B, Passcode\_Failed   
 inc DPTR   
 inc R0   
 djnz R2, Check\_Loop   
 mov FSM1\_state, #0   
 mov passcode\_index, #0   
 mov passcode\_ptr, #passcode\_buffer  
 lcall LCD\_Clear  
 lcall Display\_Setup\_Info  
 ret  
   
Passcode\_Failed:  
 Set\_Cursor(1,1)  
 Send\_Constant\_String(#passcode\_fail)   
 Set\_Cursor(2,1)  
 Send\_Constant\_String(#passcode\_fail2)  
 mov passcode\_index, #0  
 mov passcode\_ptr, #passcode\_buffer  
 Wait\_Milli\_Seconds(#200)   
 Wait\_Milli\_Seconds(#200)   
 Wait\_Milli\_Seconds(#200)   
 Wait\_Milli\_Seconds(#200)   
 Wait\_Milli\_Seconds(#200)   
 Wait\_Milli\_Seconds(#200)   
 Wait\_Milli\_Seconds(#200)   
 Wait\_Milli\_Seconds(#200)   
 Wait\_Milli\_Seconds(#200)   
 Wait\_Milli\_Seconds(#200)   
 Wait\_Milli\_Seconds(#200)   
 ret  
   
LCD\_SendCommand:  
 clr LCD\_RS   
 mov B, A   
 swap A   
 anl A, #0x0F   
 mov P0, A   
 setb LCD\_E   
 nop   
 nop  
 clr LCD\_E   
 mov A, B   
 anl A, #0x0F   
 mov P0, A   
 setb LCD\_E   
 nop   
 nop  
 clr LCD\_E   
 ret**