Table of Contents

Introduction

The objective of this project is to create a reflow soldering oven controller and utilize it to solder various components (including an EFM8 microcontroller) to a PCB. Figure 1.1 shows the hardware block diagram of our design. The K-type thermocouple sends voltages to the differential amplifier. The amplified voltage is converted from analog to digital on the N76E003 microcontroller and converted to a temperature in Celsius. This temperature value is then used in a Finite State Machine (FSM) that will run a program to heat the oven in certain stages. Through pulse width modulation, a signal is sent to a solid state relay (SSR) connected to a 1500W toaster oven. The FSM also sends a specific frequency to a buzzer, which will beep whenever there is a change of state. The LCD Display is used to display various information such as temperature, state, mode, etc.

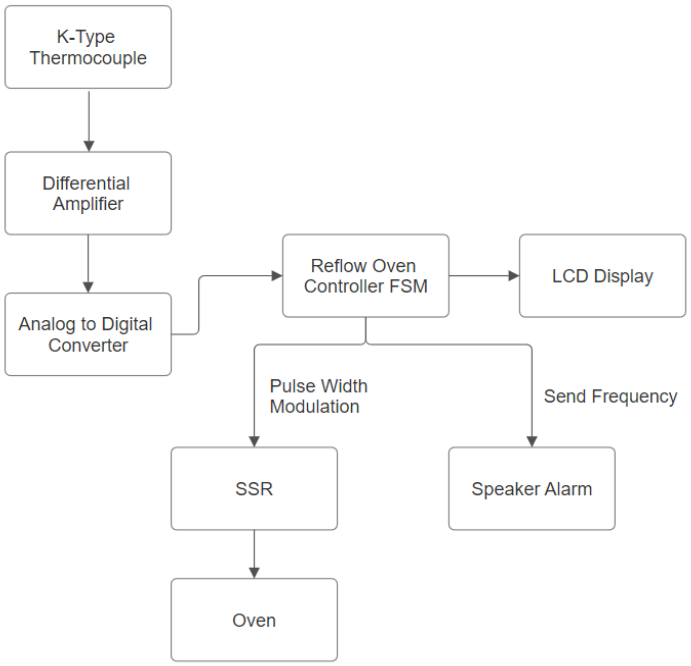


Figure 1.1 Hardware block diagram

Our design is capable of measuring and controlling temperatures between 20°C and 240°C. Figure 1.2 shows the software block diagram of our design. Once initialized, our first state will increase oven temperature until a specified “soak” phase is reached. During this stage there is also an automatic abort sequence that activates if the oven doesn’t reach at least 50°C in the first 60 seconds of operation. When the soak temperature is reached, the oven is toggled at approximately 20% of its power level to maintain temperature for a programmed time in seconds. After the soak time, the oven is set to full power until it reaches a programmed “reflow” temperature. The oven is once again held at this temperature for programmed reflow time so that the solder paste on the PCB may melt. After this reflow time ends, the oven shuts down and enters the cooling phase until it reaches safe-handling temperatures.

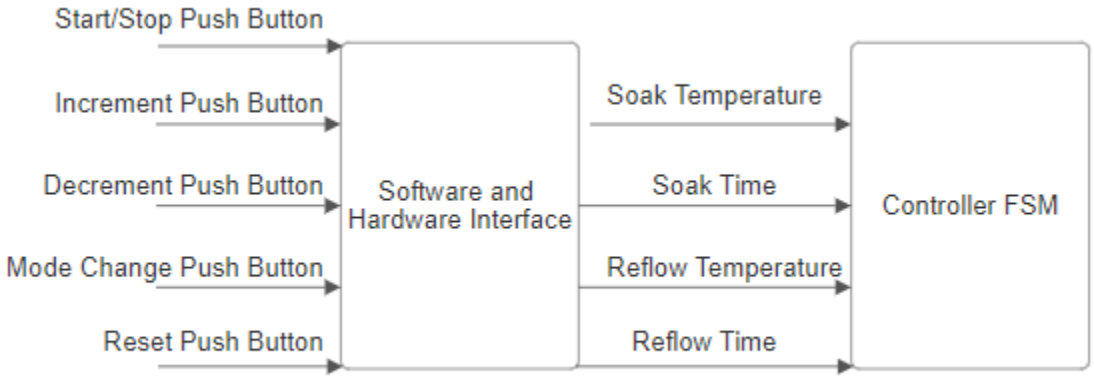


Figure 1.2 Software block diagram

Investigation

Idea Generation - Our group's idea generation process began with a thorough review of lecture slides and project materials. These resources served as the foundation for our understanding of basic operating principles, including concepts like op-amps and thermocouples. We brainstormed various possibilities, considering the feasibility of each idea within the constraints of our project requirements and technical expertise. We considered using a lighter as explained in lecture slides but we settled on a soldering iron as a more reliable source of heat.

Investigation Design - To design our investigations effectively, our group isolated and accounted for controlled and uncontrolled variables during temperature validation. Some of our controlled factors were ambient temperature, heat transfer between soldering iron, and the effect of the multimeter's generated heat. Specifically, we ensured that the cold junction temperature remained consistent throughout all our trials by measuring the in-lab temperature the day of. We also made sure to not leave the multimeter running connected for more than 10 minutes at a time to avoid another source of error. By prioritizing careful data gathering, analysis, and experimentation, we aimed to ensure the reliability and validity of our investigative process.

Data Collection - In our data collection process, we employed appropriate procedures, tools, and techniques to ensure accuracy and reliability. To measure temperature, we utilized a standard Fluke multimeter and the given program to accurately compare with our microcontroller temperature. This method allowed us to obtain precise temperature measurements within our desired range. Additionally, we maintained consistency by

using the same soldering iron for our experiments, ensuring uniformity across trials. By adhering to established practices and employing standardized tools and equipment, we minimized potential sources of error and facilitated the accurate analysis of our data.

Data Synthesis - To synthesize the data gathered from our experiments and reach appropriate conclusions, our group adopted a systematic approach. Initially, we compiled all our data points into a comprehensive table with differences listed as a separate column. To enhance our understanding and interpretation of the data, we created a graphical representation to visualize the variations and trends more effectively. This graphical analysis facilitated the identification of any patterns or anomalies in the data, enabling us to draw informed conclusions regarding the performance and behavior of the system under investigation.

Analysis of Results - In evaluating the validity of our conclusions, our group took into account the inherent degrees of error and limitations present in both theory and real-life. Recognizing that no experiment is entirely devoid of error, we acknowledged that a certain margin of error was reasonable to expect in our results. Additionally, we remained mindful of potential sources of disruption, such as noise and small offsets in components like op amps and LEDs, which could introduce variability into our measurements. We found that our system was fairly accurate at all temperatures but inaccuracies were found with temperature changes. This can be partially explained by the soldering iron heating up much faster than the oven. Additionally, our system

managed to quickly catch up once temperature plateaued and hence any temperature fluctuations are minimized.

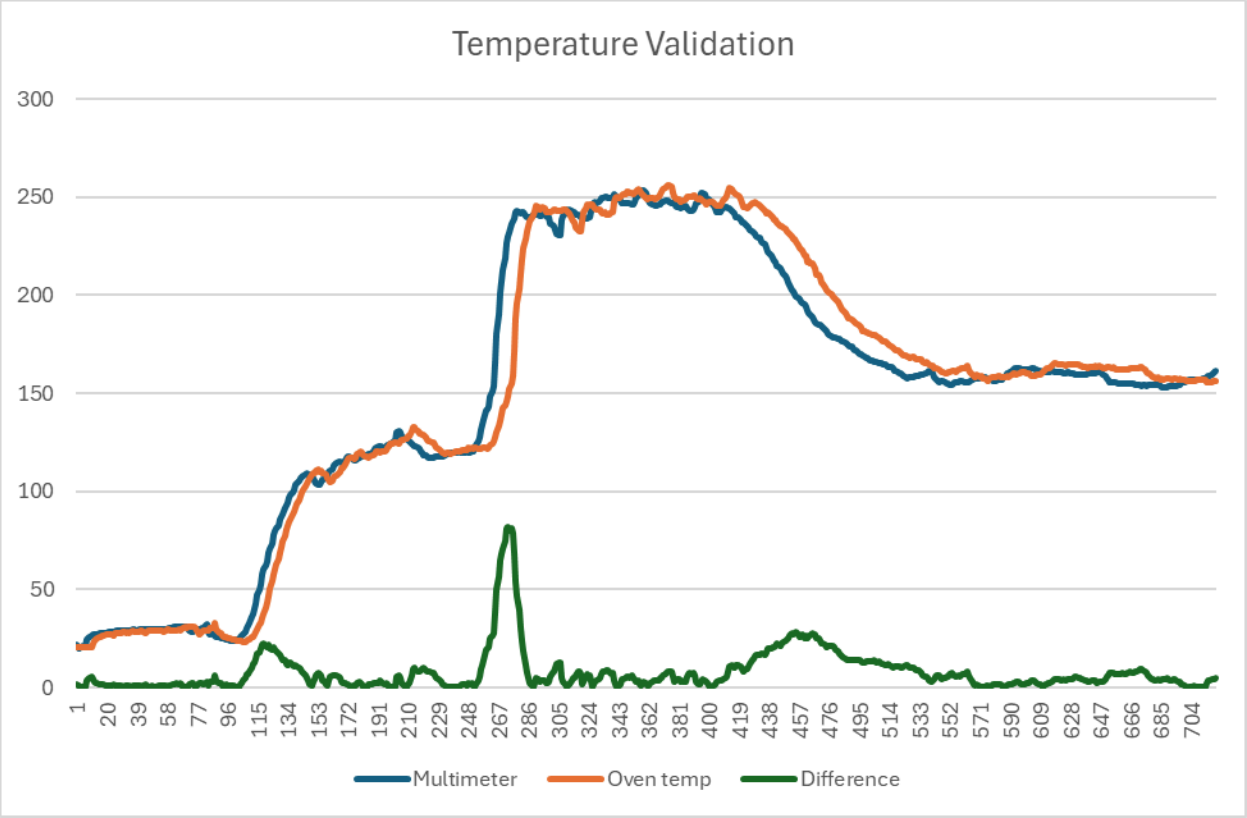


Figure 1.3 Temperature Validation Graph

Design

Use of Process - In addressing the complex, open-ended problem presented by the Reflow Oven Controller project, our group adapted and applied a structured design process that emphasized iterative development, component selection based on accessibility and functionality, and creative problem-solving strategies. We began with a thorough analysis of the project requirements, identifying key challenges such as the need for precise temperature control and the integration of a user-friendly interface.

Throughout the design and development phases, we employed an iterative process, prototyping and testing individual subsystems, such as the temperature measurement and control mechanism, the solid-state relay interface, and the innovative visual feedback system, before integrating them into a cohesive unit. This approach allowed us to systematically address and solve the complex problems encountered, making adjustments based on real-world testing and feedback.

Need and Constraint Identification - Our group systematically identified customer, user, and enterprise needs, alongside applicable constraints for the Reflow Oven Controller project, by engaging in comprehensive research and a detailed analysis of the project's specifications and industry standards. Recognizing the necessity for a device that offers precise temperature control, reliability, and user-friendliness for reflow soldering, we prioritized features such as accurate temperature measurement, an intuitive user interface, and innovative visual feedback mechanisms. Additionally, we acknowledged user needs by ensuring our design complied with safety standards, and utilized readily accessible components. Technical constraints, such as the limitations of the toaster oven, the capabilities of the K-type thermocouple, and the use of assembly language for programming, were considered to ensure our project was feasible. This approach to identifying needs and constraints allowed us to craft a project plan that was well-informed, user-centric, and aligned with the practical realities of electronics manufacturing and education.

