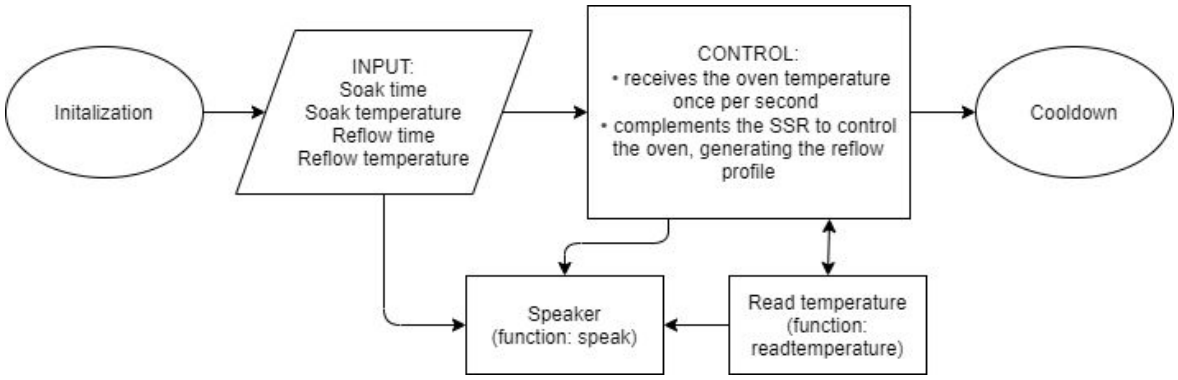
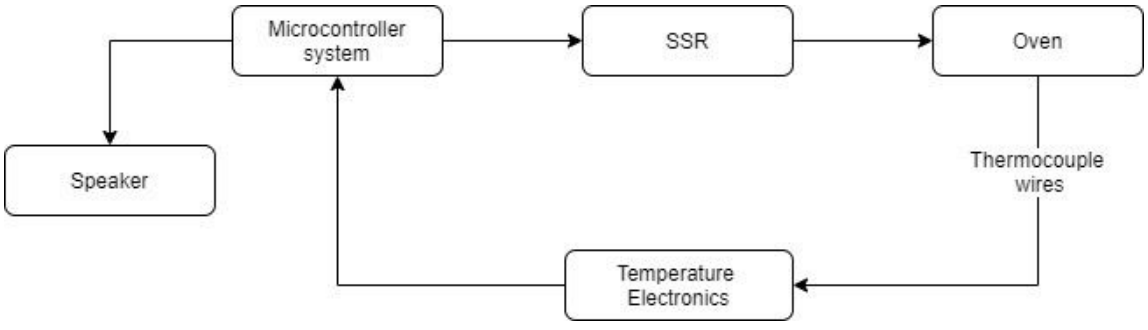
1.0 - INTRODUCTION

Reflow soldering is the process by which electrical components are connected to a PCB. This is achieved by applying soldering paste to the board and then heating it up to certain temperatures for a given period of time in a reflow soldering oven. Its reliability, speed and ease of use has made it a standard process for industrial and hobbyist electronics projects. Our goal was to develop a user-friendly controller capable of generating an optimal reflow soldering profile. In this report we detail our process, from initial investigations to our final design and implementation. Our design approach was guided by two objectives: optimizing development efficiency and maximizing end-user experience. The specifications should target ease-of-use and aesthetic from the user's perspective. As seen in the system block diagrams below, designing our hardware and software systems modularly enabled easy testing, development and integration:



Software



Hardware

2.0 - INVESTIGATION

2.1 - Idea generation

Before beginning the project, we divided it into three sections for efficient research: oven control, speaker, and circuit integration. Each member generated ideas for the assigned topic using information gathered from Canvas and the Internet. We then met and shared what we learned to ensure we all equally understood all aspects of the project. After the discussion, we integrated our ideas and came up with a working plan. To organize our thinking, we used a project workbook where we had rough flow sketches of the code for the whole team to continually refer to.

2.2 - Investigation design

A major component of our design investigation was experimentation. We needed to find a suitable resistor ratio for our op-amp circuit so that we could appropriately amplify the voltage signal from the thermocouple sensor. We initially chose too low a ratio so we did not get appropriate temperature readings. Other times, we inadvertently damaged our op-amp. After considerable experimentation, we modified our ratio - considering the combination of resistors we had at our disposal - ultimately settling on a ratio of 1MΩ to 3.3kΩ (gain: 303). This maintained the functionality of the circuit whilst providing sensible readings. Since we had more than 3 serial ports interacting with the circuit for bonus features, we judiciously allocated pins for the microcontroller to the ADC, arduino sensor, and the GUI [1]. This required designing the flow of the code in a way that incorporated all of these components. Furthermore, we designed basic tests such as switching on an LED whenever we successfully transitioned from one state to the next. These tests gave us dynamic feedback as we gradually built our code.

2.3 - Data Collection

We made extensive use of oscilloscopes and multimeters to diagnose problems with the breadboard

or equipment. To validate our design, we collected feedback from equipment like the op-amp by manually and periodically supplying it with power using the power supply.While we considered all experimental results, outliers were ultimately rejected, as they are negligible sources of error. We believe that data collection techniques go beyond just using appropriate coding and experimental practice. For instance, our ADC output was very noisy and the fluctuations made it practically unusable. Through calculation and experimentation, we determined a way to average 60 temperature values over a period of 1 second and transmit these averaged values every second to the LCD display and back to our state machine so that we could use the output signal. Moreover, we could only use one serial port at a time but needed three to implement our bonus, so we devised a method to sequentially transfer information to the stripchart and the complex GUI we developed for the oven. By innovating and applying appropriate lab safety standards, we were able to collect useful data.

2.4 - Data synthesis

Through experimentation, we determined the proper duty cycle for the soak and reflow states of the soldering process (for more information about our PWM[2] algorithm, see Section 3.7.3). Once we were able to index audio files and use them to speak temperatures, we also used the speaker to give us dynamic feedback for troubleshooting our main FSM. By using the feedback we got from the speaker, we were able to integrate the speaker code with the controller FSM. We also adjusted the clock frequency based on this feedback to synchronize the two FSMs, allowing smooth operation. Similarly, we used intermediate data throughout the stages of the project to help us make crucial decisions and ease debugging.

2.5 - Analysis of Results

We determined all sources of error on the hardware side (such as the thermocouple and the oven)

and factored their limitations into our code to optimize the reflow profile. First, we included a one unit error margin in all our state transition conditions to account for the sensitivity in the thermocouple. We also average the temperature readings from the thermocouple to account for fluctuations and error. By averaging every 255 values, our temperature readings became more stable. One physical limitation we encountered was the oven's accumulation of latent heat, which caused our algorithm to overshoot by as much as 20 °C. Using PWM resulted in a more stable and accurate reflow profile.. Our motion sensors are very sensitive so we included a delay to stabilize readings before transmitting. However, there were some unavoidable fluctuations due to random errors and minor experimental inaccuracies. Our preventative strategies resulted in a very accurate oven controller, as evidenced by our profiles (see Appendix 7.4).

3.0 - DESIGN

3.1 - Use of Process

Our group implemented rapid prototyping techniques in order to quickly develop and test ideas. We rapidly built circuits and software functions which were continuously tested and improved upon to achieve the final system. This helped us quickly solve complex, multi-faceted problems - as our tests immediately identified shortcomings in our design, thus pointing us directly to what we needed to improve. Each open-ended problem was sub-divided into tasks handled by teams of two, greatly reducing human error and decreasing debugging time.

