University of British Columbia

Department of Electrical and Computer Engineering

ELEC 291/292 L2A 2023 W2: Electrical Engineering Design Studio I

Dr. Jesus Calvino-Fraga P.Eng

Project 1 – Reflow Oven Controller

Group Number: A2

Name	Student Number	Score %	Signature
Jeffrey He	32613788	105	JH
Yuqian Song	90293630	110	YS
Sidharth Sudhir	11255635	80	SS
Ao Sun	25909979	105	AS
Harry Zhang	13013727	100	HZ
Sam Zhao	30678635	100	SZ

Date of Submission: March 1st, 202

Table of Content

8. Appendices

1. Introduction

Our project aimed to design and implement a reflow oven controller, utilizing the N76E003 microcontroller and programming it in 8051 assembly language. Our objectives were multi-fold: to accurately control the power output to a Solid-State Relay (SSR) depending on the reflow process phase, to gather and display precise temperature readings via our microcontroller, and to generate audio alerts corresponding to different temperature thresholds.

Our team embarked on this challenge by first laying out the microcontroller's pin configurations and constructing a functional circuit. We then focused on achieving precise temperature measurements from the thermocouple through the careful selection and configuration of operational amplifiers and resistors. Following this, our efforts shifted towards programming, where we developed timers, non-blocking Finite State Machines (FSMs) for managing the LCD and the speaker output and Pulse Width Modulation (PWM) signals for effective power control. The culmination of our project saw the integration of WAV audio feedback into our assembly language code, enhancing the user interface and experience. Below are the detailed specifications that guided our project development.

2. Investigation

2.1 Idea Generation

Our team initiated the brainstorming process by closely examining the project's design requirements. We recognized that for our soldering oven controller to be effective, it was essential to establish adjustable settings such as: reflow and soak times, along with reflow and soak temperatures, and to control the voltage output to meet these specifications accurately. With these considerations in mind, we mapped out various logical frameworks for the designs, evaluating the transitions between different states of the soldering cycle and the integration of hardware and software components. This strategic analysis allowed us to select what we believed to be the most suitable design for achieving our objectives.

In investigating additional functions beyond design requirements, a comprehensive examination of user input and experience is crucial. We need to identify functionalities that could enhance the overall user experience. Key considerations might include streamlining input methods and implementing intuitive features that anticipate user needs. By prioritizing user convenience and engagement, the aim is to identify innovative functionalities that not only meet design requirements but also elevate the user experience to new heights.

2.2 Investigation Design

Our team will initiate the design process for the Reflow Oven Controller by conducting a comprehensive review of existing literature on controllers coded in assembly language to identify established best practices and common challenges. We will then gather and examine detailed specifications of critical components like the K-type thermocouple, solid-state relays, and the chosen microcontroller to understand

their capabilities and constraints. Additionally, we'll delve into industry standards for temperature measurement to ensure our design adheres to the necessary range of 25°C to 240°C. A meticulous analysis will be carried out to deconstruct each project requirement, thereby grasping the technicalities such as the necessity for cold junction compensation which is pivotal for precise temperature control. We plan to undertake a feasibility study to confirm if the available technology can fulfill our project's stringent requirements, including the crucial ±3°C temperature control precision. Concurrently, we will perform a risk assessment to identify and address potential safety hazards, such as fire risks associated with the thermocouple, to ensure a safe and reliable design.

2.3 Data Collection

1. We employed a multimeter to verify the integrity of the circuit connections. This was a fundamental step in our procedure, which we conducted routinely before and after each modification to the circuit to ensure there were no open circuits or unintended short circuits.
2. An oscilloscope was critical for visualizing the Pulse Width Modulation (PWM) signals. This allowed us to observe the duty cycle and frequency of the signals controlling the solid-state relay, ensuring that the power delivered to the heater was modulated accurately according to the temperature control algorithm.
3. We used PuTTy, a terminal emulator, to interface with the microcontroller for direct communication. This was particularly useful for testing the thermocouple-to-temperature conversion routines, enabling us to monitor the output of the assembly language program in real time and verify its accuracy.

4. To expedite testing between different states of the reflow process, such as ramp-up and cool-down phases, we used a lighter. This provided a quick method to artificially and safely increase the temperature at the thermocouple, allowing us to test the system's response without having to wait for the oven to heat up and cool down naturally.

2.4 Data Synthesis

1. We utilized a working K-type thermocouple connected to a pocket multimeter to serve as a temperature reference. By comparing the readings from the pocket multimeter with the data from our Reflow Oven Controller, we could validate the accuracy of our temperature measurements.
2. During the testing phases, we collected data logs from the serial output of the microcontroller. This data was analyzed to assess the performance of the temperature control system, identifying any discrepancies or lag in the temperature regulation.
3. The collected PWM signal data from the oscilloscope was used to refine our control algorithms. We analyzed the signal characteristics and adjusted the control parameters to achieve a more stable and responsive control system.
4. By comparing the thermocouple voltage readings obtained via PuTTy with standard thermocouple conversion tables, we verified the accuracy of our temperature conversion routines implemented in assembly language.

2.5 Analysis of Results

Our group's appraisal of the validity of conclusions drawn from our results involved a multifaceted approach, focusing on the degrees of error and the limitations of our theoretical understanding and measurement capabilities.

The Python code used to interpret thermocouple voltage readings and automatically create a temperature validation table was scrutinized for accuracy.

We verified the code against known values and checked for potential bugs that could affect the conversion accuracy. We compared the output of our Python code with standard thermocouple reference tables to ensure that the conversion from voltage to temperature was within acceptable error margins. The code was also peer-reviewed to validate the algorithms used for conversion.

3. Design

3.1 Use of Process

First, we adopted an iterative method to enhance our design through prototyping, testing, and analysis cycles. After each iteration, we examined the results and made some required modifications, which would improve the system's efficiency and reliability. Second, we divided the system into four modular components: temperature sensors, power control, user interface, and safety mechanisms. This enabled simultaneous development and testing, which not only saved time but also assisted in isolating and resolving issues inside particular modules without impacting the whole system.

To access design systems and components, we used standardized components including the K-type thermocouple and SSRs. This ensures compatibility of those parts and ease of replacement, as well as access to abundant community knowledge and troubleshooting advice. To address open-ended problems, we had brainstorming sessions that resulted in creative coding solutions, involving creating new subroutines for repeated operations, optimizing the code for performance, and ensuring optimal memory utilization. Feedback systems and a user-friendly interface were also incorporated since we recognized the importance of controller usability. This was

accomplished by creating an LCD system with real-time feedback and a pushbutton mechanism for simple reflow operation. Furthermore, we included strong error detection and automated cycle termination features to mitigate the risks associated with poor temperature control. These not only functioned as a safeguard against potential overheating problems but also inspired confidence in the oven's ability to do diverse PCB soldering jobs. Finally, we applied rigorous validation and verification techniques to confirm that our design satisfied the requirements. This consisted of comparing our system's readings to lab-standard multimeters and running stress tests to check the controller's endurance and dependability under long-term operation.

3.2 Need and Constraint Identification

The initial phase of our project involved a detailed analysis of the requirements necessary for ensuring both the functionality of the designed system and adherence to academic assessment standards. The project mandated the development of the program in 8051 assembly language, emphasizing the critical need for proficiency in low-level programming and microcontroller manipulation. Key requirements include:

- Programming Language: Utilization of 8051 assembly language for the entire project development, underscoring the importance of direct hardware control and efficiency in programming.
- User Interface Requirements: Capability for users to configure soak and reflow temperatures and durations via push buttons, with immediate feedback provided through an LCD.