Problem Specification - We want a controller that is capable of starting, stopping, resetting, incrementing the time, and increasing/decreasing the temperature of the oven. Using push buttons, we can customize the oven controller to whatever reflow parameters are needed. To solder the EFM8 boards, we ensured the reflow time was less than 45 seconds and the maximum reflow temperature was not more than 240°C. If any unexpected circumstances arose during the soldering process, the emergency stop button would ensure a quick way to shut down the oven. The LCD will display the time and temperature during reflow soldering and the screen on the PC will display the plot of temperature against time.

Solution Generation - To measure the temperature inside the oven, we used a K-type thermocouple connected to an op-amp. The op-amp sends an output voltage to the analog to digital converter (ADC) on the microcontroller and is then converted to temperature using math functions. This live temperature data is then used to change the states of the FSM in combination with a timer that runs when the program is started. The FSM is made of 6 states:

-State_0: Has the start screen where we can use push buttons to edit settings for the reflow process such as temperatures at which state changes or time spent for each state.

-State_1: Ramps up temperature, sets the power of the oven to 100%, goes up to set temperature and has instructions to abort if temperature does not hit at least 50 degrees C in under 60 seconds, then disconnects power to the oven and pauses FSM.

-State_2: Once desired temperature is reached, it sets power to the oven at 20% and

lets the solder paste soak onto the PCB for a fixed time set initially.

-State_3: Is the second ramp stage, sets power to the oven to 100% again and increases temperature to a set value. There is also an abort check here to make sure the temperature does not increase above a certain temperature and does not remain at a high temperature for a long time to avoid burning the PCB.

-State_4: This is the reflow stage, sets power to the oven at 20% and keeps temperature constant.

-State_5: This is the final cooling stage of the process, switches off power to the oven and waits for the oven to cool down and goes back to State 0 when the temperature is below a certain temperature.

During this entire process, the LCD screen displays the state of the FSM, the current temperature inside the oven, the overall timer since the FSM has started and also the time spent individually in each state. To control the power to the oven, an SSR box is connected to the power source which sends power only when a 5 volt source is connected. So, we used another timer in order to use pulse width modulation to send square wave signals of 5V at fixed intervals, which then controlled the power to the oven in percentage by sending short pulses of signal.

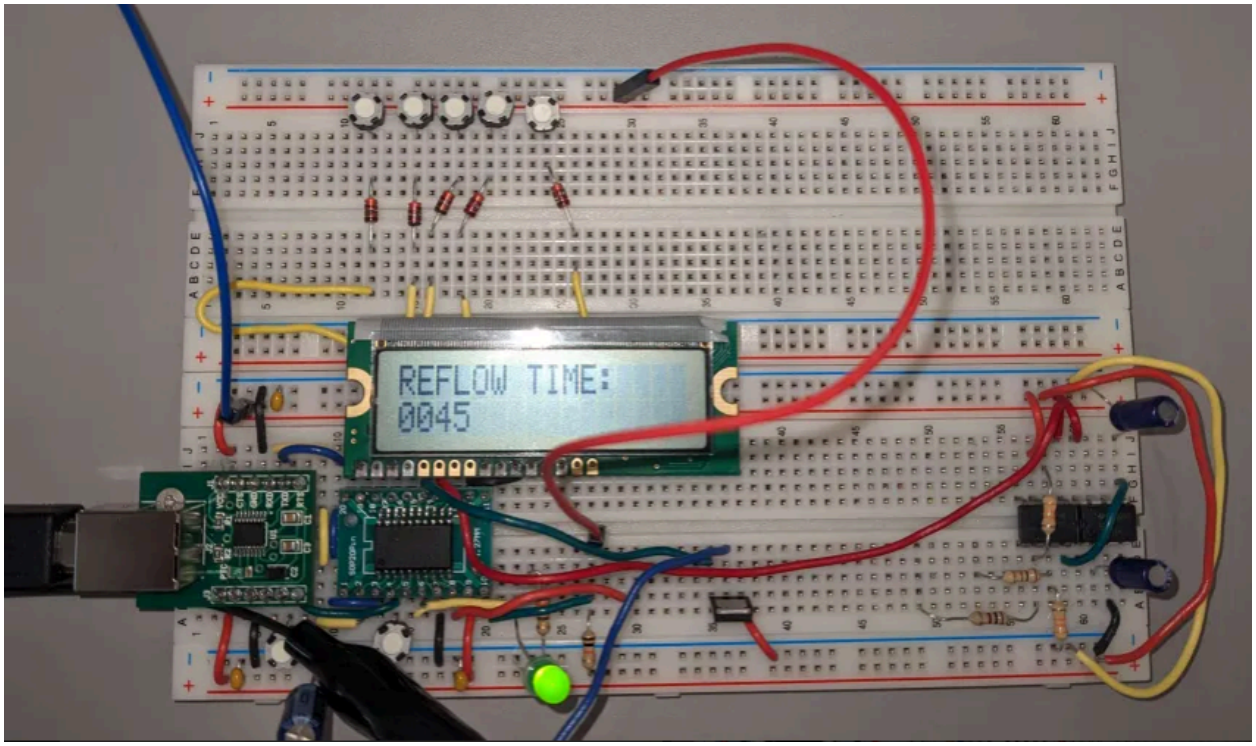


Figure 1.3 Complete oven controller showing reflow time editing in State_0

(From left to right)

Push button 1: Increment

Push button 2: Decrement

Push button 3: Mode change (reflow time and temperature, soak time and temperature, start/edit mode)

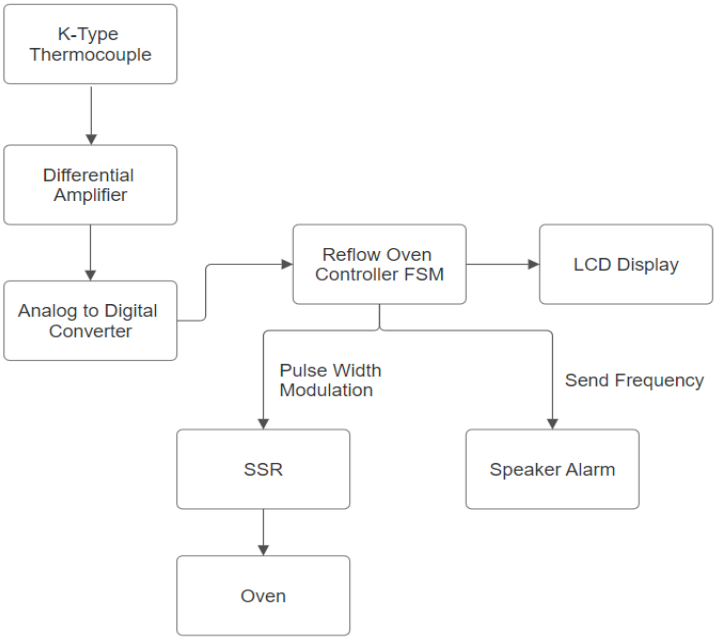
Push button 4: Start/stop

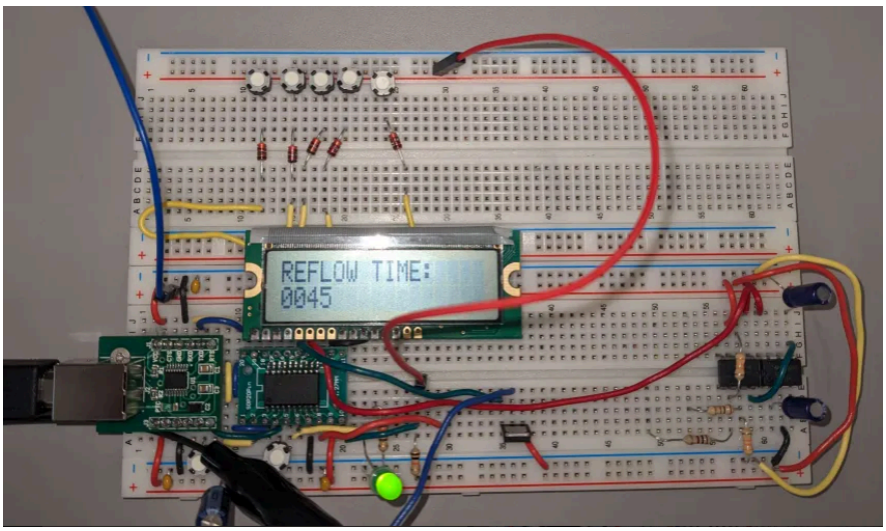
Push button 5: Reset

Solution Evaluation - During the development of our Reflow Oven Controller, we evaluated various design concepts against key project criteria, including temperature control precision, user interface usability, safety, and component availability. Our project not only met all requirements, but also had extra features in the form of extra timer within all states of the FSM, improved user interface, and also an attempt in using a

servo motor with PWM to create an analog display for states. On the Python side, we also had extra features in the form of a temperature dependent color coded graph and also an animation that brought up different images for different ranges of temperatures just like the temperature-time graph.

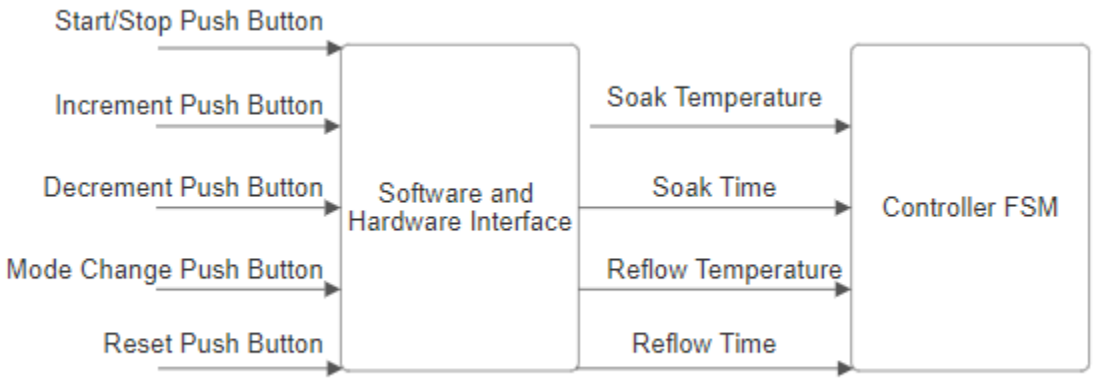
Detailed Design -





Circuit (Hardware) Explanation

The hardware of the reflow oven consists of a K-type thermocouple that measures the oven's temperature, whose signal is then amplified by a differential amplifier to make it readable by an analog-to-digital converter (ADC). The ADC converts the analog temperature signal into serial digital data that the microcontroller can process. The microcontroller, programmed with a finite state machine, uses this data to control each state of the reflow process by sending pulse-width modulation signals to a solid-state relay. The SSR, in turn, switches the oven's heating element on and off to maintain the desired temperature profile. Temperature readings and system status are displayed to the user via an LCD display, and a speaker is connected to the system to provide alarms when the reflow cycle changes state. Our hardware allows for precise temperature management needed for successful soldering in the reflow process.



