THE UNIVERSITY
OF BRITISH COLUMBIA

# Project 1 - Reflow Oven Controller

Group B8

| Student # | Student Name | % Point | Signature |
|---|---|---|---|
| 59822890 | Will Chaba | 115 | |
| 46046041 | Bryan Nsoh | 85 | |
| 12521589 | Isabelle André | 120 | |
| 95040986 | Debby Lin | 95 | |
| 75286914 | Eric Wu | 85 | |
| 50266873 | Daniel Nadeem | 100 | |

University of British Columbia
Electrical and Computer Engineering
ELEC 291 L2B Winter 2020
Instructor: Dr. Jesus Calvino-Fraga

Date of submission: February 28, 2020

# Table of Contents:

# 1.0 Introduction

In this project, a Reflow Oven Controller was designed, programmed, and connected to a Reflow Oven to perform reflow soldering of electronic components onto a printed circuit board (PCB). Reflow soldering is a common soldering technique used to assemble surface mount devices (SMDs) onto the PCB's pads coated with solder paste. In effect, the PCB is placed inside a reflow oven, then a heating process with multiple stages of differing temperature range begins to melt the paste and thus linking the components. The designed reflow oven controller sets and executes the settings of a complete reflow process consisting of 6 states numbered 0 to 5. Initially, state 0 allows the soak and reflow temperature to be set before starting the process. Sequentially, state 1 follows where time is allocated to heat the oven until a soak temperature is reached, after which state 2 begins, where temperature is kept stable and the soak timer is enabled. Continuing this process, state 3 initiates where the temperature is required to begin a steep ramp towards the peak temperature required for the reflow state in state 4. Similarly to state 2, state 4 uses a reflow timer counts down at peak temperature until state 5 is reached. Finally at state 5, the oven is allowed to cool until the process ends. Aside from controlling the finite state machine, the controller also measures temperature between 25-240 degrees celsius using a K-type thermocouple with cold junction compensation. Using the temperature readings from the thermocouple, the controller regulates the amount of power delivered by the oven using Pulse Width Modulation (PWM). The Reflow Oven Controller features vocal feedback revealing temperature every 5 seconds with the current state as well as an LCD screen displaying both soak and reflow timers, switching displays to view current temperature, room temperature, and percent output power by push button input.

# 2.0 Investigation

## 2.1 Idea Generation

Ideas and solutions for individual components of the project were first investigated, evaluated, and optimized appropriately following a problem solving process before being implemented. For instance, one of the most important criteria for thermocouple reading was accuracy. In order to convert data read from analog to digital converter (ADC) to accurate decimal values, multiple conversion and calculations were required. Each group member shared their ideas one by one. Ideas were ranked, then eliminated one by one as we compared their functionalities to the project guidelines. Thus, two possible software designs were brainstormed: one that completes the temperature calculation all as hexadecimals in the microcontroller and one that sends the hexadecimal data to the Python program to do the calculation before sending the value back as binary coded decimals (bcd) into the microcontroller via the serial port. The latter design was chosen due to its accuracy and reliability. Regarding the finite state machine used in the assembly code, timing and functionality was a concern. Two main hypotheses surfaced that the team voted upon; we could either implement a macro subroutine to use for each state or hard code the state functionalities one by one. Due to the fact that the idea of a finite state machine implemented separately was explained to us in class at a later date after most of the code was completed, it would be difficult to fully integrate it in the current code, thus we considered using a simple macro schematic to cut down on lines of code written in the main program. Both options were tested and individual state coding showed superior performance as it made debugging each state much simpler without compromising other states.

## 2.2 Investigation Design

To test the aforementioned ideas and to choose between one or the other, investigations and tests were designed for each idea to test their plausibility to implement. During the investigation process, we utilized resources online to understand ADC values and how hexadecimal numbers differ from bcd

numbers. In addition, we inquired to professor Jesus Calvino Fraga about the process of passing information from Python to a microcontroller via the serial port. He confirmed our hypothesis and gave us guidance as to how to correctly pass on such values. Different versions of the code were then created and tested for the thermocouple value conversion. As for the finite state machine, it was inferred through the official 8051 assembly style guideline that our hypothetical finite state machine using macro subroutines was a plausible design to implement. Thus we began by coding the two different versions of the finite state machine as well to be tested.

### 2.3 Data Collection

During testing, we collected some data or reported some observations which we used to complete the two versions of each idea for accuracy and reliability. For the thermocouple, we created two separate codes (one with and one without Python calculation) using the AT89LP51RC2 microcontroller to output our calculated temperature on the liquid crystal display (LCD) against the thermometer as well as against the expected ramp soak temperature at a given time when we integrate the oven controller to the temperature unit. As for the finite state machine, we created multiple versions of the same finite state machine using different codes (one with macro and one without) with a light-emitting diode (LED) on P3.7 of the AT89LP51RC2 microcontroller as the dependent variable, as the LED shines at different speed and brightness in different states and thus can be practical in debugging.

### 2.4 Data Synthesis

In order to generate more reliable data collected from the thermocouple testing, the code was restructured and the resistors were rearranged and connected to our operational amplifier (op-amp) to adjust the amplification until there was little to no discrepancies in our average temperature calculation. Initially, the data demonstrated a great deviation (approximately a 10℃ jump) in the temperature reading per unit time (approximately 18 clock cycles on the MCP3008 10-bit, 8-channel ADC). A solution was

found to remove potential noise causing the readings disturbances by taking averages of 100 readings taken in quick succession. In addition, the LED flickering sync with our timer signified a possible delay, aiding in debugging and ensuring consistency when adding or removing code without disrupting the oven control. At a later stage in process, the LED flickering frequency was synced to that of the desirable frequency for power output. The oven power output from the Solid State Relay (SSR) box was connected to P0.0 of the LPC9351 microcomputer, with the LED displaying different states of power impulses from 0% to 20% to 100% power outputs.

### 2.5 Analysis of Results

The result of the investigation step helped shape the final project design. We used this data to compare the two versions of the thermocouple conversion code and came to the conclusion that it was slightly more preferable to use an external Python calculation given that the reading could be graphed on the personal computer screen for ease of data collecting. Furthermore, it was easier to debug the code when issues would arise using the external Python code. The error within the thermocouple readings were reduced to below +/-3°C of error with an average error margin of +/-0.89°C and a maximum error of +/-2.1°C, therefore adhering to the project guidelines. Although there was no inherent issue with implementing the macro, it would be difficult to customize each state of the state machine to different power output and timer limit. As a result, we opted to hardcoding each individual state rather than using a macro that accounts for similar functions within each state.Throughout the investigation process, decisions on fundamental structures of the project were made regarding the thermocouple and finite state machine assembly code.

# 3.0 Design

## 3.1 Use of Process

Different aspects of the project were considered individually before their integration into the final design. Ideas and solutions for individual components of the project were first investigated, evaluated, and optimized appropriately following an engineering problem solving process before being implemented. Features to include in the controller were first listed and brainstormed, while ensuring that we would still have the correct number of pins on the P89 microprocessor for the prioritized design goals. We looked into the capabilities and restraints of 8051 assembly language as it pertained to this project. Dealing with the jump instruction index bounds limitation proved to be challenging as we wanted to keep our code simplified. A finite state machine was suggested to manage the reflow process and pushbuttons adjusting the reflow parameters and LCD display mode. We prototyped, and planned our development of the necessary code through the use of AT89 microprocessors, before migrating to P89 microprocessors. Testing our system involved performing trial runs of the reflow parameters, as well as ensuring that our temperature readings were accurate to within reasonable error.

## 3.2 Need and Constraint Identification

A 1500 watt oven toaster was to be controlled by a Reflow Oven Controller, designed and assembled by our team, and was to be controlled by a microcontroller that is not part of the AT89LP series from Atmel. Following the reflow process, the oven must be able to solder components onto a PCB board. Controlling the oven power required sending a variable square wave signal that would control the power output, a method known as Pulse-Width Modulation (PWM). The chosen microcontroller was to be programmed in assembly language and incorporated an interactive, updating LCD display with parameters such as soak and reflow temperature. Voice feedback was required to provide the oven temperature every 5 seconds, with an audible readout. We generated sound files that stored the audio for

numbers and stages of the reflow process. The interactive LCD display was controlled through pushbuttons, allowing for the parameters of the reflow process to be adjusted. The functionality of this machine was to controllably heat a printed circuit board, which would solder the applied components such as capacitors and microprocessors to the board.

### 3.3 Problem Specification

Our task was to make a reflow oven controller that would carefully heat up and hence attach components to the solder pads of an EMF8 PCB. This could be tested by inserting a piece of paper into the oven, and observing its lightly roasted colour. The oven used in this process had to maintain set soak and reflow temperatures for specified time periods for each cycle of the reflow process (see Appendix B). Our controller had to be able to vary the amount of power delivered to the reflow oven via a SSR box[1]. A finite state machine was necessary to move through each state of the reflow process, using different settings and requirements. The general specifications of each state of the reflow process is outlined below. The optimal settings were later found through testing, and are thoroughly summarized in Appendix C.

| | **State 0 Setup** | **State 1 Ramp to Soak** | **State 2 Soak** | **State 3 Ramp to Peak** | **State 4 Reflow** | **State 5 Cooling** |
|---|---|---|---|---|---|---|
| **Temperature** | 0°C | Rising 1-3°C/sec | 150°C+/-20°C | Rising 1-3°C/sec | ~217°C | Decreasing to RT |
| **Duration** | NA | ~120 sec | 60-120 sec | 45-75 sec | 25-45 sec | N/A |

Table 3.3.1 General Reflow Settings

[1] Jesus Calvino-Fraga, “Project 1 - Reflow Oven Controller 2020”, University of British Columbia, 2020