- Control Mechanism: The microcontroller's ability to modulate the toaster oven's power through a solid-state relay (SSR) box, illustrates the necessity for precise thermal management during the reflow soldering process.
- Feedback and Monitoring: Continuous display of the oven’s temperature, the elapsed time, and the current state of the reflow process on the LCD, complemented by auditory temperature notifications via a speaker.
- Safety Features: Integration of an abort mechanism to halt the reflow process if the predefined conditions are not met within a specific timeframe, ensuring the safety of both the system and the user.

3.3 Problem Specification

Unique challenges presented along with the project specification listed, our teams encounter challenges such as:

- Correct operation of the reflow oven: The process of the oven to reflow at the correct temperature and time is controlled using Finite-State-Machine(FSM) in the assembling. The logic flow and requirements in triggering the entrance of each state need to be exact to achieve the ideal functionality.
- Temperature Control: A precise temperature increase rate of 1 °C/s at full power, demanding accurate control mechanisms to maintain this rate.
- Component Specifications: The operational amplifier (opamp) used must exhibit minimal offset voltage to ensure the integrity of temperature readings.
- Concurrent Processing: The need for implementing the LCD and speaker outputs through two independent, non-blocking finite state machines (FSMs) to prevent delays in display updates due to speaker output processing.

3.4 Solution Generation

The solution our team provided to meet the project requirements includes FSM to control the power output and user interface(UI). The power output FSM operates automatically based on input of real-time temperature of the oven and running time which was generated by the timer 2 ISR. 4 pushbuttons are connected to the N76E003 to toggle the reflow temperature, reflow time, soak temperature, and soak time. We also set a start and a stop pushbutton to start the whole program to stop the reflow process and turn off the toaster oven at any moment of the reflow process. The LCD screen shows the information on the reflow profile parameters. After the start button is pushed, the LCD then shows information about the real-time temperature in the oven and program running time.

3.5 Solution Evaluation

Our solution was rigorously evaluated against the project's criteria, successfully achieving:

- Non-Blocking FSM Design: The utilization of two non-blocking FSMs ensured uninterrupted LCD updates, regardless of speaker output operations.
- Precision in Temperature Readings: The system’s temperature conversion mechanism delivered accurate readings from the thermocouple, which is critical for the success of the reflow soldering process.

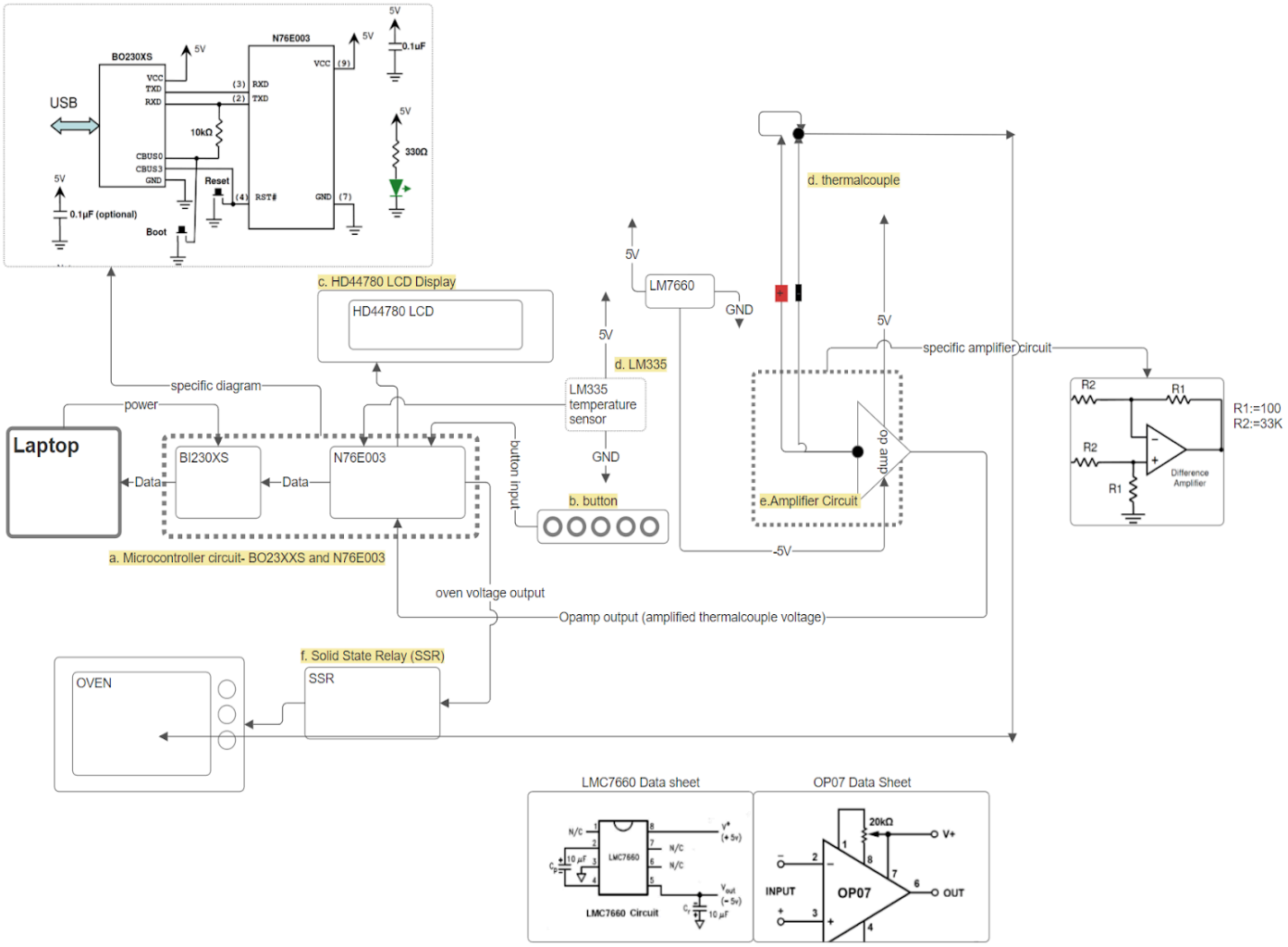
- Accurate Auditory Feedback: The speaker output was designed to accurately reflect the temperature to the nearest degree, ensuring clear and precise feedback.
- Effective Timing and Memory Management: The introduction of separate variables for timing and strategic use of long jump instructions mitigated potential issues related to timing inaccuracies and memory overflow.

3.6 Safety/Professionalism

Safety and professionalism practices are critical to not just the success of our project, but also ensuring the safety standards of our group members. Throughout the project, our team dedicated itself to upholding strict safety protocols and maintaining a high level of professional conduct. This commitment was evident in our proactive measures to mitigate risks, which included the implementation of comprehensive safety measures such as the utilization of protective gear and the adoption of safe handling practices for all electrical equipment. Additionally, we prioritized maintaining a respectful and constructive communication environment within our team. This was particularly important during discussions about the equitable distribution of grade points, where our commitment to professionalism and collaborative success was reflected. Our approach ensured not only the technical success of the project but also fostered a safe and professional working environment for all team members.