For circuits, our main debugging strategy was continuity testing. This often revealed misplaced connections which we could quickly rectify. We also used the multimeter in the lab to check whether the voltages at element outputs were correct. Similarly, we used the function generator to generate

known inputs to elements, which - in tandem with measuring the outputs - helped us identify whether elements were working correctly.

Our design approach for software related tasks was similar. We used pair programming, in which our lead programmer would type and the “backseat” programmer would provide quality control. Commenting every 3-4 lines of code enabled other members to instantly understand unfamiliar code. For uniquely challenging problems, we allocated more members of the team to finding a solution, maximizing the team’s focus on solving our hardest challenges. We extensively used simple design components like LEDs to test the logic of the low level code, and used simple flags through serial communication to periodically test the higher level GUI code; examples of these tests are explored in Section 3.7 of the report.

3.2 - Needs and Constraint Identification

To identify needs and constraints, we analyzed two cases. First, we analyzed the industry standard accuracy for reflow soldering ovens. Based on research and lab documentation, we found this to be within +/- 3 °C, and improved this to within a 1 degree tolerance. Qualities such as ease of use, attractive design and cross-platform integration are all key to enhancing user experience. From an enterprise perspective, the controller needs to be easily manufacturable and cheap, which we ensured by using AutoCad designs and readily available equipment. Any constraints stemmed primarily from the sensitivity of the oven and the thermocouple but we were able to counteract this by taking this sensitivity into account in our software. Also, the temperature range of the oven is well within the capacity that the oven can handle so we were not limited by significant physical constraints besides experimental inaccuracies.

3.3 - Problem Specification

The project needs and constraints were primarily dealt with on the software side as it is cheaper than

hardware. To counteract the sensitivity factor from the equipment mentioned above, we used a local variable to sample averaged temperature values instead of sampling every second which fluctuated significantly more. To facilitate a user friendly interface and improve aesthetics, we implemented a Python GUI and used custom 3D printed components. We wanted our 3D printed solutions to be cheap, quickly manufactured and easily replaceable in case we want to change something within our system. These added a mechanical aspect to our design requirements.

3.4 - Solution Generation

Brief description of software and hardware design concepts that helped meet specifications:

Software design	Desirable outcomes
Oven Controller FSM: calls *readtemperature* (a local) function for condition checks to transition states and the *speak* FSM for sound integration.	States have individually defined temperature specifications, PWM instructions and set conditions for next state logic so it is easy to debug the project modularly.
Set_Value function: allows users to enter parameters which are processed using hundreds, units, and tens calculation to input in their correct numerical places. *lcall ADC_to_PB* function utilised here.	Selectable reflow profile parameters. Buttons make the input interface more user-friendly by entering each parameter step-by-step and enabling users to go back to change values.
Readtemperature block to output usable temperature values from voltage input from the thermocouple and display temperature on LCD using hundreds, tens and units calculation.	**Conversion of voltage to temperature** and subsequent transmission occurs with almost no fluctuations because we add temperatures to a local variable and then average that over 1000 ms before transmitting it.
Speak: Performs mathematical calculations to index and combine numbers responsible for playback.	It is designed while keeping the oven FSM in mind to match timings and help us interchange between the two without unexpected delays.

On the hardware side, we used a rocker switch to mechanically cut power from the oven as a safety feature to meet the automatic cycle termination specification. We also customized volume control to

the speaker using a potentiometer. We were able to extend the Python stripchart functionality to a GUI (tkinter library) which formed a key component of our bonus. We did run into some failures in the process, which we eventually solved. For instance, we initially had trouble with the speaker FSM because it was blocking, which didn't allow us to return to the oven FSM. We did a 'flashing LED test' to see if our code was getting stuck because of faulty branching. To solve this, we restructured our code to have less frequent calls to the speak function from the oven FSM.

3.5 - Solution evaluation

We have developed a very comprehensive reflow oven controller with a user-friendly and customizable Python GUI. The final design is aesthetically pleasing with custom 3D printed parts to house sensors, breadboards, speakers etc. Our system:

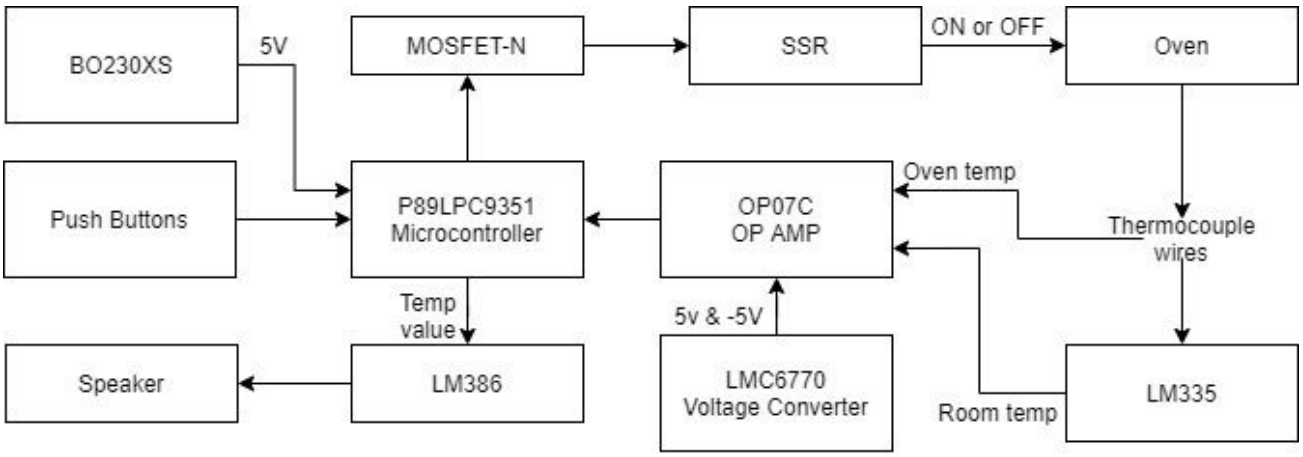
- Fulfills binary requirements (such as presence of emergency switch on/off).
- Displays the current state, temperature, running time, along with messages on the LCD.
- Successfully displayed a motion-sensor controlled Python stripchart with dynamically changing axes which shows the exact reflow profile based on customizable parameters.
- Integrated speaker capability by storing audio PCM to SPI flash memory and modularizing the speaker function to be usable by the main program. Used personally recorded audio with additional messages played within certain states and a beginning and end indication.

Nearly every aspect of our project improves upon the assigned project criteria, so not only have we implemented the required functionality, but also made it a lot more customizable and easy-to-use. By adding motion sensors, a servo motor, a 3D casing and electronic equipment like potentiometers for volume control, our final design is truly a state of the art reflow oven controller.