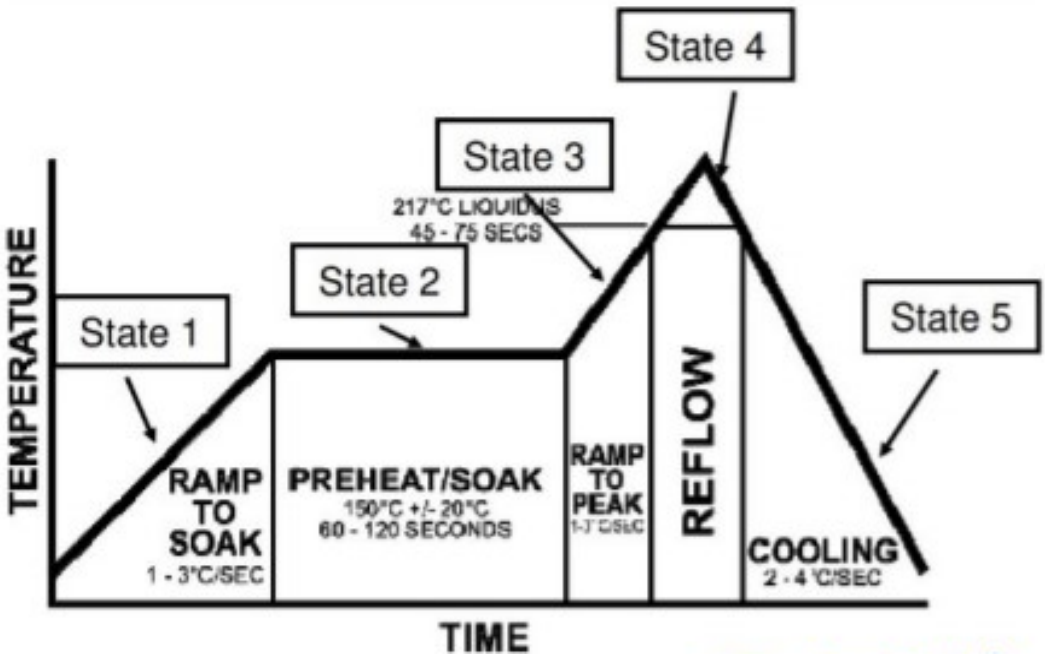



Figure 3.3.1 Reflow Process

Further information can be observed in Appendix B, depicting a full diagram of the reflow process as well as a figure demonstrating the different states of finite state machine in Appendix C. Temperature was to be read by inserting one end of a k-type thermocouple with cold junction compensation into the oven during the reflow process. In order to get temperature feedback, we had to setup and calibrate a thermocouple to read temperatures between 25℃ and 250℃ with a maximum error of ±3℃. As a safety feature , if upon starting the process, the temperature read by the thermocouple did not rise to at least 60C within the first 60 seconds of operation, the controller had to terminate the process.

### 3.4 Solution Generation

Two potential solutions were considered for varying the power delivered to the SSR box to control the oven power output:

1. Pulse width modulation (PWM) as discussed in class, outputting a digital signal allowing to reduce the power loss. PWM signal is a method for generating an analog signal using a digital source to control the oven.

2. Alternatively switching it on full power for a few seconds and then off for a few seconds based on temperature feedback. This allows a more accurate temperature control, but is less reliable than the pulse width modulation method due to error margins.

Two options were also presented for the buttons and pin management used in setting up soak and reflow time and temperature and switching displays:

1. Connecting each pushbutton to a different pin of the P89LPC9351 microcontroller. This method is the simplest one as it simply involves assigning a single pin to a push button at a time.
2. Connecting in parallel with a resistor to a single pin of the P89LPC9351 microcontroller. This method shows better pin management and allows the use of more buttons than the latter by assigning one pin to multiple buttons. However, additional code is required to define the voltage divider push buttons connected to an ADC channel. It is a newer and less intuitive method.

Regarding the voice feedback functionality, two different solutions were brainstormed to enable the feature every 5 seconds as requested in the guidelines.

1. Creating a separate bcd counter counting down from 5 seconds and enabling voice feedback of the current temperature every time the timer reaches zero before resetting the counter. This option proved to be more complex as a new timer, timer 0 was to be initialized, though it was unsure whether it could be used for this purpose, as it was not mentioned in class.
2. Using the same bcd counter as the soak and reflow timers, counting down seconds. In this option, we would take the counter in question with modulus of 5 and compare the resulting value to zero. When the resulting value would be equal to zero, voice feedback would be enabled. This function was implemented, but ultimately failed to perform during the testing portion.

### 3.5 Solution Evaluation

After taking into consideration both options for the oven power delivery, a combination of the two solutions was designed. The second option required the oven to be run at full power for longer intervals and led to high temperature fluctuations, however combining it with PWM allowed not only full control of delivered power, but also more security in ensuring that the temperature remained within set boundaries by constantly comparing the current temperature to the goal temperature.

Regarding the two possible methods of pin management with push buttons, option one was chosen, as we had more practice in working with single pin assignments, and had no shortage of pins and therefore no need to attach further pins in parallel, adding unneeded and cluttering code. The six buttons accomplished all functions required to set, start, stop, and change displays of the LCD.

Both options for voice feedback were attempted, however neither a second timer nor manipulating the current bcd counter were fully functional. The latter method was implemented and tested, however it did not deliver in time. Alternatively, voice feedback was enabled at the push of a button, stating the current temperature during the soak and reflow states. This function did not meet the project criteria, as it was required to state the current temperature every 5 seconds.

### 3.6 Safety and Professionalism

To avoid any safety issues, proper lab safety practice was used throughout the project timeline, specifically when soldering and using the oven. Hair was tied neatly, closed shoes were worn, and lab benches were cleaned and wiped neatly after each use. When soldering in the lab, the team ensured to wear safety goggles and made others aware when soldering. When using the oven to test the reflow process, pliers were used to take out paper pieces. To prevent fire hazards, the oven, soldering iron, and other lab equipment were turned off promptly after each use.

## 3.7 Detailed Design

**Hardware Block Diagram:**

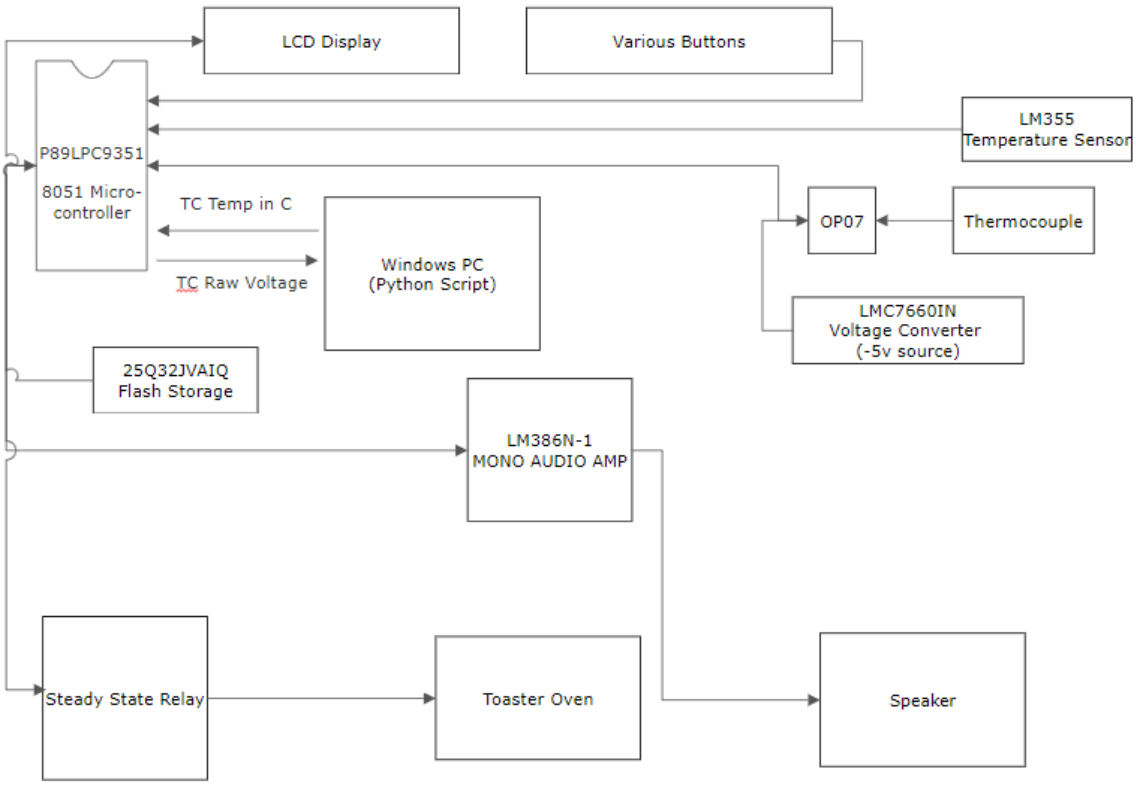



Figure 3.7.1 Hardware Block Diagram

**Circuit Explained:**

The circuit we utilized to complete our design can be found outlined in the following hardware block diagram. Everything is controlled by the P89LPC9351 microcontroller, with several branches expanding from there to incorporate smaller elements which all perform a specific task. The first main branch is the one responsible for collecting temperature data. First, the LM355 is connected to the onboard Analog to Digital Converter (ADC), returning the temperature of the cold junction of the thermocouple. This is required because the thermocouple measures a difference in temperature between the tip and the cold junction, therefore to get the total temperature we must add the cold junction temperature to the one received from the thermocouple. The thermocouple voltage is first amplified by a OP07 OpAmp due to its small output being too far out of spec to be read by the ADC directly. The OP07

requires both a +5V and -5V input, to supply -5V, we used a LMC7660IN voltage converter which takes a supplied 5V and outputs -5V. Once the voltage is read from the thermocouple, it is sent to a windows computer via the USB serial interface, and a python script converts the voltage into a temperature in Celsius[2]. We did the conversion this way to allow for higher accuracy, using the same function provided in the multimeter temperature reading python script shown in class (see Appendix D and E for full python script). This function is based on a table of results that the National Institute of Standards and Technology (NIST) created while testing K-type thermocouples, which we are using for our controller. This value is sent back to the microcontroller to be referenced for control logic. The next branch relates to the speaker and voice feedback system. To store the audio files a 25Q32JVAIQ flash storage chip is connected via the Serial Peripheral Interface (SPI) to the P89 microcontroller. The speaker is connected first to a LM386N-1 mono audio amplifier, then to the DAC on the p89 microcontroller. The last branches are smaller, one being a LCD display to display information about the reflow process and parameters, and a connection to the provided Steady State Relay (SSR) which allows us to switch the power to the toaster oven on or off.

