TABLE OF CONTENTS

1 Introduction

1.1 Design Objective

The project aimed to design, build, program, and test a Reflow Oven Controller using the N76EE03 microcontroller. The Reflow Soldering process is used to integrate surface mount devices (SMDs) onto printed circuit boards (PCBs) to help ease the soldering process.

1.2 Design Specification

Hardware Component Specifications:

General:

- N76EE03 Microcontroller
- USB Adapter: BO230X
- Various Resistors of 5% Tolerance (Look at Hardware Circuit for specifics)
- Tactile Push Buttons
- CEM-1203 Buzzer

Op Amp Circuit:

- OPO7CP Op Amp

Temperature Measurement:

- K-type/J-type Thermocouple
- LM335 Temperature Sensor
- LM4040 Voltage Reference

Software Specifications:

- Auto Safety Shutoff (If 50°C not reached in 60 seconds)
- Shut-Off Button
- User Controlled Temperature and Time Control
- Temperature Strip Chart

2 Investigation

2.1 Idea Generation

Before generating ideas and working hypotheses, our group ensured a thorough understanding of the project requirements and the factors influencing the final outcome. We began by controlling the oven using the provided PWM code to observe how the temperature increased and decreased. To further validate our understanding, we tested the thermocouple wires with a multimeter to analyze how voltage readings corresponded to temperature changes. These initial steps helped us to brainstorm ways to integrate the sensed temperature from the thermocouple into our control system. Once we understood how the state machine should function, we hypothesized that by processing the voltage from the thermocouple wires, we could enable state transitions that adjust the PWM signal, thereby controlling the oven's temperature effectively.

2.2 Investigation Design

To amplify the voltage drop from the thermocouple wire, we incorporated an operational amplifier (op-amp), which then fed the signal into the N76EE03 microcontroller for processing. To ensure accurate temperature readings, we compared our measured values with actual temperature readings obtained from a multimeter. We also implemented a cold junction compensation to account for ambient temperature variations.

Our first objective was to obtain a stable room temperature reading before testing in the oven environment. Additionally, we identified hot and cold spots within the oven to determine the optimal placement for the thermocouple. On the software side, we developed and tested each component of the system separately using identical circuit boards, ensuring proper functionality before integrating all components. This approach maximized efficiency and allowed us to troubleshoot issues in isolation before assembling the complete system.

2.3 Data Collection

For button testing and timer integration, we used trial and error, carefully noting issues and iterating solutions. To ensure accurate temperature readings, we placed the thermocouple in the oven's center after determining the ideal location through a hot and cold spot analysis. We validated our measurements by manually controlling the oven's PWM settings throughout the reflow stage while tracking temperature changes. We decided to store this information on a spreadsheet after every test. Our goal was to achieve readings within ±3°C of the actual temperature, repeating the process until this accuracy was consistently met.

2.4 Data Synthesis

After consistently obtaining accurate temperature readings without code modifications, we consolidated all individual program components into a single integrated system. We merged the state machine with the timer, PWM control, button inputs, and temperature readings, incorporating each component sequentially to ensure stability. Once fully integrated, we tested the temperature sensing functionality and overall project performance after every major code update. To refine our system, we plotted the oven's temperature curve while using our automatic PWM control, analyzing how long each state should last to achieve an optimal reflow process. We validated our design by comparing our temperature and timing data with reference resources and additional hot and cold spot tests. Finally, we conducted real-world testing by processing three PCB boards with solder paste and components, ensuring proper heating and successful reflow soldering.

2.5 Analysis of Results

Our final tests confirmed that we could successfully regulate the oven's temperature using thermocouple readings to trigger state transitions and adjust PWM signals. We validated our approach by consistently achieving proper reflow cycles without burning or discoloring the PCB components. However, we encountered some errors throughout the process. Initially, placing the thermocouple in the oven's air rather than in contact with a heated surface led to unstable temperature readings with significant fluctuations. Additionally, while plotting our temperature graph in real-time, we observed a slight delay in data updates. Despite these challenges, we effectively processed voltage changes from the thermocouple into meaningful temperature readings, enabling a fully functional state machine with timers and PWM control to operate the reflow oven.

3 Design

3.1 Use of Process:

Our group used design strategies to break up the project into smaller components that could be done by specific team members. When trying to come up with solutions to various parts of the project, we would ask other team members for ideas. We would always share the file for the latest version of our project parts so that everyone on the team had access to all project components. We also created two circuits for our project so that different parts of the team could test their code at the same time.

3.2 Need and Constraint Identification:

The need for our project was to build a controller for the oven that would allow us to reflow solder with adjustable parameters. The technical requirements were found in the project pdf on Canvas. Since we were going to be the users of our product, we decided to make the controller efficient, adjustable, and consistent. Some constraints of the project were the microcontroller having to be programmed in 8051 assembly, the accuracy of the components we used, and the time we had for the project.

3.3 Problem Specification:

The requirements for the project included:

- measuring temperature between 25℃ and 240℃ with 3℃ of error
- Push buttons to adjust reflow soldering parameters
- start and stop button
- Safety stop feature
- displaying the temperature on an LCD screen as well as a Python stripchart of the temperature.

After looking at the needs and limitations, we wanted to increase the efficiency of setting the reflow parameters we wanted, add modifications to the Python chart, and increase the accuracy of our temperature readings.

3.4 Solution Generation:

Some design solutions that we came up with are:

- a preset reflow parameter button
- having the graph sample temperature data more often
- having the graph change colour depending on the temperature
- using the precise formula on the NIST site to calculate temperature from the thermocouple wire

3.5 Solution Evaluation:

- The preset reflow parameter button would meet the requirement of increased efficiency for reflow parameter setting
- Having the graph sample temperature data more often would increase the accuracy of the graph
- Having the graph change colour would modify our Python chart
- and using the NIST formula for temperature calculation would increase the accuracy of our temperature readings