3.6 - Detailed Design

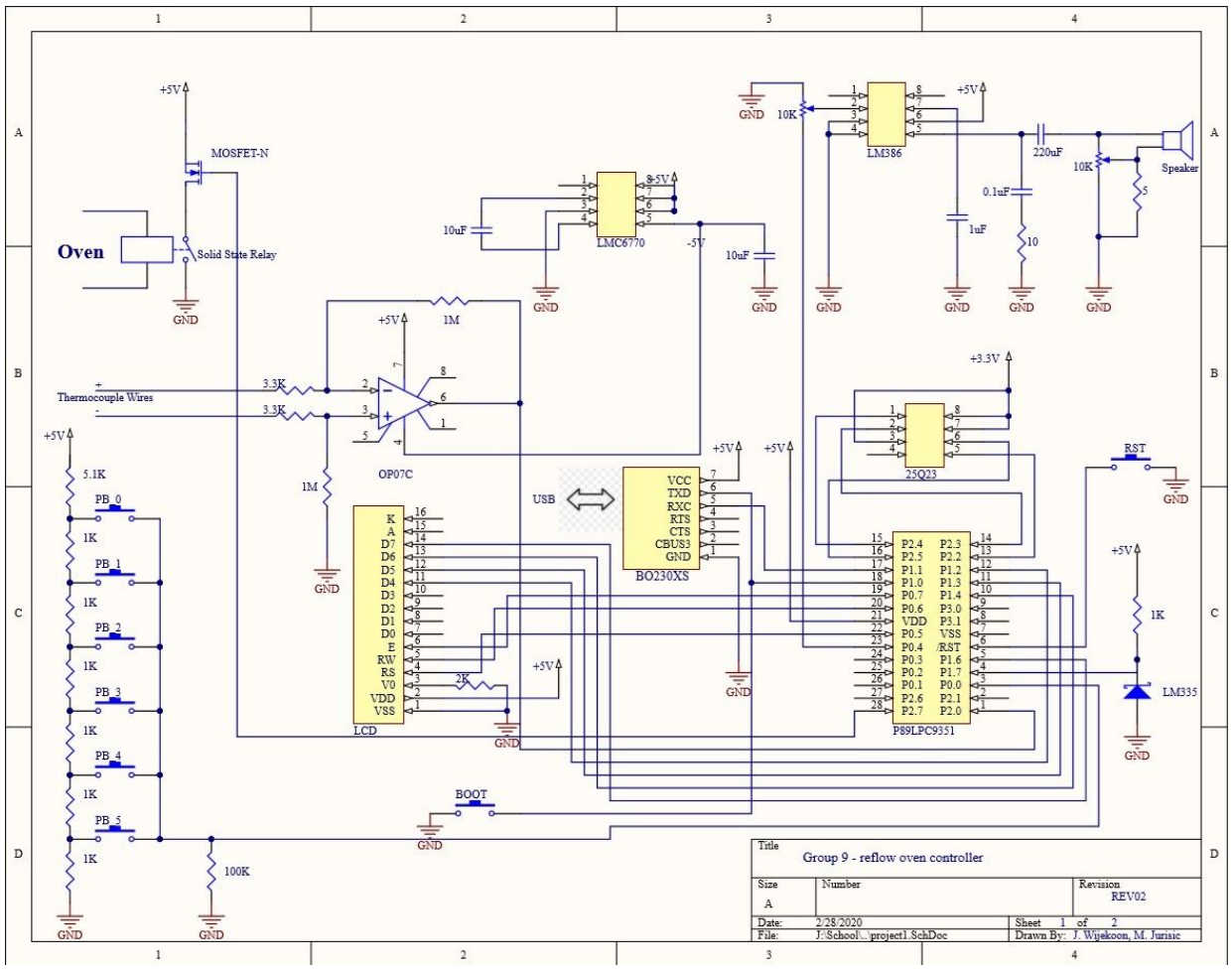
3.6.1 - Hardware block diagram

The following block diagram provides an overview of the essential hardware components; a more detailed schematic is displayed in the circuit diagram below.



3.6.2 - Circuit Diagram

The following circuit drawing maps the components used to carry out the core functionality of the reflow oven controller.



Component	Purpose
P89LPC9351 microcontroller	CPU of the oven controller
LM335 Thermostat	Measuring room temperature
K-Type Thermocouple	Measuring oven temperature
25Q23 flash memory	Store audio for speaker functionality
LCD	Provide a basic user interface
Analog input push-button circuit	Provide multiple push-button inputs using a single GPIO pin in the microcontroller
BO230XS USB to serial adapter	Easily interface with the microcontroller and flash to the LPC9351. Receive serial data from the microcontroller to be processed and presented by a Python script using a GUI.
OP07C operational amplifier	Amplify the thermocouple reading to a value large enough to be reliably read by the microcontroller ADC
LMC7660 voltage converter	Obtain a -5V output to power the negative rail of the OP07C
Solid State Relay (SSR)	Control the off/on state of the oven

3.6.3 - Circuit Explained

I. OP07C Operational Amplifier

The OpAmp rails were powered with (+/-)5V due to the simplicity in obtaining these values reliably with the BO230XS and the LMC7660. The amplifier had a non-inverting configuration with a resistor ratio of 1MΩ : 3.3KΩ, giving a gain of 303.0303 to the thermocouple reading. The thermocouple gained a potential difference of 41μV/℃ temperature increase, and was expected to fluctuate between 0 (at room temperature) - 9.84mV (at 240℃). This sets the OpAmp output in the range of 0-2.98V. This voltage range and resistor ratio were chosen to accommodate the inherent voltage limit of the LPC9351's ADC, which was 3V.

II. Analog push-button circuit

We decided to use analog push button inputs to preserve as many GPIO ports as possible for any possible bonus functionality for the controller. Having this implementation also provides clean and easy integration of any number of push-buttons in the future if necessary. This implementation did have the drawback of making the push-button input hard to correct for mechanical bounce. During the final stages of the project, the group decided to use a new set of push-buttons which were larger, more visually appealing, and had more bounce. Adding capacitors to minimize the error due to bounce proved ineffective, forcing the group to find a solution on the software end.

III. Solid State Relay

The SSR was driven by an N-MOSFET, due to the LMC's GPIO output being incapable of reliably draining the continuous 10mA current required to activate the SSR.

IV. Bonus Circuits

In addition to the standard audio amplifier configuration for the LM386, a potentiometer was also included in parallel with the speaker to control the voltage drop across the speaker, which in turn controlled the volume of the output. *The bonus circuit can be found in Appendix 7.3.*

3.6.4 - Software design and explanation

1. **Input Code**

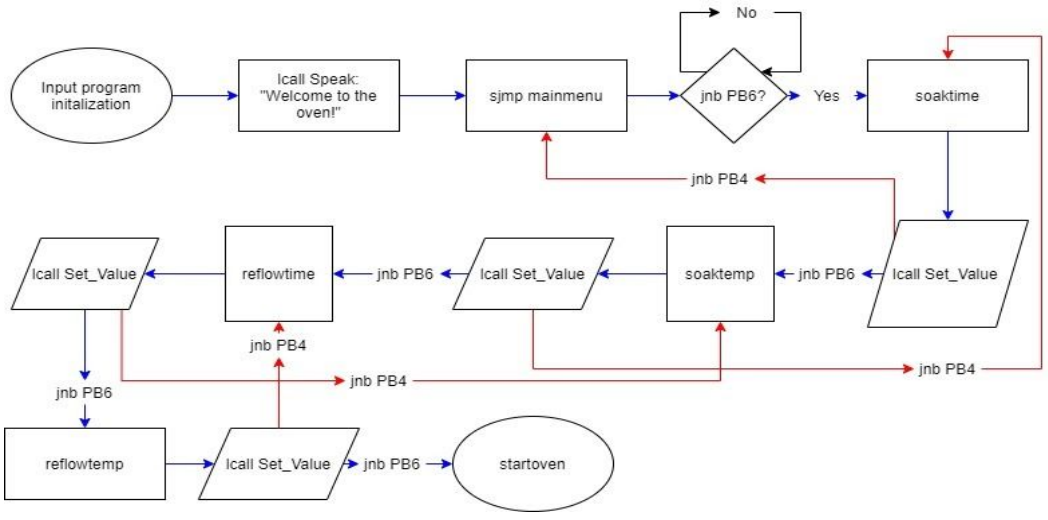


Figure 1.0 - Input Control Code

Four principal variables are used within the input code cycle: timesoak, tempsoak, timereflow and tempreflow. The 'temp' variables determine the peak temperature of the ramp while the 'time' variables determine the duration of the plateau. These variables are modified within '**mainmenu'** (Final assembly code.asm[1], line 924), where **Set_Value** is called.