[2] Jesus Calvino-Fraga, Project 1 - EFM8 board, FSM, EEPROM, and tips. University of British Columbia, Vancouver 2020

**Software Block Diagram:**

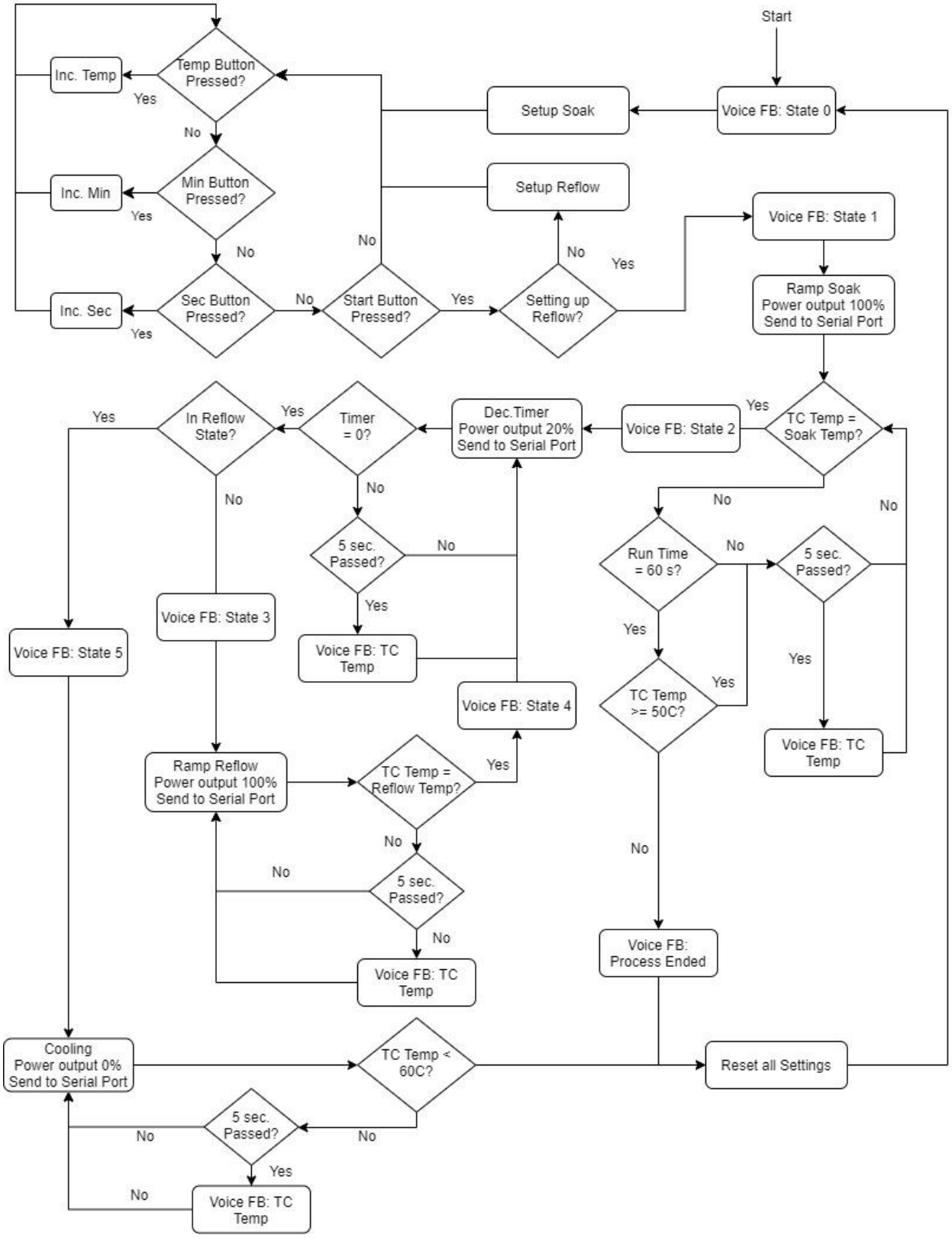



Figure 3.7.2 Software Block Diagram

**Software Explained:**

After flashing the program to the board, the microcontroller automatically follows a state machine as shown in Appendix C, beginning with state 0. At State 0, the buttons for temperature, seconds and minutes are checked in a loop continuously to see if they are pressed. When pressed, they increment their individual counters. After the soak settings are set, a fourth button, the start button, is pressed to flag the start of reflow settings. The reflow settings are set in a similar way, and once again the start button is pressed to proceed to state 1, enabling voice feedback and beginning the reflow process with ramping to the soak state depicted in Appendix B. The power is set to 100% output, and the increasing temperature repeatedly compared to the set soak temperature, until the goal temperature is reached, proceeding to state 2 and enabling voice feedback. In state 2, power is set to 20%, the soak timer is enabled, counting down until zero and enabling voice feedback every five seconds. The timer is tracked and decremented until it reaches zero. Once the timer is over, a flag is set to signify the end of state 2, and state 3 ramp to reflow begins. Similarly to state 1, power is set to 100%, voice feedback is enabled and the temperature is constantly checked against the goal temperature until reached. Once reached, state 4 reflow begins, power is set to 20%, voice feedback is enabled and the reflow timer counts down to zero, after which state 5 cooling begins. The power is set to 0%, voice feedback is enabled, the oven remains in this state until oven temperature reaches below 60 degrees celsius. After the oven cools, the program resets to state 0 to set up soak and reflow settings. The full program for the reflow process can be observed in Appendix F, consisting of all 6 states and the functioning of the timer interrupts used to decrement the bcd counters.

### 3.8 Solution Assessment

**Temperature Data Testing:**

Experimental testing of our temperature results found that our P89 was receiving temperature data well within the allowed error range of ±3 degrees. Testing methodology consisted of using clamping multimeter probes (similar to those found on the lab oscilloscopes) attached to the nodes where the two

ends of the thermocouple were attached to the breadboard using the provided multimeter temperature conversion python script to display accurate thermocouple temperatures. Simultaneously we checked the values being displayed on our LCD on the P89 and recorded our findings in a table (Appendix A). It should be noted that the LM335 temperature sensor was used to find the cold junction temperature, and was calibrated using a generic thermometer. The cold junction temperature was also inputted into the python script used with the multimeter. Throughout our testing we did find that the error was always around one degree, and maxed out at around 2.1 degrees. There was a slight amount of human error involved with the tests due to the rapidly changing numbers, however it should be noted that we did not notice any large deviations that this error may have caused. The findings of the first test conducted of the first reflow process tested with the reflow oven  is outlined in the diagram below, where the temperature fails to stop at the set soak temperature or 150°C, indicating a bug in the temperature comparison code.

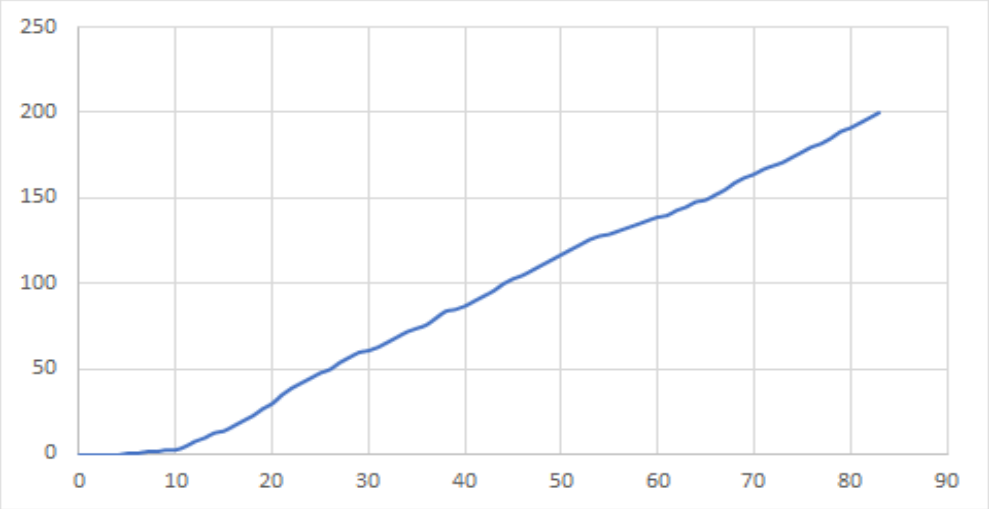



Figure 3.8.1.Ramp to Soak Bug, Temperature vs Time plot

The code was reviewed, and changes were made to compare both upper and lower byte of the bcd values of the current temperature and goal soak temperature as indicated in the code sample in Figure 3.8.2 below before proceeding to the Soak state.

```
; Compare upper byte
CompareUpperB_S1:
	mov bcd+0,SoakTemp+0
	mov bcd+1,SoakTemp+1
	mov bcd+2,#0
	mov bcd+3,#0
	mov bcd+4,#0
	lcall bcd2hex
	mov a, x+1 ;SoakTemp+1
	clr c
	subb a,Result+1	;Soak-Temp
	jnc CompareLowerB_S1		;SoakTemp>Result UP, check LB, else end
state
	ljmp End_S1
CompareLowerB_S1:
	mov a, x+0 ;SoakTemp+0
	clr c
	subb a,Result+0
	jnc jumperepreprepepre ; if SoakTemp<Result LB, loop, else end state
; If Soak Temp reached, proceed
```

Figure 3.8.2 Sample Temperature comparison code

After fixing most bugs such as temperature comparison, and successful soak and reflow state testing timer completion, the entire reflow process was once again tested, successfully reaching the cooling state. The settings were set for soak at 150°C for 60 seconds and reflow at 220°C for 45 seconds as outlined in the class notes[3].

