ELEC 291 Reflow Oven Controller

Instructor: Jésus Calviño-Fraga

Section: L2A

Group: A07

Date: Feb. 28, 2025

Student	Student Number	Points (%)	Signature
Rishi Upath	18259374	100	
Chathil Rajamanthree	32523201	100	
Colin Yeung	70611371	100	
Santo Neyyan	55096309	100	
Eric Feng	70120548	100	
Warrick Lo	47938733	100	

Table of Contents

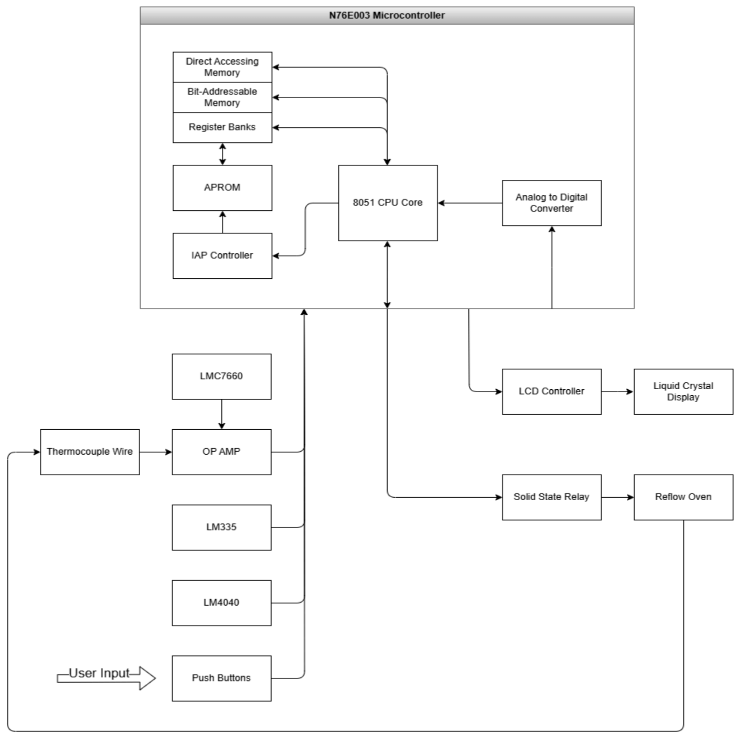
I. Introduction

The objective of this project was to design and develop a reflow oven controller for soldering surface mount devices (SMD) on printed circuit boards (PCB). The controller needed to regulate a standard 1500W toaster oven using a solid-state relay (SSR) and accurately measure temperature using a K-type thermocouple with cold-junction compensation. The design met the following specifications:

1. Utilized the N76E003 microcontroller system and A51 Assembly language.
2. Accurately measured temperatures between 25° and 240°C within +/- 3°C, validated using lab multimeters.
3. The user interface included adjustable reflow parameters using push buttons, and push button reflow start/stop.
4. During the reflow process an LCD displayed real-time oven temperature, ambient temperature, elapsed time, and current state.
5. Temperature strip chart plotted live temperature readings, with data transmitted via the serial port.
6. Automatic reflow termination if the oven temperature does not reach 50°C within the first 60 seconds of operation.

Hardware Design:

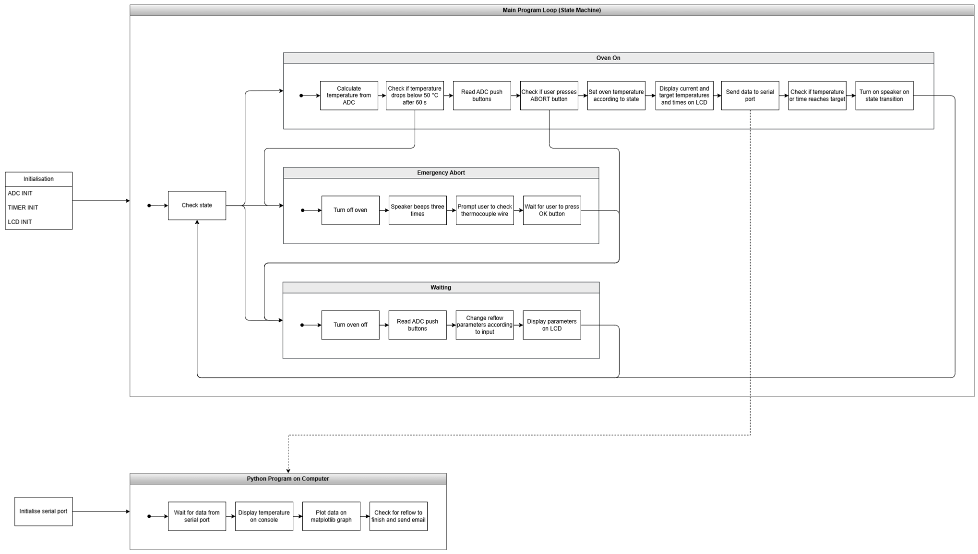
The system used a K-type thermocouple in tandem with an OP07 and LMC7660 to provide real-time temperature readings from the oven. Along with an LM335 to provide accurate ambient temperature data. Both readings then utilize an LM4040 voltage reference for accuracy and keeping stable readings. The N76E003 microcontroller processed sensor data and controlled the oven through the SSR box. User interaction was handled through pushbuttons and an LCD display. The CEM-1302 buzzer provided audio feedback to indicate different stages of the reflow process. A high-level diagram of the hardware system is provided below:



Hardware Block Diagram (Appendix C)

<u>Software Design:</u>

The firmware was written in the A51 Assembly language. The program processed raw data from temperature sensors and performed necessary calculations into temperature data. The data is transmitted in real-time through the serial port for monitoring on a computer and graphed using Python. A finite state machine managed reflow cycle transitions based on temperature and time readings. To regulate the oven's power according to the set reflow profile, a PWM signal is output by the microcontroller. A high-level diagram of the software system is provided below:



Software Block Diagram (Appendix D)

II. Investigation

Idea Generation

We created a communication channel through Discord for brainstorming project specifications and additional functionalities. This served as a centralized space to document ideas, build on each other's suggestions, and continuously expand our concept list as the project progressed.

To generate ideas, we researched existing reflow oven controllers to understand their functionality and identify what we can implement in our project. We then explored ways to enhance different aspects of our design with added features.

When considering extra functionalities, we prioritized ideas based on difficulty, time required, and overall project benefit. This structured approach ensured that everyone was aligned on the project direction and aware of the implementation priorities.

Investigation Design

Before integrating new features into the main code, we first tested them individually on a separate code file and test board. This approach allowed us to debug efficiently and work on different components in parallel without affecting the overall system functionality. We applied this method to various subsystems, including:

- Buzzer operations - proper activation and timing.
- Python Script - data logging and real-time serial temperature plotting.
- Thermocouple and LM4040 reference - confirmed temperature calculations for accuracy.
- PWM operations - confirmed working PWM signal with oven.
- LM335 temperature sensor - confirmed reliable ambient temperature readings.

After the modules worked individually, we integrated them into the finite state machine logic. Although debugging the finite state machine took some time, this method significantly narrowed down where issues occurred, making the troubleshooting process much more manageable.

Data Collection

The microcontroller reads 4 different analog inputs: input from ADC push buttons, input from the LM4040, input from the LM335, and voltage readings from the thermocouple wire amplified by the OP07. For the ADC buttons, the microcontroller switches to a proper analog input channel and correlates each button press to a specific voltage. As the buttons are pressed, the voltage in the pin changes and thus the software recognizes which button is pressed. Both the data from the LM4040 and LM335 are read periodically from their respective analog channels each time the microcontroller cycles through the main loop.

Finally, to collect data and voltage readings from the thermocouple, we used the operational amplifier. We power the op-amp with ± 5V using the LMC7660 to ensure proper amplification. Specifically, with amplifying the voltage input as much as possible, we chose our resistors to a calculated max gain value.

Data Synthesis

After the data is read and input to the microcontroller, the Assembly program synthesizes each analog input using various calculations. For each reading, we specify the analog input channel connected to the correlated input pin within our program. With the ADC pushbuttons, the program opens the proper analog input channel and identifies which push button is being pressed by comparing the voltage reading. Next, the readings from the LM4040 temperature reference are stored in the microcontroller for later use when synthesizing the temperature data. Similarly, the voltage reading from the LM335 ambient temperature sensor is first converted to temperature using specific calculations and then stored in memory.

To synthesize the temperature readings of the toaster oven, it is necessary to find the sum of the temperature of the hot and cold junctions of the thermocouple wire. The cold junction temperature correlates to the temperature value from the LM335. To obtain the temperature of the hot junction, we used the voltage readings from the operational amplifier, OP07. Specifically, the exact calculations can be viewed in Appendix B. In summary, the OP07 readings were first divided by the stored reference voltage from the LM335 to keep our data stable. Then, temperature was calculated by multiplying a ratio of the resistor values connected to the OP07 and constants. These calculations were derived by relating voltage output from the amplifier and the input voltage correlating to temperature from the thermocouple wire.

Furthermore, our group synthesized temperature reading data from the laboratory multimeter and its measurement of voltage data directly from the thermocouple wire. The python test code provided in Canvas allowed us to visualize the changing temperature readings. The multimeter provides a more accurate way of measuring the thermocouple wire voltage as it is capable of reading very small voltage. By comparing the temperature readings from the multimeter and the OP07, we concluded that our calculations for the temperature readings were valid and realistic.

Analysis of Results

To validate our measurement accuracy and conclusions, we conducted multiple tests measuring temperature readings from both the laboratory multimeter and the operational amplifier circuit. In particular, there are many limitations in theory and variations in calculations that occur when synthesizing temperature data using voltage input from the OP07. The conversion from voltage to temperature is a derived theoretical equation and there could be many different offsets in measurement due to the hardware. However, the multimeter provides a far more accurate testing method as it directly measures the low input voltages from the thermocouple.

In general, we assure accuracy relative to these errors by comparing the two sets of temperature data from the microcontroller and multimeter. These tests were performed over a temperature range of 20°C to 240°C, simulating realistic operating conditions of the reflow process. To be specific, we ran a python script which converted and logged the temperature readings from the multimeter after adjusting to the correct ambient temperature. As well, the separate set of data the microcontroller and OP07 was logged and received after being written through the serial port. Then, we moved the various sets of data to Excel and inspected the degree of difference between the data. Throughout multiple tests, the degree of error consistently stayed below 3°C and stayed particularly low at higher temperatures. The variations that did occur could be caused by inconsistencies due to ambient conditions and slight errors in calculations or interference in signals. One set of test data is shown in Appendix E. Overall, the low degree of error suggests that our approach provides precise temperature control suitable for reflow soldering applications.

III. Design

Use of Process

First, our group defined the problem together and researched possible methods to achieve each objective using the lecture slides and project document. We documented our materials and components to properly understand what solutions we could apply for different objectives. With generation of ideas, each individual idea was documented and assessed based on a time effort ratio. While developing the various sections of the project, we communicated with each other to see how our pieces fit into the overall system. In particular, we tried to optimize the integration of different hardware and software components while keeping our design efficient and organized. Iterations were made when developing each module independently and collaboratively for larger design decisions.

Key Design aspects considered:

- Efficient usage of pins on the microcontroller
- Simplicity of user interface
- Reliability and accuracy of mathematical calculations
- Extensive set of unique and practical features

Need and Constraint Identification

The purpose of this reflow oven controller is to carry out the reflow soldering process for students in ELEC 291. To ensure successful operation, the controller must reliably control the oven, accurately regulate temperature, and follow the reflow profile selected. The system must be user-friendly, allowing the user to easily set temperatures and times with push buttons. Necessary information such as temperature, time, and current state should be clearly indicated to the user at all times. The user should be able to quickly understand how to operate the system without extensive training. Considering important safety measures is also critical as this project involves working with high temperatures.

The microcontroller itself was constrained with a limited number of input pins and lesser processing power. The software had to be written in Assembly language which complicated the implementation of features to the controller. The project also had to be designed, built, and tested within a limited time, which influenced the scope of additional features we added.