A. Set_Value (Final Assembly.asm, lines 732-819)

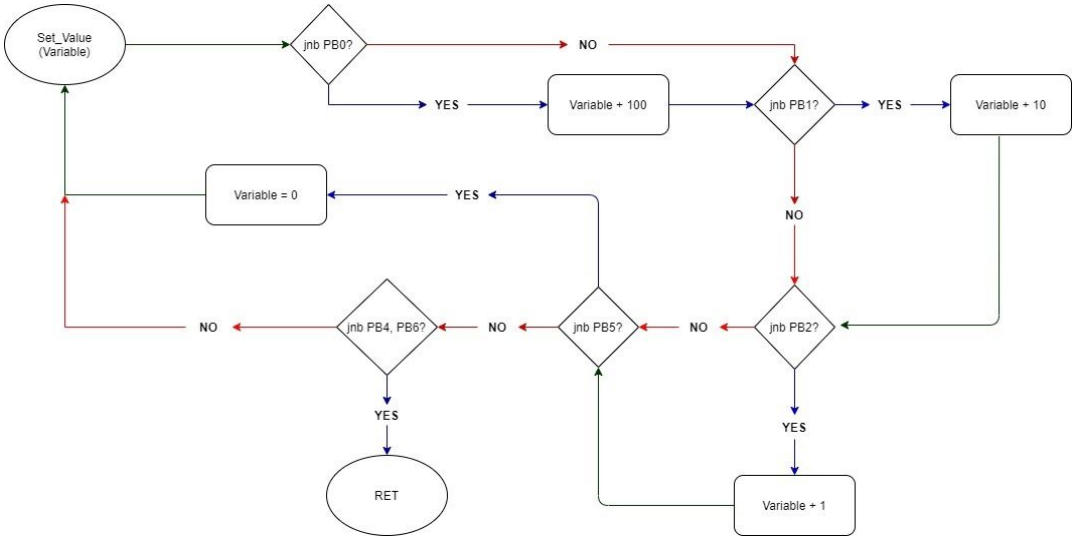


Figure 1.1 - Set_Value Flowchart

As shown above, the **Set_Value** function is used to set the value of one of the four aforementioned control variables. Each PB0 press increments the value of the variable by one hundred, for a maximum of two presses (Final assembly.asm, lines 761-775). Should the user try to input a number greater than two hundred, the variable immediately zeroes itself, because temperatures will never exceed 230 and time is always less than 200. Each PB1 press increments the variable by ten, and PB2 increments by one. Pressing PB5 clears the variable, while pressing PB4 or PB6 exits the program, because they are used to transition to the next portion of the code. This convention was chosen because it is intuitive (the 'enter' button PB6 is the rightmost), and significantly shortened

[1] See Appendix 7.8 for Assembly Source Code

our code. Instead of having multiple redundant variable manipulations, each time **Set_Value** is called, register 2 is manipulated within the code, and the new value is moved back into the variable when the code returns.

After the Set_Value function has been called for each variable, the program jumps to **startoven** (Final Assembly, line 1201) where the control cycle of the code begins.

2. **The Control Code** (Final Assembly.asm, Lines 1201)

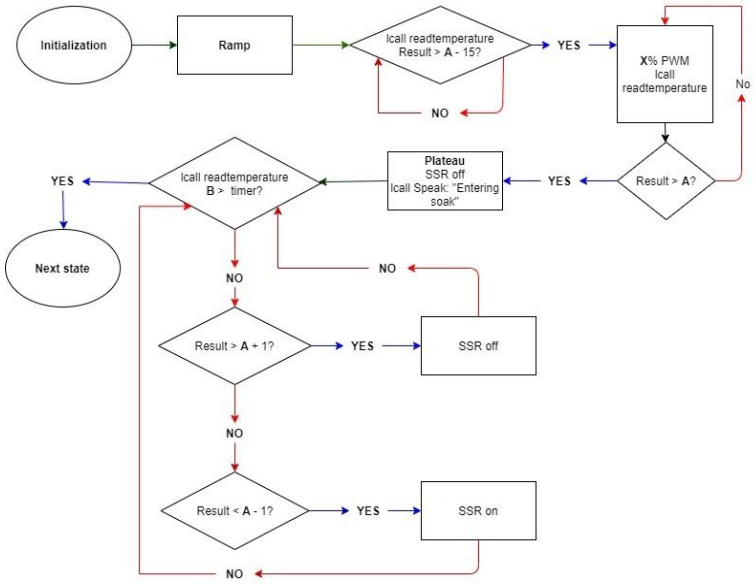


Figure 1.3 - General Controller Code

The control code consists of two identical sections: soak and reflow. Since these sections are so similar, Figure 1.3 provides an overview of the process that applies to both. During the ramp cycle, the temperature of the oven is constantly compared to the given temperature[2] (tempsoak or tempreflow) to see if it has exceeded the threshold. Instead of waiting until the temperatures are exactly equal, we check if the temperature is within 15 °C of our target. If it is, we enter PWM[3] (for more information see Section 3.7.3) and call readtemperature**,** which provides up-to-date temperature readings (see Section 2A for more information). During PWM, we continue to check the

[2] Temperature represented by **A** in Figure 1.3
[3] PWM Duty Cycle represented by **X** in Figure 1.3

temperature until it exceeds the tempsoak or tempreflow values; once this happens, we transition into the time cycle, or plateau.

During the plateau, the temperature is maintained for the duration of timesoak/timereflow[4] (for more information about the plateau algorithm, see Section 3.7.2). Within the plateau, we constantly compare the time to the target, but we also check to ensure we are truly maintaining the temperature, as this is not always the case due to latent heat. Within the plateau there is a one degree tolerance, so if the current temperature[5] is too high, the SSR is turned off, and if it is too low, the SSR is turned on. If the timer exceeds the given time, then we transition into the next cycle for reflow, which is identical to soak.

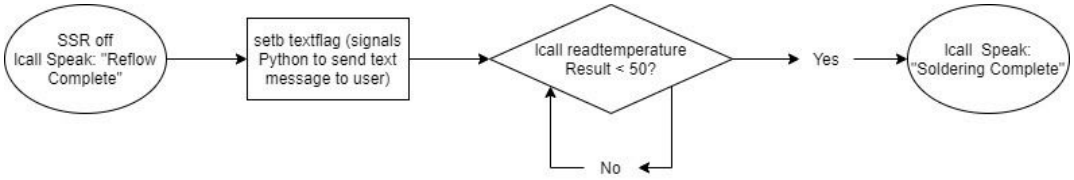


Figure 1.4 - Cooldown Code

After reflow we enter cool, where we turn off the SSR and send a flag to Python to send the user a text message (Final Assembly.asm, Lines 1563-1572). Once the temperature is less than 50 °C, the oven says “Soldering Complete”.

For more information about speaker integration and software, see Sections 2A (readtemperature) and Section 2B (speak function). See Appendix 7.9 for more information about the Python code (which deals with the Graphical User Interface and stripchart generation).

A. Readtemperature (Final Assembly.asm, Lines 1615-1796)

(For a detailed layout of the ReadTemperature algorithm, see Appendix 7.6)

The readtemperature function takes the voltage inputted to the ADC and converts it into a

[4] Time represented by **B** in Figure 1.3
[5] Represented by **Result** in Figure 1.3

temperature in Celsius. AD0DAT0 and AD0DAT3 were used to measure the temperature of the reference and oven respectively. To get more accurate readings we averaged the readings of the oven over 255 iterations. Once the appropriate conversions were completed, we displayed the average temperature, which is stored in the variable '*Result*'.