3.7 Detailed Design

3.7.1 Hardware Block Diagram



3.7.2 Circuit Explained

a. Microcontroller circuit- BO23XXS and N76E003

The microcontroller circuit is the central processing unit of this system. It interfaces with various peripherals to control the oven's temperature. It receives voltage from temperature sensors and the amplified voltage from the thermocouple then uses calculations(discussed in detail in the next section) to calculate the current temperature. The system is also designed to make autonomous decisions regarding the activation and deactivation of the oven's heating element to maintain the set temperature, facilitated by a meticulously designed Finite State Machine (FSM). The

principles of sending power to the Oven through SSR are discussed in the Solid State Relay (SSR) section. The microcontroller is connected to a serial port, which can be programmed or can communicate with a laptop. The microcontroller also consistently sends temperature data to the laptop through the serial port.

b. Buttons

Between the 5 LCD Push button and 8 ACD button, we choose to use the 5 LCD button. This is chosen based on our design of the user interface, which we found is sufficient to support our Oven Controller operations. The details of the design process can be found in section (?).

c. HD44780 LCD Display

This is a standard display used to show information such as the current temperature, target temperature, or system status. It is responsible for the majority of user interfaces. The microcontroller sends commands data and information, and the LCD will displays that information.

d. Temperature Sensors - LM335 and Thermocouple

The LM335 sensor is a precision temperature sensor with an analog output proportional to temperature. (used in Lab 3) The thermocouple is a type of temperature sensor used for measuring a wide range of temperatures and is great for high temperatures. The tip of the thermocouple will the placed in the oven to measure the temperature and the other will be connected to the circuit. However, the thermocouple's signal is very small, so it is sent to a specific amplifier circuit to boost the signal to a more usable level. The voltage can be then calculated through our program calculations.

e. Amplifier Circuit

The Amplifier Circuit is built to amplify the thermocouple voltage to a more usable level, it is built with resistors, capacitors, a voltage converter, and an op-amp. The gain of the Amplifier circuit is mainly determined by the ratio between resistor 1(R1) and resistor 2(R2),. In this project, through testing and data analysis, we found out that when R1= 100 ohms and R2 = 33000 ohms, this circuit works the best. This ratio between R1 and R2 should give us about 330 of gain, however, due to the error of the resistance, our actual gain is around 319. We then use this value to calculate our temperature, which will be discussed in detail in this section.

–operational amplifier: OP07

OP07 is a high-precision op-amp. It is used to amplify the small voltage generated by the thermocouple so that the microcontroller can read it more accurately. We connect it with the thermocouple through the positive and negative input. In addition, the OP07 requires a positive and a negative voltage to work properly. So we need to add a voltage converter to supply the negative 5 volts of voltage.

–Voltage Converter LMC7660

The LMC7660 is used to generate a negative voltage from a positive power supply. This negative voltage is required by the operational amplifier that requires dual supplies (+V and -V) to function correctly.

f. Solid State Relay (SSR)

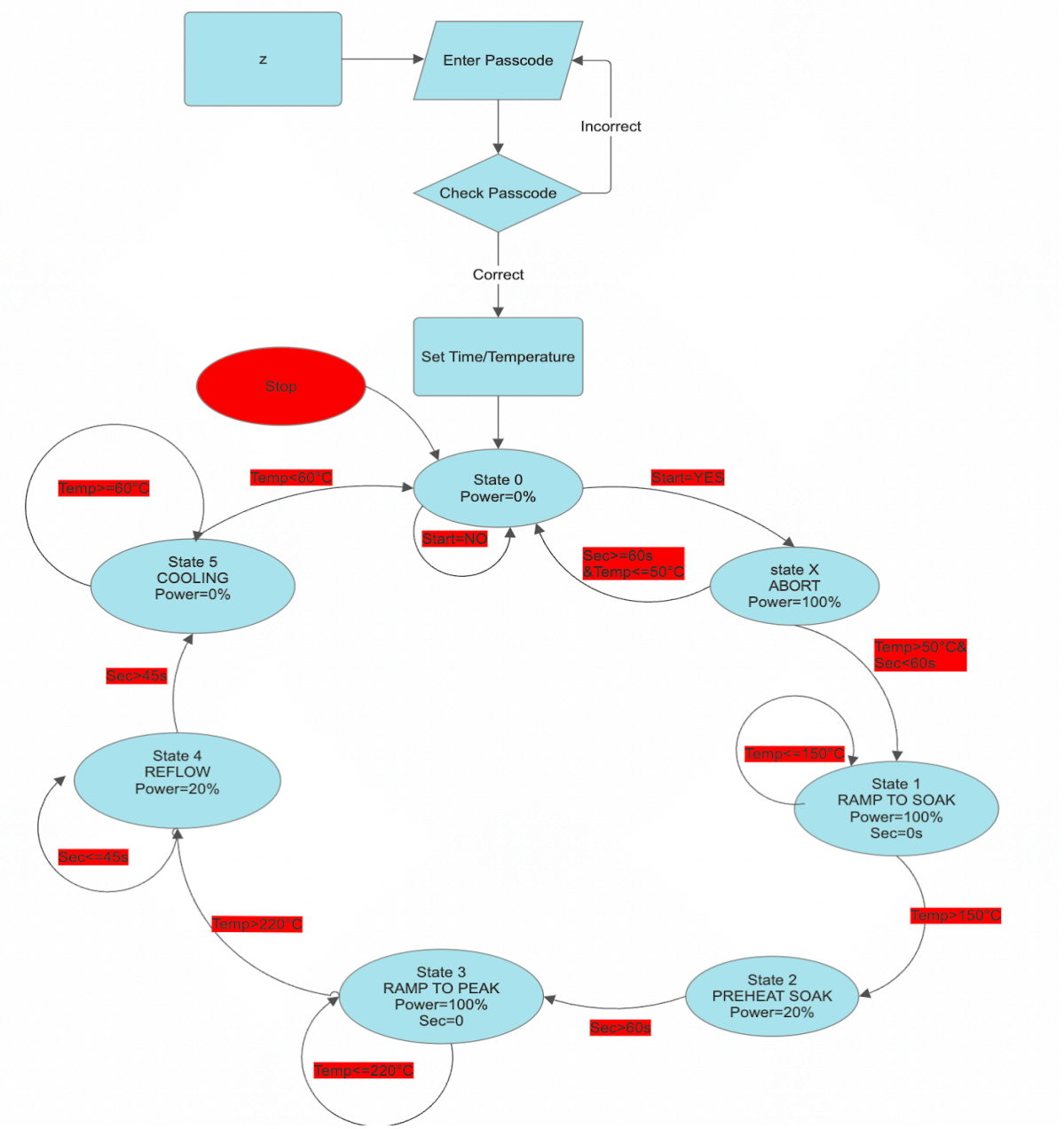
The SSR is an electronic switching device that turns the oven on and off, which is controlled by the microcontroller. The N76E003 will provide a square wave, which will determine how long the oven will be turned on or off. For example, when we enter 100 percent in the power output, for a period of 1 second, the oven will be on for one second. However, if we enter 80 percent, for a period of 1 second, the oven

will only be on for 0.8 seconds, and off for 0.2 seconds. By doing so, we can control the total power sent to the oven, therefore allowing us to control the temperature of the oven.