Problem Specification

Since the microcontroller had a limited number of pins, we defined a requirement of allocating necessary inputs based on what features would be added. Recognizing the importance of usability, a shift button was incorporated along with ADC push buttons to increase the number of available buttons while minimising pin usage. As a safety precaution, an emergency abort state was made so the oven will turn off if a certain temperature is not reached in a specified amount of time. As well, for practicality in assisting users, an email is sent to the user when the reflow soldering process is finished.

Solution Generation

To meet the functional specifications of our project, we explored multiple design solutions, incorporating various features to enhance usability, safety, and efficiency. Below is a summary of the design choices we made.

Implemented Design Solutions:

1. ADC Push Buttons for Feature Control:
   - We employed Analog-to-Digital Converter (ADC) push buttons to control different features efficiently. This approach minimized the number of input pins while allowing multiple functions to be mapped to specific buttons.
2. Shift Button for Parameter Adjustment:
   - A shift button was included to allow users to decrement parameter values. This feature facilitated precise adjustments when setting specific values for the system.
3. Emergency Abort State for Safety:
   - The system continuously monitors the oven temperature to ensure it adheres to the reflow curve. If the temperature exceeds safe limits, an emergency abort state is triggered to prevent overheating of the circuit board.
4. Buzzer Notification System:
   - A buzzer, amplified by a MOSFET, is activated at specific transition times to audibly notify users of important process changes.
5. Email Alerts and Temperature Data Logging:
   - A Python-based email notification system was integrated to alert users when the reflow temperature in the oven is reached. Additionally, temperature readings are included in the email allowing users to verify the accuracy of the reflow curve.

6. Preset Buttons for Reflow Profiles:
    - We implemented preset buttons that enable users to select predefined reflow profiles. This feature is particularly useful when working with different solder pastes, as each may require a unique heating profile.

Through these design iterations, we successfully developed a functional and user-friendly system while also learning from the challenges encountered in our attempted solutions.

<u>Solution Evaluation</u>

The final design that was chosen encompassed most of the functions which we originally planned to implement. The various push buttons met the project requirements of selecting parameters, temperature, running time, and reflow state. The emergency abort state was a feature which was specifically mentioned as a requirement, so its implementation was prioritized. In addition, a buzzer notification system was implemented to notify the user when different states of the reflow soldering process transition. For user functionality, an email is sent to the user when the reflow soldering temperature is reached, with an attachment of the temperature data. These additional features were implemented to provide more practicality for the controller and efficiency throughout the reflow process. We also gave the user more flexibility by allowing them to preset different reflow profiles.

Our team was limited by time constraints and technical difficulties in implementing other features of our design solution. Another proposed idea was playing a custom song from the buzzer which was too complex due to intricacies of Assembly syntax.

Our final design was selected for its minimization of I/O pins, convenient and accessible controls, and its additional features which serve as improvements to the pre-existing requirements.

Detailed Design

This section will outline the methodology and engineering principles employed throughout the development of this project's key components. We will provide an in-depth explanation of each block and the approaches that were taken to design each part, supported by relevant diagrams and source code.

General Overview:

The primary purpose of our system is to control the solid state relay (SSR) that powers the reflow oven. This is achieved by sending a pulse width modulated (PWM) signal to the SSR as the output. A thermocouple wire is installed inside of the oven to maintain a closed feedback loop. We employ the N76E003 chip from Nuvoton for our microcontroller chosen for its low cost and the compatibility with the 8051-based platform.

Timer Initialisation:

In the project, timers were needed for time tracking, PWM output for the SSR box, output for the speaker, output for the serial port, and delay functions. As a result, 4 timers were utilised: timers 0, 1, 2, and 3. Timer 0 (line 238) is initialised to be in 16-bit mode (mode 1). The timer starts off disabled as it will be used for delay functions. Timer 1 is used for the baud rate, at 115200 bit/s. The input for timer 1 is set to the system clock (line 226). PCON.1 is set to HIGH to enable double baud rate and serial port receiving is enabled with SCON.4. Finally, timer 1 is set to 16-bit mode (mode 1) and the reload value is moved into TH1 for auto-reloading. Timer 2 (line 260) is used to keep track of time. Timer 2's clock divider is set to 1/16. Timer 2's interrupt is then enabled, followed by initialising its reload values. Timer 3 (line 255) is used purely for the output speaker.

Reflow Oven Parameter Control:

Eight push buttons were configured using the ADC system in the microcontroller as described in the Data Synthesis section. Four buttons are used to control each reflow parameter including soak time, soak temperature, reflow time, and reflow temperature. In the IDLE state, the system checks for presses of the push buttons. Once a button is pressed, the system checks if the shift button is also concurrently held to have decrementing.

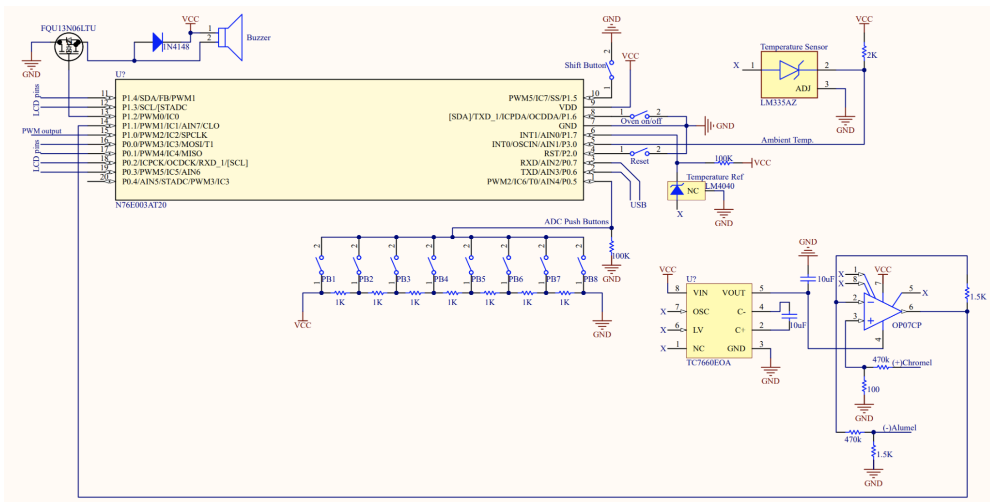
The remaining four buttons are used to implement 4 loadable reflow profiles. The program also listens for the SHIFT button. If this button is pressed when one of the four preset buttons are activated, it will save the current temperature to the preset that corresponds to the button. Otherwise, it will load that button's parameters.

To write data to APROM, the page must first be cleared. Thus, all other preset values must be saved beforehand. This is done on line 1054. Afterwards, the current values are stored in registers that correspond to the profile. In our implementation, we used two register banks of eight registers each to store the 16 different values (4 profiles, 4 values per profile). We then clear page 0x400 to prepare it for writing. The current register bank is switched to bank 1, and the registers are moved into APROM one by one (util.inc, line 7; util.inc, line 59). The process is then repeated for register bank 2. Finally, the hex code 0x55AA is written as a signature to the end of the page (util.inc, line 83). This is to ensure that the values in RAM are properly initialised on the first boot.

Temperature Measurement:

A K-type thermocouple wire is used to measure the temperature inside the oven. The wire has a temperature range from -200 °C to 1350 °C, making it more than sufficient for our use case [1].

This type of thermocouple consists of a chromel wire and an alumel wire joined together at a junction (the “hot” junction). The differing wire materials produce a voltage difference due to the Seebeck effect [2]. As shown in Appendix B, the chromel (positive) and alumel (negative) legs are connected to the non-inverting and inverting inputs of the OP07 op-amp, respectively.



Detailed Diagram of Circuit (Appendix G)

The OP07 is configured as a differential amplifier, with input resistors measuring approximately 1.474 kΩ and feedback resistors at 464.2 kΩ. Thus, a gain of about 314.9 (24.98 dB). The output of the differential amplifier is connected to pin 1.1 of the microcontroller, which is the analog-to-digital converter (ADC) on channel 7. The positive power rail of the op-amp is connected to VCC (+5 V), while the negative rail connects to the TC7660 voltage converter. This device converts a voltage between +1.5 V and +10 V to its corresponding negative voltage. As the TC7660 is also connected to VCC, the output which feeds into the op-amp will be -5 V.

Thermocouple wires give a voltage difference proportional to the temperature difference between the hot and cold junctions [3]. Thus, we needed another device to read the ambient temperature.

We chose the LM335 temperature sensor for this task, primarily for its low cost and less than 1 °C error at 25 °C, making it ideal to measure the room temperature [4]. The output of the temperature sensor is connected to pin 3.0 and ADC channel 1.

As a voltage reference, we used the LM4040. Particularly, this gives a stable output voltage of 4.095 V when the temperature is between -40 °C and 85 °C. We compare the LM335's output voltage to the reference 4.095 V to obtain an accurate measurement of the ambient temperature. The LM4040 voltage reference output is connected to pin 1.7, ADC channel 0.

During testing, an issue arose when connecting the thermocouple wire to the multimeter. Without the multimeter connected, the temperature readings would maintain a steady number of around 20 °C. However, after attaching the multimeter probes, the temperature would fluctuate ranging from 20 °C to over 50 °C.

During debugging, we determined that the cause of the fluctuations was from the hardware, although the exact cause was unknown at that time. After conducting many tests and consulting with the professor, we determined that the probable cause is the multimeter raising the voltage at the non-inverting input of the amplifier. The professor suggested placing a small resistor connecting the non-inverting input to ground. We chose a resistor of ~100 Ω for stable values.

In the microcontroller software, the temperature readings are computed in the `READ_TEMP` subroutine, as shown in Appendix H starting on line 804. The subroutine follows the shown equation to calculate the oven temperature:

$$T_H = 4.096\,\mathrm{V} \cdot \frac{R_2}{R_1 \cdot 41\,\mu\mathrm{V/°C}} \cdot \frac{\mathrm{ADC_{OP07}}}{\mathrm{ADC_{ref}}}. \tag{1}$$

The result of equation (1), along with the ambient temperature, is stored into RAM for later use. Utility subroutines were written (line 778) for clarity when switching between ADC input channels. These readings are then sent through the serial port where they are logged and graphed using the Python Script in Appendix H.

Finite State Machine:

The majority of the main program loop is controlled by the finite state machine (FSM). This section of the code is responsible for all outputs to the reflow oven, LCD, and speaker, along with managing inputs from the push buttons and temperature sensors. A diagram of the state machine and example state transitions are provided in Appendix A. State machine values that are used internally are defined at the top (line 75).

The FSM state is initialized to the WAIT state. This state listens for push button presses, either from the ADC, or directly from the GPIO pins. The ADC push buttons are processed as specified. When a push button press is detected (line 388), it jumps to the button's handler subroutine. If the start/stop oven button is pressed, the FSM transitions to the next state.

The finite state machine recognises that if the state is not WAIT, the oven must be turned on. Displaying the temperature is common to all stages of the reflow process (when the oven is turned on). Therefore, if the FSM is not in the wait state, will output to the LCD display the current oven and ambient temperatures (line 351). The program then jumps to the line containing code for the current state.

In the PREHEAT state, the program checks if the elapsed time is over 60 seconds. If this condition is met, the program would then check if the temperature is over 50 °C. If this second condition is not met, the program will turn off the oven and transition to the EMERGENCY state. This is included for safety reasons as the thermocouple wire may be improperly connected. The logic is also written such that any time the temperature drops below 50 °C after 60 s (e.g. at

90 s), the oven will still abort. This is again for safety, in case a scenario occurs where the thermocouple wire is disconnected well into the reflow process.

The four stages of the reflow process work similarly. In the PREHEAT and RAMP states, the system provides 100% PWM to the SSR and continuously checks if the user set temperature is reached. Once met, the system transitions to the next stage. In the SOAK and REFLOW states, the system provides 20% PWM to the SSR to maintain a steady temperature and utilizes timer 2 to check if the user set times are met.