Because we call readtemperature so frequently, we decided to integrate speaker calls within this call. The second part of readtemperature ensures that the speaker says something every five seconds (Final Assembly.asm, Lines 1698-1703); the message is determined by two flags: "*instruction*" and "*statechange*". *statechange* is always either high (state change in progress) or low (no state change). If the *statechange* is high, then "instruction" is set to 2 and the state transition is announced, otherwise it is set to 1 and the temperature is declared. The speaker code requires three variables: *hundredsj*, *tensj* and *unitsj*[6], which represent the hundreds, tens and units of the temperature. The algorithm used to calculate can be found at Lines 1686-1712 in Final Assembly.asm.

B. Speak Function (Final Assembly.asm, Lines 1813-1965)

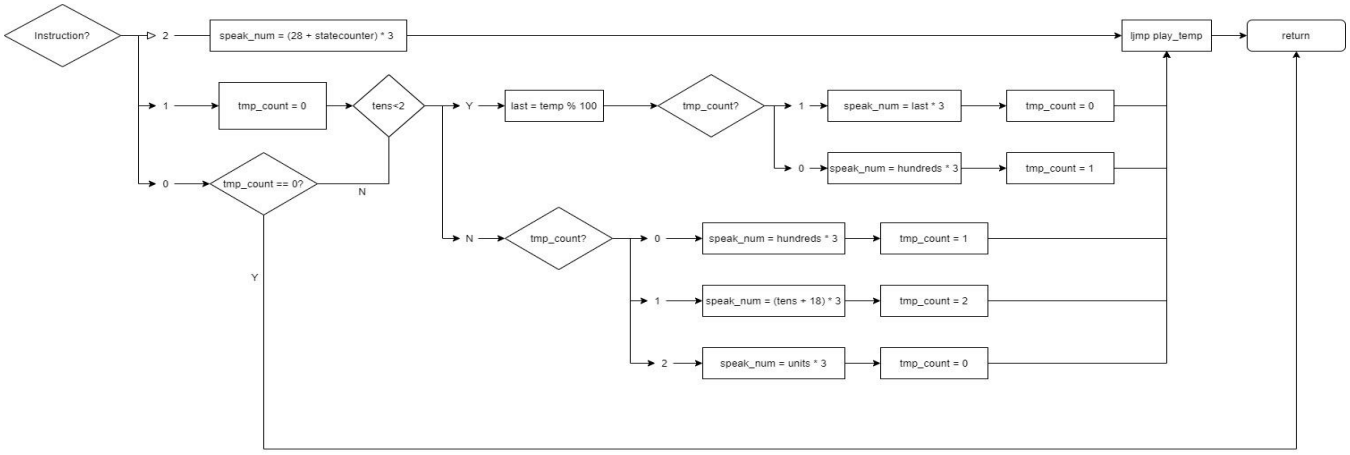


Figure 1.5 - Speaker Code

This function is responsible for announcing the current temperature or the oven state at the appropriate time using the speaker circuit. The speaker function is designed to be called by the main source code consistently every one second.

[6] Represented by hundreds, units and tens in Figure 1.4

Each temperature is “spoken” sequentially in segments across several function calls (first hundreds, then tens, etc).

On each call, the instruction informs the function what to do based on its value:

- 0 → Resumes speaking current temperature if *tmp_count* is not 0, else return
- 1 → Starts speaking a new temperature, set *tmp_count* to 0
- 2 → Speaks current state of the oven and returns

The “*tmp_count*” variable is set to zero to signify the start of a temperature reading. This value is incremented throughout the reading of the temperature, and is reset to 0 after the FSM is done “speaking” the number.

The arrays “*sound_index*” (Final Assembly.asm, Line 494) and “*size_length*” (Final Assembly, Line 533) have indexed the positions and the lengths of all the voice lines of the numbers from 1 - 19, 30 - 90 (increments of 10), 100, 200, and oven state announcements, all recited by our trusty announcer “HARSH3” . The code refers to the relevant position in the array with the variable “*speak_num*”, to access the required voice line.

Before the speaker speaks the temperature, it should know whether the (temp % 100) is greater or less than 2 because the audio file has the record from 1 to 20, and from 20, the number goes up to 100 with an interval of 10. Therefore if (temp % 100) is less than 20, the speaker will only be required to speak two times, including the hundreds, whereas if it is greater than 20, the speaker function will need to be called three times in order to speak the value: hundreds, tens, and units.

3.7 - Solution Assessment

This section summarizes the complete evaluation of our system, including input testing, speaker testing, reflow profile control testing and the primary strengths and drawbacks of our design.

3.7.1 - Input Testing

We tested both temperature and user inputs to ensure consistency and reliability:

- User input: Upon integration of our oven control code and speaker code, an LED test (code in Appendix 7.7b) revealed that variables from the speaker code were over writing the input parameters in memory (at dseg 0x30).
- Thermostat : Tested approximate room temperature, and observed fluctuations in reading while holding an active soldering iron to the pins of the thermostat.
- Thermocouple: We set the oven to a specific temperature and compare the thermocouple reading with the known temperature. Repeated for a range of temperatures.

3.7.2 - The Plateau

One major issue we encountered when writing and testing the code was the production of latent heat, which prevented the oven from maintaining a chosen temperature. To solve this problem, we took constant temperature readings to see if the oven is over (in which case we turned it off) or undershooting (so we turned it on). Because our temperature readings are averages, we thought it would be better to have a wider error margin (+/- 3 °C, but this range proved too wide. As our stripcharts in Figures 7.7 to 7.11 of Appendix 7.4 demonstrate, our plateau became more stable as the tolerance range decreased. Eventually we settled on a one degree tolerance.

3.7.3 - PWM

PWM is used in the ramp to each plateau to ensure that the correct temperature is reached. Without using PWM, the controller would never reach the plateau because it exceeded the threshold value of the plateau and its error margin. Thus our first major test was determining how much the controller overshot for a given temperature. To test this, we entered different peak temperatures and allowed the oven to heat naturally. As shown in Appendix 7.7a there is a linear relationship between the

target temperature and the actual value. For lower temperatures, the difference is roughly 20 °C, but for temperatures around the maximum temperature of the oven (230 °C), peak temperatures are not reached. As a result, a lower duty cycle was required for lower temperatures (36%) than higher temperatures (44%).

3.7.4 - Speaker tests

- Hardware: Tested our speakers response to a range of frequencies by connecting it to a signal generator. This also tested our volume control feature with the potentiometer.
- The "speak" function was isolated, and a temperature value was hard-coded in. Had a series of LED's light up for each stage of the FSM (speak, hundreds, tens, units, idle, oven state). Tested if the correct state transitions were happening for the following test cases:

1) Temperature value with tens < 2
2) Temperature value with tens > 2
3) Instruction = 2
4) Instruction = 1

3.7.5 - Overall Software Design Evaluation

Strengths:

1. **Highly user-friendly:** The custom GUI provides a user-friendly input-output interface. Setting the graph axes using motion control is innovative and intuitive, and receiving a text message when soldering is complete allows users to 'set and forget' the oven. The 3D printed case protects and elegantly covers the project, and its hidden compartment provides users with a convenient storage space for soldering essentials.
2. **Accurate reflow profile:** The combination of PWM in the ramp stages and +/-1 °C control in the plateau stages produces an accurate reflow profile that can easily be calibrated to a particular oven.

Drawbacks:

1. **Calibration:** Calibrating the PWM to a specific oven currently requires adjusting the source code directly. This can be rectified in future iterations by writing a PWM algorithm based on the linear relationship above.
2. **Push-button bounce:** The custom buttons used in our final design produce a lot of bounce; we eliminated most of it, but aim for more robust future solutions.