3.6 Safety and Professionalism

Safety and Professionalism

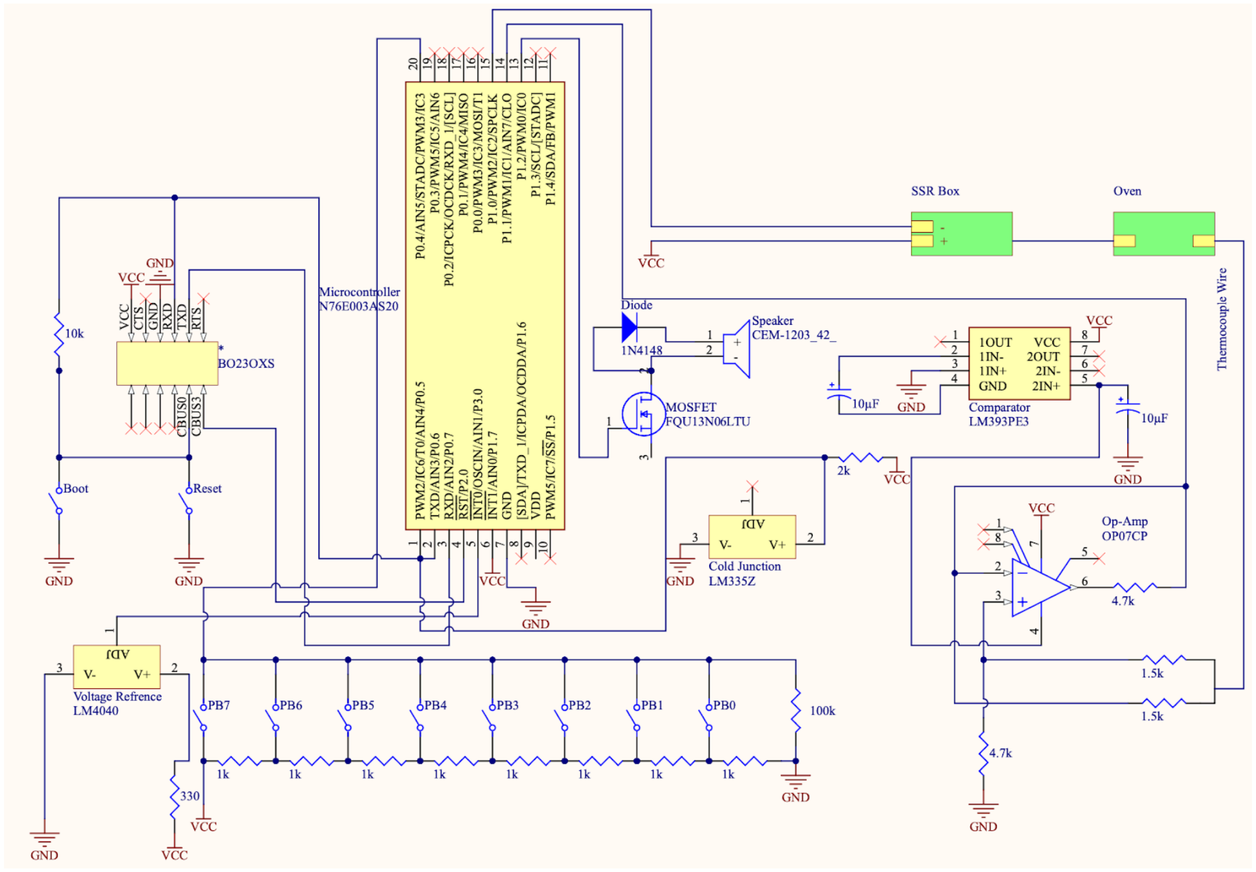
This project involved dealing with high temperatures, low voltage, and current. There were also fumes involved from the solder and flux. To satisfy proper safety, all members took precautions.

1. Safety Goggles were worn at all times when dealing with live voltage, soldering, and the oven.
2. Ventilation was prioritized when soldering to avoid the inhalation of fumes.
3. Hands were washed thoroughly after soldering activities.
4. No food and water were present in the lab or around lab activities outside the lab.
5. Proper attire was worn, covering any exposed skin. Loose hair was tied back, and closed-toed shoes were worn at all times.
6. Any flammable substance or clothing was removed and replaced with more appropriate equipment.
7. The station was cleaned after use, and lab equipment was powered off and returned to their appropriate slots.
8. Any safety issues that were noticed were brought to the lab TA's attention.

All members adhered to the protocols set by the professor before the start of the project. Members worked in groups, being accountable for each other for safety reasons.

3.7 Detailed Design

3.7.1 Hardware Block



3.7.2 Detailed Hardware Specifications

The thermocouple wire measures the oven temperature by carrying a voltage difference across its leads that is based on the oven's heat level. The operational amplifier (op-amp) detects the voltage from the thermocouple wire, amplifies it, and sends it to the microcontroller (Pin 14) for further calculations. The op-amp is configured in a difference amplifier circuit, which not only amplifies the signal but also filters out unwanted noise, producing accurate temperature readings.

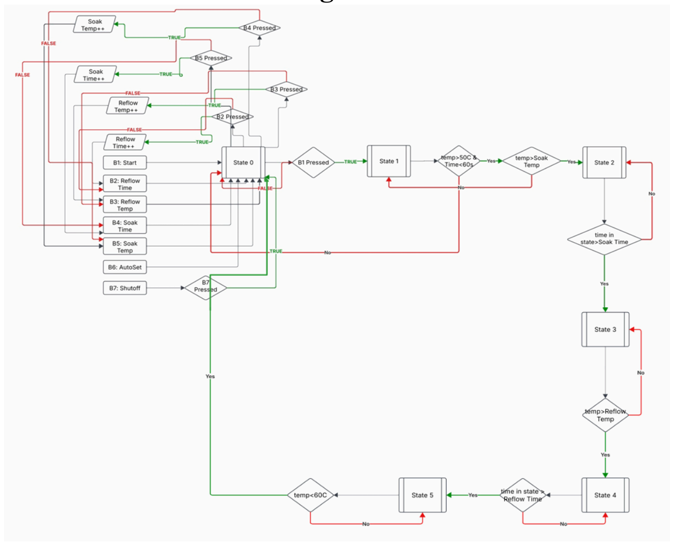
The amplifier's output voltage is scaled by a factor of 314, the gain of the amplifier circuit, which is then sent to the microcontroller for processing. Once the microcontroller determines the required power level and duration, it sends a signal (Pin 15) to the SSR box, which adjusts the oven to the specified temperature and timing. The microcontroller also outputs a signal to the speaker via Pin 13, triggering sounds at various checkpoints to signal important events during the process.

The comparator provides the negative voltage rail from pin 5 of the comparator to pin 4 of the Op-Amp. The cold junction provides the current room temperature reading, and with the hot junction read from the thermocouple wire, it will give the total temperature. There is also a voltage reference to provide accurate conversion for the ADC in the microcontroller. The LM4040 was used to provide that reference voltage.

Eight push buttons were also set up in a voltage divider format, since the microcontroller lacked the number of pins needed for the whole process. PB0 started the state machine process for the controller. PB1 is used as an emergency stop of the oven in case of a safety hazard. A bonus feature was used to preset the parameters for the reflow process. PB4- PB7 were used to increment reflow time, reflow temperature, soak time, and soak temperature, respectively.

The speaker is connected with a diode and a MOSFET. The diode is there to protect any flyback voltage going back into the MOSFET. The microcontroller will send the signal to the MOSFET, signalling whether the speaker is on or off. The diode will regulate the voltage back to the MOSFET, thus ensuring its safety. The signal to the MOSFET is sent from pin 13 of the microcontroller.

3.7.3 Software Block Diagram



3.7.4 Detailed Software Specifications

The Reflow Oven is controlled through the Reflow Process. The Reflow Process follows multiple stages of heating up to a desired temperature, holding at the temperature, heating up to another temperature, holding once again at that temperature, and lastly, cooling down to regular room temperature.