The EMERGENCY state (line 324) handles automatic safety aborts in the reflow process. This state simply plays three beeps on the speaker and displays a message on the LCD screen telling the user to check the thermocouple wire connection. The user can then press the oven start/stop button to go back to the WAIT state.

After completing the reflow process, the system returns to the IDLE state.

Solution Assessment

Temperature Accuracy:

To validate the accuracy of temperature measurements, we compared temperature readings from our microcontroller with those from a lab multimeter. The goal was to ensure that the microcontroller's temperature readings remained within the project specifications of ± 3° error range between temperatures of 25°C to 240°C. To ensure consistency, 3 trials were conducted with this temperature range. A table summarizing one validation test is provided in Appendix E.

Finite State Machine (FSM) Logic:

The oven controller functions through a FSM which transitions between the different states of the reflow process. This was central to the system's functionality and was tested extensively by cycling through multiple states and verifying transitions. Special attention was given to “edge” cases, such as transitions at the end of a minute (59s) and temperatures ending in 0. The system reliably and correctly advanced through the FSM states performing the tasks within each state.

<u>User Interface System:</u>

The user interface of the oven controller is critical for usability of the system. All push buttons were tested in each state they were required to operate in. Specifically, we confirmed parameters could be incremented and decremented. Additionally, we tested the loading of each predefined reflow profile to ensure the presets worked properly. The LCD was analyzed in each state to confirm that relevant information was displayed properly.

<u>Safety Mechanism:</u>

Safety is critical to this project as it involves working with high temperatures. The abort state safety features were tested by deliberately creating scenarios where the oven would fail to reach 50°C within the first 60 seconds of operation. The system correctly detected that condition and automatically terminated the reflow process. Once the system aborts, the user is required to press the “oven-on” button to return back to the reflow profile selection.

<u>Overall System Performance:</u>

The reflow oven controller’s performance was assessed by evaluating how accurately it followed the reflow temperature profile. Temperature readings from the microcontroller system were plotted using Python and compared to the expected profile. The temperature graph provided a clear visualization of each state at the expected temperatures. This confirmed the system was working properly. The reflow temperature plot is shown in Appendix F.

IV. Life-Long Learning

This project provided us with a unique opportunity to expand our knowledge in designing microcontroller circuits, programming with Assembly, and understanding the reflow process. The main topic of this oven reflow controller was an innovative concept that all our team members found to be efficient for assembling printed circuit boards. Beyond the technical aspects, we also gained valuable experience in time management, collaborative project development, and debugging strategies. The multitude of tasks forced us to develop and apply strict time management strategies to meet deadlines. Through facing many integration issues, we learned how to leverage each other's strengths to complete objectives as efficiently as possible. This project also gave us chances to develop our knowledge of storage memory with custom preset adjustments and pulse width modulation. Overall, the experiences that we gained from this project were arduous but also rewarding.

One course in particular that prepared us for this project was ELEC 201, where we developed a strong foundation in operational amplifiers and general circuit design. The knowledge from this course helped us quickly identify hardware errors and understand the functionality of different components. The laboratory experiences from this course also helped us feel familiar using different tools such as multimeter and oscilloscopes. In general, the circuit analysis tool that we learned in ELEC 201 allowed us to excel in this oven reflow controller project.

Similarly, CPEN 211 equipped us with the skills necessary for assembly programming and finite state machine development. In particular, our previous experiences with Assembly in this course allowed us to approach this project with confidence in writing efficient and organized code. The collaborative labs taught us how to maintain strong communication within a team environment.

V. Conclusion

The oven reflow controller of group A07 was designed to automate the reflow soldering process. Notable features included a buzzer notification system, email alerts, temperature data logging, and presets for reflow profiles. ADC push buttons were implemented for user practicality, alongside an emergency abort state for safety precautions.

In terms of problems we encountered, most occurred when integrating new features into the main file. Certain obscure issues relating to syntax and memory allocation also arose and became time-consuming. For example, a finite state machine state variable was not initialised in the correct DSEG location. As well, the thermocouple temperature was jumping irregularly only when connected to a multimeter. When consulted with Dr. Calviño-Fraga, the latter was fixed using a pull down resistor at the thermocouple's negative terminal. Other issues were minute, and attributed to human error. In general, we were able to fix most errors through comprehensive debugging and review.

The project required approximately 55 hours of work:

- Hardware assembly: 10 hours
- Assembly firmware development and module integration: 30 hours
- Serial interface through Python and bonus features: 7 hours
- Final testing and calibration: 8 hours

VI. REFERENCES

[1] Wikipedia contributors, "Thermocouple—Type K," *Wikipedia, The Free Encyclopedia*, Feb. 25, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Thermocouple#Type_K. [Accessed: Feb. 24, 2025].

[2] Wikipedia contributors, "Thermoelectric effect," *Wikipedia, The Free Encyclopedia*, Feb. 29, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Thermoelectric_effect. [Accessed: Feb. 24, 2025].

[3] W. F. Roeser, A. I. Dahl, and G. I. Gowens, "Standard tables for Chromel-Alumel thermocouples," J. Res. Natl. Bur. Stand., vol. 14, Mar. 1935.

[4] Texas Instruments, "LM335: Precision temperature sensor," *Texas Instruments*, [Online]. Available: https://www.ti.com/product/LM335. [Accessed: Feb. 24, 2025].

VII. BIBLIOGRAPHY

C. K. Alexander and M. N. O. Sadiku, *Fundamentals of Electric Circuits*, 5th ed. New York, NY, USA: McGraw-Hill, 2013.

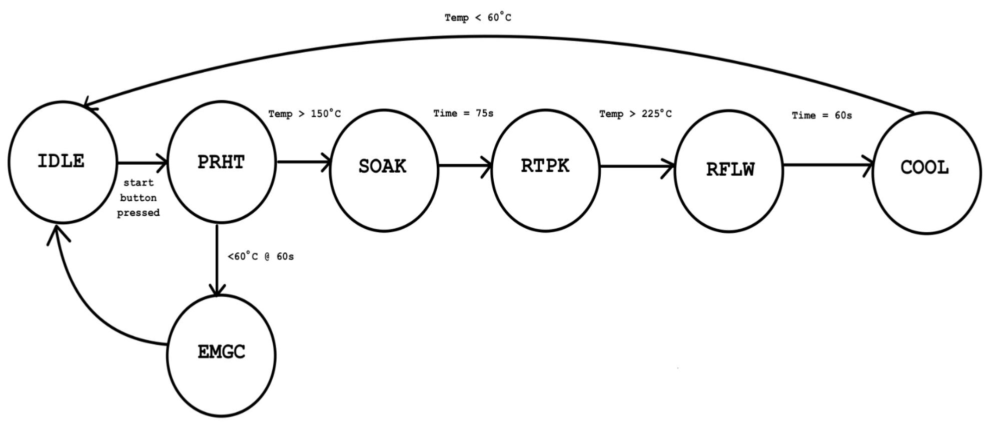
Eindhoven University of Technology, "8051 Instruction Set," *Eindhoven University of Technology*, [Online]. Available: https://aeb.win.tue.nl/comp/8051/set8051.html. [Accessed: Feb. 23, 2025].

Nuvoton Technology Corporation, *N76E003 Datasheet*, [Online]. Available: https://www.nuvoton.com/export/resource-files/DS_N76E003_EN_Rev1.09.pdf/N76E003_DS_EN.pdf. [Accessed: Feb. 20, 2025].

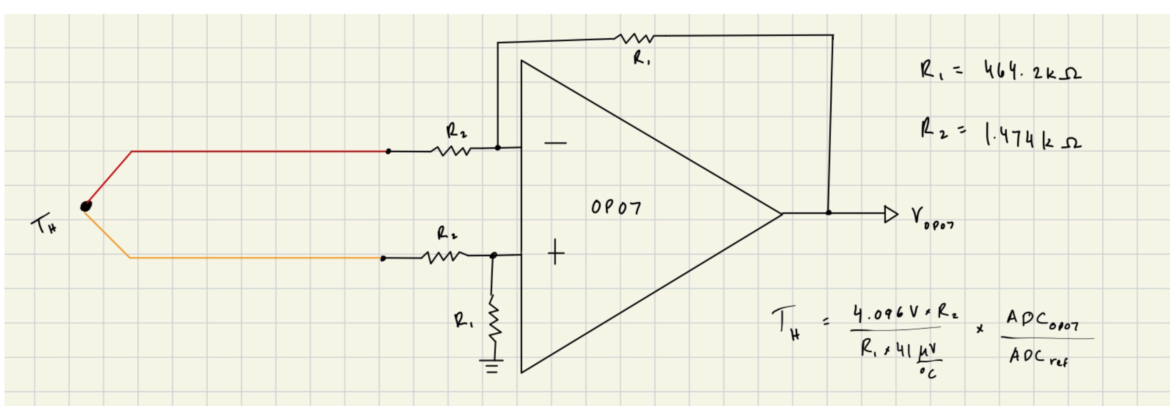
Wikipedia contributors, "Reflow soldering," *Wikipedia, The Free Encyclopedia*, Feb. 25, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Reflow_soldering. [Accessed: Feb. 26, 2025].

VIII. APPENDIX

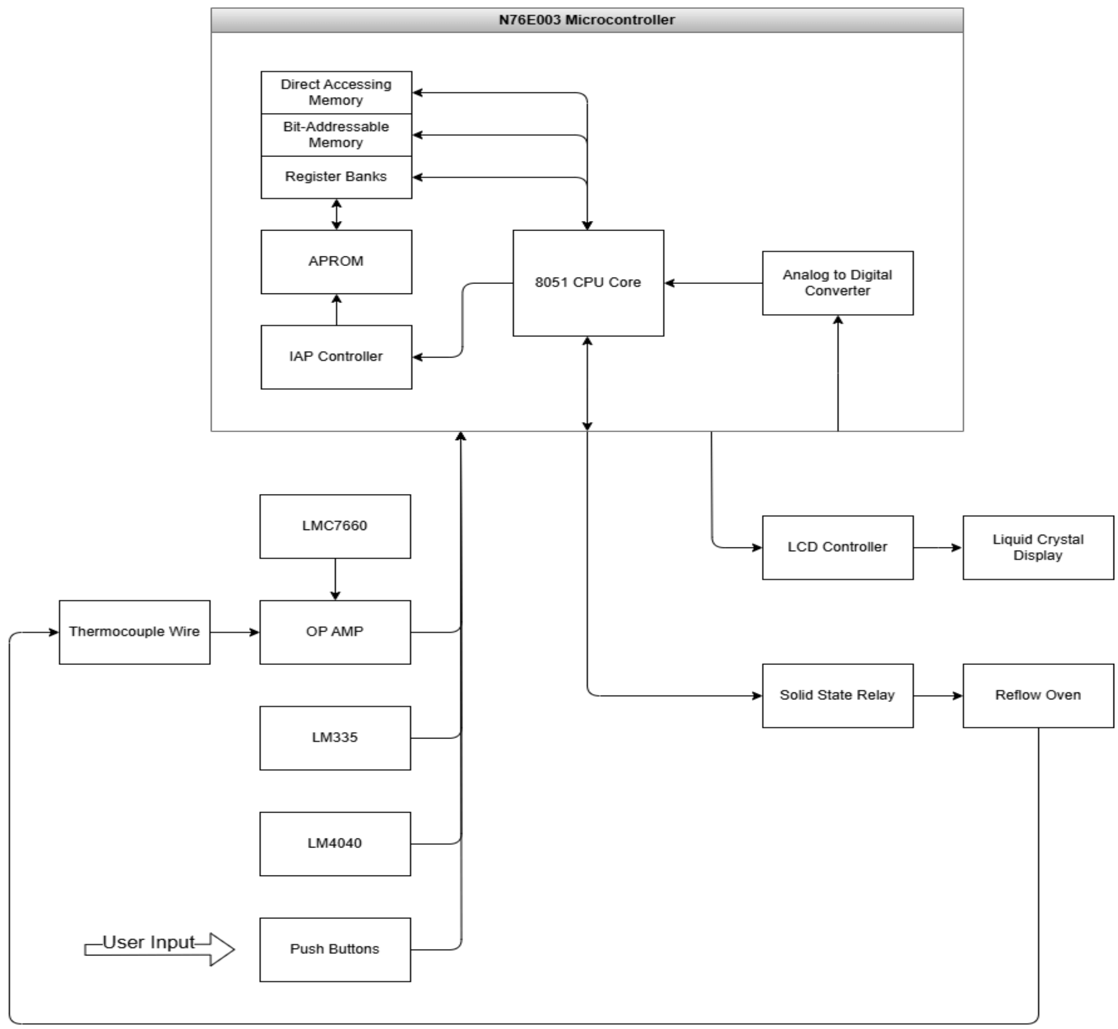
Appendix A: Finite State Machine State Transition Diagram