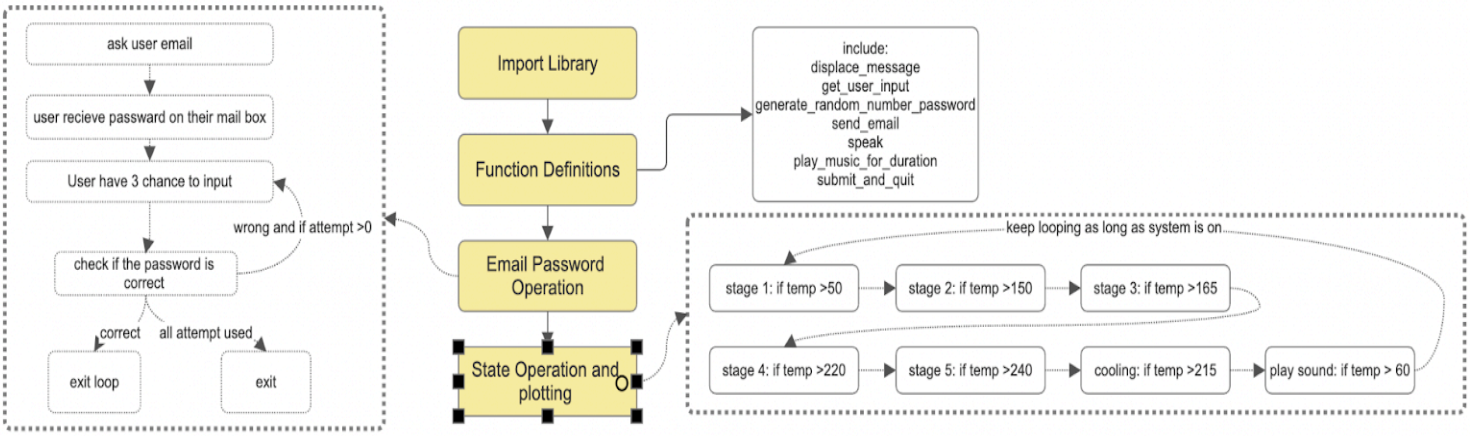
3.7.3 Software block diagram

3.7.3.1 ASM Block Diagram

Figure 1



3.7.3.2 Python Block Diagram



3.7.4 Software Explained

3.7.4.1 ASM Software

a. Timers

Our team uses timer 2 to count every second, update temperature, control PWM output, safety check, and communicate with the serial port. The timer 0 is used to generate a time delay.

b. passcode validation

For advanced user safety, we designed a passcode validation program. One has to enter the correct passcode shown on the LCD screen using 3 push buttons to enter the time and temperature setting interface.

c. safety check

Safety is our highest priority concern. After unlocking the program and starting the reflow process, the program enters the safety check program. The reflow process is aborted and jumps back to state 0(Figure 1) if the oven doesn't reach at least 50°C in the first 60 seconds of operation.

d. Temperature

Our program reads the analog input from channel 7 of the ADC which is assigned to P1.1 and connected pin to 14. We use VCC as a voltage reference and then convert the read value to voltage. Finally, we convert the voltage into temperature in the form of BCD through the ratio of two resistors composed of an operational amplifier.

$T: = \frac{10^{-6}*41*R1}{R2}$ °C R1=33kΩ, R2=100Ω

e. state control

The state control is implemented by a finite state machine shown in Figure 1.

f. oven system

We use a Pulse Width Modulation signal transferred to a solid state relay (SSR) to regulate the amount of power delivered by the oven. In the ASM program, we use timer 2 ISR to generate the pulse width signals.

3.7.4.2 Python Software (source code)

a. Import Library

In our Python program, we import various libraries and modules, each serving a specific purpose in a potentially complex application that involves data visualization, serial communication, text-to-speech conversion, audio playback, graphical user interface (GUI) creation, and security and email functionality. Here's a breakdown of the imports and their typical uses:

matplotlib.pyplot & matplotlib.animation: These are modules from the matplotlib library used for creating static, interactive, and animated visualizations in Python.

NumPy: A fundamental package for scientific computing in Python, used in the project for math operations.

serial: This module is used for communication with the serial port to connect to our oven controller device.

pyttsx3: A text-to-speech conversion library in Python that works offline. It is used to broadcast the state in our program.

pygame: provides functionality for playing music when the reflow process is ended.

string: used to generate out random password.

smtplib: These are used for sending out emails that contain passcode.

sys: Used “sys.exit()” for terminating the Python application when the temperature is too high or too low.

b. Function Definitions

We have written and created multiple functions to help with the operation of the Python program. In addition to the function provided, we also added a function to generate random number passwords, as well as multiple functions that get the user input through an additional tkinter window. Moreover, we also add a function for state broadcast and a function for play sound.

c. Email Password Operation

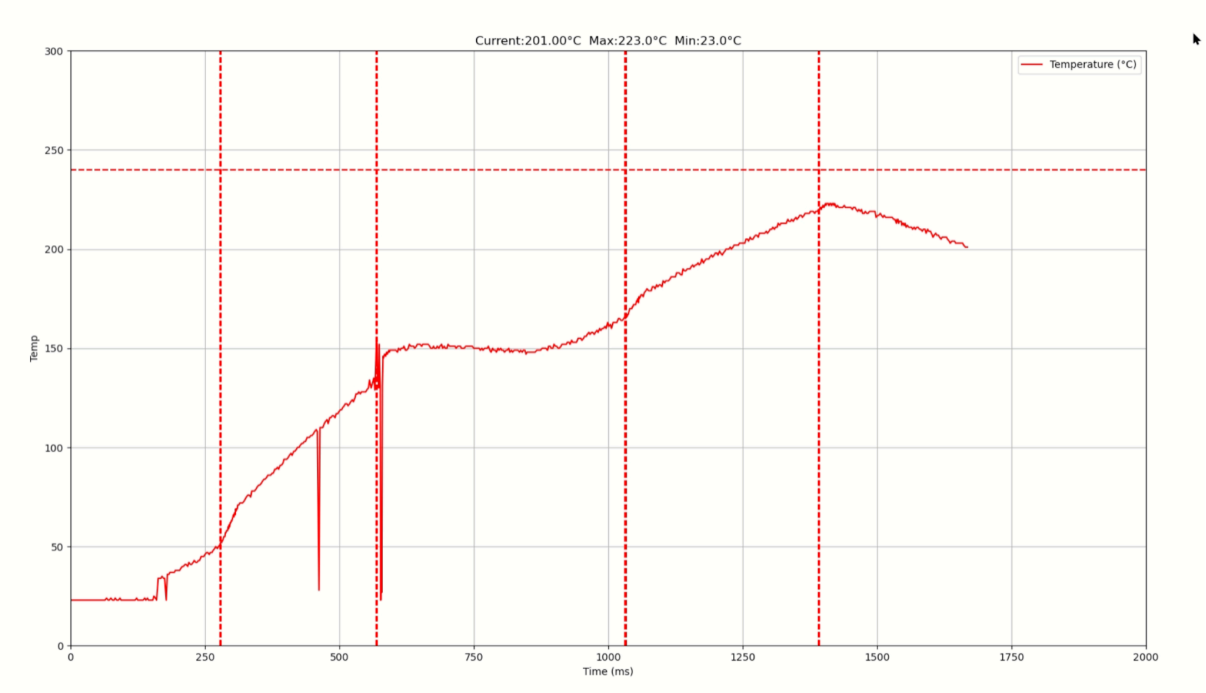
The system generates a random number password through a function we defined previously. We then prompted them to enter the email address where they wished to receive the password. Next, our function will send an email to the entered email with the randomly generated password. Users need to enter this password correctly to proceed. The user is given three chances to enter the correct password. If the user enters the correct password, a success message is displayed using displace_message("password correct :)"), and the loop is exited. If all attempts have been used, it will exit by calling sys.exit().

d. State Operation and plotting

Every time Python reads a new temperature value, it will run through the if statements to check if the condition is matched for that stage. If matched, it will

broadcast the current state two times. In addition, it draws a vertical line exactly when the state changes, which can be seen in section 3.8. Furthermore, it will draw a horizontal red line on temperature(y-axis) = 240, when our temperature goes over this line, it will ask the user to turn off the oven for safety. In the end, when the temperature cools down to 60, it will play a selected music for 20 seconds.

3.8 Solution Assessment (include data/plots)



4. Life-Long Learning

In the early stages of ELEC 291, our team embarked on a journey through the basics of LCD technology, alarm systems, timing mechanisms, and the art of temperature monitoring. This initial voyage laid the foundation for our ambitious endeavor to construct a reflow oven controller, seamlessly merging these components. Our exploration didn't stop there; we ventured further into the realm of reflow soldering techniques and the intricate world of nonblocking modules. These advanced topics were vital for our project, especially in managing multiple finite-state machines in harmony.