The parameters and specifications are set in the beginning, and the state machine processes within the set parameters. The state machine can control the power of the oven to turn on to 100% power or turn of the oven at 0% power or to remain somewhat stagnant at some power in between.
The current state is stored inside a variable called “FSM1State” and is read at the start of the function call. Then, the finite state machine (FSM) will go to the appropriate code branch for the appropriate state. The branches of the FSM are programmed to follow the Reflow process.
In FSM state 0, the FSM is waiting for the enable signal from PB7 to start the state machine, which starts the reflow process. In this state, no power is sent to the oven, and the user can set the state machine parameters to configure the temperatures, soak time, and reflow time.
Once the configurations are set, a signal is sent from the push button to start the FSM. The FSM enters state 1 and provides 100% power to the oven, increasing the temperature.
Once the soak temperature is reached, the FSM exits stage 1 and enters stage 2. During state 2, the power is reduced to 20% thus, the temperature is no longer increasing but instead remains consistent, and the soak process begins.
The soak process will go on for the amount of time that was set in state 0; once that time has expired, the FSM exits state 2 and enters state 3.
In state 3, the power to the oven is set to 100%, and the temperature once again starts to increase; once the reflow temperature is reached, the FSM exits state 3 and enters state 4.
During state 4, the power to the oven is reduced to 20%, and thus the temperature no longer increases but instead remains consistent, and the reflow process begins. The reflow process will go on for the amount of time set in state 0; once that amount of time has expired, the FSM will exit state 4 and enter state 5.
In state 5, the power to the oven is now reduced even further to 0%, thus allowing for the cooling off process to begin, resulting in the temperature to fall rapidly to a safe room temperature.

Pushbuttons
The pushbuttons are integrated throughout the software into the states. The buttons are most extensively used in state 0 since the reflow parameters are set there. A button labelled ‘start’ in Figure 1 is used to start the whole reflow process. There is also a button labeled Shut off in Figure 1, which acts as the emergency shutoff of the entire process. A larger schematic of the buttons and their integration into the circuit is provided in Figure 1.

Temperature Calculations
The output of our opamp is read through a pin of our ADC. The ADC would then give our microcontroller an integer representation of the voltage. To convert this integer into a voltage reading, we multiplied by 4.096 and then divided by the external voltage reference reading. After getting the voltage, we performed the calculations listed below:

- multiply by 1000 (first part of dividing by 10^-6)
- divide by 314 (314 was our r1/r2 of our op amp)

- multiply by 1000 (We multiplied by 1000 twice to have the same effect as dividing by 10^-6 while not losing any decimal points)
- divide by 41 (this constant was due to the thermocouple wire)

After these calculations, we would have the hot temperature.
We also calculated the cold temperature using the LM335. We converted the integer input of the ADC using the same method as above. Then, we applied the following calculations to the voltage:

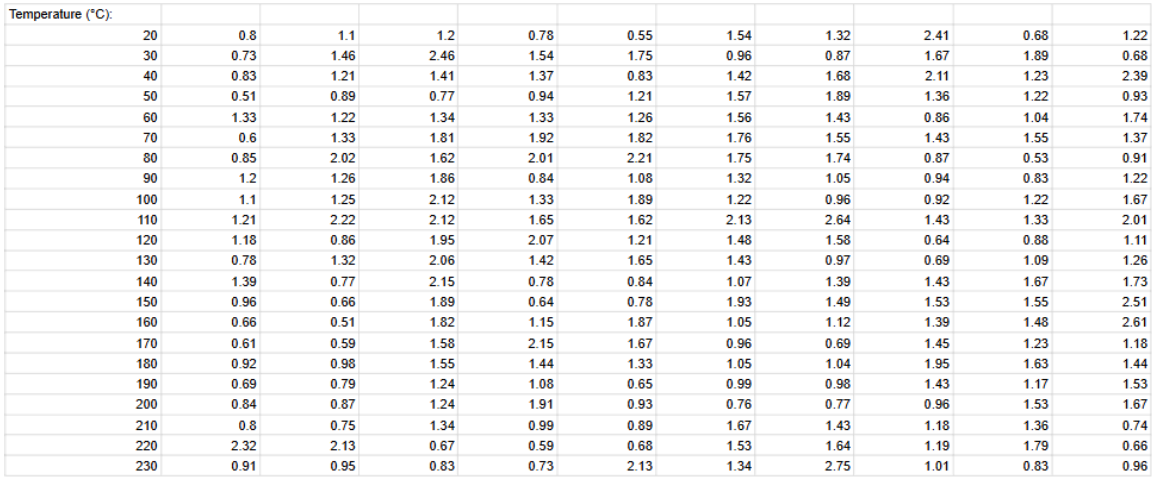
- Divide by 100
- subtract by 273

Then, to get the actual temperature, we just did hot temp + cold temp

3.8 Solutions Assessment:

Multiple design requirements and constraints need to be met for the reflow oven to work as expected.
Temperature Validation:

The inputted temperature received from the Thermocoupling wire is required to have an offset less than 3°C relative to the computed voltage measured converted to temperature compared with the data received from the thermal coupling wire in the oven. To test if our data received meets the 3°C degree celsius offset requirement, we had 10 different trials with the thermal coupling wire inside of the oven with temperatures ranging from 20°C degrees celsius to 230°C degrees celsius. The temperatures at which we took data for validation data is a 10°C difference. For example:
20°C, 30°C, 40°C …. 230°C
When the thermocouple wire is within the X0°C range with ± 1°C, we take all data based on the ± 1°C from the X0°C temperature and get an average. Additionally, we would receive data from the voltage in the same node as the thermocouple to get the computed data. From obtaining the computed data and the data from the thermocouple wire in the oven, we can obtain the offset with the equation:
Offset = | (Computed voltage data) - (Thermocouple wire data) |
After computing the average offset temperature at each range, we can determine that our inputted data in terms of the offset meets the requirement of being less than 3°C.



Temperature (°C):										
20	0.8	1.1	1.2	0.78	0.55	1.54	1.32	2.41	0.68	1.22
30	0.73	1.46	2.46	1.54	1.75	0.96	0.87	1.67	1.89	0.68
40	0.83	1.21	1.41	1.37	0.83	1.42	1.68	2.11	1.23	2.39
50	0.51	0.89	0.77	0.94	1.21	1.57	1.89	1.36	1.22	0.93
60	1.33	1.22	1.34	1.33	1.26	1.56	1.43	0.86	1.04	1.74
70	0.6	1.33	1.81	1.92	1.82	1.76	1.55	1.43	1.55	1.37
80	0.85	2.02	1.62	2.01	2.21	1.75	1.74	0.87	0.53	0.91
90	1.2	1.26	1.86	0.84	1.08	1.32	1.05	0.94	0.83	1.22
100	1.1	1.25	2.12	1.33	1.89	1.22	0.96	0.92	1.22	1.67
110	1.21	2.22	2.12	1.65	1.62	2.13	2.64	1.43	1.33	2.01
120	1.18	0.86	1.95	2.07	1.21	1.48	1.58	0.64	0.88	1.11
130	0.78	1.32	2.06	1.42	1.65	1.43	0.97	0.69	1.09	1.26
140	1.39	0.77	2.15	0.78	0.84	1.07	1.39	1.43	1.67	1.73
150	0.96	0.66	1.89	0.64	0.78	1.93	1.49	1.53	1.55	2.51
160	0.66	0.51	1.82	1.15	1.87	1.05	1.12	1.39	1.48	2.61
170	0.61	0.59	1.58	2.15	1.67	0.96	0.69	1.45	1.23	1.18
180	0.92	0.98	1.55	1.44	1.33	1.05	1.04	1.95	1.63	1.44
190	0.69	0.79	1.24	1.08	0.65	0.99	0.98	1.43	1.17	1.53
200	0.84	0.87	1.24	1.91	0.93	0.76	0.77	0.96	1.53	1.67
210	0.8	0.75	1.34	0.99	0.89	1.67	1.43	1.18	1.36	0.74
220	2.32	2.13	0.67	0.59	0.68	1.53	1.64	1.19	1.79	0.66
230	0.91	0.95	0.83	0.73	2.13	1.34	2.75	1.01	0.83	0.96