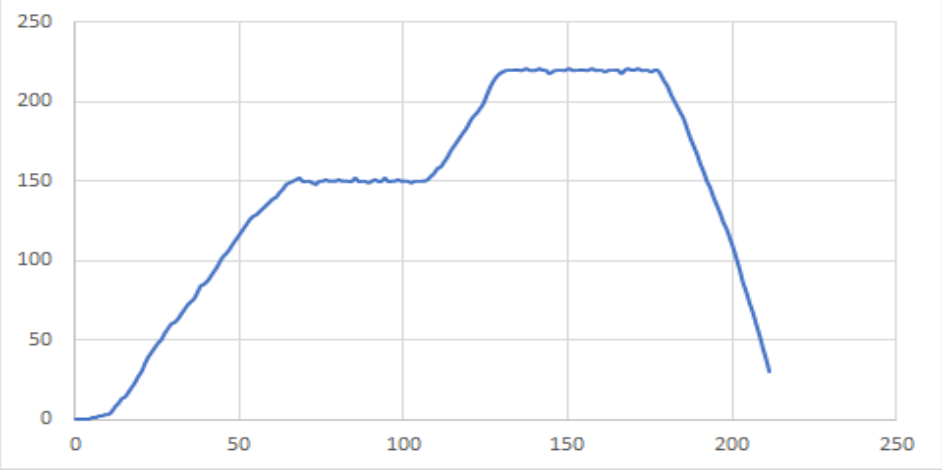



Figure 3.8.3 Complete Reflow Process, Temperature vs Time plot

[3] Jesus Calvino-Fraga, Project 1 - EFM8 board, FSM, EEPROM, and tips. University of British Columbia, Vancouver 2020

When placing a piece of paper in the oven, the test yielded in the paper a lightly brown roasted colour, shown in the far left of Figure 3.8.4, signifying a successful reflow process.

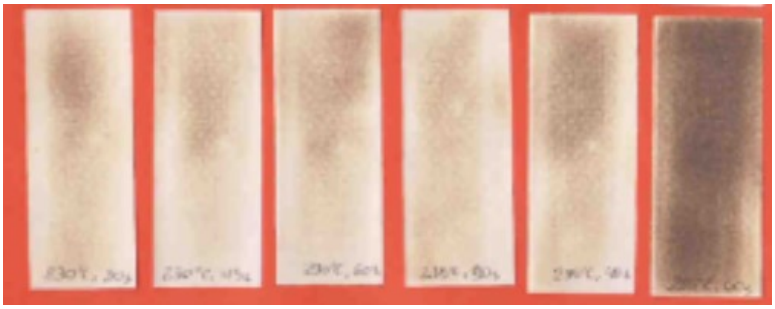


Figure 3.8.4 Reflow Oven Process Paper Test

# 4.0 Lifelong Learning

This project has been very valuable as a learning opportunity for everyone on the team. Everyone benefited from different aspects throughout the process. For instance, we learnt a lot from each other. Many were new to Github and had to learn its interface in order to communicate our ideas back and forth and debug each other's code in an orderly manner. Those with a little more experience with Github were able to help us navigate our way through the system and optimize our communication efficiency using the tool. Likewise, we were able to apply material learned in prior courses. Most have had a background from the course CPEN 211 last semester, which was beneficial for understanding the ins and outs of assembly languages, and for quickly being able to find the root of compilation error messages. This course was also helpful in analysing and understanding the raw code provided by Dr. Calvino-Fraga in class as well as integrating our own solutions into the provided code.

Moreover, some of us had to teach ourselves how to quickly navigate through large chunks of code in our source file or through a clutter of text in the data sheets to find what we are looking for. In fact, there were quite a few instances of self learning during our creative process. Those of us who are not as familiar with the syntax of assembly language in 8051 microprocessors utilized online discussion

boards to see real life examples of some of the instructions being used, we then inferred from those examples ways we could use the instruction to reconstruct the desired functionality.

Ultimately, these experiences throughout this project will have lasting effects on our work efficiency in the future. Be it another group project or not, we have learnt tools to use for better communication between our teammates, we have applied classroom knowledge directly to our project, and have learnt how to deliver efficient and quality work in a timely manner by utilizing internet resources along with our data organization skills.

# 5.0 Conclusion

This report has discussed the development of a reflow oven controller with voice feedback displaying information on an LCD screen. The objectives of this lab were to assemble the necessary hardware and write the necessary software to control a reflow oven to safely solder an EFM8 PCB. Different aspects of the project were considered individually before their integration into the final design. Each feature and individual components of the project were first investigated, evaluated, and optimized appropriately following an engineering problem solving process before being implemented. Some of these features include a safety feature, controlling the termination of the reflow process if the temperature read by the thermocouple did not rise to at least 60°C within the first 60 seconds of operation. The project objectives were met and six PCB's were successfully soldered using our system.

The designed oven controller follows the reflow model displayed in Appendix B and uses the states outlined in Appendix C in its state machine, namely State 0 to set up soak and reflow settings, state 1 to ramp the temperature up to soak, state 2 to preheat at soak temperature, state 3 to ramp the temperature up to reflow, state 4 at peak reflow temperature, and lastly state 5 to cool the oven back to room temperature.

To make a countdown timer for each stage, we used the built in timer in the P89LPC9351 microcontroller. Using a 16-bit variable and a 1-bit flag, we were able to calculate how many seconds passed, and by adding several “compare” assembly functions, a countdown timer was successfully implemented. To measure room and current temperature, an LM335 was used with a thermocouple and an ADC converter. To convert the values read by the thermocouple to the actual temperature, a python program was used to calculate, and send the temperature back to the board to be displayed onto the LCD as well as on a live plot run with Python.

To generate the sound files for the speaker, sound files were retrieved from example code provided in class and combined into a wav. file. Correct configurations were also adjusted to match the speaker. These two steps were completed using a software named Audacity. However this step proved to be more difficult than previously thought. Two different methods of implementation were designed and tested. The first consisted creating a separate bcd counter counting down from 5 seconds and enabling voice feedback of the current temperature every time the timer reaches zero before resetting the counter, while the second consisted in using an existing bcd counter and taking the modulus of 5 before compare the resulting value to zero. However neither options were fully functional. The latter method was implemented in the code, but as time became a concern, it was not debugged in time. Alternatively, voice feedback was enabled at the push of a button, stating the current temperature during the soak and reflow states. This project encouraged a heavy use of the assembly coding skills with microcontrollers that we had practiced in prior labs with the practical as well as theoretical knowledge about operational amplifiers and transistors investigated last semester. This project required several weeks, including a few all nighters of work debugging assembly code in order to meet the project deadline. Approximately 70 hours of work was done on this project, most of which were spent debugging current problems in the finite state machine, thermocouple readings, or voice feedback.

# 6.0 References

Calvino-Fraga, Jesus, “Project 1 - Reflow Oven Controller”, University of British Columbia, Vancouver, 2020

Calvino-Fraga, Jesus, “Project 1 - EFM8 board, FSM, EEPROM, and tips”, University of British Columbia, Vancouver, 2020

# 7.0 Bibliography

Microchip Technology Incorporated, “2.7V 4-Channel/8-Channel 10-Bit A/D Converters with SPI Serial Interface”, MCP3004/3008 datasheet, 2008

ARM Ltd. “8051 Instruction Set Manual.” 8051 Instruction Set Manual: Instructions, www.keil.com/support/man/docs/is51/is51_instructions.htm.

Atmel Corporation, “8-bit Flash Microcontroller with 24K/32K bytes Program Memory”, AT89LP51RB2 AT89LP51RC2 AT89LP51IC2 Preliminary, 2011

National Institute of Standards and Technology (NIST), “ITS-90 Table for type K thermocouple”, 2008

Texas Instruments, “LMx35, LMx35A Precision Temperature Sensors”, LM335, datasheet, 2015

Texas Instruments,”LMC7660 Switched Capacitor Voltage Converter”, LMC7660, datasheet, 2013

Texas Instruments, “LM386M-1/LM386MX-1 Low Voltage Audio Power Amplifier”, LM386N-1, datasheet, 2017

Analog Devices, “Ultralow Offset Voltage Operational Amplifier”, OP07, datasheet, 2011

NXP, “P89LPC9331/9341/9351/9361 8-bit microcontroller with accelerated two-clock 80C51 core, 4 kB/8 kB/16 kB 3 V byte-erasable flash with 8-bit ADCs”, P89LPC9351, datasheet, 2012

Winbond Electronics Corporation, “3V 32M-BIT SERIAL FLASH MEMORY WITH DUAL, QUAD SPI”, 25Q32JVAIQ, datasheet, 2014