4.0 - Life-Long Learning

While working on the reflow oven project, our group achieved several learning outcomes. Finishing a project of this scale within three weeks enhanced our technical, collaborative and project organization skills. Building a reflow oven FSM improved our assembly skills - benefitting our debouncing methodology with analog push buttons, dynamic memory allocation, and code organization skills. By dealing with the speaker, we learnt more about timers, clocks, interrupts and how to control audio files in assembly. We did identify a learning gap in being efficient when distributing the coding workload amongst the six of us. We plan to rectify this by using version control software like Git more heavily in the upcoming project. Assembly skills and experience with building FSMs from CPEN 211 and circuit analysis skills from ELEC 201 proved useful in completing the technical requirements of the reflow oven controller.

5.0 - Conclusion

The goal of the project was to design and build an optimal reflow oven controller that consistently communicates with the oven. Our final system succeeded in meeting and surpassing the design requirements significantly. We produced a reflow oven controller that was functional, presentable,

interactive, and highly user-friendly. Beyond being able to reach and hold any temperature in the range of 25 - 240 ℃ with robust PWM control loops, the controller allows users to customize and interact with their reflow soldering experience in a variety of different ways. Notable examples of such features that made our controller stand out were the 3D printed casing (see Appendix 7.5, Figures 7.12 and 7.13), the Python GUI for graphing and displaying the oven status, and the text message alert system for a completed soldering cycle.

Several problems were encountered during the project. Some parts such as op-amps were more error prone than others. Integrating the input circuits with the speaker proved challenging as well, as each circuit was designed in an isolated module and the issue of overlapping GPIO ports became prevalent. There were also several challenges to overcome in the software part of the project such as converting ADC to DAC and vice versa in the microcontroller and figuring out button debouncing. Moreover, integrating the speaker and oven FSM assembly codes was time-consuming and required several debugging stages until it worked.

This project was extremely time consuming to work on, with the whole group having collectively put in approximately 90 work hours over the span of 3 weeks. However, the outcome of our endeavor was rewarding, satisfactory, and a valuable learning experience we were all glad to be a part of.

6.0 - References

[1] Tkinter — Python interface to Tcl/Tk — Python 2.7.17 documentation", *Docs.python.org*, 2020. [Online]. Available: https://docs.python.org/2/library/tkinter.html. [Accessed: 28- Feb- 2020]

[2] J. Heath, "PWM: EPulse Width Modulation: What is it and how does it work?", Analog IC tips, Nov. 2018

[3] Texas Instruments, "OP07x Precision Operational Amplifiers", OP07C datasheet, Oct. 1983 [Revised Nov.2014]

[4] Texas Instruments, LMx35, "LMx35A Precision Temperature Sensors", Feb. 2015 [Revised Mar. 2016]

[5] Texas Instruments, "LMC7660 Switched Capacitor Voltage Converter", LMC7660 datasheet, Apr. 1997 [Revised April 2013].

6.1 Bibliography

- Arar, S. , The Basics of SSRs (Solid-State Relays): The Switching Device - Technical Articles, 2016
- Walsh, B. , Programming Historian: Editing Audio with Audacity, 2015

7.0 - APPENDICES

7.1 - Mechanical Case Designs

We designed and 3D printed a custom casing for our oven controller. This casing improved the aesthetic value of our oven controller, as well as providing a robust protective environment for our electronic circuits. The detailed designs, as well as 3D renderings may be found below. All designs were made using Solidworks, and all measurements are in mm.

- 7.1.1 - Dimensioned Drawings

Figure 7.1 shows the dimensioned front view of our case. The two knobs respectively served as volume control (potentiometer) and a linear actuator controller. The linear actuator opens a secret compartment within the case, which can be used to hold circuit components. On the top, we have the motion sensor which was used to scale our graph and control the GUI:

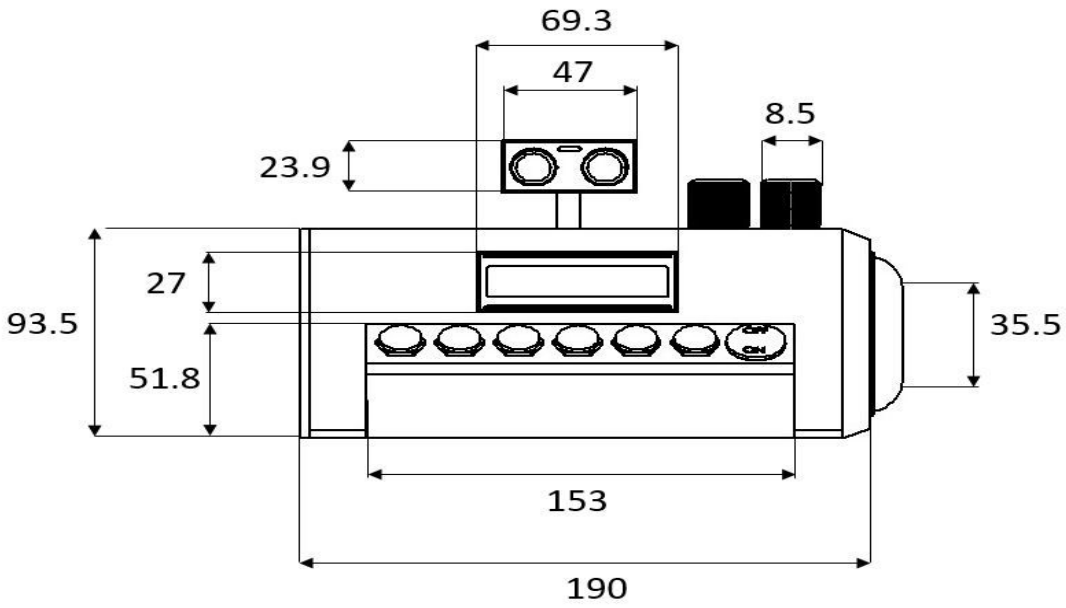


Figure 7.1

Figure 7.2 shows the side view, which holds the speaker:

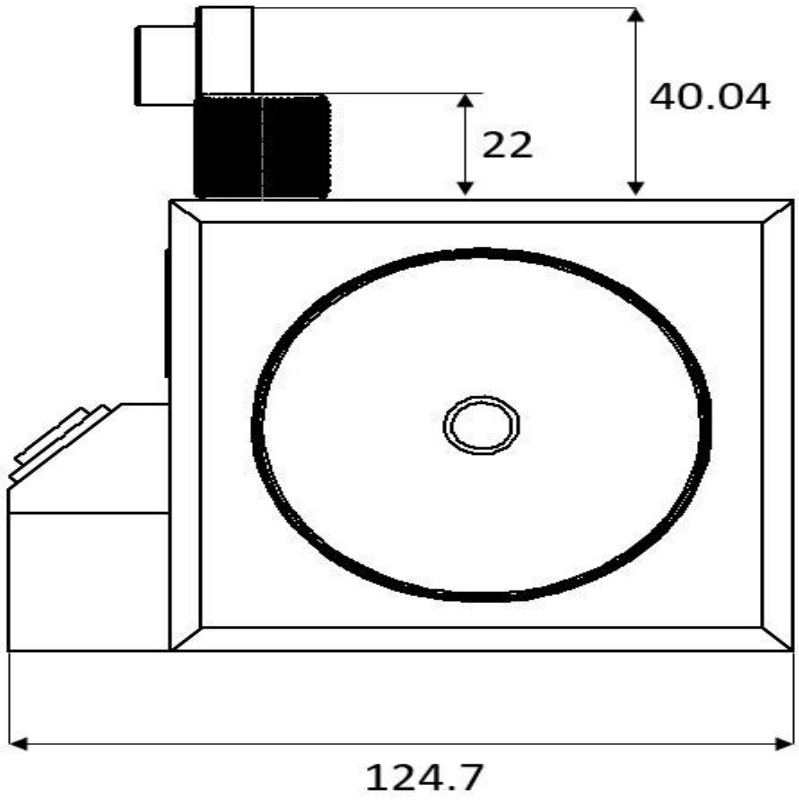


Figure 7.2

7.2 - 3D Renders

We created 3D renders of our oven controller using Solidworks. We recreated the oven controller in different colourways to pick the best design. The final product was designed using the black colourway:

Figure 7.3

Figure 7.4

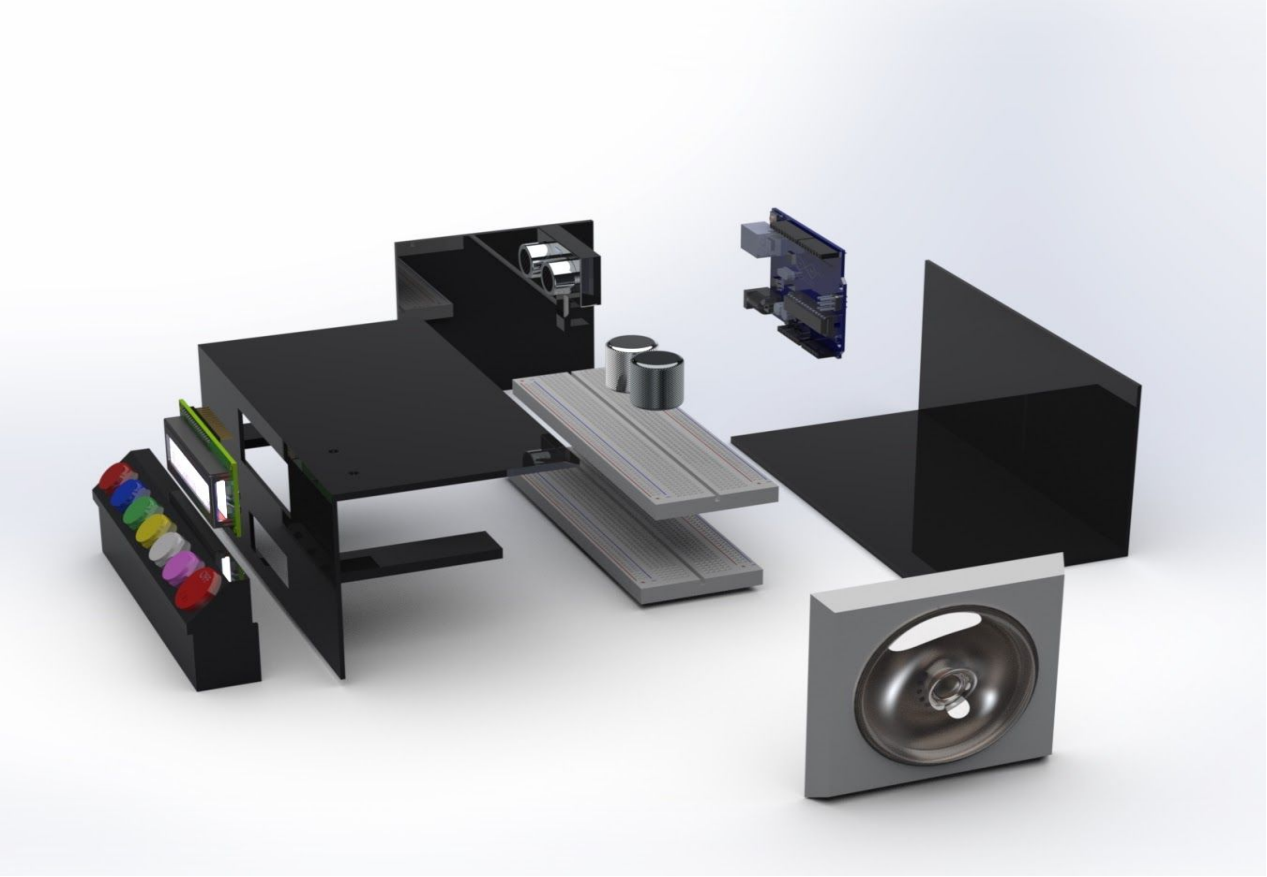


Figure 7.5

7.3 - Bonus Circuit

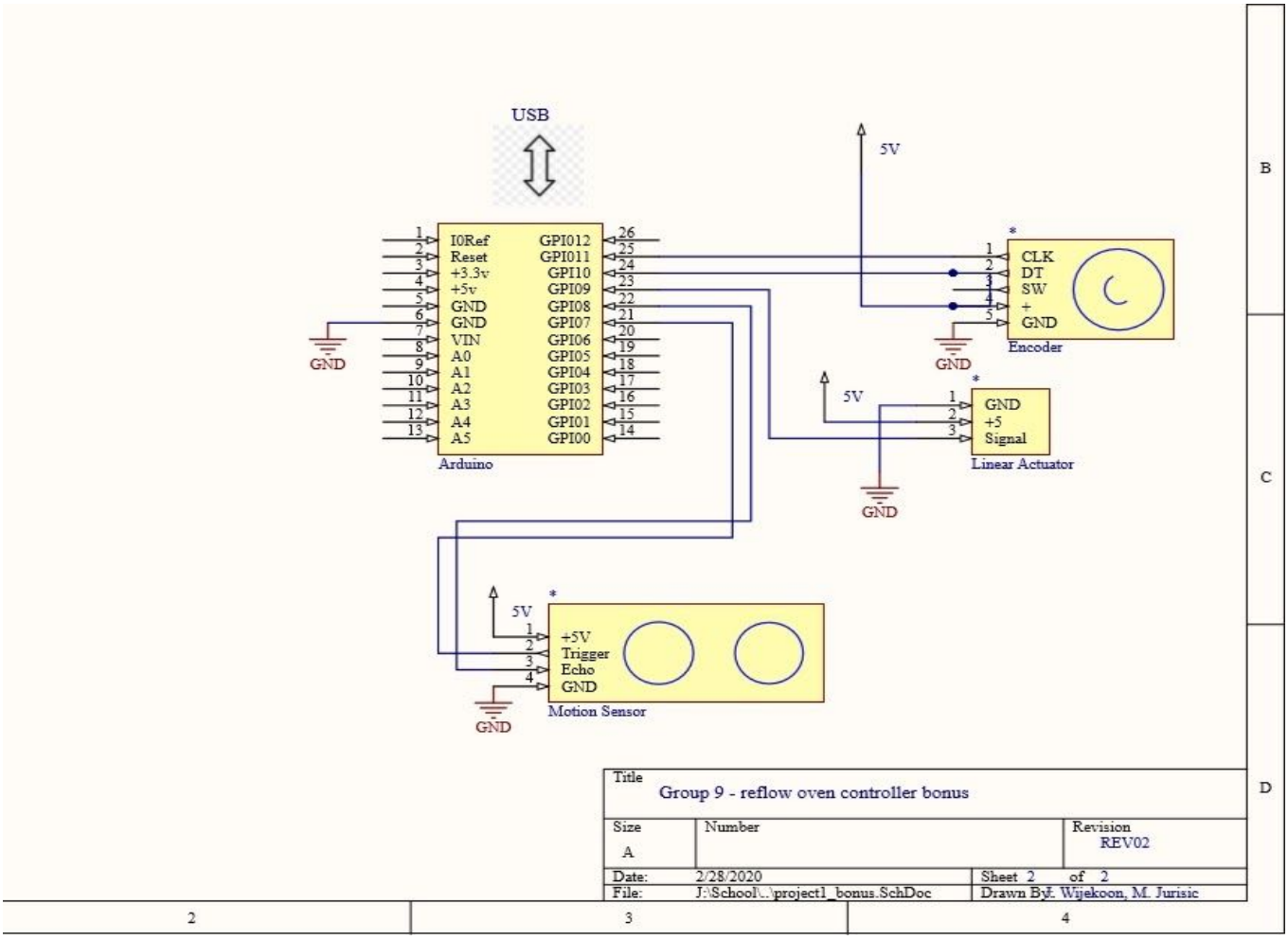


Figure 7.6

Above is part of our bonus circuit. We connected a rotary encoder, distance sensor and a linear actuator to an arduino and then interfaced the arduino with our project 1. The rotary encoder controlled the position of the linear actuator. We 3D printed a small box and stuck it onto the actuator. This made a retractable