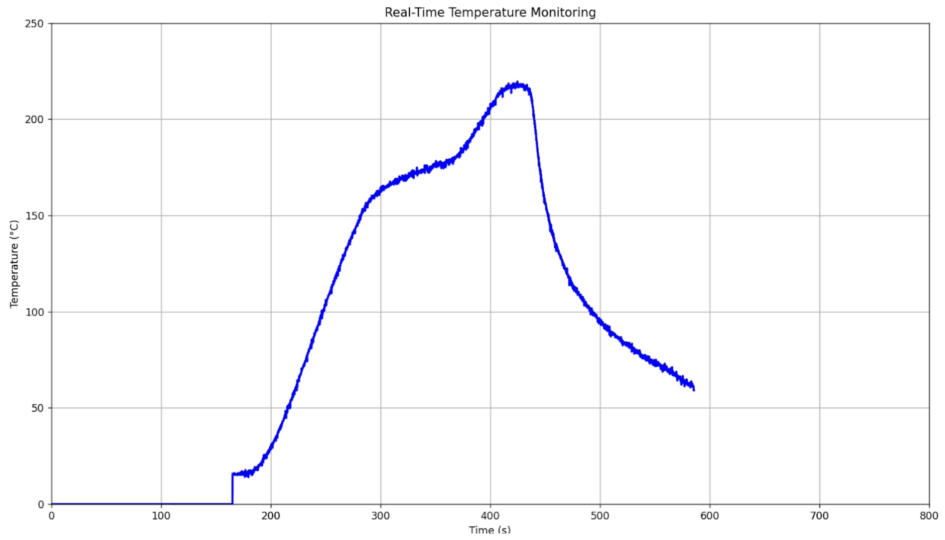
Row - Temperature, Column - Trial #x
The 10x10 represents the offset from the trial attempts along with the offset value at temperature X°C.

Push buttons:

The project is required to have a push button that acts as a kill switch, stopping the entire reflow process. There should also be buttons that allow the user to incrementally change the soak time, soak temperature, reflow time, and reflow temperature. To test the functionality of the kill switch, we would test if the reflow process would stop at each state of the FSM (Finite state machine). As for testing the functionality of the incrementation, we had four separate buttons for the four different parameters. We tested to see if the responsiveness of pressing onto the button would appear on the LCD and checked if there was any form of bouncing that occurred on the buttons.

Finite State Machine Design Implementation:

In order for the reflow process to be successful, it required 6 different states. To test the different states, we would make it so the conditions required to move to the next state were required and also see the LCD displaying the new state. Additionally, we would expect to hear a beeping sound caused by the buzzer, which makes a sound at each new state. We would test to see if a signal for PWM (Pulse width modulation) was sent to the SSR box controlling the temperature of the reflow oven. To determine if the SSR box received signals, we can visually see the inside of the oven appearing orange, indicating heat being produced, meaning the oven works as intended. Moreover, an important requirement for the design of the FSM is that if the temperature inputted from the thermocouple wire does not reach a temperature greater or equal to 50°C within 60 seconds, the entire reflow process will stop. To test this, we would intentionally have the thermocouple wire not be in an environment that surpassed this 50°C temperature threshold, allowing us to determine the functionality of the FSM within the specific state worked. Moreover, to visually inspect the process of the FSM when activated, we can use a Python strip chart, which plots the temperature of the thermocouple wire over time and allows us to see when we can expect state changes.



The image shows a real-time display of the temperature of the thermocouple wire over time.

Weakness:

A weakness in our design would be an accurate temperature from the data of the thermocouple wire being displayed to the LCD. To determine the temperature measured from the thermal coupling wire, we had contributing factors such as resistor values. With the resistor value contributing to the input value of the thermocouple wire, there could be inaccuracies such as the resistor value offset. The problem with determining the accuracy of the thermocouple wire is when testing, the thermocouple wire was not placed at the same spot, and due to the nature of the reflow oven, there tend to be hot spots, which can cause spikes of temperatures. As a result of hotspots of temperature in the oven, this can change the difference in how fast we can expect the reflow oven to change states.

Strengths:

The state machine works accurately at each state. It switches states as expected when the conditions are met. The FSM sends a signal for the buzzer to make a sound at each new state and also changes the LCD for us to visually see that a state change has occurred. Additionally, all the safety features that were part of the FSM worked. The automatic kill switch from the button to shut off the reflow process at any state works as intended. The auto abort safety feature worked since if the temperature of the thermocouple wire did not reach 50℃ within 60 seconds, it would shut off.

4 Lifelong Learning:

At the beginning of the project, we were less familiar with the concepts required to build the reflow controller. We needed to learn about the process of reflow soldering and the use of thermocouple wires to measure temperature. By working on the project and collaborating effectively as a team, we were able to bridge the knowledge gap and learn everything we needed to construct the reflow oven controller. By the end of the project, we had gained confidence in the required concepts and enhanced our knowledge of assembly programming and circuit design. We worked on every part of the project collaboratively, learning together and helping each other when needed. All members of our team went into the project with a strong foundational knowledge of electronics provided by the other courses we are taking and took last semester. A course that helped us complete the project was ELEC 201. It taught us about op-amps, which we needed to use to convert the voltage from the thermocouple wire into a readable signal to input to the microcontroller. The previous projects in ELEC 291 were also helpful since they gave us practice programming in 8051 assembly, which was heavily utilized in the project.

Conclusion:

Our team was able to complete the project and construct a working reflow oven controller. Our system monitors and regulates the temperature in real time, progresses through the stages of the reflow solder process, and can be interacted with by push buttons on a breadboard. We included all functionality described in the project document, although we included no bonus features. A problem we encountered was that we had issues when integrating the different parts of the project. We fixed this by going back and testing each part of the system separately and being confident that everything works independently before combining them. Another issue we had was that we underestimated the scope of the project. We ended up spending long hours in the lab in the days preceding the project demonstration, and it would have been more efficient if we had started earlier. Our team spent a total of approximately 80 hours working on the project spread across 10 days.

BIBLIOGRAPHY

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

[2] J. Calvino-Fraga. ELEC 281. Lecture, “Reflow Oven Controller.” Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Feb. 02, 2024.

[3] J. Calvino-Fraga. ELEC 281. Lecture, “EFM8_ FSM_NVMEM” Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Feb. 09, 2024.