The knowledge we previously acquired in CPEN 211 emerged as a beacon of guidance for this project. The course endowed us with a deep understanding of finite state machines and the nuances of assembly language programming, which became our project's backbone, allowing us to navigate challenges with confidence.

ELEC 201, another milestone in our academic journey, significantly bolstered our circuit design prowess. It was here that we delved into the complexities of constructing breadboard circuits and harnessing the power of operational amplifiers. This course not only enriched our technical arsenal but also empowered us to bring our circuit designs to life with precision and creativity, laying the electrical foundations of our project with unmatched expertise.

5. Conclusions

5.1 Design and functionality summary

Our team was able to build a Reflow Oven Controller that not only meets all the specified design requirements but also introduces additional features enhancing its functionality and user experience. Moreover, the integration of a serial port connection for temperature monitoring on a personal computer is a significant advancement, allowing for the live display of temperature data in the top left corner of the plot, the current state of the reflow process as the title. Additionally, the current state of the reflow process is not only displayed as the title but is also articulated through a speech, thanks to a Python program utilizing a text-to-speech library. This feature provides an auditory cue to users, keeping them informed about the process state without the need to constantly monitor the visual display.

5.2 Problems encountered

Despite the project's successes, we encountered many challenges as well. One of the primary issues encountered was coding the state machine for the controller. Another significant issue arose from integrating the state machine with the initialization of the controller in the 8051 assembly language. Some jumps became out of range, leading to bugs that compromised the controller's functionality. Addressing these issues required additional hours of debugging.

5.3 Time taken for completion of the project

In total, the project took approximately 48 hours of work. This time investment reflects not only the technical challenges overcome but also the learning and growth experienced by our team throughout the project.

6. References

-[1] J. Calvino-Fraga, Project 1 - Reflow Soldering Oven Controller, University of British Columbia, 2024

[2] J. Calvino-Fraga. ELEC 291/292. Class Lecture, “Project 1 - Reflow Oven Controller.” Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Feb. 02, 2024

[3] J. Calvino-Fraga. ELEC 291/292. Class Lecture, “Project 1 – EFM8 board, FSM, NVMEM, and tips.” Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Feb. 09, 2024

7. Bibliography

Silicon Labs. (n.d.). 8051 instruction set. Retrieved from:

https://www.silabs.com/documents/public/presentations/8051_Instruction_Set.pdf

Nuvoton Technology Corporation. (n.d.). N76E003 Datasheet (Rev 1.08). Retrieved from