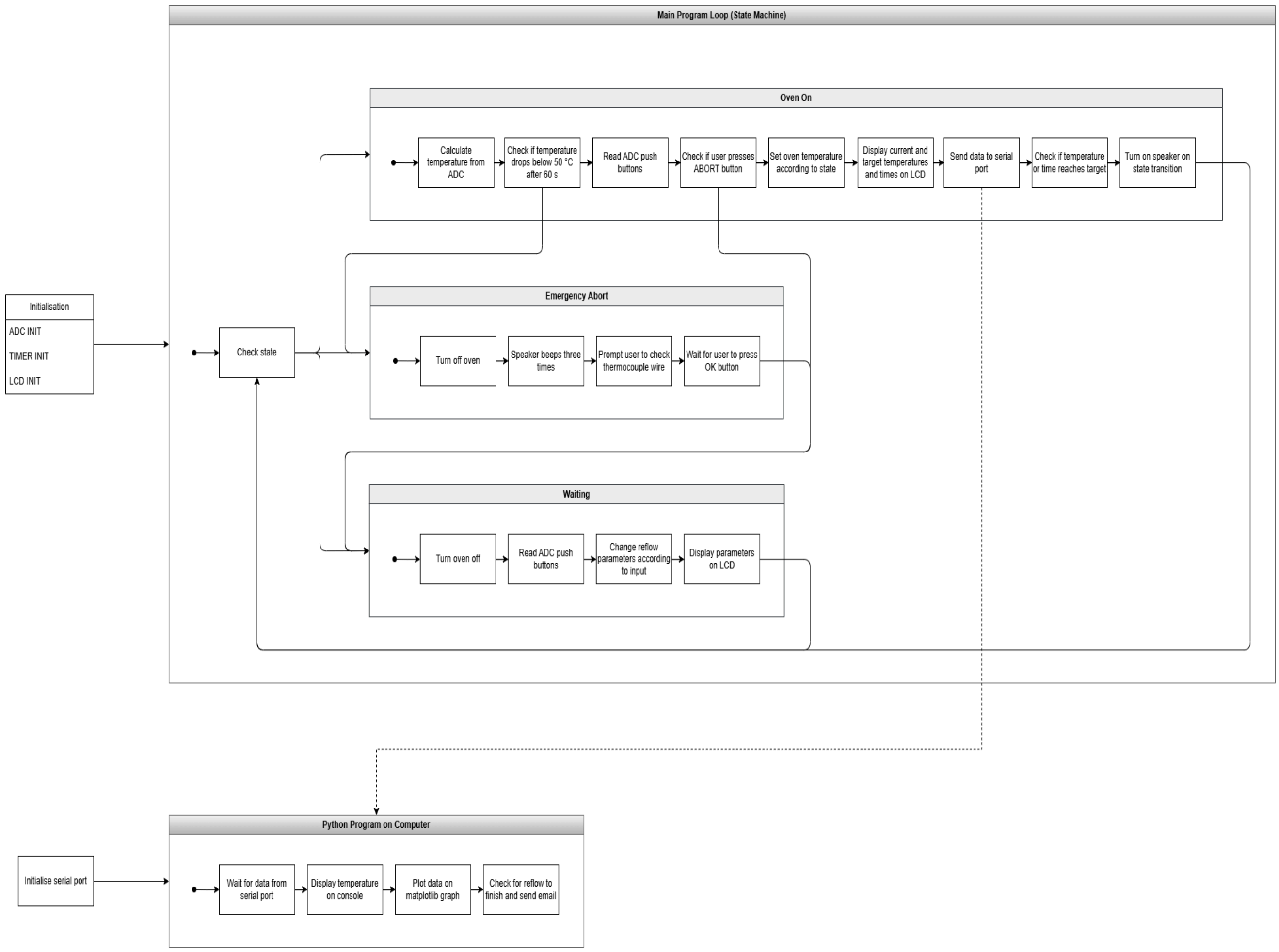
Appendix B: Diagram for Operational Amplifier and Temperature Calculations



Appendix C: Hardware System Block Diagram



Appendix D: Software System Block Diagram



Appendix E: Temperature Validation Data

Multimeter	Microcontroller	Error
20.5	22.78	2.28
21	22.89	1.89
21.1	22.93	1.83
22	23.2834	1.28
22.2	23.5512	1.35
23	24.5804	1.58
24	25.4763	1.48
25.5	26.8587	1.36
27	28.1714	1.17
28.1	29.2397	1.14
29.3	30.7575	1.46
30	30.6953	0.7
31.7	32.9231	1.22
32.1	33.3536	1.25
33	34.956	1.96
34.3	36.2104	1.91
36.1	37.8329	1.73
37.3	38.4395	1.14
38.3	40.0092	1.71
39.3	41.1709	1.87
40.2	41.5221	1.32
41.2	42.5153	1.32
42.7	44.1831	1.48
43.4	44.9051	1.51
43.9	45.5956	1.7
45	46.5641	1.56
46.3	48.8163	2.52
47.4	48.8275	1.43
48	49.5787	1.58
49.4	50.5234	1.12
50.4	52.1739	1.77
51.7	52.9407	1.24
52.4	54.2541	1.85
52.9	54.5846	1.68
54.2	55.8119	1.61
55.1	57.3665	2.27
56.6	57.988	1.39
57.1	59.0629	1.96
58.9	60.1882	1.29
60.6	62.7417	2.14
61.7	63.4495	1.75
62.4	64.6748	2.27
63.5	65.4868	1.99

Multimeter	Microcontroller	Error
63.5	65.4868	1.99
64.6	66.52	1.92
65	67.0773	2.08
66.8	68.5697	1.77
67.4	69.4695	2.07
68	70.6012	2.6
69.8	71.7448	1.94
70.4	72.7021	2.3
71	73.1	2.1
72	73.3724	1.37
72.5	74.238	1.74
73	75.0565	2.06
74.9	77.1921	2.29
75.4	77.2429	1.84
76	78.2958	2.3
77	79.4758	2.48
78.3	81.0146	2.71
79.8	81.6885	1.89
80.3	82.3587	2.06
80.9	83.2185	2.32
82	84.2536	2.25
83.2	85.61	2.41
84.9	86.9224	2.02
85.4	88.1953	2.8
86.1	88.8329	2.73
87.3	89.7792	2.48
88.4	90.9839	2.58
89	91.2424	2.24
90.2	92.7914	2.59
91.4	93.98	2.58
92	94.53	2.53
93.2	95.7809	2.58
94.2	96.937	2.74
95.3	97.6865	2.39
96.5	99.25	2.75
97.1	99.7648	2.66
98.2	100.4663	2.27
99.3	101.9978	2.7
100.5	103.0368	2.54
101	103.7619	2.76
102.1	104.8378	2.74
103.2	105.8526	2.65
104.3	106.87	2.57
105.4	108.0385	2.64

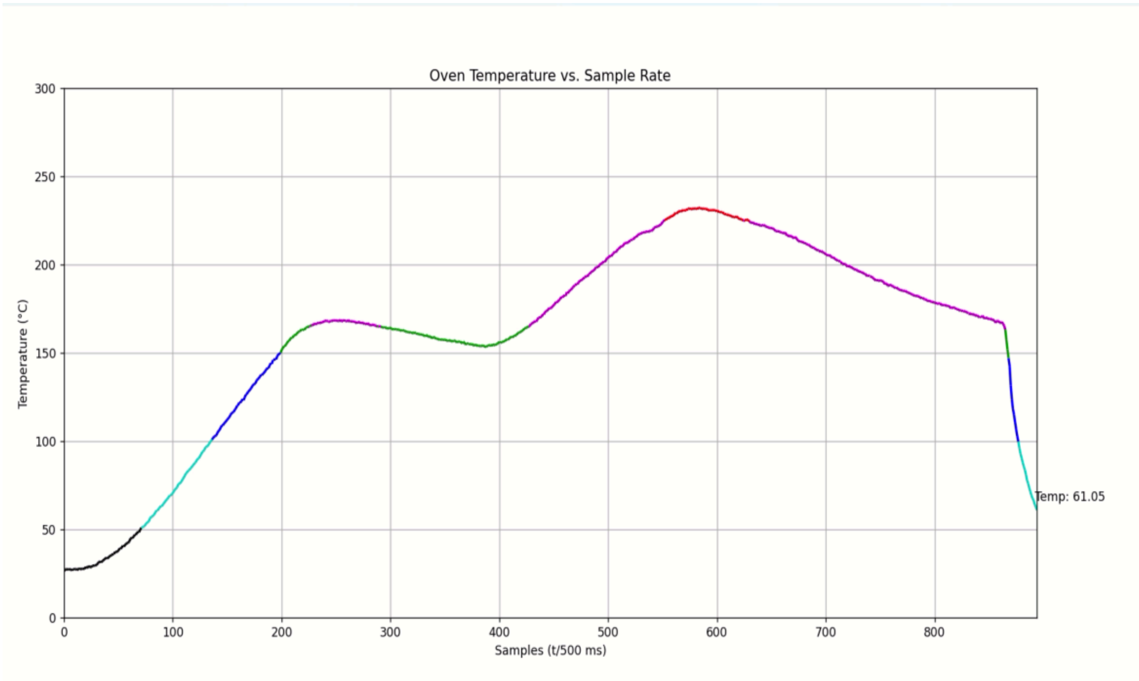
Multimeter	Microcontroller	Error
162.6	165.38	2.78
163.5	166.0409	2.54
163.9	166.4614	2.56
164.2	166.9258	2.73
164.9	167.0612	2.16
165.3	167.4917	2.19
165.7	168.1929	2.49
166	168.4556	2.46
167	169.4673	2.47
168.3	170.6392	2.34
169.1	171.3502	2.25
170.2	172.6648	2.46
171.2	173.296	2.1
172.2	174.7768	2.58
173.2	175.3053	2.11
174.3	176.8297	2.53
175	177.3473	2.35
176.4	178.3321	1.93
177.1	179.3312	2.23
178.1	179.3565	1.26
179.2	180.7826	1.58
180.1	181.3721	1.27
181.1	182.7431	1.64
182	183.577	1.58
183.1	184.437	1.34
184.2	185.886	1.69
185.3	186.6217	1.32
187.2	189.0463	1.85
188.2	189.5629	1.36
189.7	191.3263	1.63
190	191.5931	1.59
191.1	192.3797	1.28
193.5	194.7117	1.21
194.1	195.6346	1.53
195	196.0941	1.09
196.2	198.406	2.21
197	198.7185	1.72
198	199.1063	1.11
199	199.939	0.94
200.2	201.34	1.14
201.1	202.0451	0.95
202.2	203.5009	1.3
203.5	204.727	1.23
203.7	205.03	1.33

Multimeter	Microcontroller	Error
105.4	108.0385	2.64
106.4	108.99	2.59
107	109.62	2.62
108.2	110.79	2.59
109.2	112.01	2.81
110.4	112.9	2.5
111	113.74	2.74
112.2	114.83	2.63
113.3	115.99	2.69
114.4	117.14	2.74
115.6	118.13	2.53
116.1	118.7478	2.65
117.3	119.98	2.68
118.2	120.95	2.75
119.4	121.94	2.54
120.5	123.22	2.72
121	123.63	2.63
122.1	124.61	2.51
123.1	125.61	2.51
123.7	126.43	2.73
124.3	126.86	2.56
125.5	128.1841	2.68
126	128.65	2.65
126.5	129.2051	2.71
127	129.8292	2.83
128.1	130.89	2.79
128.7	131.22	2.52
129.1	131.79	2.69
130.1	132.6904	2.59
130.6	133.11	2.51
131.2	133.83	2.63
131.8	134.57	2.77
132.9	135.3029	2.4
133.3	135.91	2.61
133.8	136.67	2.87
134.9	137.65	2.75
135.4	137.93	2.53
135.9	138.51	2.61
136.5	139.16	2.66
137.6	139.8482	2.25
138.1	140.73	2.63
138.6	141.15	2.55
139.5	142.21	2.71
140.1	142.71	2.61
140.6	143.34	2.74