[4]“N76E003 Datasheet Nuvoton 1T 8051-based Microcontroller N76E003 Datasheet,” 2019. Accessed: Mar. 02, 2025. [Online]. Available: https://www.nuvoton.com/export/resource-files/DS_N76E003_EN_Rev1.08.pdf

[5]“LM135, LM135A, LM235, LM235A, LM335, LM335A LMx35, LMx35A Precision Temperature Sensors 1 Features 3 Description Basic Temperature Sensor Simplified Schematic Calibrated Sensor.” Accessed: Mar. 02, 2025. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm335.pdf?ts=1740864511431&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLM335

[6]“LM393B, LM2903B, LM193, LM293, LM393 and LM2903 Dual Comparators.” Available: https://www.ti.com/lit/ds/symlink/lm393.pdf

[7]“Ultralow Offset Voltage Operational Amplifier Data Sheet OP07.” Available: https://www.analog.com/media/en/technical-documentation/data-sheets/OP07.pdf

APPENDIX:

```
; 76E003 ADC test program: Reads channel 7 on P1.1, pin 14
; This version uses an LED as voltage reference connected to pin 6 (P1.7/AIN0)

$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
;                               -------
;


BAUD              EQU 115200 ; Baud rate of UART in bps
TIMER1_RELOAD     EQU (0x100-(CLK/(16*BAUD)))
TIMER0_RELOAD_1MS EQU (0x10000-(CLK/1000))
CLK           EQU 16600000 ; Microcontroller system frequency in Hz
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)))


BsTime equ P1.3
BsTemp equ P0.0
BrTime equ P0.1
BRTemp equ P0.2
B01 equ P0.3
; Reset vector
org 0x0000
    ljmp main

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
org 0x002B
       ljmp Timer2_ISR

title:     db 'RC:     RT:', 0
title2:    db 'SC:     ST:', 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
SOUND_OUT equ P1.2
PWM_OUT equ P1.0
$NOLIST
$include(LCD_4bit.inc) ; A library of LCD related functions and utility macros
$LIST

; These register definitions needed by 'math32.inc'
DSEG at 30H
x:   ds 4
y:   ds 4
bcd: ds 5
VLED_ADC: ds 2
sec: ds 3
Count1ms:     ds 2 ; Used to determine when half second has passed
FSM1_state: ds 1
temp_soak: ds 2
time_soak: ds 2
temp_refl: ds 2
time_refl: ds 2
refltime: ds 2
refltemp: ds 2
soaktime: ds 2
soaktemp: ds 2
shutoff: ds 2

seconds:         ds 1
pwm: ds 3
pwm_counter:  ds 1


temp: ds 4
VAL_LM4040: ds 2
VAL_LM335: ds 2

BSEG
beep_enable: dbit 1
mf: dbit 1
PB0: dbit 1
PB1: dbit 1
PB2: dbit 1
PB3: dbit 1
PB4: dbit 1
PB5: dbit 1
```

```
PB6: dbit 1
PB7: dbit 1

s_flag:                  dbit 1
half_seconds_flag: dbit 1
flag1:                   dbit 1
flag2:                   dbit 1
flag0:                   dbit 1


$NOLIST
$include(math32.inc)
$LIST





;                        1234567890123456
rt:    db ' RT ',0  ;Reflow time,Reflow Temp, Soak Time
rc:    db 'RC ',0
st:    db ' ST ',0
sc:    db 'SC ',0
;ti:    db 'Ti',0
To:    db 'T0',0
state1messagetop:    db 's1     RAMP2SOAK', 0
state1messagebottom: db 'tt   ts    t    ', 0
state2messagetop:    db 's2          SOAK', 0
state3messagetop:    db 's3     RAMP2PEAK', 0
state4messagetop:    db 's4        REFLOW', 0
state5messagetop:    db 's5     cooldown ', 0
errorshutoffmessage: db '****SHUT OFF****', 0
CSEG
;---------------------------------;
; 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 TL0, #High(TIMER0_RELOAD)
       mov TH0, #low(TIMER0_RELOAD)
       ; Enable the timer and interrupts
    setb ET0  ; Enable timer 0 interrupt
    setb TR0  ; Start timer 0
       ret


; ==============================
; Timer0 ISR for Buzzer Control
; ==============================
Timer0_ISR:
    ;clr TR0                 ; Stop the timer to prevent timing issues
   ; mov TH0, #high(TIMER0_RELOAD)
   ; mov TL0, #low(TIMER0_RELOAD)
   ; setb TR0                ; Restart the timer
beep_check:
    jb beep_enable, toggle_sound  ; If beep_enable is set, toggle sound
    ;clr SOUND_OUT           ; Otherwise, ensure it's off
    reti

toggle_sound:
       clr TR0                  ; Stop the timer to prevent timing issues
    mov TH0, #high(TIMER0_RELOAD)
    mov TL0, #low(TIMER0_RELOAD)
    setb TR0
    cpl SOUND_OUT           ; Toggle buzzer state
    reti

; ==============================
; Start Beep Function
; ==============================
Start_Beep:
 setb beep_enable        ; Enable beep flag              ; Start Timer 0 if not already running
 ljmp beep_check

; ==============================
; Stop Beep Function
; ==============================
Stop_Beep:
    clr beep_enable         ; Disable beep flag
    clr SOUND_OUT           ; Ensure buzzer is off
    clr TR0                 ; Stop Timer 0 to save power
  reti

;---------------------------------;
; Routine to initialize the ISR   ;
; for timer 2                     ;
;---------------------------------;
Timer2_Init:
       mov T2CON, #0 ; Stop timer/counter.  Autoreload mode.
       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
       mov pwm_counter, #0

       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
```

```
Skip_Count:
        inc pwm_counter
        clr c
        mov a, pwm
        subb a, pwm_counter
        cpl c
        mov PWM_OUT, c

        mov a, pwm_counter
        cjne a, #100, Timer2_ISR_continue
        mov pwm_counter, #0
        inc seconds
        setb s_flag

Timer2_ISR_continue:
        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
        ljmp Inc_done

go_to_Timer2_ISR_done:
        ljmp Timer2_ISR_done
Inc_Done:
        ; Check if half second has passed
        mov a, Count1ms+0
        cjne a, #low(1000), go_to_Timer2_ISR_done ; Warning: this instruction changes the carry
flag!
        mov a, Count1ms+1
        cjne a, #high(1000), go_to_Timer2_ISR_done

        ; 500 milliseconds have passed.  Set a flag so the main program knows
        setb half_seconds_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 ;this block of code is for getting it to stop at 0000
    mov a, shutoff+1 ;loads most significant digit into a
    cjne a, #0x00, test1 ;checks if a which has most significant digit is 0, if not 0 regular
action, if 0 then check least significant digit
        clr a ;clears a
        mov a, shutoff+0 ;loads least sigficant digit into a
        cjne a, #0x00, test0 ;check if least significant digit is 0, if not 0 regular action, if 0
then should should bcd counter should be 0
        mov shutoff+0, a ;load the 0 into a.
        ljmp test0

check0: ;this block of code is for getting it to stop at 0000
    mov a, shutoff+1 ;loads most significant digit into a