https://www.nuvoton.com/export/resource-files/DS_N76E003_EN_Rev1.08.pdf

8. Appendices

8.1 ARM Source Code

```
      ; N76E003 ADC test program: Reads channel 7 on P1.1, pin
14(Cold Junction)
      ; Reads channel 1 on P3.0, Pin3.0(hot junction)

      $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
;                               -------
;

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_RELOAD_1MS EQU (0x10000-(CLK/1000))
TIMER2_RATE       EQU 100      ; 100Hz or 10ms
TIMER2_RELOAD     EQU (65536-(CLK/(16*TIMER2_RATE))) ; Need to
change timer 2 input divide to 16 in T2MOD

; Output
PWM_OUT    EQU P1.0 ; Logic 1=oven on

BSEG
s_flag: dbit 1 ; set to 1 every time a second has passed

mf: dbit 1 ; £¿£¿£¿
beep: dbit 1
```

```
	; These five bit variables store the value of the pushbuttons
after calling 'LCD_PB' below
	PB0: dbit 1
	PB1: dbit 1
	PB2: dbit 1
	PB3: dbit 1
	PB4: dbit 1


	DSEG at 0x30
	pwm_counter:  ds 1 ; Free running counter 0, 1, 2, ..., 100, 0
	pwm:          ds 1 ; pwm percentage
	seconds:      ds 1 ; a seconds counter attached to Timer 2 ISR
	sec:          ds 1
	FSM1_state:   ds 1

	temp: ds 1

	temp_soak: ds 2

	time_soak: ds 2

	temp_refl: ds 2

	time_refl: ds 2

	temp_cool: ds 1

	safetemp: ds 1

	; These register definitions needed by 'math32.inc'
```

```
	x:   ds 4
	y:   ds 4
	bcd: ds 5


	cseg
	; These 'equ' must match the hardware wiring
	LCD_RS equ P1.3
	;LCD_RW equ PX.X ; Not used in this code, connect the pin to
GND
	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


	; Reset vector
	org 0x0000
	    ljmp main

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

	; Timer/Counter 0 overflow interrupt vector (not used in this
code)
	org 0x000B
		reti

	; 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



	$NOLIST
	$include(LCD_4bit.inc) ; A library of LCD related functions and
utility macros
	$include(math32.inc);A library of mathematic calculation of
variables
	$LIST



	Left_blank mac
		mov a, %0
		anl a, #0xf0
		swap a
		jz Left_blank_%M_a
```

```
            ljmp %1
      Left_blank_%M_a:
            Display_char(#' ')
            mov a, %0
            anl a, #0x0f
            jz Left_blank_%M_b
            ljmp %1
      Left_blank_%M_b:
            Display_char(#' ')
      endmac

      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

            ; 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 timer 2 for periodic interrupts
            mov T2CON, #0 ; Stop timer/counter.  Autoreload mode.
            mov TH2, #high(TIMER2_RELOAD)
            mov TL2, #low(TIMER2_RELOAD)
            ; Set the reload value
            mov T2MOD, #0b1010_0000 ; Enable timer 2 autoreload, and
clock divider is 16
            mov RCMP2H, #high(TIMER2_RELOAD)
            mov RCMP2L, #low(TIMER2_RELOAD)
            ; Init the free running 10 ms counter to zero
            mov pwm_counter, #0
            ; Enable the timer and interrupts
            orl EIE, #0x80 ; Enable timer 2 interrupt ET2=1
          setb TR2  ; Enable timer 2



            ; Initialize the pin used by the ADC (P1.1) as input.
            orl   P1M1, #0b00000010
            anl   P1M2, #0b11111101

            ; Initialize and start the ADC:
            anl ADCCON0, #0xF0
            orl ADCCON0, #0x01 ; Select channel 7
```

```
            ; AINDIDS select if some pins are analog inputs or
digital I/O:
            mov AINDIDS, #0x00 ; Disable all analog inputs
            orl AINDIDS, #0b00000001 ; P1.1 is analog input/Activate
AIN0 and AIN7 channel inputs
            orl ADCCON1, #0x01 ; Enable ADC

            setb EA ; Enable global interrupts
            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.
            push psw
            push acc

            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, exit_a
            sjmp exit1_b
```

```
exit_a:
    ljmp Timer2_ISR_done
exit1_b:
    mov pwm_counter, #0

      inc sec ; It is super easy to keep a seconds count here
    clr a
    mov a,seconds
    add a,#1
    da a
    mov seconds, a
    clr a
      setb s_flag

Timer2_ISR_done:
      pop acc
      pop psw
      reti

wait_1ms:
      clr   TR0 ; Stop timer 0
      clr   TF0 ; Clear overflow flag
      mov   TH0, #high(TIMER0_RELOAD_1MS)
      mov   TL0,#low(TIMER0_RELOAD_1MS)
      setb TR0
      jnb   TF0, $ ; Wait for overflow
      ret

; Wait the number of miliseconds in R2
waitms:
      lcall wait_1ms
```

```
        djnz R2, waitms
        ret

; function for pushbutton
LCD_PB:
        ; Set variables to 1: 'no push button pressed'
        setb PB0
        setb PB1
        setb PB2
        setb PB3
        setb PB4
        ; 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 PB4, c
	setb P1.3

	clr P0.0
	mov c, P1.5
	mov PB3, c
	setb P0.0

	clr P0.1
	mov c, P1.5
	mov PB2, c
	setb P0.1

	clr P0.2
	mov c, P1.5
	mov PB1, c
	setb P0.2

	clr P0.3
	mov c, P1.5
	mov PB0, c
	setb P0.3

LCD_PB_Done:
	ret
```

```
;---------------------------------;
; Send a BCD number to PuTTY????? ;
;---------------------------------;
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


; We can display a number any way we want.  In this case with
; four decimal places.
```

```
Display_formated_BCD:
      Set_Cursor(1, 4)
    Display_BCD(bcd+3)
      Display_BCD(bcd+2)
      ; Replace all the zeros to the left with blanks
      Set_Cursor(1, 4)

      Left_blank(bcd+3, skip_blank)
      Left_blank(bcd+2, skip_blank)
      Left_blank(bcd+1, skip_blank)
      Left_blank(bcd+0, skip_blank)
      mov a, bcd+0
      anl a, #0f0h
      swap a
      jnz skip_blank
      Display_char(#' ')
skip_blank:
      ret



putchar:
    jnb TI, putchar
    clr TI
    mov SBUF, a
    ret

Read_temp:
    ;receive temperature data
      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, ADCRH
swap a
push acc
anl a, #0x0f
mov R1, a
pop acc
anl a, #0xf0
orl a, ADCRL
mov R0, A

; Convert to voltage
  mov x+0, R0
  mov x+1, R1
  mov x+2, #0
  mov x+3, #0
  Load_y(50300) ; VCC voltage measured
  lcall mul32
  Load_y(4095) ; 2^12-1
  lcall div32

  Load_y(100000) ;
  lcall mul32

  Load_y(1353) ;
  lcall div32

  Load_y(230000) ;
  lcall add32
```

```
            lcall wait_1ms

              mov temp,x+0

              ; Convert to BCD and display
              lcall hex2bcd
              lcall Display_formated_BCD
              ;Send value to putty
              Send_BCD(bcd+3)
              Send_BCD(bcd+2)

              mov a, #'\r'
              lcall putchar

              mov a, #'\n'
              lcall putchar
              ; Wait 500 ms between conversions
              mov R2, #250
              lcall waitms
              mov R2, #250
              lcall waitms
              ret

      ;                     1234567890123456    <- This helps
determine the location of the counter
      soak_time_msg:    db 'St:****', 0
      soak_temp_msg:    db 'ST:****', 0

      reflow_time_msg:  db          'Rt:****', 0
      reflow_temp_msg:  db          'RT:****', 0
```

```
start_msg:        db 'STARTING...', 0
clear:            db '                ', 0
fsm_message:      db 'FSM', 0

test_message:     db 'To=    C  ', 0
value_message:    db '', 0
blank:            db '                ', 0
state0:           db 'set up',0
state1:           db 'State1',0
state2:           db 'State2',0
state3:           db 'State3',0
state4:           db 'State4',0
state5:           db 'State5',0
safety_measure:     db 'safety',0
complete:         db 'Reflow Complete', 0

main:

      mov sp, #0x7f
      lcall Init_All
    lcall LCD_4BIT


    ; initial messages in LCD
      Set_Cursor(1,1)
    Send_Constant_String(#soak_temp_msg)
    Set_Cursor(2,1)
    Send_Constant_String(#soak_time_msg)
      Set_Cursor(1,10)
    Send_Constant_String(#reflow_temp_msg)
    Set_cursor(2,10)
```

```
            Send_Constant_String(#reflow_time_msg)

              clr beep

              mov FSM1_state,#0
              ;mov pwm_counter,#0x00 ; Free running counter 0, 1, 2,
...., 100, 0
            mov pwm, #0         ; pwm percentage
            mov seconds, #0x00    ; a seconds counter attached to Timer
2 ISR
            ;mov sec,#0x00

            mov temp,#0x00

            mov temp_cool,#0x00
            mov a,temp_cool
            add a,#0x60
            da a
            mov temp_cool,a

              mov safetemp, #0x0
              mov a, safetemp
              add a, #0x50
              da a
              mov safetemp, a


            mov temp_soak+0,#0x00
            mov temp_soak+1,#0x00
            mov time_soak+0,#0x00
            mov time_soak+1,#0x00
```

```
            mov temp_refl+0,#0x00
            mov temp_refl+1,#0x00
            mov time_refl+0,#0x00
            mov time_refl+1,#0x00

        loop1:
            lcall LCD_PB
              jb PB4, loop2
              Wait_Milli_Seconds(#50) ; Debounce delay.  This macro is
also in 'LCD_4bit.inc'
              jb PB4, loop2

              clr a
              mov a, temp_soak+0
            add a, #1
            da a
            mov temp_soak+0,a
            cjne a, #0x99,exit1
            mov temp_soak+0, #0
            clr a
            mov a,temp_soak+1
            add a,#1
            da a
            mov temp_soak+1,a
            cjne a, #0x99,exit1
            mov temp_soak+1, #0
            sjmp loop2
        exit1:
            nop

        loop2:
```

```
          lcall LCD_PB
            jb PB3, loop3
            Wait_Milli_Seconds(#50) ; Debounce delay.  This macro is
also in 'LCD_4bit.inc'
            jb PB3, loop3

            clr a
            mov a, time_soak+0
          add a, #1
          da a
          mov time_soak+0,a
          cjne a, #0x99, exit2
          mov time_soak+0, #0
          clr a
          mov a,time_soak+1
          add a,#1
          da a
          mov time_soak+1,a
          cjne a, #0x99,exit2
          mov time_soak+1, #0
          sjmp loop3
      exit2:
          nop

      loop3:
          lcall LCD_PB
            jb PB2, loop4
            Wait_Milli_Seconds(#50) ; Debounce delay.  This macro is
also in 'LCD_4bit.inc'
            jb PB2, loop4
```

```
            clr a
            mov a, temp_refl+0
          add a, #1
          da a
          mov temp_refl+0,a
          cjne a, #0x99, exit3
          mov temp_refl+0, #0
          clr a
          mov a,temp_refl+1
          add a,#1
          da a
          mov temp_refl+1,a
          cjne a, #0x99,exit3
          mov temp_refl+1, #0
          sjmp loop4
      exit3:
          nop

      loop4:
          lcall LCD_PB
            jb PB1, loop5
            Wait_Milli_Seconds(#50) ; Debounce delay.  This macro is
also in 'LCD_4bit.inc'
            jb PB1, loop5

            clr a
            mov a, time_refl+0
          add a, #1
          da a
          mov time_refl+0,a
          cjne a, #0x99, exit4
```

```
            mov time_refl+0, #0x00
            clr a
            mov a,time_refl+1
            add a,#0x01
            da a
            mov time_refl+1,a
            cjne a, #0x99,exit4
            mov time_refl+1, #0
            sjmp loop5
      exit4:
            nop

      loop5:
            lcall LCD_PB

              jb PB0, loop_b ; if the 'CLEAR' button is not pressed
skip
              Wait_Milli_Seconds(#50) ; Debounce delay.  This macro is
also in 'LCD_4bit.inc'
              jb PB0, loop_b ; if the 'CLEAR' button is not pressed
skip
              ljmp fsm_start

      loop_b:

            Set_Cursor(1,4)
            Display_BCD(temp_soak+1)
            Set_Cursor(1,6)
            Display_BCD(temp_soak+0)
            Set_Cursor(2,4)
            Display_BCD(time_soak+1)
```

```
            Set_Cursor(2,6)     ; the place in the LCD where we want
the BCD counter value
            Display_BCD(time_soak+0) ; This macro is also in
'LCD_4bit.inc'
            Set_Cursor(1,13)     ; the place in the LCD where we want
the BCD counter value
            Display_BCD(temp_refl+1) ; This macro is also in
'LCD_4bit.inc'
            Set_Cursor(1,15)     ; the place in the LCD where we want
the BCD counter value
            Display_BCD(temp_refl+0) ; This macro is also in
'LCD_4bit.inc'
            Set_Cursor(2,13)     ; the place in the LCD where we want
the BCD counter value
            Display_BCD(time_refl+1) ; This macro is also in
'LCD_4bit.inc'
            Set_Cursor(2,15)     ; the place in the LCD where we want
the BCD counter value
            Display_BCD(time_refl+0) ; This macro is also in
'LCD_4bit.inc'
            ljmp loop1

      fsm_start:

            mov pwm, #0
            Set_Cursor(1, 1)
            Send_Constant_String(#clear)
            Set_Cursor(2, 1)
            Send_Constant_String(#clear)
            Wait_Milli_Seconds(#250)
            Wait_Milli_Seconds(#250)
```

```
      Wait_Milli_Seconds(#250)
      Wait_Milli_Seconds(#250)

      Set_Cursor(2, 1)
      Send_Constant_String(#start_msg)
      Wait_Milli_Seconds(#250)
      Wait_Milli_Seconds(#250)
      Wait_Milli_Seconds(#250)
      Wait_Milli_Seconds(#250)
      Wait_Milli_Seconds(#250)
      Wait_Milli_Seconds(#250)
      Wait_Milli_Seconds(#250)
      Wait_Milli_Seconds(#250)

      ; initial messages in LCD
      Set_Cursor(2, 1)
      Send_Constant_String(#clear)
      Wait_Milli_Seconds(#250)
      Wait_Milli_Seconds(#250)
      Set_Cursor(1, 1)
    Send_Constant_String(#test_message)
      Set_Cursor(2, 1)
    Send_Constant_String(#value_message)
    mov seconds, #0x00
      sjmp Forever



Forever:
    Set_Cursor(2,10)
      Display_BCD(seconds)
```

```
            lcall Read_temp
            sjmp FSM1

        FSM1:
            mov a, FSM1_state

        FSM1_state0:
            cjne a, #0, FSM1_safety
            Set_Cursor(2, 1)
            Send_Constant_String(#state0)
            mov pwm, #0
            mov seconds, #0
            lcall LCD_PB
            mov R2, #50
              lcall waitms
            jb PB0, FSM1_state0_done
            mov FSM1_state, #6

        FSM1_state0_done:
            ljmp FSM2

        FSM1_safety:
              cjne a, #6, FSM1_state1
            mov pwm, #100
            ;mov seconds, #0
              Set_Cursor(2,1)
              Send_Constant_String(#safety_measure)


              ;lcall soak_time_check ;check to see if the maximum time
for soaking has been reached
```

```
            ;jb abort, FSM1_state1 ;abort if abort bit set to 1

            mov a, temp_cool
            clr c
            subb a, seconds ;check if time has been exceeded
threshold
            jnc jump_temp

            mov a, safetemp
            clr c
            subb a, bcd+2
            jnc jump_state
            mov FSM1_state, #1
            ljmp safety_done

      jump_state:
            mov FSM1_state, #0
            ljmp safety_done
      jump_temp:
            mov a, safetemp
            clr c
            subb a, bcd+2
            jnc safety_done
            mov FSM1_state, #1
            ljmp safety_done

      safety_done:
            ljmp FSM2

      FSM1_state1:
          cjne a, #1, FSM1_state2
```

```
    Set_Cursor(2, 1)
    Send_Constant_String(#state1)
    mov pwm, #100
    mov seconds, #0
    mov sec, #0
    clr a
    clr c
    mov a,temp_soak+1
    mov R0,bcd+3
    subb a,R0
    jnc FSM1_state1_done
    clr a
    mov a, temp_soak+0
    mov R0,bcd+2
    clr c
    subb a, R0
    jnc FSM1_state1_done
    mov FSM1_state, #2
FSM1_state1_done:
    ljmp FSM2

FSM1_state2:
    cjne a, #2, FSM1_state3
    Set_Cursor(2, 1)
    Send_Constant_String(#state2)
    mov pwm, #20
    mov a, time_soak+0
    clr c
    subb a, seconds
    jnc FSM1_state2_done
    mov FSM1_state, #3
```

```
FSM1_state2_done:
    ljmp FSM2

FSM1_state3:

    cjne a, #3, FSM1_state4
    Set_Cursor(2, 1)
    Send_Constant_String(#state3)
    mov pwm, #100
    mov seconds, #0
    mov sec, #0
    clr a
    clr c
    mov a,temp_refl+1
    mov R0,bcd+3
    subb a,R0
    jnc FSM1_state3_done
    clr a
    mov a, temp_refl+0
    mov R0,bcd+2
    clr c
    subb a, R0
    jnc FSM1_state3_done
    mov FSM1_state,#4
FSM1_state3_done:
    ljmp FSM2

FSM1_state4:
    cjne a, #4, FSM1_state5
    Set_Cursor(2, 1)
    Send_Constant_String(#state4)
```

```
    mov pwm, #20
    mov a,time_refl+0
    clr c
    subb a,seconds
    jnc FSM1_state4_done
    mov FSM1_state,#5
FSM1_state4_done:
    ljmp FSM2

FSM1_state5:
    cjne a, #5, FSM1_state5
    Set_Cursor(2, 1)
    Send_Constant_String(#state5)
    mov pwm, #0

    clr a
    mov a,bcd+3
    cjne a,#0,FSM1_state5_done

    clr a
    mov a, temp_cool
    mov R0,bcd+2
    clr c
    subb a, R0
    jc FSM1_state5_done
    mov FSM1_state,#0
    mov seconds,#0
    mov sec, #0
;    mov temp,#0
    mov pwm,#0
FSM1_state5_done:
```

```
        ljmp FSM2



FSM2:


          lcall LCD_PB
          jb PB1, FSM1_state_done
        mov seconds,#0
        mov sec,#0
        mov pwm,#0
        mov FSM1_state, #0


FSM1_state_done:
          ljmp Forever


END
```