Software Explanation

The software component of the reflow oven controller serves as an interface between the user's input commands and the physical control of the oven's temperature profile. It

is structured around a Finite State Machine (FSM) which transitions through various states such as ‘Soak Temperature’, ‘Soak Time’, ‘Reflow Temperature’, and ‘Reflow Time’ based on user interactions through push buttons. These buttons include ‘Start/Stop’, ‘Increment’, ‘Decrement’, ‘Mode Change’, and ‘Reset’, each triggering different actions in the software. When a button is pressed, the software interprets the input through a debouncing mechanism to ensure accurate signal processing, and then adjusts the control parameters of FSM state accordingly. For instance, pressing the ‘Increment’ or ‘Decrement’ button will alter the soak or reflow temperatures of times, whereas ‘Mode Change’ might toggle between setting different parameters. The ‘Start/Stop’ button initiates or halts the reflow process, and ‘Reset’ returns the system to an initial state. This structure allows for real-time control of the reflow process, essential for achieving the precise thermal profiles needed for effective soldering on PCBs.

Solution Assessment - Before soldering our EFM8 boards, we conducted basic tests on the controller system to ensure correct operation.

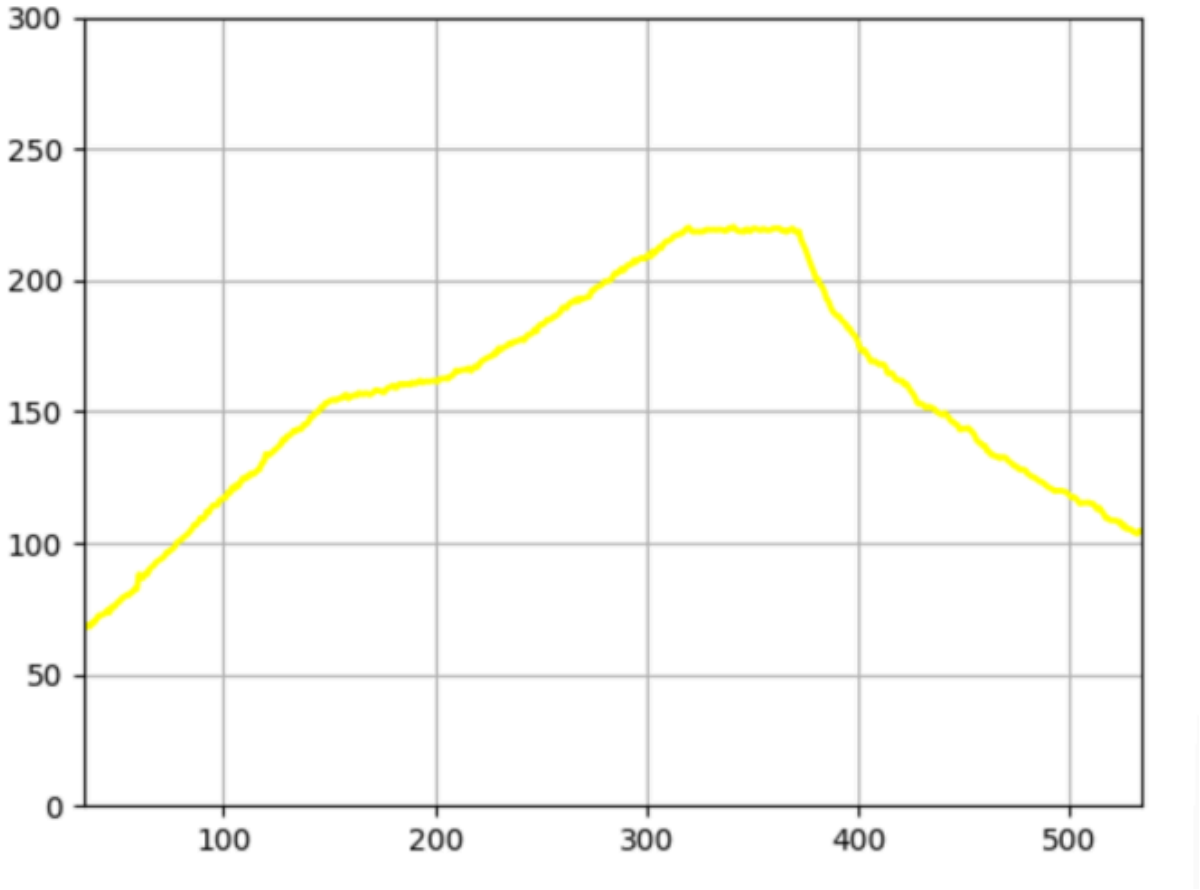


Figure 1.4 Reflow curve on Python

We first tested the reset, ensuring time would return to zero and temperature would match the room's. During this stage, we waited around five minutes but the FSM would keep resetting and the LCD would be unreadable at 55 seconds. Upon trying to debug, and finding no direct logical reason for the error, we realized that we were facing stack overflow errors where some parts of the memory of the system were being overwritten.This error was fixed by thoroughly going through all of the assembly code

again, and enabling a “pop” for every “push”, and also using returning after every function call.

We then tested the entire FSM running through all the stages based on programmed conditions using first using a soldering iron and later with the actual oven. We also extensively tested the safety abort feature where if the controller did not change states at the right temperature or time, it would immediately switch off the oven and pause the state machine until fixed. Using the oven, we tested a soak temperature of 150°C and found it to be relatively accurate using the multimeter temperature program as our baseline. After completing the above test cases, we were confident that our reflow oven controller worked correctly and thus moved onto soldering the EFM8 PCBs.

We had a few issues with the quality of our boards but most were unrelated to the oven controller. Two of our boards shifted at some point during the soldering process and thus had to be resoldered by hand. However, as seen in figure 1.5, our boards had good solder connections and the PCB itself was not greatly discoloured. Our boards were found to have also worked properly using the Lab 4 “autotest.c”.

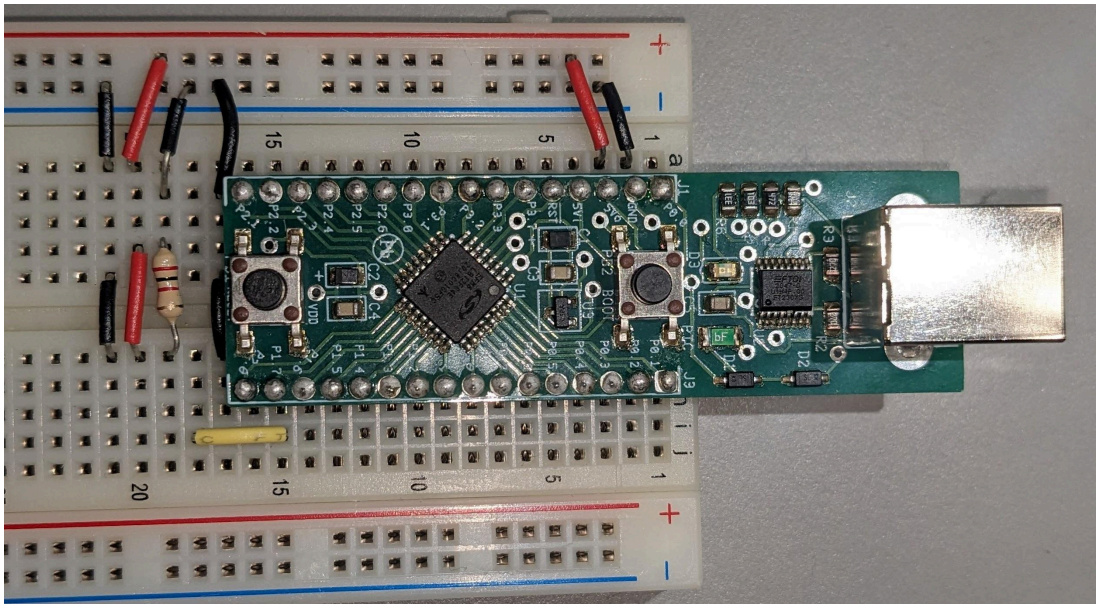


Figure 1.5 Soldered EFM8

In conclusion, our final design not only met the project requirements but also provided an engaging and informative experience for users.

Live-Long Learning

In undertaking the Reflow Oven Controller project, we learned the practical integration of hardware interfacing, programming in 8051 assembly language, and thermal management within a real-world application. Despite our foundational knowledge from courses such as CPEN 211 Introduction to Microcomputers and ELEC 201/202 Circuit Analysis 1 and 2, we encountered a knowledge gap in applying these principles to a temperature-based system's design and development. The CPEN 211 course provided us with a good understanding of microcontrollers, the project's requirement for programming in 8051 assembly necessitated a deeper proficiency in low-level

programming to control hardware and manage real-time temperature data. Similarly, while ELEC 201/202 equipped us with the theoretical background in circuit design, this project required practical applications to design a system capable of accurately measuring temperatures and controlling a high-power appliance, such as a toaster oven, using a solid-state relay.

We realized the importance of bridging theoretical knowledge with practical skills in electronics design, thermal management, and programming to create a functional and safe reflow oven controller. The absence of a specific course focusing on the integration of these skills highlighted the need for self-directed learning and adaptation throughout the project, especially in areas such as assembly language programming and the use of thermocouples for accurate temperature measurement and control. Through this project, we not only expanded our technical knowledge but also developed a deeper appreciation for the interdisciplinary approach required in modern engineering projects, blending electronics, computer programming, and thermal dynamics to meet the project's specifications and safety requirements.

Conclusion

In conclusion, our project was designed to facilitate the assembly of surface mount devices (SMDs) onto printed circuit boards (PCBs) using reflow soldering. This controller uses an N76003 microcontroller to accurately measure and control the temperature of a toaster oven, ranging from 25°C to 240°C. We used a K-type thermocouple with cold junction compensation for temperature measurement. The

system was programmed in 8051 assembly language, allowing for precise control over the heating process, including the implementation of a solid-state relay (SSR) to regulate the oven's power.

A user interface was developed to select reflow profile parameters, display temperature and reflow states on an LCD, and start/stop the reflow process. An innovative feature of our project is the visual feedback system that displays three different photos of a professor (colored green, yellow, and red) depending on the current temperature, enhancing the user interaction by providing an intuitive understanding of the oven's status.

Throughout the project, we encountered several challenges, particularly in programming the microcontroller in assembly language, which required a steep learning curve. The precise temperature control and the integration of the thermocouple also presented difficulties, necessitating careful calibration and testing to ensure accuracy. Implementing the extra functionality of displaying colored images based on temperature involved both hardware and software creativity, requiring us to efficiently manage memory and processing power.

The project demanded approximately 30 hours of work, reflecting the time invested in design, programming, assembly, and testing. This time was spread across various stages, including initial planning, coding in assembly language, hardware integration, debugging, and final testing. The effort to implement the extra visual feedback

functionality required innovative thinking and additional hours of work, but it significantly enhanced the project's usability and educational value.

Overall, the Reflow Oven Controller project not only tested our technical skills and knowledge in microcontroller electronics and programming but also taught us valuable lessons in project management, problem-solving, and the importance of user-centered design. Despite the challenges encountered, the successful completion of the project and the implementation of unique features have been immensely rewarding.

References

[1] N76E003 datasheet.
https://www.nuvoton.com/resource-files/DS_N76E003_EN_Rev1.08.pdf

[2] Sample assembly code and instructions provided by Professor Jesus Calviño-Fraga for the ELEC291 course at UBC.

[3] Calviño-Fraga, J., "Project 1 – EFM8 board, FSM, NVMEM", University of British Columbia, Electrical and Computer Engineering, ELEC291/ELEC292, Department of Electrical and Computer Engineering, UBC, February 9, 2024.