    cjne a, #0x00, test0 ;checks if a which has most significant digit is 0, if not 0 regular
action, if 0 then check least significant digit
        clr a ;clears a
        mov a, shutoff+0 ;loads least sigficant digit into a
        cjne a, #0x00, test0 ;check if least significant digit is 0, if not 0 regular action, if 0
then should should bcd counter should be 0
        mov shutoff+0, a ;load the 0 into a.
        ljmp test1
test0:
        jnb flag0, test1
        clr a
        mov a, shutoff+0
        add a, #0x99 ;subtracts
        cjne a, #0x99, Timer2_ISR_da0
        add a, #0x01 ; if i dont add this it starts at 98, instead of 99
        mov shutoff+0, a
        clr a
        mov a, shutoff+1
        add a, #0x99 ;subtracts from the most significant digit
        da a
        mov shutoff+1, a
        ;clr a
        ljmp check0

Timer2_ISR_da0:
        da a ; Decimal adjust instruction.  Check datasheet for more details!
        mov shutoff+0, a
        ; Increment the BCD counter


check1: ;this block of code is for getting it to stop at 0000
    mov a, time_soak+1 ;loads most significant digit into a
    cjne a, #0x00, test1 ;checks if a which has most significant digit is 0, if not 0 regular
action, if 0 then check least significant digit
        clr a ;clears a
        mov a, time_soak+0 ;loads least sigficant digit into a
        cjne a, #0x00, test1 ;check if least significant digit is 0, if not 0 regular action, if 0
then should should bcd counter should be 0
        mov time_soak+0, a ;load the 0 into a.
        ljmp test2

test1:
        jnb flag1, test2
        clr a
        mov a, time_soak+0
        add a, #0x99 ;subtracts
        cjne a, #0x99, Timer2_ISR_da
        add a, #0x01; if i dont add this it starts at 98, instead of 99
        mov time_soak+0, a
        clr a
        mov a, time_soak+1
        add a, #0x99 ;subtracts from the most significant digit
        da a
        mov time_soak+1, a
        ;clr a
        ljmp check1

Timer2_ISR_da:
        da a ; Decimal adjust instruction.  Check datasheet for more details!
        mov time_soak+0, a

check2: ;this block of code is for getting it to stop at 0000
    mov a, time_refl+1 ;loads most significant digit into a
    cjne a, #0x00, test2 ;checks if a which has most significant digit is 0, if not 0 regular
action, if 0 then check least significant digit
        clr a ;clears a
        mov a, time_refl+0 ;loads least sigficant digit into a
        cjne a, #0x00, test2 ;check if least significant digit is 0, if not 0 regular action, if 0
then should should bcd counter should be 0
        mov time_refl+0, a ;load the 0 into a.
        ljmp check2
```

```
test2:
       jnb flag2, Timer2_ISR_done
       clr a
       mov a, time_refl+0
       add a, #0x99 ;subtracts
       cjne a, #0x99, Timer2_ISR_da2
       add a, #0x01; if i dont add this it starts at 98, instead of 99
       mov time_refl+0, a
       clr a
       mov a, time_refl+1
       add a, #0x99 ;subtracts from the most significant digit
       da a
       mov time_refl+1, a
       ;clr a
       ljmp check2

Timer2_ISR_da2:
       da a ; Decimal adjust instruction.  Check datasheet for more details!
       mov time_refl+0, a

Timer2_ISR_done:
       pop psw
       pop acc
       reti

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 and start the ADC:

       ; Initialize (P0.4, P0.5) as input.
       orl P0M1, #0b00110000
    anl P0M2, #0b11001111
    ; Initialize the pins used by the ADC (P1.1, P1.7) as input.
       orl     P1M1, #0b10000010
       anl     P1M2, #0b01111101



       ; AINDIDS select if some pins are analog inputs or digital I/O:
       mov AINDIDS, #0x00 ; Disable all analog inputs
```

```
        orl AINDIDS, #0b10110001 ; Using AIN0, AIN4, AIN5, AIN7
        orl ADCCON1, #0x01 ; Enable ADC
        setb EA   ; Enable global interrupts
        setb ET0  ; Enable Timer 0 interrupt
        setb TR0  ; Ensure Timer 0 is running



        ret
load_temp:
        jnb TI, load_temp ; wait for SBUF to be ready
        clr TI                  ; clr TI so we know SBUF is getting our new value
        mov SBUF, a             ; move new value into SBUF
        ret

send_temp:
        mov a, bcd+3
        swap a                  ; upper digit of bcd+3
        anl a, #0x0F    ; Mask the lower nibble (idk what this means)
        add a, #48      ; convert to ASCI
        lcall load_temp ; load the value
    mov SBUF, a     ; send value

    mov a, bcd+3      ; lower digit of bcd+3
    anl a, #0x0F
    add a, #48
    lcall load_temp
    mov SBUF, a

        mov a, bcd+2
        swap a                  ; upper digit of bcd+2
        anl a, #0x0F
        add a, #48
        lcall load_temp
    mov SBUF, a

    mov a, bcd+2      ; lower digit of bcd+2
    anl a, #0x0F
    add a, #48
    lcall load_temp
    mov SBUF, a

    mov a, #'.'       ; decimal point
    lcall load_temp
    mov SBUF, a

    mov a, bcd+1      ; upper digit of bcd+1
        swap a
        anl a, #0x0F
        add a, #48
        lcall load_temp
    mov SBUF, a

    mov a, bcd+1      ; lower digit of bcd+1
    anl a, #0x0F
    add a, #48
    lcall load_temp
    mov SBUF, a

    mov a, bcd+0      ; upper digit of bcd+0
        swap a
        anl a, #0x0F
        add a, #48
        lcall load_temp
    mov SBUF, a

    mov a, bcd+0       ; lower digit of bcd+0
    anl a, #0x0F
    add a, #48
    lcall load_temp
    mov SBUF, a

        mov a, #'\r'   ; cursor to far left
    lcall load_temp
    mov SBUF, a

    mov a, #'\n'       ; new line
    lcall load_temp
    mov SBUF, a

        ret
send_temp_done:
        ret
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


ADC_to_PB:
        anl ADCCON0, #0xF0
        orl ADCCON0, #0x05 ; Select AIN5

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

        setb PB7
        setb PB6
        setb PB5
        setb PB4
        setb PB3
        setb PB2
        setb PB1
        setb PB0

        ; Check PB7
ADC_to_PB_L7:
        clr c
        mov a, ADCRH
        subb a, #0xf0
        jc ADC_to_PB_L6
        clr PB7
        ret
        ; Check PB6
ADC_to_PB_L6:
        clr c
        mov a, ADCRH
        subb a, #0xd0
        jc ADC_to_PB_L5
        clr PB6
        ret

        ; Check PB5
ADC_to_PB_L5:
        clr c
        mov a, ADCRH
        subb a, #0xb0
        jc ADC_to_PB_L4
        clr PB5
        ret

        ; Check PB4
ADC_to_PB_L4:
        clr c
        mov a, ADCRH