8.2 Python Source Code

```
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import matplotlib;
matplotlib.use("TkAgg")
import time
import serial
import  pyttsx3
import pygame
import tkinter as tk
from tkinter import simpledialog
from tkinter import messagebox
```

```
from tkinter import ttk
import secrets
import string
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import sys

def displace_message(message):

    root = tk.Tk()
    root.withdraw()

    messagebox.showinfo("Important notice", message)

    root.destroy()

def get_user_input(prompt):
    root = tk.Tk()
    root.withdraw()

    user_input = simpledialog.askstring("Input", prompt)

    return user_input

def generate_random_number_password(length=6):
    # Use only digits for the password
    digits = string.digits
    password = ''.join(secrets.choice(digits) for i in range(length))
    return password

def send_email(subject, message, to_email):
    from_email = "jeffrey.he0418@gmail.com"
    password = "jkvksukxtbeboddi"

    # Create the email message
    msg = MIMEMultipart()
    msg['From'] = from_email
    msg['To'] = to_email
    msg['Subject'] = subject
    body = MIMEText(message, 'plain')
    msg.attach(body)

    # Send the email
    with smtplib.SMTP('smtp.gmail.com', 587) as server:
        server.starttls()
        server.login(from_email, password)
        text = msg.as_string()
        server.sendmail(from_email, to_email, text)

def speak(text):
    engine = pyttsx3.init()  # Initialize the converter
    engine.setProperty('rate', 100)  # Speed percent (can go over 100)
    engine.setProperty('volume', 1)  # Volume 0-1

    engine.say(text)  # Add the text to the speech queue
    engine.runAndWait()

def play_music_for_duration(file_path, duration):
    pygame.mixer.init()  # Initialize the mixer module
    pygame.mixer.music.load(file_path)  # Load the music file
    pygame.mixer.music.play()  # Start playing the music

    time.sleep(duration)  # Wait for the duration (in seconds) you want the music to play
    pygame.mixer.music.stop()  # Stop the music after the duration has passed

def submit_and_quit():
    global speed_1_int, volume_1_int, repeat_1_int
    # Retrieve the inputs from the entry widgets
    speed_1 = speed.get()
```

```
    volume_1 = volume.get()
    repeat_1 = repeat.get()

    repeat_1_int = int(repeat_1.strip())
    volume_1_int = int(volume_1.strip())
    speed_1_int = int(repeat_1.strip())

    # Print the inputs for demonstration purposes
    print(f"Speed: {speed_1}, Volume: {volume_1}, Repeat: {repeat_1}")

    root.quit()
    root.destroy()

repeat_1_int =0
volume_1_int = 0
speed_1_int = 0



password = generate_random_number_password()

#get email
ask_email = "Please Enter The Email Where You Want To Get Your Password:"
email = get_user_input(ask_email)

#end email
send_email("Oven Controller Password", f"Your password is: {password}", email)

#check email
password_chance = 3
while password_chance >0:
    get_password = f"please enter the password sent to your email:\nyou have {password_chance} chances."
    email_password = get_user_input(get_password)
    if email_password == password:
        displace_message("password correct :)")
        break
    else:
        displace_message("password wrong :(")
        password_chance = password_chance - 1

if password_chance == 0:
    displace_message("all attempt used, please restart the program")
    sys.exit()

# Set up the serial port connection
ser = serial.Serial(
    port='COM6',  # Change to your port
    baudrate=115200,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_TWO,
    bytesize=serial.EIGHTBITS
)

# Initialize lists to store time and formatted_number values
time_data = []
formatted_numbers = []

fig, ax = plt.subplots()
line, = ax.plot([], [], 'r-', label='Temperature (°C)')
ax.set_xlabel('Time (ms)')
ax.set_ylabel('Temp')
ax.set_title('Temp')
ax.legend()
ax.grid(True)

a = 0
b = 0
c = 0
d = 0
e = 0
```

```
f = 0
g = 0
def init():
    ax.set_xlim(0, 2000)  # Initial x-axis limit
    ax.set_ylim(0, 300)  # Initial y-axis limit
    return line,

def update(frame, formatted_number=None):
    global a, b, c, d, e, temp_counter, f, g

    if ser.in_waiting:
        strin = ser.readline()
        decoded_strin = strin.decode('utf-8').strip()

        def draw_vertical_line():
            ax.axvline(x=frame, color='r', linestyle='--')

        try:
            if len(decoded_strin) == 4:
                formatted_str = decoded_strin[:2] + '.' + decoded_strin[2:]
                formatted_number = float(formatted_str)*100
                if formatted_number > 50 and a < 2:
                    speak("stage one: ramp to soak")
                    a += 1
                    draw_vertical_line()

                if formatted_number > 150 and b < 2:
                    speak("stage two: soaking")
                    b += 1
                    draw_vertical_line()

                if formatted_number > 165 and c < 2:
                    speak("stage three: ramp to peak")
                    c += 1
                    draw_vertical_line()

                if formatted_number > 220 and d < 2:
                    speak("stage four: reflow")
                    d += 1
                    temp_counter = 1
                    draw_vertical_line()
                    ax.axhline(y=240, color='r', linestyle='--')

                if formatted_number > 240:
                    speak("temperature too high")
                    displace_message("temperature too high, please terminate program")

                if formatted_number < 215 and d==2 and f < 2:
                    speak("cooling ")
                    f+=1

                if formatted_number < 40 and d==2 and g < 2:
                    speak("process is complete")
                    music_file = "C:/291/project1/祖海-好运来.ogg"
                    play_music_for_duration(music_file, 20)


                # Update the data lists
                time_data.append(frame)
                formatted_numbers.append(formatted_number)

                # Update plot data
                line.set_data(time_data, formatted_numbers)

                ax.set_title(f'Current:{formatted_number:.2f}°C  Max:{max(formatted_numbers)}°C  Min:{min(formatted_numbers)}°C')
                # Adjust limits
                if frame >= ax.get_xlim()[1]:
                    ax.set_xlim(0, ax.get_xlim()[1] + 10)  # Keep x-axis minimum fixed at 0

                if formatted_number >= ax.get_ylim()[1] or formatted_number <= ax.get_ylim()[0]:
```

```
                ax.set_ylim(min(formatted_numbers) - 10, max(formatted_numbers) + 10)

        except ValueError as e:
            print(f"Error processing input {decoded_strin}: {e}")

    return line,



ani = animation.FuncAnimation(fig, update, init_func=init, frames=np.arange(1, 2000), blit=False, interval=200)

plt.show()
```