[4] Calviño-Fraga, Jesús. "Project 1 - Reflow Oven Controller." University of British Columbia, Electrical and Computer Engineering, February 2, 2024.

[5] Texas Instruments. "LM393-N: Low Power Low Offset Voltage Dual Comparators." Texas Instruments, July 2019.

Bibliography

Alexander, C. K., and Sadiku, M. "Fundamentals of Electric Circuits." McGraw-Hill, 4th ed., 2009.

Dally, William J., and R. Curtis Harting. "Digital Design: A Systems Approach." Cambridge University Press, 2012.

Appendix A

Temperature Validation Data (C°) (Tc=20.7C°) (Date:15/02/2024)								
0s ~ 120s			121s~240s			241s ~ 360s		
Multimeter	LCD	Difference	Multimeter	LCD	Difference	Multimeter	LCD	Difference
22.1	20.7	1.4	119.8	119.7416	0.06	178.4	198.8254	20.43
20	20.7	0.7	119.8	120.3336	0.53	178.2	197.7598	19.56
20.7	20.7	0	120	120.3632	0.36	178	196.5832	18.58
20.8	20.7	0.1	120.1	120.5112	0.41	177.6	195.1624	17.56
20.9	20.7	0.2	119.7	120.6888	0.99	177.4	193.823	16.42
21	20.7	0.3	119.6	121.1698	1.57	176.7	192.8462	16.15
24.4	20.7	3.7	119.6	121.2586	1.66	176.4	191.862	15.46
25.2	20.7	4.5	120	121.3474	1.35	175.9	190.8778	14.98
25.6	20.7	4.9	120.1	121.3474	1.25	175.4	189.6272	14.23
26.2	20.7	5.5	120	122.2428	2.24	174.3	188.4654	14.17
26.8	22.8386	3.96	119.8	121.6138	1.81	174	187.9326	13.93
26.9	24.0892	2.81	120.3	122.331	2.03	173.6	187.843	14.24

				6			8	
27	24.7996	2.2	120.7	121.7026	1	172.8	187.0372	14.24
27.4	24.9772	2.42	122.3	122.2428	0.06	172.2	186.3194	14.12
27.7	25.8726	1.83	123.7	122.0282	1.67	171.7	185.6978	14
27.6	25.695	1.91	125.4	121.7914	3.61	170.6	184.8024	14.2
27.8	26.4054	1.39	127.4	121.7026	5.7	170.2	183.9958	13.8
27.8	26.4128	1.39	131.5	121.8802	9.62	169.9	183.3742	13.47
27.8	26.9456	0.85	133.9	122.2428	11.66	169.5	182.0348	12.53
27.9	27.1232	0.78	136.6	122.2428	14.36	168.5	181.4946	12.99
28.2	26.9456	1.25	141.3	122.0652	19.23	168.1	181.317	13.22
28.3	27.3008	1	141.9	121.8802	20.02	167.7	181.2282	13.53
28.4	27.3008	1.1	143.2	122.4796	20.72	167.1	180.6954	13.6
28.6	26.6792	1.92	148.3	123.486	24.81	166.8	180.3328	13.53
28.7	27.8336	0.87	150.9	124.2038	26.7	166.4	179.8888	13.49
28.8	27.4784	1.32	153.6	125.8984	27.7	165.9	180.0664	14.17
28.8	27.841	0.96	165.6	127.4154	38.18	166	179.7112	13.71
28.9	27.8336	1.07	180.6	130.3606	50.24	166	178.8158	12.82
28.9	28.0112	0.89	190	133.3058	56.69	165.5	178.727	13.23

28.9	28.3738	0.53	201.2	136.1622	65.04	165.1	177.654	12.55
29	28.1888	0.81	207.4	139.3738	68.03	165.1	177.3876	12.29
29	27.841	1.16	213	142.3856	70.61	164.9	176.766	11.87
29.1	28.1	1	218.6	143.6584	74.94	164.6	176.3146	11.71
29.1	28.0112	1.09	226.8	145.5306	81.27	164.1	175.7818	11.68
29.1	28.4626	0.64	230	148.1946	81.81	163.2	175.4266	12.23
29.5	28.729	0.77	231.8	152.0426	79.76	163.2	174.5312	11.33
29.4	28.3738	1.03	236.2	155.158	81.04	163.2	174.0798	10.88
29.3	28.3738	0.93	237.6	158.8284	78.77	162.7	172.918	10.22
29.3	28.1962	1.1	239	171.0458	67.95	161.7	172.7404	11.04
29.5	28.6402	0.86	242.1	188.1916	53.91	161.4	172.1188	10.72
29.5	28.8178	0.68	242.9	196.043	46.86	161	171.9412	10.94
29.5	28.9066	0.59	242.7	203.1692	39.53	160.1	170.957	10.86
29.6	28.3738	1.23	241.8	211.2944	30.51	159.9	170.3354	10.44
29.5	28.0186	1.48	242.4	218.3392	24.06	159.6	169.9728	10.37
29.8	28.5514	1.25	242.5	224.1408	18.36	158.9	169.5288	10.63
29.7	29.173	0.53	240.7	228.9582	11.74	158	169.2624	11.26
29.7	29.1804	0.52	240	232.532	7.47	157.8	168.988	11.19

				4			6	
29.7	28.729	0.97	239.9	235.2926	4.61	157.9	168.6334	10.73
29.6	29.2618	0.34	240.1	238.0602	2.04	158	168.2782	10.28
29.7	29.0916	0.61	240.7	239.9324	0.77	158.2	168.6334	10.43
29.6	29.0842	0.52	240.9	241.3088	0.41	158.5	168.6334	10.13
29.7	28.9066	0.79	241	243.329	2.33	158.7	167.9156	9.22
29.8	28.729	1.07	240.9	245.6452	4.75	158.7	167.6492	8.95
30	29.0916	0.91	240.8	245.1124	4.31	158.9	167.5604	8.66
29.9	29.0916	0.81	240.7	243.6842	2.98	159.3	167.4716	8.17
29.9	28.6476	1.25	240.7	244.4834	3.78	159.5	167.2052	7.71
30	28.8178	1.18	240.9	244.7498	3.85	159.6	166.0434	6.44
29.7	29.4468	0.25	241.1	244.4834	3.38	160	165.4144	5.41
30.6	29.543	1.06	240.4	242.6112	2.21	160.2	165.777	5.58
30.5	29.1804	1.32	240	242.1672	2.17	160.6	165.5994	5
30.5	29.0916	1.41	239.5	242.1672	2.67	161.3	164.8816	3.58
30.7	29.0916	1.61	236.8	242.5224	5.72	161.1	164.1638	3.06
30.8	29.0916	1.71	235.8	243.329	7.53	160.2	163.8086	3.61
31.3	28.9066	2.39	235	243.4178	8.42	158.4	162.8244	4.42

31.2	29.7946	1.41	233.5	243.5066	10.01	157.4	163.0982	5.7
31.3	29.173	2.13	231.3	243.144	11.84	156.6	162.7356	6.14
31.2	29.4394	1.76	230.7	243.144	12.44	155.6	162.0252	6.43
31.1	30.5124	0.59	230.6	243.2402	12.64	156.3	161.3074	5.01
31	30.69	0.31	238.3	243.1884	4.89	156.5	160.6858	4.19
30.9	30.3422	0.56	240.7	243.4548	2.75	156.1	160.6858	4.59
30.3	30.69	0.39	242	243.4178	1.42	155.3	160.5008	5.2
28.9	30.69	1.79	243.5	243.144	0.36	155.1	160.412	5.31
28.4	30.875	2.48	243.3	242.5594	0.74	154.9	160.6858	5.79
28.7	30.69	1.99	243.4	241.7232	1.68	154.2	161.041	6.84
29.9	30.69	0.79	242.9	239.0444	3.86	154.1	161.485	7.39
29.9	30.1572	0.26	242.6	237.7938	4.81	154.8	161.2186	6.42
29.8	28.9066	0.89	242.1	237.113	4.99	155.6	161.7514	6.15
29.4	26.9456	2.45	241.3	234.8486	6.45	155.7	160.9596	5.26
30	27.6634	2.34	241	232.9764	8.02	155.9	161.9438	6.04
30.6	28.3738	2.23	240.5	232.3844	8.12	156.2	162.0252	5.83
30.5	29.0842	1.42	239.6	232.5324	7.07	156.3	162.7356	6.44
31.5	29.2618	2.24	239.6	238.0972	1.5	156.2	162.7356	6.54

32.1	29.1804	2.92	239.2	242.256	3.06	155.9	162.913 2	7.01
28.3	29.2692	0.97	239	244.527 8	5.53	155.7	163.364 6	7.66
27.4	30.3348	2.93	239.1	246.178	7.08	155.7	163.897 4	8.2
27.6	30.5198	2.92	240.1	245.778 4	5.68	155.9	161.936 4	6.04
27.2	30.246	3.05	240.1	246.000 4	5.9	156.3	160.952 2	4.65
26.5	32.651	6.15	246.1	246.222 4	0.12	156.7	159.790 4	3.09
26.1	29.4468	3.35	247.1	245.29	1.81	157.3	158.983 8	1.68
25.9	28.1	2.2	247.6	244.483 4	3.12	157.4	158.095 8	0.7
25.6	27.6634	2.06	247.2	243.861 8	3.34	157.6	159.250 2	1.65
25.3	27.841	2.54	247.7	243.721 2	3.98	157.6	158.628 6	1.03
25.1	26.4054	1.31	248.4	243.366	5.03	157.9	158.539 8	0.64
24.9	25.5174	0.62	249.4	242.344 8	7.06	158.2	158.717 4	0.52
24.6	25.9614	1.36	249.8	241.937 8	7.86	158	157.733 2	0.27
24.5	24.8958	0.4	250.2	242.293	7.91	157.9	157.385 4	0.51
24.3	25.1548	0.85	250	241.271 8	8.73	157.7	156.660 2	1.04
24.1	25.2436	1.14	249.8	241.271 8	8.53	157.2	156.038 6	1.16
24	24.7996	0.8	249.3	241.183	8.12	157.1	157.733 2	0.63
23.9	24.7996	0.9	249.3	242.344	6.96	156.9	158.095	1.2

				8			8	
24.1	24.2668	0.17	250.2	242.5224	7.68	156.5	158.1846	1.68
24.2	23.8228	0.38	251.3	247.6506	3.65	156.4	158.2734	1.87
24.5	23.9116	0.59	251.1	249.5228	1.58	156.4	158.3622	1.96
25.6	24.0892	1.51	249.8	250.2406	0.44	156.9	158.8062	1.91
26.4	23.8228	2.58	248.9	249.7966	0.9	157.1	158.895	1.8
27.2	23.4676	3.73	248.3	249.9298	1.63	157.1	158.5398	1.44
28.2	23.2826	4.92	247.1	251.0472	3.95	157.2	158.2734	1.07
30.4	23.734	6.67	247	251.4912	4.49	158.3	158.2734	0.03
31.5	24.4444	7.06	247.2	251.802	4.6	159.1	158.3696	0.73
32.7	24.2668	8.43	247	252.6974	5.7	159.9	158.6286	1.27
36	25.6062	10.39	247.2	252.875	5.68	160.5	158.5472	1.95
37.8	25.8726	11.93	247.2	252.3866	5.19	160.6	158.7174	1.88
40	26.9456	13.05	246.5	252.3866	5.89	161.3	159.6128	1.69
42.5	28.1074	14.39	246.6	251.2248	4.62	162.2	160.0568	2.14
47.2	29.802	17.4	247.9	251.9352	4.04	162.6	160.153	2.45
50	32.0368	17.96	249.7	252.8306	3.13	162.9	159.7978	3.1
52.7	33.1098	19.59	250.7	253.9924	3.29	162.9	160.153	2.75