        subb a, #0x90
        jc ADC_to_PB_L3
        clr PB4
        ret

        ; Check PB3
ADC_to_PB_L3:
        clr c
        mov a, ADCRH
        subb a, #0x70
        jc ADC_to_PB_L2
        clr PB3
        ret

        ; Check PB2
ADC_to_PB_L2:
        clr c
        mov a, ADCRH
        subb a, #0x50
        jc ADC_to_PB_L1
        clr PB2
        ret

        ; Check PB1
ADC_to_PB_L1:
        clr c
        mov a, ADCRH
        subb a, #0x30
        jc ADC_to_PB_L0
        clr PB1
        ret

        ; Check PB0
ADC_to_PB_L0:
        clr c
        mov a, ADCRH
        subb a, #0x10
        jc ADC_to_PB_Done
        clr PB0
        ret

ADC_to_PB_Done:
        ; No pusbutton pressed
        ret


jump_to_loop:
        ret

increment_value_rc:
        mov a, temp_refl+0
        add a, #1
        da a
        mov temp_refl+0, a

    cjne a, #0x99, jump_to_loop
    mov a, temp_refl+1
    add a, #1
    da a
    mov temp_refl+1, a
    mov temp_refl+0, #0x00
        ret

increment_value_rt:
        mov a, time_refl+0
        add a, #1
        da a
        mov time_refl+0, a

    cjne a, #0x99, jump_to_loop
    mov a, time_refl+1
    add a, #1
    da a
    mov time_refl+1, a
    mov time_refl+0, #0x00
        ret

increment_value_sc:
        mov a, temp_soak+0
        add a, #1
        da a
        mov temp_soak+0, a

    cjne a, #0x99, jump_to_loop
    mov a, temp_soak+1
    add a, #1
    da a
    mov temp_soak+1, a
    mov temp_soak+0, #0x00
        ret

increment_value_st:
        mov a, time_soak+0
        add a, #1
        da a
        mov time_soak+0, a

    cjne a, #0x99, jump_to_loop
    mov a, time_soak+1
    add a, #1
    da a
    mov time_soak+1, a
    mov time_soak+0, #0x00
       ret
loop:

       lcall ADC_to_PB
    lcall Display_Value
    Set_Cursor(1, 1)
    Send_Constant_String(#rc)
    Set_Cursor(1, 9)
    Send_Constant_String(#rt)
       Set_Cursor(2, 1)
    Send_Constant_String(#sc)
    Set_Cursor(2, 9)
    Send_Constant_String(#st)
    jb PB7, loop_rt
    Wait_Milli_Seconds(#100)
    jb PB7, loop_rt
    mov R2, #100
    lcall waitms

       lcall increment_value_rc

loop_rt:
       jb PB6, loop_sc
       Wait_Milli_Seconds(#100)
    jb PB6, loop_sc
    mov R2, #100
    lcall waitms

       lcall increment_value_rt

loop_sc:
       jb PB5, loop_st
       Wait_Milli_Seconds(#100)
    jb PB5, loop_st
    mov R2, #100
    lcall waitms

       lcall increment_value_sc

loop_st:
       jb PB4, loop_out
       Wait_Milli_Seconds(#100)
    jb PB4, loop_out
    mov R2, #100
    lcall waitms

       lcall increment_value_st
loop_out:
    ret

Display_Value:
    Set_Cursor(1, 4)
    Display_BCD(temp_refl+1)
    Display_BCD(temp_refl+0)

       Set_Cursor(1, 13)
       Display_BCD(time_refl+1)
       Display_BCD(time_refl+0)

       Set_Cursor(2, 4)
```

```
        Display_BCD(temp_soak+1)
        Display_BCD(temp_soak+0)

        Set_Cursor(2, 13)
        Display_BCD(time_soak+1)
        Display_BCD(time_soak+0)
    ret

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


update_temp:
        anl ADCCON0, #0xF0
        orl ADCCON0, #0x00 ; Select channel 0

        lcall Read_ADC
        ; reads signal from pin 6
        mov VAL_LM4040+0, R0
        mov VAL_LM4040+1, R1

        anl ADCCON0, #0xF0
        orl ADCCON0, #0x04 ; Select channel 6
        lcall Read_ADC
        ; reads signal from pin 1
        mov VAL_LM335+0, R0
    mov VAL_LM335+1, R1

        ; Read the signal from pin 14
        anl ADCCON0, #0xF0
        orl ADCCON0, #0x07 ; Select channel 7
        lcall Read_ADC

    ; moves the voltage at AIN7 into x
        mov x+0, R0
        mov x+1, R1
        mov x+2, #0
        mov x+3, #0

        Load_y(00040959) ; The MEASURED voltage reference: 4.0959V, with 4 decimal places
        lcall mul32 ; 4.096 * ADCop07
        ; moves ADC ref into y
        mov y+0, VAL_LM4040+0
        mov y+1, VAL_LM4040+1
        mov y+2, #0
        mov y+3, #0
        lcall div32 ; (4.096 * ADCop07) / ADCref
        ;vout is in x
op_v_to_htemp:

        Load_y(1000)
        lcall mul32
        ; the r1/r2
        Load_y(314)
        lcall div32

        Load_y(1000)
        lcall mul32
        Load_y(41)
        lcall div32



        mov temp+0, x+0
        mov temp+1, x+1
        mov temp+2, x+2
        mov temp+3, x+3
    ; temp = hot_temp

lm335_v_to_ctemp:

        mov x+0, VAL_LM335+0
        mov x+1, VAL_LM335+1
        mov x+2, #0
        mov x+3, #0

        Load_y(00040959)
        lcall mul32
        mov y+0, VAL_LM4040+0
        mov y+1, VAL_LM4040+1
        mov y+2, #0
        mov y+3, #0
        lcall div32
        Load_y(100)
        lcall mul32
        Load_y(02730000)
        lcall sub32
        Load_y(00040000)
        lcall sub32

actual_temp:

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

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

        lcall add32

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

        ret

Display_formated_BCD:
        Set_Cursor(2, 10)
        Display_BCD(bcd+3)
        Display_BCD(bcd+2)
        Display_char(#'.')
        Display_BCD(bcd+1)
        ret

LCD_Display:
        lcall hex2bcd
        lcall Display_formated_BCD
        ret
go_to_continue_FSM1State1:
        ljmp continue_FSM1State1
check_if_shutoff:
        load_y(00500000)
        mov x+0, temp+0
        mov x+1, temp+1
        mov x+2, temp+2
        mov x+3, temp+3

        lcall x_lt_y

        jnb mf, go_to_continue_FSM1State1
        ljmp shutoff_instr

shutoff_instr:
        lcall re_initialize
        ljmp forever

re_initialize:
        Load_x(0225)
    lcall hex2bcd
    mov temp_refl+0, bcd+0
    mov temp_refl+1, bcd+1

        Load_x(0045)
    lcall hex2bcd
        mov time_refl+0, bcd+0
        mov time_refl+1, bcd+1

        Load_x(0170)
        lcall hex2bcd
        mov temp_soak+0, bcd+0
        mov temp_soak+1, bcd+1

        Load_x(0090)
    lcall hex2bcd
        mov time_soak+0, bcd+0
        mov time_soak+1, bcd+1

        Load_x(0060)
    lcall hex2bcd
        mov shutoff+0, bcd+0
        mov shutoff+1, bcd+1

    mov FSM1_state, #0
    clr flag1
    clr flag2
    clr flag0

    ljmp forever


main:
       mov sp, #0x7f
       lcall Init_All
       lcall Timer2_Init

    setb EA   ; Enable Global interrupts
    lcall LCD_4BIT

    Load_x(0200)
    lcall hex2bcd
    mov temp_refl+0, bcd+0
    mov temp_refl+1, bcd+1

       Load_x(0030)
    lcall hex2bcd