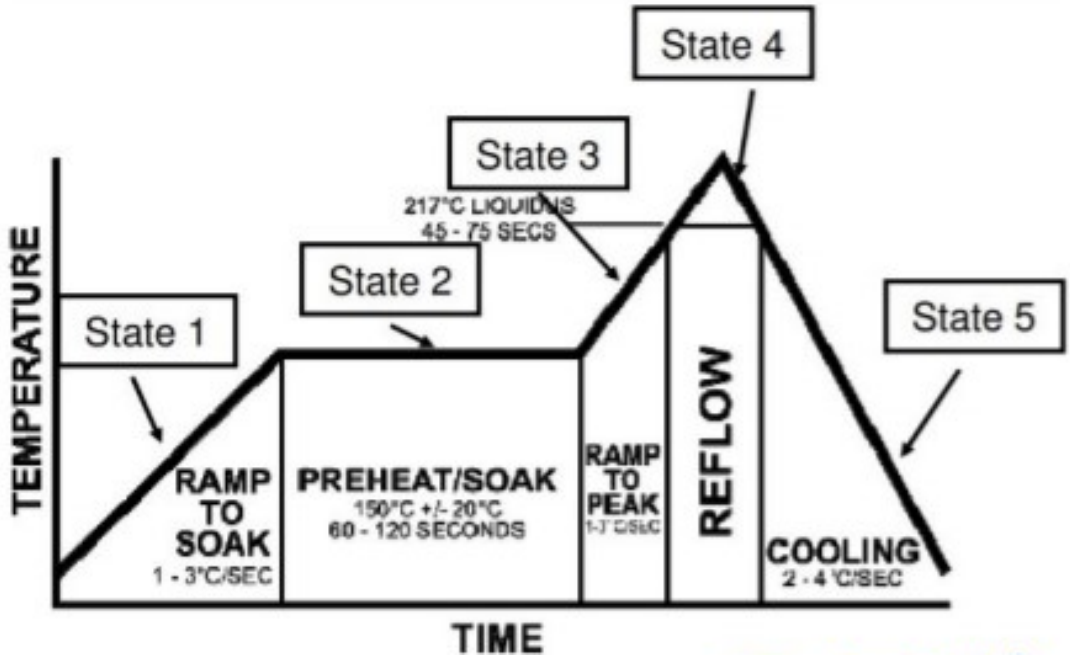
# 8.0 Appendices

## Appendix A: Temperature Validation Data

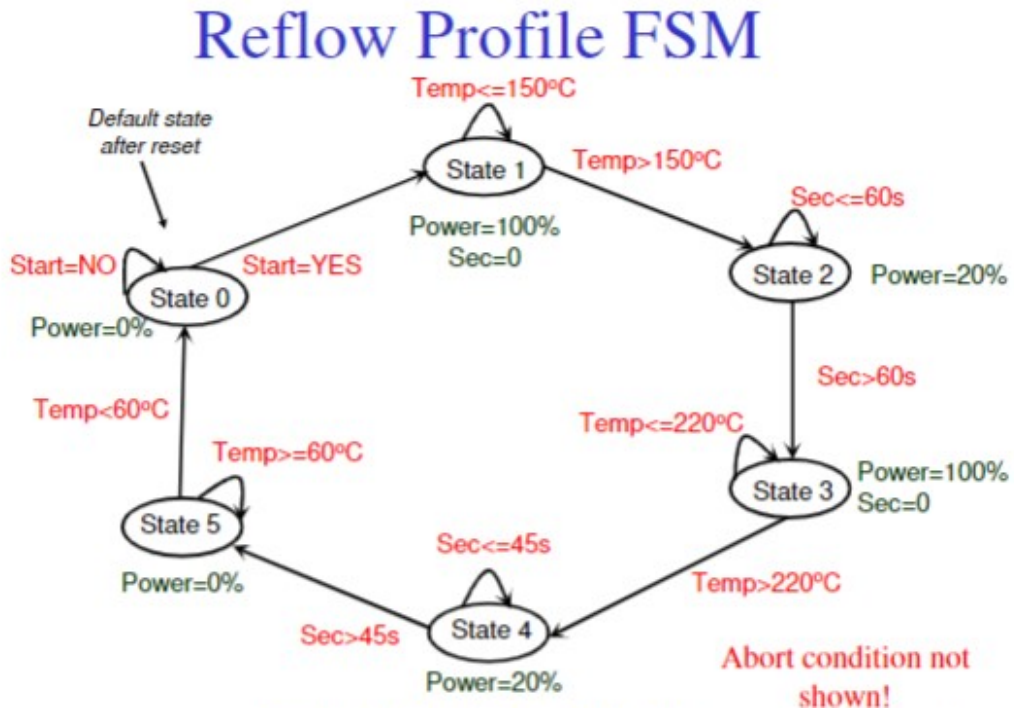
| Multimeter Temp | Thermocouple Temp | Absolute Error (#of Degrees) | MAXIMUM ERROR (#of Degrees) |
|---|---|---|---|
| 27.4 | 28 | 0.6 | 2.1 |
| 31.2 | 31 | 0.2 | AVERAGE ERROR (#of Degrees) |
| 33.1 | 32 | 1.1 | 0.89 |
| 34.5 | 35 | 0.5 | |
| 36.3 | 37 | 0.7 | |
| 41.3 | 41 | 0.3 | |
| 42.4 | 43 | 0.6 | |
| 44.7 | 46 | 1.3 | |
| 49.4 | 50 | 0.6 | |
| 52.1 | 53 | 0.9 | |
| 54.2 | 56 | 1.8 | |
| 55.4 | 57 | 1.6 | |
| 58.2 | 59 | 0.8 | |
| 60.5 | 61 | 0.5 | |
| 62.5 | 63 | 0.5 | |
| 64.7 | 66 | 1.3 | |
| 67.7 | 69 | 1.3 | |
| 71.3 | 73 | 1.7 | |
| 75.9 | 77 | 1.1 | |
| 79.6 | 80 | 0.4 | |
| 83.9 | 86 | 2.1 | |
| 88.6 | 90 | 1.4 | |
| 95.1 | 96 | 0.9 | |
| 103.6 | 104 | 0.4 | |
| 107.7 | 108 | 0.3 | |
| 111.9 | 113 | 1.1 | |
| 115.5 | 117 | 1.5 | |
| 120.9 | 121 | 0.1 | |
| 125.7 | 127 | 1.3 | |
| 132.7 | 133 | 0.3 | |
| 136.8 | 138 | 1.2 | |
| 143.5 | 144 | 0.5 | |
| 144.5 | 145 | 0.5 | |

| | | | |
|---|---|---|---|
| 149.3 | 150 | 0.7 | |
| 155.6 | 157 | 1.4 | |
| 160.9 | 161 | 0.1 | |
| 164.2 | 164 | 0.2 | |
| 168.2 | 170 | 1.8 | |
| 174.3 | 175 | 0.7 | |
| 180.7 | 180 | 0.7 | |
| 184.4 | 183 | 1.4 | |
| 187.3 | 188 | 0.7 | |
| 192.5 | 192 | 0.5 | |
| 195.8 | 195 | 0.8 | |
| 199.9 | 199 | 0.9 | |
| 202.6 | 201 | 1.6 | |
| 204.9 | 204 | 0.9 | |
| 208.1 | 207 | 1.1 | |
| 212.2 | 211 | 1.2 | |
| 215.8 | 216 | 0.2 | |
| 218.9 | 220 | 1.1 | |
| 222.3 | 222 | 0.3 | |
| 224.7 | 224 | 0.7 | |
| 227.6 | 228 | 0.4 | |
| 229.9 | 229 | 0.9 | |
| 231.9 | 230 | 1.9 | |
| 232.4 | 232 | 0.4 | |
| 233.9 | 233 | 0.9 | |
| 236.3 | 236 | 0.3 | |
| 238.1 | 238 | 0.1 | |
| 240.6 | 239 | 1.6 | |
| 242.9 | 242 | 0.9 | |
| 244.5 | 244 | 0.5 | |
| 246.6 | 245 | 1.6 | |
| 248.8 | 248 | 0.8 | |
| 249.2 | 248 | 1.2 | |
| 248.8 | 248 | 0.8 | |
| 249.4 | 248 | 1.4 | |
| 250.5 | 249 | 1.5 | |
| 251.6 | 251 | 0.6 | |
| 252.1 | 251 | 1.1 | |
| 252 | 251 | 1 | |

## Appendix B: Reflow Process




## Appendix C: Finite State Machine



## Appendix D: Stripchart_Final.py

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import sys, time, math
import time
import serial
import struct
import kconvert
xsize=150

ser = serial.Serial(
 port='COM8',
 baudrate=115200,
 parity=serial.PARITY_NONE,
 stopbits=serial.STOPBITS_ONE,
 bytesize=serial.EIGHTBITS)
ser.isOpen()
def data_gen():
    t = data_gen.t
    while True:

        strin = ser.readline()
        t+=1
        val=float(strin)
     #  val/=1000
        val*=150
        val/=33000
        val*=3.3
        cj=0
        val/=255
        val*=1000
        val=round(kconvert.mV_to_C(val,cj),1)
        val =int(val)

        temp = str(val).zfill(4)+'\n'
```

```
        ser.write(temp.encode('ascii'))
        print(strin)
        print(val)
        val+=25
        yield t, val

def run(data):
    # update the data
    t,y = data

    if t>-1:
        xdata.append(t)
        ydata.append(y)
        if t>xsize: # Scroll to the left.
            ax.set_xlim(t-xsize, t)
        line.set_data(xdata, ydata)

    return line,

def on_close_figure(event):
    sys.exit(0)

data_gen.t = -1
fig = plt.figure()
fig.canvas.mpl_connect('close_event', on_close_figure)
ax = fig.add_subplot(111)
line, = ax.plot([], [], lw=2)
ax.set_ylim(0,250 )
ax.set_xlim(0, xsize)
ax.grid()
xdata, ydata = [], []


# Important: Although blit=True makes graphing faster, we need blit=False to prevent
# spurious lines to appear when resizing the stripchart.
ani = animation.FuncAnimation(fig, run, data_gen, blit=False, interval=100, repeat=False)
plt.show()
```

## Appendix D: Kconvert.py

```
import math

# Evaluate a polynomial in reverse order using Horner's Rule,
# for example: a3*x^3+a2*x^2+a1*x+a0 = ((a3*x+a2)x+a1)x+a0
def PolyEval(lst, x):
    total = 0
    for a in reversed(lst):
        total = total*x+a
    return total

def mV_to_C(mVolts, ColdJunctionTemp):
    total_mV=mVolts+C_to_mV(ColdJunctionTemp)
    if total_mV < ranges_mV_to_C[0]:
        return -200.1 # indicates underrange
    elif total_mV < ranges_mV_to_C[1]:
        return PolyEval(mV_to_C_1, total_mV)
    elif total_mV < ranges_mV_to_C[2]:
        return PolyEval(mV_to_C_2, total_mV)
    elif total_mV < ranges_mV_to_C[3]:
        return PolyEval(mV_to_C_3, total_mV)
    else:
        return 1372.1 # indicates overrrange

def C_to_mV(tempC):
    if tempC < ranges_C_to_mV[0] or tempC > ranges_C_to_mV[2]:
        raise Exception("Temperature out of range in C_to_mV()")
    if tempC < ranges_C_to_mV[1]:
        return PolyEval(C_to_mV_1, tempC)
    else:
        return PolyEval(C_to_mV_2, tempC) + a[0] * math.exp(a[1] * (tempC - a[2]) * (tempC -
a[2]))

# This section tests the correctness of the functions above by converting the range of
# temperatures from -199C to 1372C to millivolts, converting the millivolts back to
# temperature and then comparing the result with the original temperature.
if __name__ == '__main__':
    Num_Fails = 0
```