57.8	35.6924	22.11	251.3	253.1858	1.89	162.5	160.412	2.09
60.2	37.8384	22.36	252.8	252.1128	0.69	162.3	160.8634	1.44
62.3	40.8724	21.43	253.5	251.4024	2.1	162.2	160.7746	1.43
64.2	43.5512	20.65	253.7	250.914	2.79	162	160.597	1.4
68.7	46.748	21.95	252.2	249.7078	2.49	162.1	160.0568	2.04
70.9	50.6922	20.21	250.1	248.8568	1.24	162.2	159.8792	2.32
73.6	54.5328	19.07	248.2	249.5228	1.32	162.4	159.8792	2.52
78.1	57.6556	20.44	247	249.619	2.62	162.5	158.9838	3.52
79.9	60.4898	19.41	246.6	249.5228	2.92	162.7	158.895	3.8
81.5	63.0132	18.49	246	249.7966	3.8	162.5	158.6286	3.87
82.8	65.692	17.11	245.4	249.1676	3.77	162.2	159.08	3.12
85.6	68.8222	16.78	245.8	249.2564	3.46	162	159.7904	2.21
87	71.6786	15.32	246.2	250.063	3.86	161.4	159.7904	1.61
88.5	74.7718	13.73	246.5	251.6688	5.17	161.2	159.524	1.68
91.5	77.6282	13.87	247.5	252.5198	5.02	161	159.968	1.03
93	80.8398	12.16	247.6	253.8592	6.26	160.9	160.597	0.3
94.4	82.8378	11.56	248	254.614	6.61	160.8	162.3804	1.58
97.1	84.4436	12.66	248.2	255.6426	7.44	160.7	162.4692	1.77

98.4	85.8348	12.57	248.2	256.2198	8.02	160.8	162.9132	2.11
99.6	88.4248	11.18	247.9	255.9534	8.05	161.1	163.0982	2
100.9	89.9418	10.96	247.2	255.243	8.04	161.3	164.075	2.77
103.2	91.9916	11.21	247.2	252.4754	5.28	161.3	164.6152	3.32
104.3	93.7306	10.57	247.3	250.507	3.21	161	165.1554	4.16
105.3	95.3364	9.96	245.9	249.397	3.5	161.1	165.5032	4.4
107	97.2974	9.7	245.2	249.1676	3.97	161.1	165.0592	3.96
107.6	99.2658	8.33	244.7	248.7236	4.02	160.7	164.889	4.19
108	100.7828	7.22	244.5	247.5618	3.06	160.6	164.4376	3.84
108.6	102.5662	6.03	244.7	248.0946	3.39	160.6	164.5264	3.93
109	103.7724	5.23	245.2	248.0576	2.86	160.5	164.4376	3.94
108.9	105.2006	3.7	245.6	248.8124	3.21	160.5	164.1638	3.66
108.5	106.8952	1.6	244.4	249.9742	5.57	160.6	164.704	4.1
107.6	108.3678	0.77	243.7	250.2406	6.54	160.6	164.9778	4.38
106.5	109.2114	2.71	243.2	250.507	7.31	160.5	164.7928	4.29
105.3	109.389	4.09	243.1	250.2406	7.14	160.4	164.9704	4.57
104.4	110.5508	6.15	243.4	250.6846	7.28	160.1	164.9778	4.88
103.2	110.9578	7.76	244.8	250.7734	5.97	159.7	165.0666	5.37

103.6	110.1586	6.56	246.7	249.8854	3.19	159.5	164.8816	5.38
104.6	110.7802	6.18	247.6	249.1232	1.52	159.4	164.5264	5.13
105.8	109.6184	3.82	248.1	249.619	1.52	159.4	164.1638	4.76
106	109.5296	3.53	252.1	249.1232	2.98	159.6	163.8086	4.21
107.1	108.501	1.4	252.2	248.1464	4.05	159.5	163.4608	3.96
108.1	106.9396	1.16	251.3	247.3842	3.92	159.7	163.4534	3.75
109.8	106.2292	3.57	249.6	246.5776	3.02	159.8	163.187	3.39
110.3	104.9268	5.37	249.2	246.8884	2.31	159.9	162.9132	3.01
111.2	105.2894	5.91	248.7	247.3842	1.32	160	163.187	3.19
113.5	107.0358	6.46	248.2	247.8282	0.37	160.1	163.631	3.53
114.1	107.7832	6.32	246.6	247.6062	1.01	160.1	163.4534	3.35
114.4	108.0126	6.39	245.5	246.6664	1.17	160.2	163.631	3.43
115.1	109.4778	5.62	244.1	246.0448	1.94	160.4	163.8086	3.41
115.2	110.2474	4.95	242.6	245.512	2.91	160.6	163.187	2.59
114.6	111.8532	2.75	242.1	245.512	3.41	160.8	163.8086	3.01
114.6	112.2084	2.39	242.4	245.6896	3.29	160.7	163.4534	2.75
115.9	113.7624	2.14	243.7	247.2066	3.51	160.2	163.4534	3.25
116.9	114.976	1.92	244.6	248.546	3.95	159.6	162.831	3.23

							8	
117.7	116.2266	1.47	244.9	249.5228	4.62	158.8	163.002	4.2
117.7	116.9444	0.76	244.8	250.5958	5.8	157.7	163.187	5.49
117	117.122	0.12	244.5	252.8306	8.33	157.1	163.187	6.09
116.6	117.2996	0.7	244.1	254.9692	10.87	155.9	163.4534	7.55
115.9	116.7076	0.81	242.9	254.2588	11.36	155.7	163.002	7.3
116	116.8556	0.86	242.1	252.9194	10.82	155.6	163.3646	7.76
116.6	118.9054	2.31	241.8	252.2534	10.45	155.5	162.6468	7.15
116.8	119.7416	2.94	239.7	251.4024	11.7	155.3	162.2916	6.99
117.6	120.156	2.56	239.7	251.0028	11.3	155	161.9438	6.94
118	119.8896	1.89	239.4	249.9742	10.57	154.8	161.8476	7.05
118.2	118.3726	0.17	238.2	248.99	10.79	154.9	162.0252	7.13
118.2	117.7806	0.42	237.4	247.029	9.63	154.9	162.3804	7.48
118.3	117.7806	0.52	236.9	245.2382	8.34	155.1	161.9364	6.84
118.9	117.2108	1.69	235.8	244.9718	9.17	155.1	161.8476	6.75
119.4	117.6548	1.75	235.1	244.6166	9.52	154.9	162.2028	7.3
119.9	118.3726	1.53	234.4	245.2382	10.84	154.9	162.1214	7.22
120.4	118.3726	2.03	233.5	245.6008	12.1	154.8	162.6542	7.85

121.8	119.2606	2.54	232.6	246.9402	14.34	154.7	162.4692	7.77
122.4	120.1856	2.21	232.2	247.1178	14.92	154.7	162.4766	7.78
122.8	120.5482	2.25	231.5	247.6506	16.15	154.6	162.4766	7.88
123	119.564	3.44	230.2	246.6664	16.47	154.4	162.6542	8.25
122.9	120.1856	2.71	229.6	246.4	16.8	154.2	163.0982	8.9
122.6	120.6	2	229.1	245.6378	16.54	154	163.372	9.37
122.5	120.5482	1.95	227.6	244.7942	17.19	153.9	163.187	9.29
123.1	121.081	2.02	226.8	244.0838	17.28	154.1	162.9132	8.81
123.5	122.154	1.35	226	242.922	16.92	154.2	162.299	8.1
123.5	122.9532	0.55	223.9	241.9378	18.04	153.9	161.1372	7.24
124.1	124.0262	0.07	222.4	242.0636	19.66	154	160.3306	6.33
125.1	124.4406	0.66	221.7	241.849	20.15	154.1	159.8792	5.78
125.4	124.7366	0.66	220.4	240.332	19.93	154.3	158.7248	4.42
125.5	125.0696	0.43	218.9	239.703	20.8	154.3	158.3696	4.07
130	124.7366	5.26	217.9	238.6744	20.77	154.4	158.2734	3.87
130.7	124.263	6.44	217.1	238.4894	21.39	154.3	157.822	3.52
130.4	125.3656	5.03	214.7	237.2018	22.5	154.1	158.5472	4.45
128.5	125.9872	2.51	214.5	236.1362	21.64	154	157.8368	3.84

127.8	126.3202	1.48	214	235.2778	21.28	153.4	157.2004	3.8
127.1	126.6754	0.42	212.7	235.0114	22.31	153	157.2966	4.3
126.7	127.2082	0.51	211.2	234.4342	23.23	152.8	157.2078	4.41
126.5	128.37	1.87	209.5	233.635	24.13	153	157.4742	4.47
125.9	127.9482	2.05	208.2	232.4362	24.24	153.2	157.822	4.62
124.6	130.627	6.03	206.8	231.8516	25.05	153.4	157.563	4.16
123.4	132.588	9.19	205.1	231.4372	26.34	153.6	157.3854	3.79
123.2	133.106	9.91	202.7	230.1496	27.45	153.5	157.2078	3.71
123	131.5224	8.52	201.8	228.899	27.1	153.5	157.6518	4.15
122.6	130.8712	8.27	200.9	228.3662	27.47	153.5	157.563	4.06
121.5	129.739	8.24	199.3	227.5078	28.21	153.7	157.2966	3.6
120.6	129.1988	8.6	198.7	225.6282	26.93	154.1	157.119	3.02
119.8	128.999	9.2	198.2	224.0816	25.88	154.4	157.2966	2.9
118.7	129.0212	10.32	196.9	223.3638	26.46	155.5	157.0228	1.52
118.3	127.3266	9.03	196	222.7718	26.77	155.7	157.119	1.42
118	126.2314	8.23	195.1	220.0634	24.96	155.9	156.7638	0.86
117.4	125.632	8.23	194.1	220.4482	26.35	156.3	155.9572	0.34
117.4	125.454	8.05	191.9	217.029	25.13	156.5	156.497	0

	4			4			4	
117.3	124.9142	7.61	191.1	217.3846	26.28	156.6	156.3124	0.29
117.3	124.6478	7.35	190.3	216.5188	26.22	156.7	156.5862	0.11
117.5	123.671	6.17	188.6	216.0674	27.47	156.7	156.1348	0.57
117.6	122.302	4.7	187.6	214.8834	27.28	156.8	156.0386	0.76
117.6	121.8802	4.28	186.6	213.3664	26.77	156.7	156.5714	0.13
117.8	121.1698	3.37	185.4	210.7098	25.31	156.7	156.9414	0.24
117.9	120.5482	2.65	185.1	210.3324	25.23	156.8	156.675	0.12
118	119.564	1.56	184.8	208.9042	24.1	157	156.7638	0.24
118.5	119.2976	0.8	184.4	206.7804	22.38	157.1	156.7638	0.34
118.9	119.564	0.66	183.4	205.6852	22.29	157.3	157.2078	0.09
119.2	119.4752	0.28	182.6	204.1904	21.59	157.4	156.9414	0.46
119.4	119.564	0.16	182	202.851	20.85	158.1	155.8684	2.23
119.5	119.2088	0.29	180.5	201.778	21.28	158.8	155.4984	3.3
119.5	120.008	0.51	179.8	201.3118	21.51	159.2	155.602	3.6
			179.3	200.4386	21.14	159.5	155.5132	3.99
			178.7	199.632	20.93	160	155.9572	4.04
						160.6	156.0386	4.56