       mov time_refl+0, bcd+0
       mov time_refl+1, bcd+1

       Load_x(0150)
       lcall hex2bcd
       mov temp_soak+0, bcd+0
       mov temp_soak+1, bcd+1

       Load_x(0070)
    lcall hex2bcd
       mov time_soak+0, bcd+0
       mov time_soak+1, bcd+1

       Load_x(0060)
    lcall hex2bcd
       mov shutoff+0, bcd+0
       mov shutoff+1, bcd+1

    mov FSM1_state, #0

    clr flag1
    clr flag2
    clr flag0
       ljmp forever
gotoShutoff_instr:

ljmp shutoff_instr
;--------------start
forever:
lcall ADC_to_PB
lcall update_temp
lcall send_temp
Wait_Milli_Seconds(#50)
jnb PB1, gotoShutoff_instr
Wait_Milli_Seconds(#50)
jnb PB1, gotoShutoff_instr


; Finite State machine
FSM1:
        mov a, FSM1_state
FSM1_state0:

        cjne a, #0, FSM1_state1
        lcall loop
        mov pwm, #100


check_start_button:
        jb PB0, check_restart_button
        Wait_Milli_Seconds(#50)
        jb PB0, check_restart_button
        mov FSM1_state, #1
        ljmp FSM1_state0_done

check_restart_button:
        jb PB2, go_to_forever_1
        Wait_Milli_Seconds(#50)
        jb PB2, go_to_forever_1
        ljmp re_initialize

 FSM1_state0_done:
    lcall Start_Beep
    mov R2, #200
        lcall waitms
    lcall Stop_Beep
          ; Stop beep

    ljmp FSM2
go_to_forever_1:
ljmp forever
go_to_FSM1_state2:
ljmp FSM1_state2
 FSM1_state1:
         cjne a, #1, go_to_FSM1_state2
         mov pwm, #0
         lcall LCD_Display
         setb flag0
         clr half_seconds_flag
;       Set_Cursor(2, 1)
;    Send_Constant_String(#state1messagebottom)
    Set_Cursor(1, 1)
    Send_Constant_String(#state1messagetop)

   ; lcall x_eq_y
        mov x+0, shutoff+0
    mov x+1, shutoff+1
    mov x+2, #0
    mov x+3, #0
    Load_y(0)
    lcall x_eq_y

         jnb mf, continue_FSM1State1
         ljmp check_if_shutoff
         mov R2, #200
        lcall waitms

continue_FSM1State1:
         mov y+3, temp_soak+1
         mov y+2, temp_soak+0
         mov y+1, #0
         mov y+0, #0
        Set_Cursor(2, 1)
        mov x+3, temp+3
        mov x+2, temp+2
        mov x+1, temp+1
        mov x+0, temp+0

        lcall hex2bcd
        mov x+3, bcd+3
        mov x+2, bcd+2
        mov x+1, bcd+1
        mov x+0, bcd+0

        lcall x_gt_y ; checks if the temp > soak temp

        jnb mf, go_to_forever_2

        mov FSM1_state, #2
 FSM1_state1_done:
  lcall Start_Beep
    mov R2, #200
       lcall waitms
    lcall Stop_Beep
        ljmp FSM2
go_to_forever_2:
ljmp forever
go_to_FSM1_state3:
ljmp FSM1_state3
 FSM1_state2:
        cjne a, #2, go_to_FSM1_state3
        lcall LCD_Display
        setb flag1
        clr half_seconds_flag
       Set_Cursor(1, 1)
    Send_Constant_String(#state2messagetop)
       Set_Cursor(2, 1)
;    Send_Constant_String(#state1messagebottom)
       Display_BCD(time_soak+1)
       Display_BCD(time_soak+0)
       Set_Cursor(2, 6)
       lcall LCD_Display
        mov pwm, #60

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

        lcall hex2bcd
        mov x+3, bcd+3
        mov x+2, bcd+2
        mov x+1, bcd+1
        mov x+0, bcd+0

        mov x+3, #0
        mov x+2, #0
        mov x+1, time_soak+1
        mov x+0, time_soak+0
        load_y(0)

        lcall x_eq_y ; check if soak time goes to 0

        jnb mf, check_again
        ljmp change_to_State3
check_again:
       load_y(9000)
       lcall x_gt_y
       jnb mf, go_to_forever_3
        mov R2, #200
       lcall waitms
change_to_State3:
       mov FSM1_state, #3
 ;check the condition if it doesnt increase temp for 60 seconds or more
 FSM1_state2_done:
       lcall Start_Beep
    mov R2, #200
       lcall waitms
    lcall Stop_Beep
       ljmp FSM2
go_to_forever_3:
ljmp forever
go_to_FSM1_state4:
ljmp FSM1_state4
 FSM1_state3:

     cjne a, #3, go_to_FSM1_state4
     lcall LCD_Display
     Set_Cursor(1, 1)
    Send_Constant_String(#state3messagetop)
       Set_Cursor(2, 1)
    Send_Constant_String(#state1messagebottom)
        mov pwm, #0
        mov y+3, temp_refl+1
        mov y+2, temp_refl+0
        mov y+1, #0
        mov y+0, #0

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

        lcall hex2bcd
        mov x+3, bcd+3
        mov x+2, bcd+2
        mov x+1, bcd+1
        mov x+0, bcd+0
        lcall Display_formated_BCD
        lcall x_gt_y

        jnb mf, go_to_forever_3

        mov R2, #200
       lcall waitms
        mov FSM1_state, #4
 FSM1_state3_done:
        lcall Start_Beep
    mov R2, #200
       lcall waitms
    lcall Stop_Beep
        ljmp FSM2
go_to_forever_4:
ljmp forever
go_to_FSM1_state5:
ljmp FSM1_state5
```

```
 FSM1_state4:

        cjne a, #4, go_to_FSM1_state5
        lcall LCD_Display
        setb flag2
        clr flag1
        clr half_seconds_flag
        Set_Cursor(1, 1)
    Send_Constant_String(#state4messagetop)
       Set_Cursor(2, 1)
;    Send_Constant_String(#state1messagebottom)
       Display_BCD(time_refl+1)
       Display_BCD(time_refl+0)
        mov pwm, #50

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

        lcall hex2bcd
        mov x+3, bcd+3
        mov x+2, bcd+2
        mov x+1, bcd+1
        mov x+0, bcd+0

        mov x+3, #0
        mov x+2, #0
        mov x+1, time_refl+1
        mov x+0, time_refl+0
        load_y(0)

        lcall x_eq_y ; check if soak time goes to 0

        jnb mf, check_again2
        ljmp change_to_state4

check_again2:
        load_y(9000)
        lcall x_gt_y
        jnb mf, go_to_forever_5
change_to_state4:
        mov R2, #200
       lcall waitms
        mov FSM1_state, #5
 FSM1_state4_done:
  lcall Start_Beep
    mov R2, #200
       lcall waitms
    lcall Stop_Beep
       ljmp FSM2
 go_to_forever_5:
ljmp forever
 FSM1_state5:

        cjne a, #5, go_to_forever_5
        lcall LCD_Display
        clr flag1
        clr flag2
        Set_Cursor(1, 1)
    Send_Constant_String(#state5messagetop)
       mov pwm, #100
```

```
      load_y(600000)
       mov x+3, temp+3
       mov x+2, temp+2
       mov x+1, temp+1
       mov x+0, temp+0

       lcall hex2bcd
       mov x+3, bcd+3
       mov x+2, bcd+2
       mov x+1, bcd+1
       mov x+0, bcd+0

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

       lcall x_lt_y

       jnb mf, FSM2

       mov R2, #200
      lcall waitms
       mov FSM1_state, #0
FSM1_state5_done:
       lcall Start_Beep
   mov R2, #200
      lcall waitms
   lcall Stop_Beep
      ljmp FSM2 ;

FSM2:
ljmp forever


END
```