```
    Worst_Error = 0.0

    for TestTemp in range(-199, 1372):
        ThermocoupleVoltage = C_to_mV(TestTemp) # The thermocouple's voltage in millivolts
        ComputedTemperature=mV_to_C(ThermocoupleVoltage, 0)
        Current_Error=math.fabs(TestTemp-ComputedTemperature)
        if Current_Error > Worst_Error:
            Worst_Error = Current_Error
        if Current_Error > 0.05:  # According to the table the worst error could be 0.06
            print ("Failed at emperature:", TestTemp, "Got instead:",
round(ComputedTemperature,2))
            Num_Fails=Num_Fails+1
    print ("Test finished with ", Num_Fails, "failures(s). Worst error was: ", round(Worst_Error,
2))
```

# Appendix F: Assembly Code

```
$NOLIST
$MOD9351
$LIST

;DECLARATIONS
CLK         EQU 14746000  ; Microcontroller system clock frequency in Hz
CCU_RATE    EQU 22050     ; 22050Hz is the sampling rate of the wav file we are playing
CCU_RELOAD  EQU ((65536-((CLK/(2*CCU_RATE)))))
BAUD        EQU 115200
BRVAL       EQU ((CLK/BAUD)-16)

TIMER1_RATE   EQU 200     ; 200Hz, for a timer tick of 5ms
TIMER1_RELOAD EQU ((65536-(CLK/(2*TIMER1_RATE))))

; PINS INPUT OUTPUTS
FLASH_CE EQU P2.4
MY_MOSI EQU P2.2
MY_MISO EQU P2.3
MY_SCLK EQU P2.5

;Sound and power outputs
SOUND equ P2.7
```

```
POWER equ P2.6

;Boot Button
BOOT_BUTTON equ P2.6
;Temp Min Sec buttons
TEMP_BUTTON  equ P0.2            ; Inc temperature
ALMIN_BUTTON  equ P0.3 ; Inc minutes
ALSEC_BUTTON   equ P0.1          ; Inc seconds
;Start/Stop timer, Mode button
STARTSTOP_BUTTON equ P3.0    ; Start/Stop process immediately, Settings
MODE_BUTTON equ P3.1             ; Switch Displays between Clock, Current Temp,
Settings/timer

; Commands supported by the SPI flash memory according to the datasheet
WRITE_ENABLE     EQU 0x06  ; Address:0 Dummy:0 Num:0
WRITE_DISABLE    EQU 0x04  ; Address:0 Dummy:0 Num:0
READ_STATUS      EQU 0x05  ; Address:0 Dummy:0 Num:1 to infinite
READ_BYTES       EQU 0x03  ; Address:3 Dummy:0 Num:1 to infinite
READ_SILICON_ID  EQU 0xab  ; Address:0 Dummy:3 Num:1 to infinite
FAST_READ        EQU 0x0b  ; Address:3 Dummy:1 Num:1 to infinite
WRITE_STATUS     EQU 0x01  ; Address:0 Dummy:0 Num:1
WRITE_BYTES      EQU 0x02  ; Address:3 Dummy:0 Num:1 to 256
ERASE_ALL        EQU 0xc7  ; Address:0 Dummy:0 Num:0
ERASE_BLOCK      EQU 0xd8  ; Address:3 Dummy:0 Num:0
READ_DEVICE_ID   EQU 0x9f  ; Address:0 Dummy:2 Num:1 to infinite

CSEG

; Reset vector
org 0x0000
   ljmp MainProgram

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

; Timer/Counter 0 overflow interrupt vector
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
	ljmp Timer1_ISR

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

; Timer/Counter 2 overflow interrupt vector
org 0x005B
	ljmp CCU_ISR

; These register definitions needed by 'math32.inc'
DSEG at 0x30
w:	 ds 3
x:   ds 4
y:   ds 4
bcd: ds 5
buffer: ds 30

; THERMOCOUPLE
LM_Result: ds 2
TC_Result: ds 2
Result: ds 2
LM_TEMP: ds 2
; TEMPERATURE
SaveT: ds 4
goalTemp: ds 2
SoakTemp: ds 3		; set soak temperature
ReflTemp: ds 3		; set refl temperature
Display_Power: ds 2
; TIMER COUNTERS	; contains counters and timers
Count5ms: ds 1
Count1ms: ds 2		; Used to determine when (1) second has passed
```

```
BCD_counterSec: ds 1
BCD_counterMin: ds 1
BCD_runtimeSec: ds 1
BCD_runtimeMin: ds 1
; ALARMS
SoakMinAlarm: ds 1              ;contains set time values
SoakSecAlarm: ds 1
ReflMinAlarm: ds 1
ReflSecAlarm: ds 1


BSEG
mf: dbit 1
half_seconds_flag: dbit 1       ; Set to 1 in the ISR every time 1000 ms had passed (actually 1
second flag)
seconds_flag: dbit 1
timer_done: dbit 1              ; Set to 1 once ready to start countdown
refltimer_done: dbit 1          ; Set to 1 once refl timer starts
tempdisplay_flag: dbit 1        ; Set to 1 for temp and run time display
powerout_flag: dbit 1

CSEG
; These 'equ' must match the wiring between the microcontroller and the LCD!
LCD_RS equ P0.5
LCD_RW equ P0.6
LCD_E  equ P0.7
LCD_D4 equ P1.2
LCD_D5 equ P1.3
LCD_D6 equ P1.4
LCD_D7 equ P1.6

; LCD and Putty Strings
_Hello_World: DB 'Hello World!', '\r', '\n',0
_New_Line: DB '\r\n', 0
_Soak: DB 'Soak:',0
_Refl: DB 'Refl:',0
_Temperature_LCD: DB 'Temp:',0
_Power: DB 'Power:%',0
_C:     DB '000C',0
```

```
_blank: DB ' ',0
_default: DB '00:00',0
_clearLCD: DB '               ',0

$NOLIST
$include(LCD_4bit_LPC9351.inc)
$include(math32.inc)
$include(voice_feedback.asm)
$include(will.inc)
$include(tempcheck.inc)
$include(Jesus_stuff.inc)
;$include (Power_out.asm)
$LIST

EX1_ISR:
  clr ECCU
  reti

;---------------------------------;
; Routine to initialize the ISR   ;
; for timer 1                     ;
;---------------------------------;
Timer1_Init:
        mov a, TMOD
        anl a, #0x0f ; Clear the bits for timer 1
        orl a, #0x10 ; Configure timer 1 as 16-timer
        mov TMOD, a
        mov TH1, #high(TIMER1_RELOAD)
        mov TL1, #low(TIMER1_RELOAD)
        ; Enable the timer and interrupts
  setb ET1  ; Enable timer 1 interrupt
  setb TR1  ; Start timer 1
        ret

;---------------------------------;
; ISR for timer 1                 ;
;---------------------------------;
Timer1_ISR:
        mov TH1, #high(TIMER1_RELOAD)
```

```
	mov TL1, #low(TIMER1_RELOAD)

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

	; Increment the 8-bit 5-mili-second counter
	inc Count5ms

Inc_Done:
	; Check if half second has passed
	mov a, Count5ms
	cjne a, #200, Timer1_ISR_done ; Warning: this instruction changes the carry flag!

	; 1000 milliseconds have passed.  Set a flag so the main program knows
	setb seconds_flag ; Let the main program know half second had passed
	; Reset to zero the 5-milli-seconds counter, it is a 8-bit variable
	mov Count5ms, #0

	; decrement seconds
	mov a , BCD_counterSec
	add a, #0x99
	da a
	mov BCD_counterSec, a
	cjne a, #0x99, Timer1_ISR_done
	mov BCD_counterSec, #0x59
	;decrement minutes
	mov a , BCD_counterMin
	add a, #0x99
	da a
	mov BCD_counterMin, a
	cjne a, #0x99, Timer1_ISR_done	;If timer minutes at 0, set timerdone flag
	setb timer_done
	mov BCD_counterMin, #0x00

Timer1_ISR_done:
	pop psw
	pop acc
	reti
```

```
;----------------------;
;    MAIN PROGRAM      ;
;----------------------;
MainProgram:
   mov SP, #0x7F

   lcall InitSerialPort
   lcall Ports_Init ; Default all pins as bidirectional I/O. See Table 42.
   lcall InitADC0 ; Call after 'Ports_Init'
        lcall InitDAC1
   lcall LCD_4BIT
   lcall Double_Clk
        lcall CCU_Init; voice feedback interrupt
        lcall Timer1_Init
        lcall Init_SPI

        ; set/clear interrupts
        setb POWER
        clr TR1
        clr TMOD20 ; Stop CCU timer
        clr SOUND ; Turn speaker off
        clr T2S_FSM_Start
        mov T2S_FSM_state, #0
        setb EA ; Enable global interrupts.

        mov seconds, #0x00
        mov minutes, #0x00
   mov SoakTemp, #0x00
        mov ReflTemp, #0x00
        mov GoalTemp, #0x00
        mov SoakTemp+1, #0x00
        mov ReflTemp+1, #0x00
   mov SoakTemp, #0x00
        mov ReflTemp, #0x00
        mov BCD_counterSec, #0x00
        mov BCD_counterMin, #0x00
        mov SoakMinAlarm, #0x00
        mov SoakSecAlarm, #0x00
```

```
	mov ReflMinAlarm, #0x00
	mov ReflSecAlarm, #0x00

   ;set constant strings lcd
   Set_Cursor(1,1)
	Send_Constant_String(#_Soak)
	Set_Cursor(1,6)
	Send_Constant_String(#_C)
	Set_Cursor(1,10)
	Send_Constant_String(#_blank)
	Set_Cursor(1,11)
	Send_Constant_String(#_default)

	Set_Cursor(2,1)
	Send_Constant_String(#_Refl)
	Set_Cursor(2,6)
	Send_Constant_String(#_C)
	Set_Cursor(2,10)
	Send_Constant_String(#_blank)
	Set_Cursor(2,11)
	Send_Constant_String(#_default)

	ljmp State0_SetupSoak				; sets up all soak temp, time, refl temp, time
before counter start