						161.2	156.4086	4.79

Appendix B

Python Source Code

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
#import vlc
import sys, serial
from PIL import Image, ImageTk
import tkinter as tk
import threading  # Import threading

xsize = 500

# Configure the serial port
ser = serial.Serial(
    port='COM6',  # Update to the correct port
    baudrate=115200,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_TWO,
    bytesize=serial.EIGHTBITS
)
ser.isOpen()

# Declare global variables for Tkinter window and label
root = None
img_label = None

# Function to load images and update the Tkinter label
def update_image(state):
    global img_label
    images = {
        "green": ImageTk.PhotoImage(Image.open("green.png")),
        "yellow": ImageTk.PhotoImage(Image.open("yellow.png")),
```

```
        "red": ImageTk.PhotoImage(Image.open("red.png"))
    }
    img_label.config(image=images[state])
    img_label.image = images[state]  # Keep a reference

# Function to create and run the Tkinter window
def create_tkinter_window():
    global root, img_label
    root = tk.Tk()
    root.title("Temperature State")
    img_label = tk.Label(root)  # Placeholder for the image
    img_label.pack()
    root.mainloop()

# Start the Tkinter window in a separate thread
tkinter_thread = threading.Thread(target=create_tkinter_window,
daemon=True)
tkinter_thread.start()

def data_gen():
    t = data_gen.t
    while True:
        t += 1
        val = float(ser.readline())
        yield t, val

def run(data):
    # Update the data
    t, y = data
    xdata.append(t)
    ydata.append(y)
    if t > xsize:  # Scroll to the left.
        ax.set_xlim(t - xsize, t)

    #play audio file through vlc media player
    # p =
vlc.MediaPlayer("C:\Users\abhi1\OneDrive\Desktop\ELEC291_package\Project
1\shboom.m4a") #insert audio file location path here
    # p.play(shboom.m4a) #insert audio file name here
```

```
    # Set color based on temperature range and update image in Tkinter
window
    color = 'green' if y <= 125 else ('yellow' if 125 < y <= 200 else
'red')
    line.set_data(xdata, ydata)
    line.set_color(color)
    update_image(color)  # Call update_image function here

    return line,

def on_close_figure(event):
    if root:
        root.quit()  # Close the Tkinter window properly
    sys.exit(0)

    #p.stop(shboom.m4a) #insert audio file name here

data_gen.t = 0
xdata, ydata = [], []
fig = plt.figure()
fig.canvas.mpl_connect('close_event', on_close_figure)
ax = fig.add_subplot(111)
line, = ax.plot([], [], lw=2)
ax.set_ylim(0, 300)
ax.set_xlim(0, xsize)
ax.grid()

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

plt.show()
```

Appendix C

Assembly Source Code

```
 ; main program to be run
$NOLIST
$MODN76E003
$LIST

;  N76E003 pinout:
;                                  -------
;       PWM2/IC6/T0/AIN4/P0.5 -|1    20|- P0.4/AIN5/STADC/PWM3/IC3
;               TXD/AIN3/P0.6 -|2    19|- P0.3/PWM5/IC5/AIN6
;               RXD/AIN2/P0.7 -|3    18|- P0.2/ICPCK/OCDCK/RXD_1/[SCL]
;                    RST/P2.0 -|4    17|- P0.1/PWM4/IC4/MISO
;        INT0/OSCIN/AIN1/P3.0 -|5    16|- P0.0/PWM3/IC3/MOSI/T1
;              INT1/AIN0/P1.7 -|6    15|- P1.0/PWM2/IC2/SPCLK
;                         GND -|7    14|- P1.1/PWM1/IC1/AIN7/CLO
;[SDA]/TXD_1/ICPDA/OCDDA/P1.6 -|8    13|- P1.2/PWM0/IC0
;                         VDD -|9    12|- P1.3/SCL/[STADC]
;            PWM5/IC7/SS/P1.5 -|10   11|- P1.4/SDA/FB/PWM1
;                                  -------
;

;-----------------clock values----------------------
CLK               EQU 16600000 ; Microcontroller system frequency in Hz
BAUD              EQU 115200 ; Baud rate of UART in bps
TIMER1_RELOAD     EQU (0x100-(CLK/(16*BAUD)))
TIMER0_RATE         EQU 4096     ; 2048Hz squarewave (peak amplitude of
CEM-1203 speaker)

TIMER0_RELOAD       EQU ((65536-(CLK/TIMER0_RATE)))
TIMER2_RATE         EQU 1000    ; 1000Hz, for a timer tick of 1ms
TIMER2_RELOAD       EQU ((65536-(CLK/TIMER2_RATE)))

PAGE_ERASE_AP   EQU 00100010b   ;for flash memory
BYTE_PROGRAM_AP EQU 00100001b

ORG 0x0000
```

```
    ljmp MainProgram

; External interrupt 0 vector (not used in this code)
org 0x0003
    reti

; Timer/Counter 0 overflow interrupt vector
org 0x000B
    ljmp Timer0_ISR

; External interrupt 1 vector (not used in this code)
org 0x0013
    reti

; Timer/Counter 1 overflow interrupt vector (not used in this code)
org 0x001B
    reti

; Serial port receive/transmit interrupt vector (not used in this code)
org 0x0023
    reti

; Timer/Counter 2 overflow interrupt vector
org 0x002B
    ljmp Timer2_ISR


;               1234567890123456    <- This helps determine the location
of the counter

param1:     db 'SOAK TEMP:      ', 0

param2:     db 'SOAK TIME:      ', 0

param3:     db 'REFLOW TEMP:    ', 0

param4:     db 'REFLOW TIME:     ', 0

startmsg:     db '->START 2 START  ', 0
```

```
editmsg:     db '->EDIT 2 EDIT    ', 0

title06:     db 'CONFIRM?          ', 0

title07:     db 'REFLOW TIME?      ', 0

title08:     db 'CONFIRM?          ', 0

title1:     db '1 RAMP   Ts:       ', 0

title2:     db '2 SOAK             ', 0

title3:     db '3 RAMP   Tr:       ', 0

title4:     db '4 REFLOW           ', 0

title5:     db '5 COOLING          ', 0

blank:     db '                  ', 0

cseg
; These 'equ' must match the hardware wiring
LCD_RS equ P1.3
LCD_E  equ P1.4
LCD_D4 equ P0.0
LCD_D5 equ P0.1
LCD_D6 equ P0.2
LCD_D7 equ P0.3

PWM_OUT     equ P1.0 ; logic=1 oven on
SOUND_OUT   equ P1.6

;---------------------------------------------------
DSEG at 0x30

;---------------temp validation----------------------
```

```
x:    ds 4

y:    ds 4

bcd: ds 5

VLED_ADC: ds 2


;----------------- clock and speaker ------------------------------

Count1ms:               ds 2 ;to determine when one second has passed

secs_ctr:               ds 1

mins_ctr:               ds 1



state_secs_ctr:         ds 2


buzzer:                 ds 1

;-----------------FSM-s--------------------

FSM1_state:             ds 1 ; determine state

edit_state:             ds 1

;----------------fsm values--------------------

stemp:                  ds 2

stime:                  ds 2

rtemp:                  ds 2

rtime:                  ds 2
```

```
stime_bcd:              ds 2

rtime_bcd:              ds 2

;-----------------timer values-----------------

soak_timer:             ds 2

reflow_timer:           ds 1

;----------------PWM-----------------

pwm_counter:            ds 1

pwm:                    ds 1

;------------------oven temp-------------

oven_temp:              ds 2


;--------------------flags-----------------------------
BSEG

mf: dbit 1

one_second_flag: dbit 1

;----------------push buttons------------------

incr:           dbit 1

decr:           dbit 1

edit:           dbit 1

reset:          dbit 1
```

```
start_stop:     dbit 1

;---------------soak and reflow time flag-------------

soak_timer_flag:    dbit 1

reflow_timer_flag:  dbit 1

soak_temp_flag:    dbit 1

reflow_temp_flag:  dbit 1
;--------------PWM flag----------------------

pwm_flag:           dbit 1

$NOLIST
$include(LCD_4bit.inc) ; A library of LCD related functions and utility
macros
$include(math32.inc)
$LIST
;---------flash memory functions-------------

putchar:
    JNB TI, putchar
    CLR TI
    MOV SBUF, a
    RET


SendString:
    CLR A
    MOVC A, @A+DPTR
    JZ SSDone
    LCALL putchar
    INC DPTR
    SJMP SendString

SSDone:
    ret
```

```
; Sends the byte in the accumulator to the serial port in decimal

Send_byte:
    mov b, #100
    div ab
    orl a, #'0'
    lcall putchar
    mov a, b
    mov b, #10
    div ab
    orl a, #'0'
    lcall putchar
    mov a, b
    orl a, #'0'
    lcall putchar
    mov a, #'\r'
    lcall putchar
    mov a, #'\n'
    lcall putchar
    ret

;---------------------------------;
; Routine to initialize the pins  ;
; for input out - ADC inc         ;
;---------------------------------;

Init_All:
    ; 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

    orl CKCON, #0x10 ; CLK is the input for timer 1
    orl PCON, #0x80 ; Bit SMOD=1, double baud rate
    mov SCON, #0x52
```

```
    anl T3CON, #0b11011111
    anl TMOD, #0x0F ; Clear the configuration bits for timer 1
    orl TMOD, #0x20 ; Timer 1 Mode 2
    mov TH1, #TIMER1_RELOAD ; TH1=TIMER1_RELOAD;
    setb TR1

    mov SCON, #52H


    ; Using timer 0 for delay functions.  Initialize here:
    clr TR0 ; Stop timer 0
    orl CKCON,#0x08 ; CLK is the input for timer 0
    anl TMOD,#0xF0 ; Clear the configuration bits for timer 0
    orl TMOD,#0x01 ; Timer 0 in Mode 1: 16-bit timer

    ; Initialize the pins used by the ADC (P1.1, P1.7) as input.
    orl P1M1, #0b10000010
    anl P1M2, #0b01111101

    ; Since the reset button bounces, we need to wait a bit before
    ; sending messages, otherwise we risk displaying gibberish!
    mov R1, #200
    mov R0, #104
    djnz R0, $   ; 4 cycles->4*60.285ns*104=25us
    djnz R1, $-4 ; 25us*200=5.0ms

    ; Initialize and start the ADC:
    anl ADCCON0, #0xF0
    orl ADCCON0, #0x07 ; Select channel 7
    ; AINDIDS select if some pins are analog inputs or digital I/O:
    mov AINDIDS, #0x00 ; Disable all analog inputs
    orl AINDIDS, #0b10000001 ; Activate AIN0 and AIN7 analog inputs
    orl ADCCON1, #0x01 ; Enable ADC

    ret

;-----------functions for temp reading ADC-----------

Read_ADC:
```

```
    clr ADCF
    setb ADCS ;  ADC start trigger signal
    jnb ADCF, $ ; Wait for conversion complete

    ; Read the ADC result and store in [R1, R0]
    mov a, ADCRL
    anl a, #0x0f
    mov R0, a
    mov a, ADCRH
    swap a
    push acc
    anl a, #0x0f
    mov R1, a
    pop acc
    anl a, #0xf0
    orl a, R0
    mov R0, A
    ret