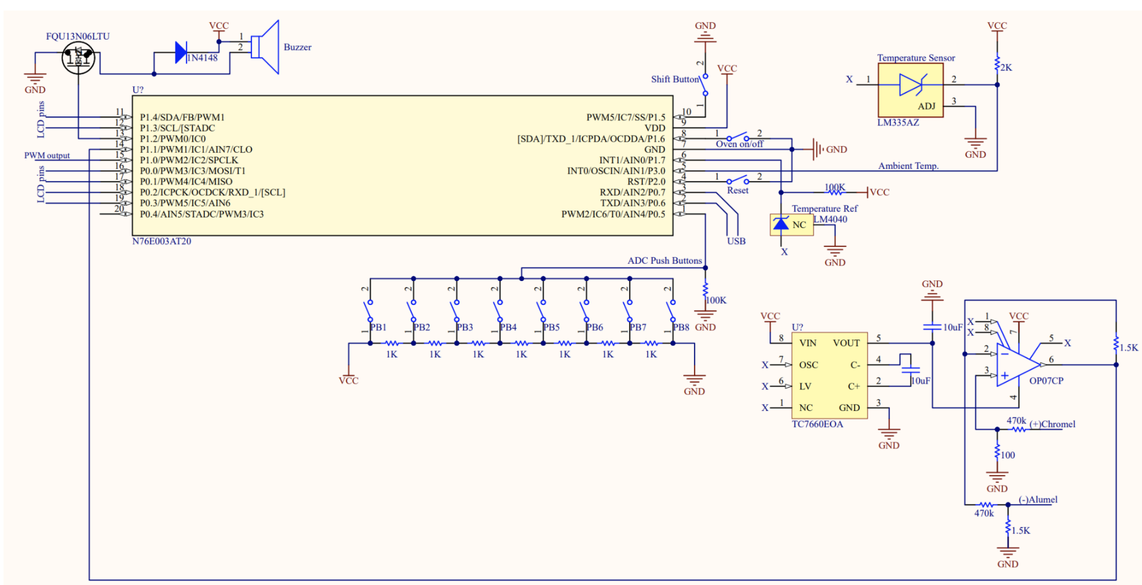
Multimeter	Microcontroller	Error
140.6	143.34	2.74
141.2	143.82	2.62
142	144.55	2.55
142.5	145.22	2.72
143	145.63	2.63
144	146.71	2.71
144.5	147.26	2.76
145	147.82	2.82
145.4	147.94	2.54
146.3	148.7131	2.41
146.8	149.3287	2.53
147.4	150.06	2.66
148.4	150.8685	2.47
148.8	151.49	2.69
149.3	152.11	2.81
149.8	152.42	2.62
150.9	152.9231	2.02
151.5	154.05	2.55
152	154.3765	2.38
152.8	155.0714	2.27
153.2	155.7873	2.59
153.7	156.2941	2.59
154.1	156.61	2.51
154.9	157.0453	2.15
155.1	157.8295	2.73
155.4	157.7004	2.3
156.1	158.1473	2.05
156.5	158.4934	1.99
156.8	159.49	2.69
157.1	159.648	2.55
157.6	159.3556	1.76
157.8	160.3558	2.56
158.1	160.6573	2.56
158.7	160.8078	2.11
159	161.4109	2.41
159.3	161.6346	2.33
159.6	162.3314	2.73
160.2	162.4985	2.3
160.5	163.23	2.73
160.9	162.958	2.06
161.6	164.0497	2.45
161.9	164.66	2.76
162.3	164.85	2.55
162.6	165.38	2.78
163.5	166.0409	2.54

Multimeter	Microcontroller	Error
203.7	205.03	1.33
204	204.6817	0.68
205	207.4134	2.41
206	206.5348	0.53
206.2	207.7121	1.51
207	208.2373	1.24
208	209.0029	1
209.4	210.4263	1.03
210	211.2502	1.25
211	212.066	1.07
212.1	213.1468	1.05
213.2	214.6421	1.44
214.1	215.237	1.14
215.1	216.4197	1.32
216.1	216.9975	0.9
217.1	218.0595	0.96
218.2	219.0302	0.83
219.2	219.8568	0.66
220.1	221.186	1.09
221.4	222.2824	0.88
222.3	223.4795	1.18
223.2	224.2578	1.06
224	225.0636	1.06
225.1	225.9346	0.83
226	226.5804	0.58
227.1	227.737	0.64
228	229.6534	1.65
229.3	229.898	0.6
230	230.1065	0.11
231.2	230.706	0.49
232	232.8892	0.89
234.1	234.519	0.42
235	235.8251	0.83
236.1	236.9997	0.9
237	238.117	1.12
238.2	238.6829	0.48
239.3	240.0997	0.8
240	241.9934	1.99

Appendix F: Python Plot of Reflow Profile



Appendix G: Circuit Diagram



Appendix H: Program Source Code

util.inc

```
; util.inc
;
; Utility subroutines and macros.

; Read/write to flash storage using IAP.

IAP_WRITE_REG MAC REG
        MOV IAPCN, #0b0010_0001
        MOV IAPFD, %0

        ; Begin IAP.
        MOV TA, #0xAA
        MOV TA, #0x55
        ORL IAPTRG, #0b0000_0001

        INC IAPAL
ENDMAC

IAP_READ_ADDR MAC REG
        MOV A, #0x00
        MOVC A, @A+DPTR
        MOV %0, A

        INC DPTR
ENDMAC

; Write R0-R7 from banks 1-2 into APROM starting at address 0x400.
IAP_WRITE:
        PUSH PSW

        ; Disable interrupts to avoid delays when writing to
        ; timed access registers.
        CLR EA

        ; Enable IAP function.
        MOV TA, #0xAA
        MOV TA, #0x55
        ORL CHPCON, #0b0000_0001

        ; Enable APROM erasing and programming by IAP.
        MOV TA, #0xAA
        MOV TA, #0x55
        ORL IAPUEN, #0b0000_0001

        ; Erase page 0x400~0x47F.
        MOV IAPCN, #0b0010_0010
        MOV IAPAH, #0x04
        MOV IAPAL, #0x00
        MOV IAPFD, #0xFF

        ; Begin IAP.
        MOV TA, #0xAA
        MOV TA, #0x55
        ORL IAPTRG, #0b0000_0001

```

```
        ; Note: ANL/ORL PSW will break the program. We need to MOV
        ; the register banks directly here.

        ; Switch to register bank 1.
        MOV PSW, #0b0000_1000

        IAP_WRITE_REG(R0)
        IAP_WRITE_REG(R1)
        IAP_WRITE_REG(R2)
        IAP_WRITE_REG(R3)
        IAP_WRITE_REG(R4)
        IAP_WRITE_REG(R5)
        IAP_WRITE_REG(R6)
        IAP_WRITE_REG(R7)

        ; Switch to register bank 2.
        MOV PSW, #0b0001_0000

        IAP_WRITE_REG(R0)
        IAP_WRITE_REG(R1)
        IAP_WRITE_REG(R2)
        IAP_WRITE_REG(R3)
        IAP_WRITE_REG(R4)
        IAP_WRITE_REG(R5)
        IAP_WRITE_REG(R6)
        IAP_WRITE_REG(R7)

        ; Write signature to 0x47F
        MOV IAPCN, #0b0010_0001
        MOV IAPAH, #0x04
        MOV IAPAL, #0x7F
        MOV IAPFD, #SIGNATURE

        ; Begin IAP.
        MOV TA, #0xAA
        MOV TA, #0x55
        ORL IAPTRG, #0b0000_0001

        ; Disable APROM erasing and programming by IAP.
        MOV TA, #0xAA
        MOV TA, #0x55
        ANL IAPUEN, #0b1111_1110

        ; Disable IAP function.
        MOV TA, #0xAA
        MOV TA, #0x55
        ANL CHPCON, #0b1111_1110

        ; Reenable global interrupts.
        SETB EA

        ; Restores register bank to its state before the subroutine call.
        POP PSW
        RET

; Reads data from APROM starting at address 0x400 and stores
; it into R0-R7 of banks 1-2.
IAP_READ:
        PUSH ACC
        PUSH PSW
        MOV DPTR, #0x400

```

```
        ; Switch to register bank 1.
        ANL PSW, #0b1110_0111
        ORL PSW, #0b0000_1000

        IAP_READ_ADDR(R0)
        IAP_READ_ADDR(R1)
        IAP_READ_ADDR(R2)
        IAP_READ_ADDR(R3)
        IAP_READ_ADDR(R4)
        IAP_READ_ADDR(R5)
        IAP_READ_ADDR(R6)
        IAP_READ_ADDR(R7)

        ; Switch to register bank 2.
        ANL PSW, #0b1110_0111
        ORL PSW, #0b0001_0000

        IAP_READ_ADDR(R0)
        IAP_READ_ADDR(R1)
        IAP_READ_ADDR(R2)
        IAP_READ_ADDR(R3)
        IAP_READ_ADDR(R4)
        IAP_READ_ADDR(R5)
        IAP_READ_ADDR(R6)
        IAP_READ_ADDR(R7)

        ; Restores register bank to its state before the subroutine call.
        POP PSW
        POP ACC
        RET

; Display a BCD number in the LCD.

DISPLAY_BCD MAC
        PUSH AR0
        MOV R0, %0
        LCALL ?DISPLAY_BCD
        POP AR0
ENDMAC

?DISPLAY_BCD:
        PUSH ACC
        ; Write most significant digit.
        MOV A, R0
        SWAP A
        ANL A, #0x0F
        ORL A, #0x30
        LCALL ?WRITEDATA
        ; Write least significant digit.
        MOV A, R0
        ANL A, #0x0F
        ORL A, #0x30
        LCALL ?WRITEDATA
        POP ACC
        RET

DISPLAY_LOWER_BCD MAC
        PUSH AR0
        MOV R0, %0
        LCALL ?DISPLAY_LOWER_BCD
        POP AR0
ENDMAC
```

```
?DISPLAY_LOWER_BCD:
        PUSH ACC
        ; Write least significant digit.
        MOV A, R0
        ANL A, #0x0F
        ORL A, #0x30
        LCALL ?WRITEDATA
        POP ACC
        RET

; Display a char in the LCD.

DISPLAY_CHAR MAC
        PUSH ACC
        MOV A, %0
        LCALL ?WRITEDATA
        POP ACC
ENDMAC

; Delay subroutines and macros.

DELAY MAC MS
        PUSH AR2
        MOV R2, %0
        LCALL WAIT_MS
        POP AR2
ENDMAC

WAIT_1_MS:
        ; Stop timer 0.
        CLR TR0
        ; Clear overflow flag.
        CLR TF0
        MOV TH0, #HIGH(TIMER0_RELOAD)
        MOV TL0, #LOW(TIMER0_RELOAD)
        SETB TR0
        ; Wait for overflow.
        JNB TF0, $
        RET

; Wait the number of milliseconds in R2.
WAIT_MS:
        LCALL WAIT_1_MS
        DJNZ R2, WAIT_MS
        RET
```

main.asm

```
; main.asm
;
; Assembly code for reflow oven controller.
; Written for N76E003 (8051 derivative).
;
; Authored by Eric Feng, Warrick Lo, Santo Neyyan,
; Chathil Rajamanthree, Rishi Upath, Colin Yeung.

; ADC channel mappings.
;
; AIN0         P1.7   Reference voltage
; AIN1         P3.0   Ambient temperature
```

```
; AIN4         P0.5   Push button
; AIN7         P1.1   Op-amp with thermocouple wire

; Mappings for push buttons.
;
; P1.5         SHIFT modifier
; P1.6         Start/stop oven
;
; PB.0         Load preset 4
; PB.1         Load preset 3
; PB.2         Load preset 2
; PB.3         Load preset 1
; PB.4         Reflow time
; PB.5         Reflow temperature
; PB.6         Soak time
; PB.7         Soak temperature

$NOLIST
$MODN76E003
$LIST

ORG 0x0000
       LJMP START

; Timer 2 ISR vector.
ORG 0x002B
       LJMP TIMER2_ISR

; Timer 3 ISR vector.
ORG 0x0083
       LJMP TIMER3_ISR

; 0b0101_1010.
SIGNATURE EQU 0x5A

; Microcontroller system frequency in Hz.
CLK EQU 16600000
; Baud rate of UART in bit/s.
BAUD EQU 115200
; Timer reload values.
TIMER0_RELOAD EQU (0x10000 - (CLK/1000))
TIMER1_RELOAD EQU (0x100 - (CLK/(16*BAUD)))
TIMER2_RELOAD EQU (0x10000 - (CLK/1600))
TIMER3_RELOAD EQU (0x10000 - (CLK/2000))

; PWM.
PWM_OUT EQU P1.0
; Speaker output.
SPKR_OUT EQU P1.2
; Start/stop oven.
OVEN_BUTTON EQU P1.6
; Shift button.
S_BUTTON EQU P1.5

; LCD I/O pins.
LCD_E EQU P1.4
LCD_RS EQU P1.3
LCD_D4 EQU P0.0
LCD_D5 EQU P0.1
LCD_D6 EQU P0.2
LCD_D7 EQU P0.3