;-------------------------------------;
;	STATE0 SET SOAK/REFL SETTINGS	;
;-------------------------------------;
;--------- SETUP SOAK ---------;
State0_SetupSoak:

	jb BOOT_BUTTON, SetSoakTemp  ; if the 'BOOT' button is not pressed skip
	Wait_Milli_Seconds(#50)	; Debounce delay.  This macro is also in 'LCD_4bit.inc'
	jb BOOT_BUTTON, SetSoakTemp  ; if the 'BOOT' button is not pressed skip
	jnb BOOT_BUTTON, $

	;Make LCD screen blink??

	clr a					; clear all settings
```

```
        mov SoakTemp, a
        mov SoakMinAlarm, a
        mov SoakSecAlarm, a
        lcall Display_Soak

        ljmp State0_SetupSoak          ;loops in Setup until Start button pressed

CheckReflSet:                   ; if startmode button pressed, set refl
        jb STARTSTOP_BUTTON, State0_SetupSoak
    Wait_Milli_seconds(#50)
    jb STARTSTOP_BUTTON, State0_SetupSoak
    jnb STARTSTOP_BUTTON, $
    ljmp State0_SetupRefl

SetSoakTemp:
        jb TEMP_BUTTON, SetSoakMin ; if 'soak min' button is not pressed, check soak sec
    Wait_Milli_seconds(#50)
    jb TEMP_BUTTON, SetSoakMin
    jnb TEMP_BUTTON, $

    ; increment Soak temp
        mov a, SoakTemp
        cjne a, #0x90, dontincrementhigherSOAK
incrementhigherSOAK:
        mov a, SoakTemp+1
        add a, #0x01
        da a
        mov SoakTemp+1, a
dontincrementhigherSOAK:
        mov a, SoakTemp
        add a, #0x10
        da a
        mov SoakTemp, a
        clr a
        lcall Display_Soak
        ljmp State0_SetupSoak

SetSoakMin:
        jb ALMIN_BUTTON, SetSoakSec
```

```
   Wait_Milli_seconds(#50)
   jb ALMIN_BUTTON, SetSoakSec
   jnb ALMIN_BUTTON, $

        ; Now increment Soak min
        mov a, SoakMinAlarm
        cjne a, #0x59, incrementSM          ;if not equal to 59, add 1
        mov a, #0x00
        da a
        mov SoakMinAlarm, a
        clr a
        lcall Display_Soak
        ljmp State0_SetupSoak
incrementSM:
        add a, #0x01
        da a
        mov SoakMinAlarm, a
        clr a
        lcall Display_Soak
        ljmp State0_SetupSoak

SetSoakSec:
        jb ALSEC_BUTTON, CheckReflSet
   Wait_Milli_seconds(#50)
   jb ALSEC_BUTTON, CheckReflSet
   jnb ALSEC_BUTTON, $

        ; Now increment Soak sec
        mov a, SoakSecAlarm
        cjne a, #0x59, incrementSS          ;if not equal to 59, add 1
        mov a, #0x00
        da a
        mov SoakSecAlarm, a
        clr a
        lcall Display_Soak
        ljmp State0_SetupSoak
incrementSS:
        add a, #0x01
        da a
```

```
	mov SoakSecAlarm, a
	clr a
	lcall Display_Soak
	ljmp State0_SetupSoak

;--------- SETUP REFLOW    --------;
State0_SetupRefl:
	jb BOOT_BUTTON, SetReflTemp  ; if the 'BOOT' button is not pressed skip
	Wait_Milli_Seconds(#50)      ; Debounce delay.  This macro is also in 'LCD_4bit.inc'
	jb BOOT_BUTTON, SetReflTemp  ; if the 'BOOT' button is not pressed skip
	jnb BOOT_BUTTON, $

	;Make LCD screen blink??

	clr a
	mov ReflTemp, a
	mov ReflMinAlarm, a
	mov ReflSecAlarm, a
	lcall Display_Refl

	ljmp State0_SetupRefl          ;loops in Setup until Start button pressed

SetReflTemp:
	jb TEMP_BUTTON, SetReflMin ; if 'soak min' button is not pressed, check soak sec
   Wait_Milli_seconds(#50)
   jb TEMP_BUTTON, SetReflMin
   jnb TEMP_BUTTON, $
   ; increment Soak temp
	mov a, ReflTemp
	cjne a, #0x90, dontincrementhigherREFL
incrementhigherREFL:
	mov a, ReflTemp+1
	add a, #0x01
	da a
	mov ReflTemp+1, a
dontincrementhigherREFL:
	mov a, ReflTemp
	add a, #0x10
	da a
```

```
        mov ReflTemp, a
        clr a
        lcall Display_Refl
        ljmp State0_SetupRefl

SetReflMin:
        jb ALMIN_BUTTON, SetReflSec
    Wait_Milli_seconds(#50)
    jb ALMIN_BUTTON, SetReflSec
    jnb ALMIN_BUTTON, $

        ; Now increment Soak min
        mov a, ReflMinAlarm
        cjne a, #0x59, incrementRM          ;if not equal to 59, add 1
        mov a, #0x00
        da a
        mov ReflMinAlarm, a
        clr a
        lcall Display_Refl
        ljmp State0_SetupRefl
incrementRM:
        add a, #0x01
        da a
        mov ReflMinAlarm, a
        clr a
        lcall Display_Refl
        ljmp State0_SetupRefl

CheckStartTimer:            ; if modestart buttup pressed, start timer and main loop

        jb STARTSTOP_BUTTON, jumpercst
    Wait_Milli_seconds(#50)
    jb STARTSTOP_BUTTON, jumpercst
    jnb STARTSTOP_BUTTON, $

        ;------------------------ TODO ---------------------------;
        ; Voice Feedback Soak stage
        ;---------------------------------------------------------;
```

```
        ; temp stuff, clear bits
        clr a
        mov x+1,a
        mov x+2,a
        mov x+3,a

        ;------------------------ TODO ----------------------------;
        ; Change display to ramp soak?
        ;------------------------------------------------------------;
        mov goalTemp, SoakTemp              ;track current vs goalTemp
        mov BCD_CounterSec, #0x60
        setb TR1
        ljmp State1_RampSoak
jumpercst:
        ljmp State0_SetupRefl

SetReflSec:
        jb ALSEC_BUTTON, CheckStartTimer
    Wait_Milli_seconds(#50)
    jb ALSEC_BUTTON, CheckStartTimer
    jnb ALSEC_BUTTON, $

        ; Now increment Soak sec
        mov a, ReflSecAlarm
        cjne a, #0x59, incrementRS          ;if not equal to 59, add 1
        mov a, #0x00
        da a
        mov ReflSecAlarm, a
        clr a
        lcall Display_Refl
        ljmp State0_SetupRefl
incrementRS:
        add a, #0x01
        da a
        mov ReflSecAlarm, a
        clr a
        lcall Display_Refl
        ljmp State0_SetupRefl
```

```
;--------------------------------;
;                   STATE1 RAMP SOAK              ;
;--------------------------------;
State1_RampSoak:
 ; 100% power

        mov a, #0x60
        clr c
        subb a, BCD_CounterSec
        jnc Skip123

        clr TR1
        setb POWER
        ljmp State0_SetupRefl
        Skip123:

        lcall ReadTemp
        clr POWER

        jb MODE_BUTTON, SwitchDisplay_S1                ; if stop button not pressed, go loop
and check for 00
   Wait_Milli_seconds(#50)
   jb MODE_BUTTON, SwitchDisplay_S1
   jnb MODE_BUTTON, $

        mov a, Result
        mov b, #100
        div ab

        cjne a, #0, AmazonServices
        sjmp Fedex
AmazonServices:
        push b
        lcall Play_Sound_Using_Index
        jb TMOD20, $ ; Wait for sound to finish playing
        pop b
Fedex:
        mov a, b
        mov b, #10
```

```
        div ab
        push b
        lcall Play_Sound_Using_Index
        jb TMOD20, $ ; Wait for sound to finish playing
        pop b

        mov a, b
        lcall Play_Sound_Using_Index
        jb TMOD20, $ ; Wait for sound to finish playing

        jb tempdisplay_flag, TimerDisplayJmp2
        jnb tempdisplay_flag, TempDisplayJmp2

SwitchDisplay_S1:
        lcall ReadTemp
        mov Display_Power, #0x99    ;power at 100%

; Compare upper byte
CompareUpperB_S1:
        mov bcd+0,SoakTemp+0
        mov bcd+1,SoakTemp+1
        mov bcd+2,#0
        mov bcd+3,#0
        mov bcd+4,#0

        lcall bcd2hex

        mov a, x+1 ;SoakTemp+1
        clr c
        subb a,Result+1         ;Soak-Temp
        jnc CompareLowerB_S1                ; if SoakTemp>Result UB, check LB, else end state
        ljmp End_S1
CompareLowerB_S1:

        mov a, x+0 ;SoakTemp+0

        clr c
        subb a,Result+0
        jnc jumperepreprepepre ; if SoakTemp<Result LB, loop, else end state
```

```
; If Soak Temp reached, proceed
End_S1:
	mov BCD_counterMin, SoakMinAlarm	; move time settings into counters
	mov BCD_counterSec, SoakSecAlarm
	clr timer_done
	clr refltimer_done; clear timer done flags
	setb TR1				;Start Timer
	ljmp Forever
	;------------------------- TODO ------------------------------;
	; Implement safety feature (if Temp < 50C in first 60s, abort) ;
	;-------------------------------------------------------------;
jumperepreprepepre:
	ljmp State1_RampSoak
;----------------------;
;	JMP FUNCS	;
;----------------------;
TempDisplayJmp2:
	ljmp TempDisplay2
TimerDisplayJmp2:
	ljmp TimerDisplay2

;-------------------------------------;
;			STATE2&4 MAIN LOOP		;
;-------------------------------------;