Send_BCD mac
    push ar0
    mov r0, %0
    lcall ?Send_BCD
    pop ar0
    endmac


    ?Send_BCD:
    push acc
    ; Write most significant digit
    mov a, r0
    swap a
    anl a, #0fh
    orl a, #30h
    lcall putchar
    ; write least significant digit
    mov a, r0
    anl a, #0fh
    orl a, #30h
```

```
    lcall putchar
    pop acc
    ret

    SendChar mac
    push acc
    mov a,%0
    lcall putchar
    pop acc
    endmac


Display_formated_BCD:
    Set_Cursor(2, 1)
    Display_BCD(bcd+3)
    Display_BCD(bcd+2)
    Display_char(#'.')
    Display_BCD(bcd+1)
    Display_BCD(bcd+0)

    ret
;---------------------------------;
; Routine to initialize the ISR   ;
; for timer 0                     ;
;---------------------------------;

Timer0_Init:
    orl CKCON, #0b00001000 ; Input for timer 0 is sysclk/1
    mov a, TMOD
    anl a, #0xf0 ; 11110000 Clear the bits for timer 0
    orl a, #0x01 ; 00000001 Configure timer 0 as 16-timer
    mov TMOD, a
    mov TH0, #high(TIMER0_RELOAD)
    mov TL0, #low(TIMER0_RELOAD)
    ; Enable the timer and interrupts
    setb ET0  ; Enable timer 0 interrupt
    setb TR0  ; Start timer 0
    ret
```

```
;---------------------------------;
; ISR for timer 0.  Set to execute;
; every 1/4096Hz to generate a    ;
; 2048 Hz wave at pin SOUND_OUT   ;
;---------------------------------;

Timer0_ISR:
    ;clr TF0  ; According to the data sheet this is done for us already.
    ; Timer 0 doesn't have 16-bit auto-reload, so
    clr TR0
    mov TH0, #high(TIMER0_RELOAD)
    mov TL0, #low(TIMER0_RELOAD)
    setb TR0
    cpl SOUND_OUT
    reti

;---------------------------------;
; Routine to initialize the ISR   ;
; for timer 2                     ;
;---------------------------------;

Timer2_Init:
    mov T2CON, #0 ; Stop timer/counter.  Autoreload mode.z
    mov TH2, #high(TIMER2_RELOAD)
    mov TL2, #low(TIMER2_RELOAD)
    ; Set the reload value
    orl T2MOD, #0x80 ; Enable timer 2 autoreload
    mov RCMP2H, #high(TIMER2_RELOAD)
    mov RCMP2L, #low(TIMER2_RELOAD)
    ; Init One millisecond interrupt counter.  It is a 16-bit variable
made with two 8-bit parts
    clr a
    mov Count1ms+0, a
    mov Count1ms+1, a
    ; Enable the timer and interrupts
    orl EIE, #0x80 ; Enable timer 2 interrupt ET2=1
    ;setb TR2  ; Enable timer 2
    ret
```

```
;---------------------------------;
; ISR for timer 2                 ;
;---------------------------------;

Timer2_ISR:
    clr TF2  ; Timer 2 doesn't clear TF2 automatically. Do it in the
ISR.  It is bit addressable.
    cpl P0.4 ; To check the interrupt rate with oscilloscope. It must be
precisely a 1 ms pulse.

    ; The two registers used in the ISR must be saved in the stack
    push acc
    push psw

    ; Increment the 16-bit one mili second counter
    inc Count1ms+0    ; Increment the low 8-bits first
    mov a, Count1ms+0 ; If the low 8-bits overflow, then increment high
8-bits
    jnz Inc_Done
    inc Count1ms+1
    ;mov a, buzzer
    ;cjne a, #0, buzzer_out


pwm_counts:

    inc pwm_counter
    clr c
    mov a, pwm
    subb a, pwm_counter ;If pwm_counter <= pwm then c=1
    cpl c
    mov PWM_OUT, c

    mov a, pwm_counter
    cjne a, #100, Inc_Done
    mov pwm_counter, #0
```

```
Inc_Done:
    ; Check if half second has passed
    mov a, Count1ms+0
    cjne a, #low(1000), jumper ; Warning: this instruction changes the
carry flag!
    mov a, Count1ms+1
    cjne a, #high(1000), Timer2_ISR_done

    ; 500 milliseconds have passed.  Set a flag so the main program
knows
    setb one_second_flag; Let the main program know half second had
passed
    cpl TR0 ; Enable/disable timer/counter 0. This line creates a
beep-silence-beep-silence sound.
    ; Reset to zero the milli-seconds counter, it is a 16-bit variable
    clr a
    mov Count1ms+0, a
    mov Count1ms+1, a
    ; Increment the BCD counter
    mov a, secs_ctr
    ;jnb UPDOWN, Timer2_ISR_decrement
    add a, #0x01
    sjmp clkup


jumper:
    ljmp Timer2_ISR_done

; clkup:
;   mov a, state_secs_ctr                       ;variables for state
timers
;   add a, #1
;   da a
;   mov state_secs_ctr, a
;   mov a, secs_ctr
;   add a, #1
;   da a ; Decimal adjust instruction.  Check datasheet for more
details!
;   mov secs_ctr, a
```

```
;    cjne a, #0x60, Timer2_ISR_done
;    mov secs_ctr, #0x00
;    mov a, mins_ctr
;    add a, #1
;    da a
;    mov mins_ctr, a


clkup:
    mov a, state_secs_ctr
    inc a   ;variables for state timers
    da a
    mov state_secs_ctr, a
    mov a, secs_ctr
    add a, #0x01
    da a
    mov secs_ctr, a
    cjne a, #0x60, Timer2_ISR_done
    mov secs_ctr, #0x00
    mov a, mins_ctr
    inc a                    ; Decimal adjust instruction.  Check
datasheet for more details!
    da a
    mov mins_ctr, a



Timer2_ISR_done:
    pop psw
    pop acc
    reti


;-------------------Generate Display--------------------------

GenerateDisplay:


MtimerDisplay:
```

```
    Set_Cursor(2, 15)
    Display_BCD(secs_ctr)
    Set_Cursor(2, 14)
    Display_char(#':')
    Set_Cursor(2, 12)
    Display_BCD(mins_ctr)


TempDisplay:
    Set_Cursor(2, 1)
    ; Read the 2.08V LED voltage connected to AIN0 on pin 6
    anl ADCCON0, #0xF0
    orl ADCCON0, #0x00 ; Select channel 0

    lcall Read_ADC
    ; Save result for later use
    mov VLED_ADC+0, R0
    mov VLED_ADC+1, R1

    ; Read the signal connected to AIN7
    anl ADCCON0, #0xF0
    orl ADCCON0, #0x07 ; Select channel 7
    lcall Read_ADC

    ; Convert to voltage
    mov x+0, R0
    mov x+1, R1
    ; Pad other bits with zero
    mov x+2, #0
    mov x+3, #0
    Load_y(20740) ; The MEASURED LED voltage: 2.074V, with 4 decimal
places
    lcall mul32
    ; Retrive the ADC LED value
    mov y+0, VLED_ADC+0
    mov y+1, VLED_ADC+1
    ; Pad other bits with zero
    mov y+2, #0
```

```
    mov y+3, #0
    lcall div32


    Load_y (74)
    lcall mul32
    Load_y (207000)
    lcall add32
    ; x is vCH
    ; Load_y(27300) ;load 2.73V into y
    ; lcall sub32
    ; Load_y(100)
    ;lcall mul32
    ;x is now T
    ; Convert to BCD and display
    lcall hex2bcd
    lcall Display_formated_BCD

    Send_BCD(bcd+3)
    Send_BCD(bcd+2)
    SendChar(#'.')
    Send_BCD(bcd+1)
    Send_BCD(bcd+0)
    SendChar(#'\n')

    mov oven_temp+0, bcd+2          ;saving the oven temp
    mov oven_temp+1, bcd+3

    ret

;-----------------lcd pushbuttons--------------------------

LCD_PB:
    ; Set variables to 1: 'no push button pressed'
    setb incr
    setb decr
    setb edit
    setb start_stop
    setb reset
```

```
; The input pin used to check set to '1'
setb P1.5

; Check if any push button is pressed
clr P0.0
clr P0.1
clr P0.2
clr P0.3
clr P1.3
jb P1.5, LCD_PB_Done

; Debounce
mov R2, #50
lcall waitms
jb P1.5, LCD_PB_Done

; Set the LCD data pins to logic 1
setb P0.0
setb P0.1
setb P0.2
setb P0.3
setb P1.3

; Check the push buttons one by one
clr P1.3
mov c, P1.5
mov reset, c
setb P1.3

clr P0.0
mov c, P1.5
mov start_stop, c
setb P0.0

clr P0.1
mov c, P1.5
mov edit, c
setb P0.1
```

```
    clr P0.2
    mov c, P1.5
    mov decr, c
    setb P0.2

    clr P0.3
    mov c, P1.5
    mov incr, c
    setb P0.3


LCD_PB_Done:
    ret


wait_1ms:
    clr TR0 ; Stop timer 0
    clr TF0 ; Clear overflow flag
    mov TH0, #high(TIMER0_RELOAD)
    mov TL0,#low(TIMER0_RELOAD)
    setb TR0
    jnb TF0, $ ; Wait for overflow
    ret

; Wait the number of miliseconds in R2

waitms:
    lcall wait_1ms
    djnz R2, waitms
    ret

;---------------------------------- Main Program
---------------------------------


MainProgram:                                ;main program

    mov SP, #0x7f
    lcall Init_All
```

```
    lcall LCD_4BIT                                          ; initialize
settings

    clr one_second_flag
    clr soak_timer_flag                                 ; flags
    clr reflow_timer_flag
    clr pwm_flag

    mov FSM1_state, #0
    mov edit_state, #0
                                        ; fsm
    mov secs_ctr, #0                                    ; clock
    mov mins_ctr, #0
    mov soak_timer, #0
    mov reflow_timer, #0
    mov buzzer, #0
    mov pwm_counter, #0
                                        ; edit settings
    ;lcall Timer1_Init                                  ; timers
    lcall Timer2_Init

    setb TR2
    setb EA



    mov stemp+0, #0x50
    mov stemp+1, #0x01
    mov stime, #0x60
    mov a, stime
    da a
    mov stime_bcd, a
    mov rtemp+0, #0x17
    mov rtemp+1, #0x02
    mov rtime, #0x45
    mov a, rtime
    da a
    mov rtime_bcd, a
```

```
                                        ;


forever:
    jnb one_second_flag, SkipDisplay            ;every second, the
screen should update
    clr one_second_flag
    lcall GenerateDisplay



SkipDisplay:                                    ;if no display needed

    ljmp FSM1
;---------------------------------------------------------------------

;-----------------------check_stop_restart-----------------------

check_stop_restart:
    lcall LCD_PB                ;evaluate the buttons   ;need to wait so
dont need lightning button presses
    mov c, start_stop
    jnc stop_protocol
    mov c, reset
    jnc reset_protocol
    ret


stop_protocol:
    mov pwm, #0
    mov FSM1_state, #0
    ret


reset_protocol:
    mov pwm, #0                 ;turn off oven
    mov FSM1_state, #0          ;go back to select stage
    ret
```