```

```
; State machine states representations.
STATE_EMERGENCY EQU 0
STATE_IDLE EQU 1
STATE_PREHEAT EQU 2
STATE_SOAK EQU 3
STATE_RAMP EQU 4
STATE_REFLOW EQU 5
STATE_COOLING EQU 6

$NOLIST
$include(util.inc)
$include(LCD.inc)
$include(math32.inc)
$LIST

DSEG at 0x24

; Bitfield for push button values.
PB: DS 1

DSEG at 0x30

counter: DS 1

; State machine related variables. Refer to defined macros.
time: DS 2
counter_ms: DS 2
FSM1_state: DS 1

; Variables for 32-bit integer arithmetic.
x: DS 4
y: DS 4
z: DS 4

; User-controlled variables.
soak_temp: DS 1
soak_time: DS 1
reflow_temp: DS 1
reflow_time: DS 1

; Reference voltage.
VAL_LM4040: DS 2
; Ambient temperature.
VAL_LM335: DS 2

; Oven controller variables.
pwm: DS 1
temp_oven: DS 2

; BCD numbers for LCD.
bcd: DS 5

BSEG

mf: DBIT 1
spkr_disable: DBIT 1

CSEG

; LCD display strings.

temp:
```

```
        DB '   C/  C    :  s', 0
str_soak_params:
        DB 'SOAK       C   s', 0
str_reflow_params:
        DB 'REFLOW     C   s', 0
str_target:
        DB 'Target:', 0
str_preheat:
        DB 'PRE ', 0
str_soak:
        DB 'SOAK', 0
str_ramp:
        DB 'RAMP', 0
str_reflow:
        DB 'RFLW', 0
str_cooling:
        DB 'COOLING DOWN', 0
str_emergency_1:
        DB 'EMERGENCY ABORT', 0
str_emergency_2:
        DB 'CHECK THERMOWIRE', 0
str_complete:
        DB 'Reflow complete', 0
str_resume:
        DB 'Resuming...', 0
str_abort:
        DB 'Aborting...', 0

; Interrupt service routines.

TIMER2_ISR:
        PUSH ACC
        PUSH PSW
        ; Reset timer 2 overflow flag.
        CLR TF2

        ; PWM.
        CLR C
        INC counter
        MOV A, pwm
        SUBB A, counter
        MOV PWM_OUT, C

        MOV A, counter
        CJNE A, #100, TIMER2_ISR_L1

        ; Executes every second.
        MOV counter, #0
        ; Increment 1 s counter.
        MOV A, time+0
        ADD A, #1
        DA A
        MOV time+0, A
        CJNE A, #0x60, TIMER2_ISR_L1
        ; Reset 1 s counter.
        MOV time+0, #0x00
        ; Increment 1 min counter.
        MOV A, time+1
        ADD A, #1
        DA A
        MOV time+1, A
TIMER2_ISR_L1:
```

```
        POP PSW
        POP ACC
        RETI

TIMER3_ISR:
        JB spkr_disable, TIMER3_ISR_L1
        CPL SPKR_OUT
TIMER3_ISR_L1:
        RETI

; Program entry point.

START:
        MOV SP, #0x7FH

        ; Configure all the pins for biderectional I/O.
        MOV P3M1, #0x00
        MOV P3M2, #0x00
        MOV P1M1, #0x00
        MOV P1M2, #0x00
        MOV P0M1, #0x00
        MOV P0M2, #0x00

        ; Enable global interrupts.
        SETB EA

        ; CLK is the input for timer 1.
        ORL CKCON, #0b0001_0000
        ; Bit SMOD=1, double baud rate.
        ORL PCON, #0b1000_0000
        MOV SCON, #0b0101_0010
        ANL T3CON, #0b1101_1111
        ; Clear the configuration bits for timer 1.
        ANL TMOD, #0b0000_1111
        ; Timer 1 Mode 2.
        ORL TMOD, #0b0010_0000
        MOV TH1, #TIMER1_RELOAD
        SETB TR1

        ; Using timer 0 for delay functions.
        CLR TR0
        ORL CKCON, #0b0000_1000
        ANL TMOD, #0b1111_0000
        ORL TMOD, #0b0000_0001

        ; Timer 2 initialisation.
        MOV T2CON, #0b0000_0000
        MOV T2MOD, #0b1010_0000
        ORL EIE, #0b1000_0000
        MOV TH2, #HIGH(TIMER2_RELOAD)
        MOV TL2, #LOW(TIMER2_RELOAD)
        MOV RCMP2H, #HIGH(TIMER2_RELOAD)
        MOV RCMP2L, #LOW(TIMER2_RELOAD)
        MOV counter, #0x00
        SETB TR2

        ; Timer 3 initialisation.
        MOV RH3, #HIGH(TIMER3_RELOAD)
        MOV RL3, #LOW(TIMER3_RELOAD)
        ORL EIE1, #0b0000_0010
        MOV T3CON, #0b0000_1000