8.3 Validation Data (20-degree celsius to 240 degree celsius)

```
T     Tt     D    (T: Temperature; Tt:Test Temp; Difference )

24.9 24.0 0.9

24.8 25.0 0.2

24.7 24.0 0.7

24.7 24.0 0.7

24.6 24.0 0.6

24.4 24.0 0.4

24.3 24.0 0.3

24.2 24.0 0.2

24.1 24.0 0.1

23.8 23.0 0.8

23.6 24.0 0.4

23.5 23.0 0.5

23.5 23.0 0.5

23.6 24.0 0.4

23.8 23.0 0.8

24.3 25.0 0.7

24.5 24.0 0.5

25.0 24.0 1.0

25.4 25.0 0.4

26.4 26.0 0.4

27.4 28.0 0.6

28.1 28.0 0.1
```

28.6 28.0 0.6

28.8 28.0 0.8

29.2 29.0 0.2

29.3 30.0 0.7

29.3 29.0 0.3

29.3 30.0 0.7

29.6 29.0 0.6

29.6 30.0 0.4

29.7 29.0 0.7

29.8 29.0 0.8

30.2 30.0 0.2

30.3 30.0 0.3

30.3 30.0 0.3

30.9 30.0 0.9

31.4 31.0 0.4

32.3 32.0 0.3

36.5 36.0 0.5

39.4 41.0 1.6

42.2 45.0 2.8

45.1 47.0 1.9

49.6 51.0 1.4

52.3 54.0 1.7

54.6 57.0 2.4

59.1 60.0 0.9

61.1 64.0 2.9

66.0 67.0 1.0

70.7 69.0 1.7

72.4 72.0 0.4

74.3 76.0 1.7

75.5 75.0 0.5

73.4 72.0 1.4

70.9 69.0 1.9

68.4 65.0 3.4

64.7 65.0 0.3

65.8 22.0 43.8

54.1 68.0 13.9

73.3 73.0 0.3

76.2 76.0 0.2

79.0 81.0 2.0

81.7 83.0 1.3

86.7 87.0 0.3

89.1 90.0 0.9

91.3 94.0 2.7

95.5 96.0 0.5

97.6 100.0 2.4

99.8 102.0 2.2

101.7 105.0 3.3

106.9 107.0 0.1

108.5 109.0 0.5

110.1 111.0 0.9

113.3 114.0 0.7

114.8 116.0 1.2

116.2 117.0 0.8

118.8 121.0 2.2

120.2 123.0 2.8

121.5 124.0 2.5

122.7 127.0 4.3

124.9 127.0 2.1

126.0 126.0 0.0

126.9 130.0 3.1

128.5 127.0 1.5