```
;----------------------------load functions to compare values------

load_soak_temp:
    push acc
    push psw
    clr mf
    clr A

    mov x+0,a
    mov x+1,a
    mov x+2,a
    mov x+3,a

    mov y+0,a
    mov y+1,a
    mov y+2,a
    mov y+3,a


    mov x+0, stemp+0
    mov x+1, stemp+1

    mov y+0, oven_temp+0
    mov y+1, oven_temp+1

    pop psw
    pop acc
    ret


load_reflow_temp:
    push acc
    push psw
    clr mf
    clr a

    mov x+0,a
    mov x+1,a
```

```
    mov x+2,a
    mov x+3,a

    mov y+0,a
    mov y+1,a
    mov y+2,a
    mov y+3,a


    mov x+0, rtemp+0
    mov x+1, rtemp+1

    mov y+0, oven_temp+0
    mov y+1, oven_temp+1

    pop psw
    pop acc
    ret


load_abort_temp:
    push acc
    push psw
    clr mf
    clr a

    mov x+0,a
    mov x+1,a
    mov x+2,a
    mov x+3,a

    mov y+0,a
    mov y+1,a
    mov y+2,a
    mov y+3,a

    mov x+0, #0x50

    mov y+0, oven_temp+0
```

```
    mov y+1, oven_temp+1

    pop psw
    pop acc
    ret


load_cooling_temp:
    push acc
    push psw
    clr mf
    clr a

    mov x+0,a
    mov x+1,a
    mov x+2,a
    mov x+3,a

    mov y+0,a
    mov y+1,a
    mov y+2,a
    mov y+3,a

    mov x+0, #0x50

    mov y+0, oven_temp+0
    mov y+1, oven_temp+1

    pop psw
    pop acc
    ret



;---------------------------------- FSM ---------------------------------

FSM1:
    mov a, FSM1_state                ; initializes which state to start
from
```

```
FSM1_state0:                            ; start or not and edit settings
    cjne a, #0, FSM1_state1             ; if state is not 0, then move to
the next
    ljmp FSM_edit
    mov pwm, #0                         ; pulse width modulation set to 0
    mov secs_ctr, #0


FSM1_state0_done:
    WriteCommand(#0x01)
    Wait_Milli_Seconds(#2)
    mov FSM1_state, #1                  ; change the state to the next
    mov secs_ctr, #0
    mov mins_ctr, #0
    mov buzzer, #1

    ljmp forever


check_abort:
    lcall load_abort_temp       ;if temp<50 after a min, abort.
    lcall x_gt_y
    jb mf, abort_process
    ret


abort_process:
    mov pwm, #0
    mov FSM1_state, #0
    ljmp forever


FSM1_state1:                            ; ramp to soak
    cjne a, #1, FSM1_state2
    Set_Cursor(1, 1)
    Send_Constant_String(#title1)
    Set_Cursor(1, 13)
```

```
    Display_BCD(stemp+1)
    Display_BCD(stemp+0)
    mov pwm, #100                   ; set power to full
    mov a, #0x00
    cjne a, mins_ctr, check_abort
    lcall load_soak_temp            ;soak temp in x , oven in y
    lcall x_gteq_y
    jb mf, FSM1_state1_done

    mov FSM1_state, #2
    WriteCommand(#0x01)             ; clear thje tiomer area
    clr soak_temp_flag
    setb soak_timer_flag            ; start the timer for soak
    mov state_secs_ctr, #0          ;set the timer for states
;--------------------------------abort subroutines---------------------



FSM1_state1_done:
    mov buzzer, #2
    ljmp forever


FSM1_state2: ;soak
    cjne a, #2, FSM1_state3
    Set_Cursor(1, 1)
    Send_Constant_String(#title2)
    Set_Cursor(1, 13)
    Display_BCD(state_secs_ctr)
    mov pwm, #20
    mov a, state_secs_ctr
    clr c
    subb a, stime
    jc FSM1_state2_done

    mov FSM1_state, #3
    clr soak_timer_flag
    WriteCommand(#0x01)         ; clear thje tiomer area
```

```
FSM1_state2_done:

    ljmp forever


FSM1_state3:                        ;ramp to reflow
    cjne a, #3, FSM1_state4
    Set_Cursor(1, 1)
    Send_Constant_String(#title3)
    Set_Cursor(1, 13)
    Display_BCD(rtemp+1)
    Display_BCD(rtemp+0)
    mov pwm, #100                   ;ramp to temp
    lcall load_reflow_temp          ;soak temp in x , oven in y
    lcall x_gteq_y
    jb mf, FSM1_state3_done

    mov FSM1_state, #4
    mov state_secs_ctr, #0          ;resetting the state_sec_ctr


FSM1_state3_done:
    mov buzzer, #4
    ljmp forever


FSM1_state4:                        ;reflow
    cjne a, #4, FSM1_state5
    Set_Cursor(1, 1)
    Send_Constant_String(#title4)
    Set_Cursor(1,13)
    Display_BCD(state_secs_ctr)
    mov pwm, #20                    ;keep temp constant

    mov a, state_secs_ctr
    clr c
    subb a, rtime
    jc FSM1_state4_done
```

```
    mov FSM1_state, #5



FSM1_state4_done:
    mov buzzer, #5
    ljmp forever


FSM1_state5:                            ;cooling
    cjne a, #5, FSM1_state0_buffer
    Set_Cursor(1, 1)
    Send_Constant_String(#title5)
    mov pwm, #0
    lcall load_cooling_temp             ;50 in x, oven temp in y
    lcall x_gteq_y
    jnb mf, FSM1_state5_done

    mov FSM1_state, #0



FSM1_state5_done:
    mov buzzer, #6
    ljmp forever


FSM1_state0_buffer: ;buffer to long jump to state0

    ljmp FSM1_state0

;---------------------------------------------------------------------

;---------------------FSM_edit--------------------

FSM_edit:
    mov a, edit_state
```

```
FSM_soaktemp:
    cjne a, #0, FSM_soaktime
    ljmp soaktemp



FSM_soaktemp_done:
    mov edit_state, #1
    ljmp FSM_edit


FSM_soaktime:
    cjne a, #1, FSM_reflowtemp
    call soaktime




FSM_soaktime_done:
    mov edit_state, #2
    ljmp FSM_edit


FSM_reflowtemp:
    cjne a, #2, FSM_reflowtime
    ljmp reflowtemp




FSM_reflowtemp_done:
    mov edit_state, #3
    ljmp FSM_edit


FSM_reflowtime:
    cjne a, #3, FSM_start
    ljmp reflowtime
```

```
FSM_reflowtime_done:
    mov edit_state, #4
    ljmp FSM_edit


FSM_start:                      ;can only start once all parameters are
selected
    cjne a, #4, FSM_soaktemp
    lcall LCD_PB
    Wait_Milli_Seconds(#75)
    Set_Cursor(1, 1)            ;messages for starting or editing
    Send_Constant_String(#startmsg)
    Set_Cursor(2, 1)            ;messages for starting or editing
    Send_Constant_String(#editmsg)
    mov c, start_stop           ;start the process
    jnc FSM1_state0_done_buffer
    mov c, edit                 ;keep on editing
    jnc FSM_start_done
    ljmp FSM_start



FSM1_state0_done_buffer:
    ljmp FSM1_state0_done


FSM_start_done:
    mov edit_state, #0          ;clear lcd
    WriteCommand(#0x01)
    Wait_Milli_Seconds(#2)
    ljmp FSM_edit

;--------------edit fsm state functions----------------

; Assumes A register contains the value to be displayed before calling
```

```
; This routine converts the binary value in A to ASCII and displays it
on the LCD


soaktemp:
    lcall LCD_PB            ;evaluate the buttons
    Wait_Milli_Seconds(#75) ;need to wait so dont need lightning button
presses
    lcall stemp_change


Display_soaktemp:
    Set_Cursor(1, 1)                    ;displaying the values.
    Send_Constant_String(#param1)
    Set_Cursor(2, 1)
    Display_BCD(stemp+1)
    Display_BCD(stemp+0)

    mov c, edit             ;since is zero when pushed, if carry is on
then replay
    jc soaktemp
    ljmp FSM_soaktemp_done


stemp_change:

    mov c, incr             ;if increment
    jnc add_stemp
    mov c, decr             ;then check if decrement
    jnc sub_stemp
    ret


add_stemp:
    clr c
    mov a, stemp+0
    addc a, #0x30
    jc Display_soaktemp
    mov a, stemp+0              ;need to re add since changed value
```

```
    add a, #0x01
    da a
    mov stemp+0, a
    ret


sub_stemp:
    clr c
    mov a, stemp+0
    subb a, #0x31
    jc Display_soaktemp
    mov a, stemp+0
    add a, #0x99
    da a
    mov stemp+0, a
    ret


soaktime:
    lcall LCD_PB            ;evaluate the buttons
    Wait_Milli_Seconds(#75)
    lcall stime_change


Display_soaktime:
    Set_Cursor(1, 1)                    ;displaying the values.
    Send_Constant_String(#param2)
    Set_Cursor(2, 1)
    Display_BCD(#0)
    Display_BCD(stime)

    mov c, edit             ;since is zero when pushed, if carry is on
then replay
    jc soaktime
    ljmp FSM_soaktime_done


stime_change:
    mov c, incr             ;if increment
```

```
    jnc add_stime
    mov c, decr             ;then check if decrement
    jnc sub_stime
    ret


add_stime:
    mov a, stime+0              ;need to re add since changed value
    add a, #0x01
    da a
    mov stime+0, a
    inc stime_bcd
    ret


sub_stime:
    mov a, stime+0
    add a, #0x99
    da a
    mov stime+0, a
    dec stime_bcd
    ret



reflowtemp:
    lcall LCD_PB            ;evaluate the buttons
    Wait_Milli_Seconds(#75)
    lcall rtemp_change


Display_reflowtemp:
    Set_Cursor(1, 1)                    ;displaying the values.
    Send_Constant_String(#param3)
    Set_Cursor(2, 1)
    Display_BCD(rtemp+1)
    Display_BCD(rtemp+0)

    mov c, edit             ;since is zero when pushed, if carry is on
```

```
then replay
    jc reflowtemp
    ljmp FSM_reflowtemp_done


rtemp_change:
    mov c, incr             ;if increment
    jnc add_rtemp
    mov c, decr             ;then check if decrement
    jnc sub_rtemp
    ret


add_rtemp:
    mov a, rtemp+0              ;need to re add since changed value
    add a, #0x01
    da a
    mov rtemp+0, a
    ret


sub_rtemp:
    mov a, rtemp+0
    add a, #0x99
    da a
    mov rtemp+0, a
    ret


reflowtime:
    lcall LCD_PB            ;evaluate the buttons
    Wait_Milli_Seconds(#75)
    lcall rtime_change


Display_reflowtime:
    Set_Cursor(1, 1)                    ;displaying the values.
    Send_Constant_String(#param4)
    Set_Cursor(2, 1)
```

```
    Display_BCD(#0x00)
    Display_BCD(rtime)

    mov c, edit             ;since is zero when pushed, if carry is on
then replay
    jc reflowtime
    ljmp FSM_reflowtime_done


rtime_change:
    mov c, incr             ;if increment
    jnc add_rtime
    mov c, decr             ;then check if decrement
    jnc sub_rtime
    ret


add_rtime:
    mov a, rtime+0              ;need to re add since changed value
    add a, #0x01
    da a
    mov rtime+0, a
    inc rtime_bcd
    ret


sub_rtime:
    mov a, rtime+0
    add a, #0x99
    da a
    mov rtime+0, a
    inc rtime_bcd
    ret
;---------------------------------------------------------

END
```