```

```
        ; Initialize and start the ADC.

        ; Configure AIN4 (P0.5) as input.
        ORL P0M1, #0b0010_0000
        ANL P0M2, #0b1101_1111
        ; Configure AIN0 (P1.7) and AIN7 (P1.1) as input.
        ORL P1M1, #0b1000_0010
        ANL P1M2, #0b0111_1101
        ; Configure AIN1 (P3.0) as input.
        ORL P3M1, #0b0000_0001
        ANL P3M2, #0b1111_1110
        ; Set AIN0, AIN1, AIN4, and AIN7 as analog inputs.
        ORL AINDIDS, #0b1001_0011
        ; Enable ADC.
        ORL ADCCON1, #0b0000_0001

        LCALL LCD_INIT

        ; Check flash memory for the program's signature so we
        ; can initialise the APROM on first boot.
        PUSH ACC
        MOV DPTR, #0x47F
        MOV A, #0x00
        MOVC A, @A+DPTR
        CJNE A, #SIGNATURE, $+5
        SJMP $+5
        LCALL APROM_INIT
        POP ACC

        ; Custom arrow character.
        WRITECOMMAND(#0x40)
        WRITEDATA(#0b00000)
        WRITEDATA(#0b01000)
        WRITEDATA(#0b01100)
        WRITEDATA(#0b00110)
        WRITEDATA(#0b00110)
        WRITEDATA(#0b01100)
        WRITEDATA(#0b01000)
        WRITEDATA(#0b00000)

        ; Initialise state machine.
        MOV FSM1_state, #STATE_IDLE
        MOV time+0, #0x00
        MOV time+1, #0x00
        SETB spkr_disable

        ; Initialise temperatures/times.
        MOV soak_temp, #0x50
        MOV soak_time, #0x90
        MOV reflow_temp, #0x25
        MOV reflow_time, #0x60

        SET_CURSOR(1, 1)
        SEND_CONSTANT_STRING(#str_soak_params)
        SET_CURSOR(2, 1)
        SEND_CONSTANT_STRING(#str_reflow_params)

MAIN:
        MOV A, FSM1_state
        LJMP IDLE

; Begin state machine logic and handling.
```

```
EMERGENCY:
        SET_CURSOR(1, 1)
        SEND_CONSTANT_STRING(#str_emergency_1)
        SET_CURSOR(2, 1)
        SEND_CONSTANT_STRING(#str_emergency_2)
        ; Check if oven on/off button is pressed.
        JB OVEN_BUTTON, EMERGENCY_L1
        DELAY(#100)
        JB OVEN_BUTTON, EMERGENCY_L1
        JNB OVEN_BUTTON, $

        WRITECOMMAND(#0x01)
        DELAY(#5)
        SET_CURSOR(1, 1)
        SEND_CONSTANT_STRING(#str_resume)
        LJMP RESET_TO_IDLE
EMERGENCY_L1:
        LJMP MAIN

OVEN_ON:
        ; This delay helps mitigate an undesired flash at
        ; the beginning of the state transition.
        DELAY(#5)

        CJNE A, #STATE_EMERGENCY, $+6
        LJMP EMERGENCY

        SET_CURSOR(1, 1)
        SEND_CONSTANT_STRING(#temp)
        LCALL READ_TEMP
        SET_CURSOR(1, 12)
        DISPLAY_LOWER_BCD(time+1)
        SET_CURSOR(1, 14)
        DISPLAY_BCD(time+0)
        DELAY(#250)
        DELAY(#250)

        ; Check if oven on/off button is pressed.
        JB OVEN_BUTTON, OVEN_ON_L1
        DELAY(#75)
        JB OVEN_BUTTON, OVEN_ON_L1
        JNB OVEN_BUTTON, $

        ; Abort reflow process.
        WRITECOMMAND(#0x01)
        DELAY(#5)
        SEND_CONSTANT_STRING(#str_abort)
        LJMP RESET_TO_IDLE

OVEN_ON_L1:
        LJMP PREHEAT

?OVEN_ON:
        LJMP OVEN_ON

IDLE:
        CJNE A, #STATE_IDLE, ?OVEN_ON
        MOV pwm, #0

        ; Convert ADC signal to push button bitfield.
        LCALL ADC_TO_PB
```

```

        ; Go to handler subroutines if button is pressed.

        ; Shift button is handled inside the subroutine.
        JB PB.4, $+6
        LCALL CHANGE_REFLOW_TIME
        JB PB.5, $+6
        LCALL CHANGE_REFLOW_TEMP
        JB PB.6, $+6
        LCALL CHANGE_SOAK_TIME
        JB PB.7, $+6
        LCALL CHANGE_SOAK_TEMP

        ; Check if SHIFT+PB.{0..3} is pressed.
        JB S_BUTTON, IDLE_L1

        JB PB.0, $+6
        LCALL SAVE_PRESET_4
        JB PB.1, $+6
        LCALL SAVE_PRESET_3
        JB PB.2, $+6
        LCALL SAVE_PRESET_2
        JB PB.3, $+6
        LCALL SAVE_PRESET_1

IDLE_L1:
        JB PB.0, $+6
        LCALL LOAD_PRESET_4
        JB PB.1, $+6
        LCALL LOAD_PRESET_3
        JB PB.2, $+6
        LCALL LOAD_PRESET_2
        JB PB.3, $+6
        LCALL LOAD_PRESET_1

        LJMP DISPLAY_VARIABLES

DISPLAY_VARIABLES:
        ; Display variables.
        SET_CURSOR(1, 9)
        DISPLAY_CHAR(#'1')
        DISPLAY_BCD(soak_temp)
        SET_CURSOR(1, 14)
        DISPLAY_BCD(soak_time)
        SET_CURSOR(2, 9)
        DISPLAY_CHAR(#'2')
        DISPLAY_BCD(reflow_temp)
        SET_CURSOR(2, 14)
        DISPLAY_BCD(reflow_time)

        ; Check if oven on/off button is pressed.
        JB OVEN_BUTTON, IDLE_L2
        DELAY(#100)
        JB OVEN_BUTTON, IDLE_L2
        JNB OVEN_BUTTON, $

PREHEAT_TRANSITION:
        WRITECOMMAND(#0x01)
        CLR spkr_disable
        DELAY(#250)
        SETB spkr_disable
        MOV FSM1_state, #STATE_PREHEAT
```

```
        MOV time+0, #0x00
        MOV time+1, #0x00
        MOV pwm, #100

IDLE_L2:
        DELAY(#100)
        LJMP MAIN

?SOAK:
        LJMP SOAK

PREHEAT:
        CJNE A, #STATE_PREHEAT, ?SOAK

        SET_CURSOR(2, 1)
        SEND_CONSTANT_STRING(#str_preheat)
        SET_CURSOR(2, 6)
        SEND_CONSTANT_STRING(#str_target)
        SET_CURSOR(2, 13)
        DISPLAY_CHAR(#'1')
        DISPLAY_BCD(soak_temp)
        DISPLAY_CHAR(#'C')

        ; Check if oven temperature reaches 50 deg C within 60 s.
        MOV A, time+1
        JNZ $+4
        SJMP PREHEAT_L1
        MOV A, temp_oven+1
        CJNE A, #0x00, PREHEAT_L1
        CLR C
        MOV A, #0x50
        SUBB A, temp_oven+0
        JC PREHEAT_L1
ABORTING:
        WRITECOMMAND(#0x01)
        MOV FSM1_state, #STATE_EMERGENCY
        MOV pwm, #0
        SET_CURSOR(1, 1)
        SEND_CONSTANT_STRING(#str_abort)
        CLR spkr_disable
        DELAY(#250)
        DELAY(#250)
        DELAY(#250)
        DELAY(#250)
        SETB spkr_disable
        LJMP MAIN

PREHEAT_L1:
        ; Check if oven temperature is more than threshold.
        CLR C
        MOV A, temp_oven+1
        SUBB A, #0x01
        JC PREHEAT_L2
        MOV A, temp_oven+0
        SUBB A, soak_temp
        JC PREHEAT_L2

SOAK_TRANSITION:
        WRITECOMMAND(#0x01)
        CLR spkr_disable
        DELAY(#250)
        SETB spkr_disable
```

```
        MOV FSM1_state, #STATE_SOAK
        MOV time+0, #0x00
        MOV time+1, #0x00
        MOV pwm, #20

PREHEAT_L2:
        LJMP MAIN

?RAMPUP:
        LJMP RAMPUP

SOAK:
        CJNE A, #STATE_SOAK, ?RAMPUP

        SET_CURSOR(2, 1)
        SEND_CONSTANT_STRING(#str_soak)
        SET_CURSOR(2, 6)
        SEND_CONSTANT_STRING(#str_target)
        SET_CURSOR(2, 14)
        DISPLAY_BCD(soak_time)
        DISPLAY_CHAR(#'s')

        CLR C
        MOV A, soak_time
        SUBB A, #0x60
        JC SOAK_L1
        ; Target soak time here is equal or more than 60 s.
        MOV R0, A
        MOV A, time+1
        CJNE A, #1, SOAK_L3
        SJMP SOAK_L2
SOAK_L1:
        MOV R0, soak_time
SOAK_L2:
        CLR C
        MOV A, time+0
        SUBB A, R0
        JC SOAK_L3

RAMP_TRANSITION:
        WRITECOMMAND(#0x01)
        CLR spkr_disable
        DELAY(#250)
        SETB spkr_disable
        MOV FSM1_state, #STATE_RAMP
        MOV time+0, #0x00
        MOV time+1, #0x00
        MOV pwm, #100

SOAK_L3:
        LJMP MAIN

?REFLOW:
        LJMP REFLOW

RAMPUP:
        CJNE A, #STATE_RAMP, ?REFLOW

        SET_CURSOR(2, 1)
        SEND_CONSTANT_STRING(#str_ramp)
        SET_CURSOR(2, 6)
        SEND_CONSTANT_STRING(#str_target)
```

```
        SET_CURSOR(2, 13)
        DISPLAY_CHAR(#'2')
        DISPLAY_BCD(reflow_temp)
        DISPLAY_CHAR(#'C')

        CLR C
        MOV A, temp_oven+1
        SUBB A, #0x02
        JC RAMPUP_L1
        MOV A, temp_oven+0
        SUBB A, reflow_temp
        JC RAMPUP_L1

REFLOW_TRANSITION:
        WRITECOMMAND(#0x01)
        CLR spkr_disable
        DELAY(#250)
        SETB spkr_disable
        MOV FSM1_state, #STATE_REFLOW
        MOV time+0, #0x00
        MOV time+1, #0x00
        MOV pwm, #30

RAMPUP_L1:
        LJMP MAIN

?COOLING:
        LJMP COOLING

REFLOW:
        CJNE A, #STATE_REFLOW, ?COOLING

        SET_CURSOR(2, 1)
        SEND_CONSTANT_STRING(#str_reflow)
        SET_CURSOR(2, 6)
        SEND_CONSTANT_STRING(#str_target)
        SET_CURSOR(2, 14)
        DISPLAY_BCD(reflow_time)
        DISPLAY_CHAR(#'s')

        CLR C
        MOV A, reflow_time
        SUBB A, #0x60
        JC REFLOW_L1
        ; Target reflow time here is equal to or more than 60 s.
        MOV R0, A
        MOV A, time+1
        CJNE A, #1, REFLOW_L3
        SJMP REFLOW_L2
REFLOW_L1:
        MOV R0, reflow_time
REFLOW_L2:
        CLR C
        MOV A, time+0
        SUBB A, R0
        JC REFLOW_L3

COOLING_TRANSITION:
        WRITECOMMAND(#0x01)
        CLR spkr_disable
        DELAY(#250)
        SETB spkr_disable
```

```
        MOV FSM1_state, #STATE_COOLING
        MOV time+0, #0x00
        MOV time+1, #0x00
        MOV pwm, #0

REFLOW_L3:
        LJMP MAIN

COOLING_L1:
        LJMP MAIN

COOLING:
        ; This condition should NEVER be met.
        CJNE A, #STATE_COOLING, $

        SET_CURSOR(2, 1)
        SEND_CONSTANT_STRING(#str_cooling)

        CLR C
        MOV A, #0x00
        SUBB A, temp_oven+1
        JC COOLING_L1
        MOV A, #0x60
        SUBB A, temp_oven+0
        JC COOLING_L1

        WRITECOMMAND(#0x01)
        DELAY(#5)
        SET_CURSOR(1, 1)
        SEND_CONSTANT_STRING(#str_complete)

RESET_TO_IDLE:
        MOV FSM1_state, #STATE_IDLE
        MOV time+0, #0x00
        MOV time+1, #0x00
        MOV pwm, #0
        CLR spkr_disable
        DELAY(#200)
        DELAY(#200)
        SETB spkr_disable
        DELAY(#250)
        CLR spkr_disable
        DELAY(#200)
        DELAY(#200)
        SETB spkr_disable
        DELAY(#250)
        CLR spkr_disable
        DELAY(#250)
        DELAY(#250)
        DELAY(#250)
        SETB spkr_disable
        SET_CURSOR(1, 1)
        SEND_CONSTANT_STRING(#str_soak_params)
        SET_CURSOR(2, 1)
        SEND_CONSTANT_STRING(#str_reflow_params)
        LJMP MAIN

BACK_TO_IDLE:
        WRITECOMMAND(#0x01)
        MOV FSM1_state, #STATE_IDLE
        MOV time+0, #0x00
        MOV time+1, #0x00
```

```
        MOV pwm, #0
        DELAY(#5)
        SET_CURSOR(1, 1)
        SEND_CONSTANT_STRING(#str_soak_params)
        SET_CURSOR(2, 1)
        SEND_CONSTANT_STRING(#str_reflow_params)
        CLR spkr_disable
        DELAY(#200)
        DELAY(#200)
        SETB spkr_disable
        DELAY(#250)
        CLR spkr_disable
        DELAY(#200)
        DELAY(#200)
        SETB spkr_disable
        DELAY(#250)
        CLR spkr_disable
        DELAY(#250)
        DELAY(#250)
        DELAY(#250)
        SETB spkr_disable
        LJMP MAIN

; Initialise APROM flash storage with default reflow profiles.

APROM_INIT:
        PUSH PSW

        ; Switch to register bank 1.
        MOV PSW, #0b0000_1000

        MOV R0, #0x50
        MOV R1, #0x90
        MOV R2, #0x25
        MOV R3, #0x60
        MOV R4, #0x80
        MOV R5, #0x60
        MOV R6, #0x30
        MOV R7, #0x45

        ; Switch to register bank 2.
        MOV PSW, #0b0001_0000

        MOV R0, #0x40
        MOV R1, #0x90
        MOV R2, #0x10
        MOV R3, #0x45
        MOV R4, #0x90
        MOV R5, #0x30
        MOV R6, #0x60
        MOV R7, #0x20

        LCALL IAP_WRITE

        POP PSW
        RET

; Subroutine code for reading ADC.

READ_ADC:
        PUSH ACC
        CLR ADCF
```

```
        SETB ADCS
        JNB ADCF, $

        ; Read the ADC result and store in [R1, R0].
        MOV A, ADCRL
        ANL A, #0b0000_1111
        MOV R0, A
        MOV A, ADCRH
        SWAP A
        PUSH ACC
        ANL A, #0b0000_1111
        MOV R1, A
        POP ACC
        ANL A, #0b1111_0000
        ORL A, R0
        MOV R0, A
        POP ACC
        RET

; ADC channel switching subroutines.

SWITCH_TO_AIN0:
        ; Select ADC channel 0.
        ANL ADCCON0, #0b1111_0000
        ORL ADCCON0, #0b0000_0000
        RET

SWITCH_TO_AIN1:
        ; Select ADC channel 1.
        ANL ADCCON0, #0b1111_0000
        ORL ADCCON0, #0b0000_0001
        RET

SWITCH_TO_AIN4:
        ; Select ADC channel 4.
        ANL ADCCON0, #0b1111_0000
        ORL ADCCON0, #0b0000_0100
        RET

SWITCH_TO_AIN7:
        ; Select ADC channel 7.
        ANL ADCCON0, #0b1111_0000
        ORL ADCCON0, #0b0000_0111
        RET

; Subroutine for reading ambient and oven temperatures from the ADC.

READ_TEMP:
        ; Read the 2.08V LM4040 voltage connected to AIN0 on pin 6.
        LCALL SWITCH_TO_AIN0

        LCALL READ_ADC
        ; Save result for later use.
        MOV VAL_LM4040+0, R0
        MOV VAL_LM4040+1, R1

        ; LM335.
        LCALL SWITCH_TO_AIN1

        LCALL READ_ADC
        MOV VAL_LM335+0, R0
        MOV VAL_LM335+1, R1
```

```

            ; Convert to voltage.
            MOV x+0, R0
            MOV x+1, R1
            ; Pad other bits with zero.
            MOV x+2, #0
            MOV x+3, #0
            ; The MEASURED voltage reference: 4.0959V, with 4 decimal places.
            LOAD_Y(40959)
            LCALL MUL32

            ; Retrieve the LM4040 ADC value.
            MOV y+0, VAL_LM4040+0
            MOV y+1, VAL_LM4040+1
            MOV y+2, #0
            MOV y+3, #0
            LCALL DIV32

            ; Convert to temperature for LM335.
            LOAD_Y(27300)
            LCALL SUB32
            LOAD_Y(100)
            LCALL MUL32

            ; Store LM335 temperature result in z.
            MOV z+0, x+0
            MOV z+1, x+1
            MOV z+2, x+2
            MOV z+3, x+3

            LCALL HEX2BCD
            ; Display ambient temperature.
            SET_CURSOR(1, 6)
            DISPLAY_BCD(bcd+2)

            ; Read the amplified thermocouple wire signal connected to AIN7.
            LCALL SWITCH_TO_AIN7
            LCALL READ_ADC

            ; Convert to voltage.
            MOV x+0, R0
            MOV x+1, R1
            MOV x+2, #0
            MOV x+3, #0
            ; The MEASURED voltage reference: 4.0959V, with 4 decimal places.
            LOAD_Y(40959)
            LCALL MUL32

            ; Retrieve the LM4040 ADC value.
            MOV y+0, VAL_LM4040+0
            MOV y+1, VAL_LM4040+1
            MOV y+2, #0
            MOV y+3, #0
            LCALL DIV32

            LOAD_Y(670)
            LCALL MUL32
            LOAD_Y(211)
            LCALL DIV32
            LOAD_Y(1000)
            LCALL MUL32
            LOAD_Y(41)
```

```
        LCALL DIV32

        ; LM335 temperature stored in z.
        MOV y+0, z+0
        MOV y+1, z+1
        MOV y+2, z+2
        MOV y+3, z+3
        LCALL ADD32

        ; Convert to BCD and display.
        LCALL HEX2BCD
        SET_CURSOR(1, 1)
        DISPLAY_LOWER_BCD(bcd+3)
        DISPLAY_BCD(bcd+2)

        ; Send to PUTTY.
        PUSH ACC
        SEND_BCD(bcd+3)
        SEND_BCD(bcd+2)
        MOV A, #'.'
        LCALL PUTCHAR
        SEND_BCD(bcd+1)
        SEND_BCD(bcd+0)
        MOV A, #'\r'
        LCALL PUTCHAR
        MOV A, #'\n'
        LCALL PUTCHAR
        POP ACC

        MOV temp_oven+0, bcd+2
        MOV temp_oven+1, bcd+3

        RET

; Check push buttons.

CHECK_PUSH_BUTTON MAC PB, HEX
        CLR C
        MOV A, ADCRH
        SUBB A, %1
        JC $+7
        CLR %0
        POP ACC
        RET
ENDMAC

ADC_TO_PB:
        LCALL SWITCH_TO_AIN4

        ; Wait for ADC to finish A/D conversion.
        CLR ADCF
        SETB ADCS
        JNB ADCF, $

        ; Initialise buttons.
        SETB OVEN_BUTTON
        SETB S_BUTTON
        MOV PB, #0xFF

        ; The accumulator is popped either in the macro expansion
        ; or at the very end of this subroutine.
        PUSH ACC
```

```
        CHECK_PUSH_BUTTON(PB.7, #0xF0)
        CHECK_PUSH_BUTTON(PB.6, #0xD0)
        CHECK_PUSH_BUTTON(PB.5, #0xB0)
        CHECK_PUSH_BUTTON(PB.4, #0x90)
        CHECK_PUSH_BUTTON(PB.3, #0x70)
        CHECK_PUSH_BUTTON(PB.2, #0x50)
        CHECK_PUSH_BUTTON(PB.1, #0x30)
        CHECK_PUSH_BUTTON(PB.0, #0x10)
        POP ACC
        RET

; Push button handling.

CHANGE_VALUE MAC VALUE, PB, ROW, COL
CHANGE_%0:
        PUSH ACC
        DELAY(#125)
        JB %1, $+18
        JB %1, $

        MOV A, %0
        JNB S_BUTTON, $+7
        ADD A, #1
        SJMP $+4
        ADD A, #0x99
        DA A
        MOV %0, A

        ; Update LCD Display at specified ROW and COL.
        LCALL CLEAR_ARROWS
        SET_CURSOR(%2, %3)
        WRITEDATA(#0x00)

        POP ACC
        RET
ENDMAC

CHANGE_VALUE(REFLOW_TIME, PB.4, 2, 13)
CHANGE_VALUE(REFLOW_TEMP, PB.5, 2, 8)
CHANGE_VALUE(SOAK_TIME, PB.6, 1, 13)
CHANGE_VALUE(SOAK_TEMP, PB.7, 1, 8)

CLEAR_ARROWS:
        SET_CURSOR(1, 8)
        DISPLAY_CHAR(#' ')
        SET_CURSOR(1, 13)
        DISPLAY_CHAR(#' ')
        SET_CURSOR(2, 8)
        DISPLAY_CHAR(#' ')
        SET_CURSOR(2, 13)
        DISPLAY_CHAR(#' ')
        RET

LOAD_PRESET MAC PRESET, PB
LOAD_PRESET_%0:
        PUSH ACC
        DELAY(#125)
        JB %1, LOAD_PRESET_%0_L1
        LCALL FETCH_PRESET_%0
LOAD_PRESET_%0_L1:
        POP ACC
        RET
```

```
ENDMAC

LOAD_PRESET(1, PB.3)
LOAD_PRESET(2, PB.2)
LOAD_PRESET(3, PB.1)
LOAD_PRESET(4, PB.0)

FETCH_PRESET MAC PRESET, ADDR, REGBANK
FETCH_PRESET_%0:
       PUSH ACC
       PUSH PSW

       MOV DPTR, %1

       ANL PSW, #0b1110_0111
       ORL PSW, #(%2 << 3)

       MOV A, #0x00
       MOVC A, @A+DPTR
       MOV soak_temp, A

       MOV A, #0x01
       MOVC A, @A+DPTR
       MOV soak_time, A

       MOV A, #0x02
       MOVC A, @A+DPTR
       MOV reflow_temp, A

       MOV A, #0x03
       MOVC A, @A+DPTR
       MOV reflow_time, A

       POP PSW
       POP ACC
       RET
ENDMAC

FETCH_PRESET(1, #0x400, 0b01)
FETCH_PRESET(2, #0x404, 0b01)
FETCH_PRESET(3, #0x408, 0b10)
FETCH_PRESET(4, #0x40C, 0b10)

SAVE_PRESET MAC PRESET, PB, REGBANK, RA, RB, RC, RD
SAVE_PRESET_%0:
       PUSH ACC
       PUSH PSW
       DELAY(#125)
       JB %1, SAVE_PRESET_%0_L1
       LCALL IAP_READ

       ANL PSW, #0b1110_0111
       ORL PSW, #(%2 << 3)

       MOV %3, soak_temp
       MOV %4, soak_time
       MOV %5, reflow_temp
       MOV %6, reflow_time

       LCALL IAP_WRITE
SAVE_PRESET_%0_L1:
       POP PSW
```

```
        POP ACC
        RET
ENDMAC

SAVE_PRESET(1, PB.3, 0b01, R0, R1, R2, R3)
SAVE_PRESET(2, PB.2, 0b01, R4, R5, R6, R7)
SAVE_PRESET(3, PB.1, 0b10, R0, R1, R2, R3)
SAVE_PRESET(4, PB.0, 0b10, R4, R5, R6, R7)