; forever loop interface with putty
Forever:
	 ; 20% pwm for soak and refl

	; check temperature
	;lcall T2S_FSM
	lcall ReadTemp
	jb refltimer_done,ReadRefl

	mov bcd+0,SoakTemp+0
	mov bcd+1,SoakTemp+1
	mov bcd+2,#0
	mov bcd+3,#0
	mov bcd+4,#0
```

```
        sjmp skipRefl
ReadRefl:

        mov bcd+0,ReflTemp+0
        mov bcd+1,ReflTemp+1
        mov bcd+2,#0
        mov bcd+3,#0
        mov bcd+4,#0
skipRefl:
        lcall bcd2hex
        mov a, x+1 ;SoakTemp+1
        clr c

        subb a,Result+1         ;Soak-Temp
        jnc CompareLowerSTATE2
        ljmp POWER_STATE2;if soak<current temp, enable power

CompareLowerSTATE2:
        mov a, x+0 ;SoakTemp+0

        clr c
        subb a,Result+0
        jnc POWER_STATE2;if soak<current temp, enable power

        ;0% POWER
   setb POWER ; led off
   sjmp STATE2POWERSKIP

POWER_STATE2:
        ;20% POWER
        clr POWER ; Led on
   Wait_Milli_Seconds(#20)
   setb POWER ; led off
   Wait_Milli_Seconds(#80)
STATE2POWERSKIP:

        ; Voice Feedback
        ;lcall T2S_FSM                  ; Run the state machine that plays minutes:seconds
```

```
        mov Display_Power, #0x20    ;power at 20% for Soak and Refl Stages 2&4

        jnb seconds_flag, CheckButtons
        ; One second has passed, refresh the LCD with new time

        jb timer_done, TimerDoneJmp                 ;check if timer done
        clr seconds_flag
        jb tempdisplay_flag, TempDisplayJmp         ; if temp mode button pressed, show temp
display

        ljmp WriteNum

        ; Do this forever
        sjmp CheckButtons

CheckButtons:
        ; TIME CHECK
        jb BOOT_BUTTON, CheckStop  ; buttons to change screen to Clock and Current Temp
later
        Wait_Milli_Seconds(#50)
        jb BOOT_BUTTON, CheckStop
        jnb BOOT_BUTTON, $

        clr TR1                 ; Stop timer 2
        clr a
        mov BCD_counterSec, #0x00
        mov BCD_counterMin, #0x00
        lcall Display_Soak
        lcall Display_Refl

        ljmp State0_SetupSoak

;---------------------;
;      JMP FUNCS      ;
;---------------------;
TempDisplayJmp:
        ljmp TempDisplay
TimerDisplayJmp:
        ljmp TimerDisplay
```

```
ForeverJmp:
        ljmp Forever
TimerDoneJmp:
        ljmp TimerDone

; add another button for display that will loop to loop_a after
CheckStop:
   jb STARTSTOP_BUTTON, VoiceFeedback                    ; if stop button not pressed, go loop
and display
   Wait_Milli_seconds(#50)
   jb STARTSTOP_BUTTON, VoiceFeedback
   jnb STARTSTOP_BUTTON, $
   clr TR1                ; Stop timer 2
   clr POWER                              ; stop power
        ;------------------------ TODO ----------------------------;
        ; Turn off power oven
        ;-------------------------------------------------------------;
        ljmp State0_SetupSoak                    ; if stop button pressed, go back to setup

SwitchDisplays:
        jb MODE_BUTTON, ForeverJmp                ; if stop button not pressed, go loop and
check for 00
   Wait_Milli_seconds(#50)
   jb MODE_BUTTON, ForeverJmp
   jnb MODE_BUTTON, $

        mov a, Result
        mov b, #100
        div ab

        cjne a, #0, AmazonServices1
        sjmp Fedex1
AmazonServices1:
        push b
        lcall Play_Sound_Using_Index
        jb TMOD20, $ ; Wait for sound to finish playing
        pop b
Fedex1:
        mov a, b
```

```
        mov b, #10
        div ab
        push b
        lcall Play_Sound_Using_Index
        jb TMOD20, $ ; Wait for sound to finish playing
        pop b

        mov a, b
        lcall Play_Sound_Using_Index
        jb TMOD20, $ ; Wait for sound to finish playing

        jb tempdisplay_flag, TimerDisplayJmp
        jnb tempdisplay_flag, TempDisplayJmp
        ljmp Forever

VoiceFeedback:
        ; Voice Feedback
        jb TEMP_BUTTON, SwitchDisplays                  ; if stop button not pressed, go loop
and display
   Wait_Milli_seconds(#50)
   jb TEMP_BUTTON, SwitchDisplays

   mov seconds, BCD_counterSec
   mov minutes, BCD_counterMin
        setb T2S_FSM_Start  ; This plays the current minutes:seconds by making the state
machine get out of state zero.
        ljmp Forever

TimerDone:              ; if timer done
        clr TR1             ; Stop timer 2
        clr a
        mov goalTemp, ReflTemp              ;track current vs goalTemp
        jnb refltimer_done, State3_RampRefl                 ; if reflow timer not done, start
reflow timer
        ;else if refltimer done, finish process
        mov goalTemp, #0x00                 ;track current vs goalTemp
        ljmp State5_Cool                ; go to Cool state
```

```
;--------------------------------;
;                STATE3 RAMP REFL              ;
;--------------------------------;
State3_RampRefl:
        ; 100% power

        lcall ReadTemp
        clr POWER

        jb MODE_BUTTON, SwitchDisplay_S3            ; if stop button not pressed, go loop
and check for 00
   Wait_Milli_seconds(#50)
   jb MODE_BUTTON, SwitchDisplay_S3
   jnb MODE_BUTTON, $

        mov a, Result
        mov b, #100
        div ab

        cjne a, #0, AmazonServices2
        sjmp Fedex2
AmazonServices2:
        push b
        lcall Play_Sound_Using_Index
        jb TMOD20, $ ; Wait for sound to finish playing
        pop b
Fedex2:
        mov a, b
        mov b, #10
        div ab
        push b
        lcall Play_Sound_Using_Index
        jb TMOD20, $ ; Wait for sound to finish playing
        pop b

        mov a, b
        lcall Play_Sound_Using_Index
        jb TMOD20, $ ; Wait for sound to finish playing
```

```
        jb tempdisplay_flag, TimerDisplayJmp3
        jnb tempdisplay_flag, TempDisplayJmp3

SwitchDisplay_S3:
        lcall ReadTemp
        mov Display_Power, #0x99    ;power at 100%

; Compare upper byte
        mov bcd+0,ReflTemp+0
        mov bcd+1,ReflTemp+1
        mov bcd+2,#0
        mov bcd+3,#0
        mov bcd+4,#0

        lcall bcd2hex


CompareUpperB_S3:
        mov a, x+1
        clr c
        subb a, Result+1        ;Soak-Temp
        jnc CompareLowerB_S3                ; if SoakTemp>Result UB, check LB, else end state
        ljmp End_S3
CompareLowerB_S3:
        mov a, x+0
        clr c
        subb a, Result+0
        jnc State3_RampRefl ; if SoakTemp<Result LB, loop, else end state
; If Soak Temp reached, proceed
        ljmp End_S3
;---------------------;
;      JMP FUNCS      ;
;---------------------;
TempDisplayJmp3:
        ljmp TempDisplay
TimerDisplayJmp3:
        ljmp TimerDisplay

;--------------------------;
```

```
;                   STATE4 REFL             ;
;-------------------------;
End_S3:
        clr timer_done
        setb refltimer_done             ; set to indicate final stage in process
        mov BCD_counterMin, ReflMinAlarm        ; move time settings into counters
        mov BCD_counterSec, ReflSecAlarm
        clr timer_done
        setb refltimer_done; clear timer done flags
        setb TR1                        ;Start Timer
        mov goalTemp, ReflTemp
        ljmp Forever

;-------------------------;
;                   STATE5 COOLING      ;
;-------------------------;
State5_Cool:
;       pwn 0%

        lcall ReadTemp
        setb POWER              ;power off

        jb MODE_BUTTON, SwitchDisplay_S5                ; if stop button not pressed, go loop
and check for 00
   Wait_Milli_seconds(#50)
   jb MODE_BUTTON, SwitchDisplay_S5
   jnb MODE_BUTTON, $

        mov a, Result
        mov b, #100
        div ab

        cjne a, #0, AmazonServices3
        sjmp Fedex3
AmazonServices3:
        push b
        lcall Play_Sound_Using_Index
        jb TMOD20, $ ; Wait for sound to finish playing
        pop b
```

```
Fedex3:
        mov a, b
        mov b, #10
        div ab
        push b
        lcall Play_Sound_Using_Index
        jb TMOD20, $ ; Wait for sound to finish playing
        pop b

        mov a, b
        lcall Play_Sound_Using_Index
        jb TMOD20, $ ; Wait for sound to finish playing

        jb tempdisplay_flag, TimerDisplayJmp3
        jnb tempdisplay_flag, TempDisplayJmp3

SwitchDisplay_S5:
        lcall ReadTemp
        mov Display_Power, #0x00    ;power at 0%
; Compare upper byte
CompareUpperB_S5:
        mov a, Result+1
        clr c
        subb a, #0x00  ;Soak-Temp
        jnc CompareLowerB_S5                ; if SoakTemp>Result UB, check LB, else end state
        ljmp End_S5
CompareLowerB_S5:
        mov a, Result+0                 ;change to 0x60 later
        clr c
        subb a, #0x60
        jnc State5_Cool ; if SoakTemp<Result LB, loop, else end state
; If Cooling temp reached, proceed
;--------------------------;
;                   STATE4 REFL             ;
;--------------------------;
End_S5:
        clr TR1
        clr timer_done
        clr refltimer_done              ; set to indicate final stage in process
```

```
        ;reset all settings
        mov SoakTemp, #0x00
        mov ReflTemp, #0x00
        mov BCD_counterSec, #0x00
        mov BCD_counterMin, #0x00
        mov SoakMinAlarm, #0x00
        mov SoakSecAlarm, #0x00
        mov ReflMinAlarm, #0x00
        mov ReflSecAlarm, #0x00
        lcall Display_Soak
        lcall Display_Refl

        ljmp State0_SetupSoak

END
```