```

graph.py

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import sys, serial, smtplib, threading
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.base import MIMEBase
from email import encoders
from matplotlib.collections import LineCollection

xsize = 150  # Initial range for the x-axis
log_file = "temperature_log.txt"  # Log file name

# Temperature thresholds
state1_val = 50
state2_val = 100
state3_val = 150
state4_val = 170
state5_val = 210
email_trigger_temp = 217 # Email trigger threshold, change based on what value want to email at

# Email Configuration, adjust this based on your mailtrap host info
smtp_server = "bulk.smtp.mailtrap.io"
port = 587
login = "api"  # Mailtrap login
password = "91f13d766ef5fe08540e0ec581c2c181"  # Mailtrap password
sender_email = "hello@demomailtrap.com"
#adjust who will recieve email here
receiver_email = "user@gmail.com"


email_sent = False  # Flag to ensure only one email is sent

#email sending function
def send_email():
    """Send an email when the temperature exceeds 217°C and attach the log file."""
    global email_sent
    if email_sent:
        return

    email_sent = True  # Set flag before sending to avoid duplicates

    # Create an email message with proper UTF-8 encoding
    msg = MIMEMultipart()
    msg["From"] = sender_email
    msg["To"] = receiver_email
    msg["Subject"] = "🔥 Temperature Alert! Oven Over 217°C! Reflow has occurred🔥"

    # Email body, adjust as necessary
```

```
    body = "Alert. Oven Temperature has reached 217°C. Cooling will begin soon and board is fully cooked. See the attached log file for more info on temperature readings."
    msg.attach(MIMEText(body, "plain", "utf-8"))  # Set encoding to UTF-8

    # Attach the log file
    try:
        with open(log_file, "rb") as attachment:
            part = MIMEBase("application", "octet-stream")
            part.set_payload(attachment.read())

        encoders.encode_base64(part)
        part.add_header("Content-Disposition", f"attachment; filename={log_file}")
        msg.attach(part)

        # Send the email
        with smtplib.SMTP(smtp_server, port) as server:
            server.starttls()
            server.login(login, password)
            server.sendmail(sender_email, receiver_email, msg.as_string())

#check message to see if email was sent or if error
        print("✅ Email with log file sent successfully.")
    except Exception as e:
        print(f"❌ Email failed: {e}")

# Configure serial port
ser = serial.Serial(
    port='COM8',
    baudrate=115200,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_TWO,
    bytesize=serial.EIGHTBITS
)
ser.isOpen()

# Ensure log file is cleared at the start
open(log_file, "w").close()

# Data generator
def data_gen():
    global email_sent
    #actual decoding of serial port and printing
    t = data_gen.t
    while True:
        try:
            strin = ser.readline()
            decoded_string = strin.decode('utf-8').strip()  # Remove newline characters
            val = float(decoded_string)  # Convert to float
            print(f"Received: {val}°C")  # Debug print

            # Save to log file, inputs readings into the text file
            with open(log_file, "a") as f:
                f.write(f"{t}, {val}\n")

            # If temp exceeds 217°C for the first time, send email with log file
            if val > email_trigger_temp and not email_sent:
                threading.Thread(target=send_email, daemon=True).start()

            yield t, val
            t += 1
            #error check
```

```
            except ValueError:
                print("⚠️ Warning: Invalid data received. Skipping.")

# Function to determine segment colors
def get_color(value):
    if value >= state5_val:
        return 'r'  # Red
    elif value >= state4_val:
        return 'm'  # Magenta
    elif value >= state3_val:
        return 'g'  # Green
    elif value >= state2_val:
        return 'b'  # Blue
    elif value >= state1_val:
        return 'c'  # Cyan
    else:
        return 'k'  # Black for lower temps

# Function to update the graph
def run(data):
    t, y = data
    xdata.append(t)
    ydata.append(y)

    # Shift x-axis dynamically while keeping left at 0
    ax.set_xlim(0, max(t, xsize))

    # Create segments with colors
    points = np.array([xdata, ydata]).T.reshape(-1, 1, 2)
    segments = np.concatenate([points[:-1], points[1:]], axis=1)
    colors = [get_color(val) for val in ydata[:-1]]

    # Update line collection instead of redrawing everything
    line_collection.set_segments(segments)
    line_collection.set_color(colors)

    return line_collection,

# Event handler for closing the figure
def on_close_figure(event):
    sys.exit(0)

# Initialize variables
#everything below is for graph
data_gen.t = -1
fig, ax = plt.subplots()
fig.canvas.mpl_connect('close_event', on_close_figure)

ax.set_ylim(0, 300)
ax.set_xlim(0, xsize)  # Fixed left boundary
ax.grid()
ax.set_title("Oven Temperature vs. Time")
ax.set_xlabel("Time (t/500 ms)")
ax.set_ylabel("Temperature (°C)")
xdata, ydata = [], []

# Initialize line collection
line_collection = LineCollection([], linewidth=2)
ax.add_collection(line_collection)

# Animation
```

```
ani = animation.FuncAnimation(fig, run, data_gen, blit=False, interval=100,
repeat=False)
plt.show()
```