box which moved in and out of the casing. We used this box to store circuit components. We used the distance sensor to control our GUI and scale our python graph axis. This involved opening both the putty serial port and arduino as well, which was handled within python.

7.4 - Sample Strip Charts

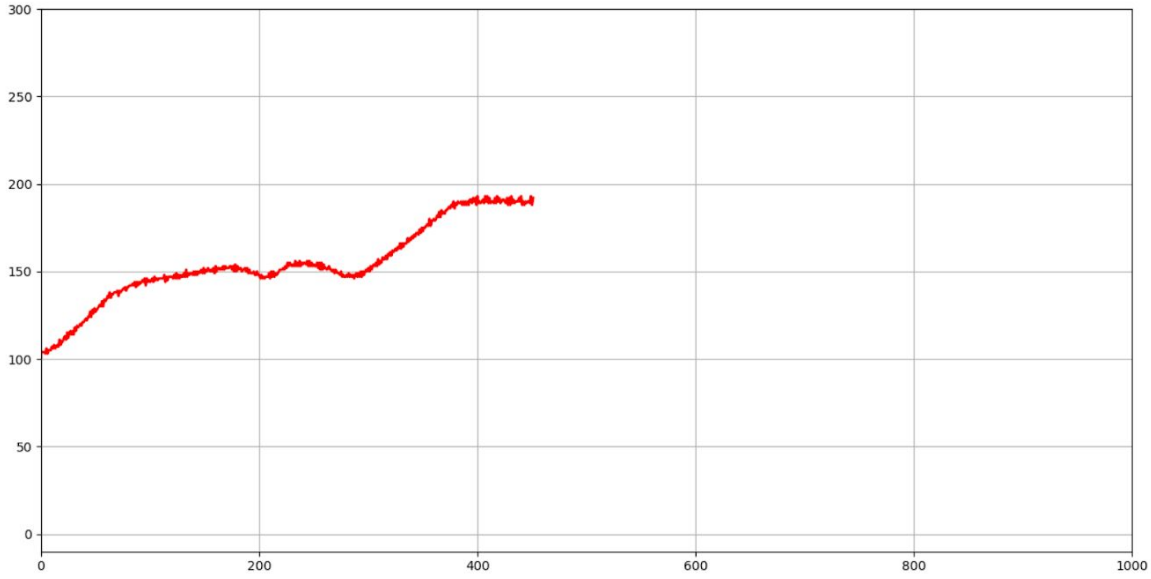


Figure 7.7

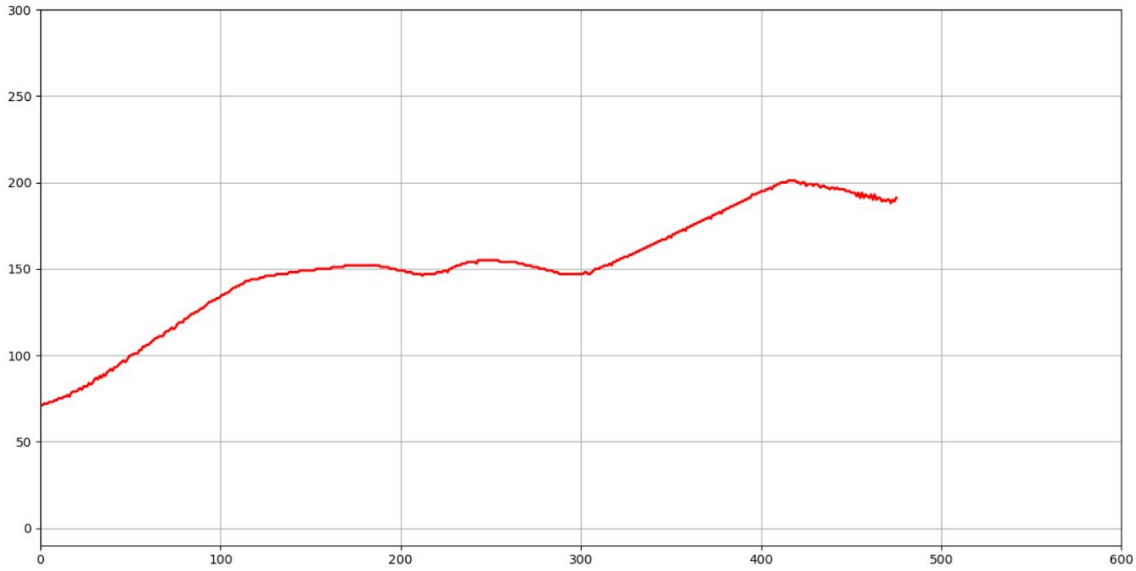


Figure 7.8

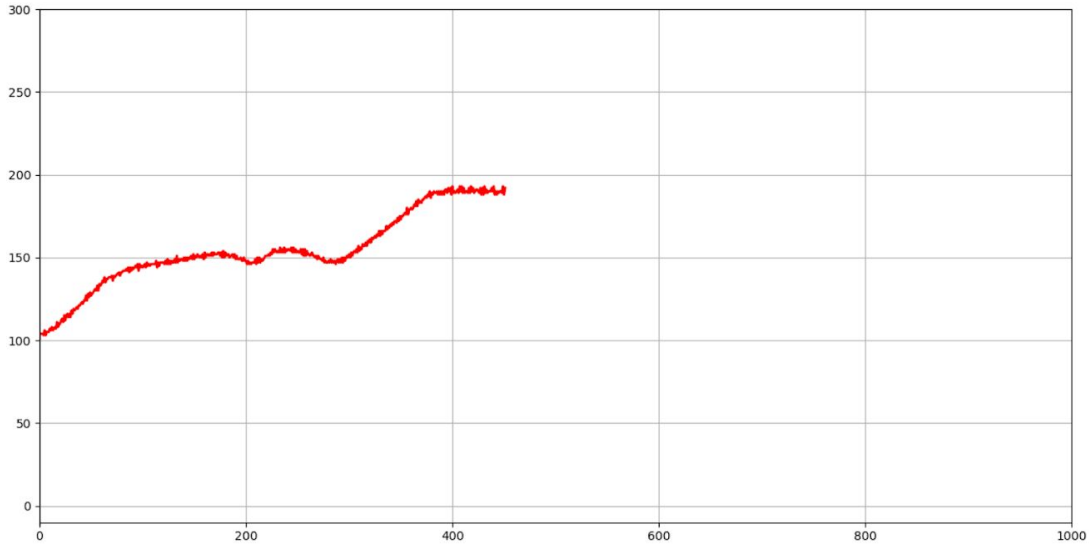


Figure 7.9

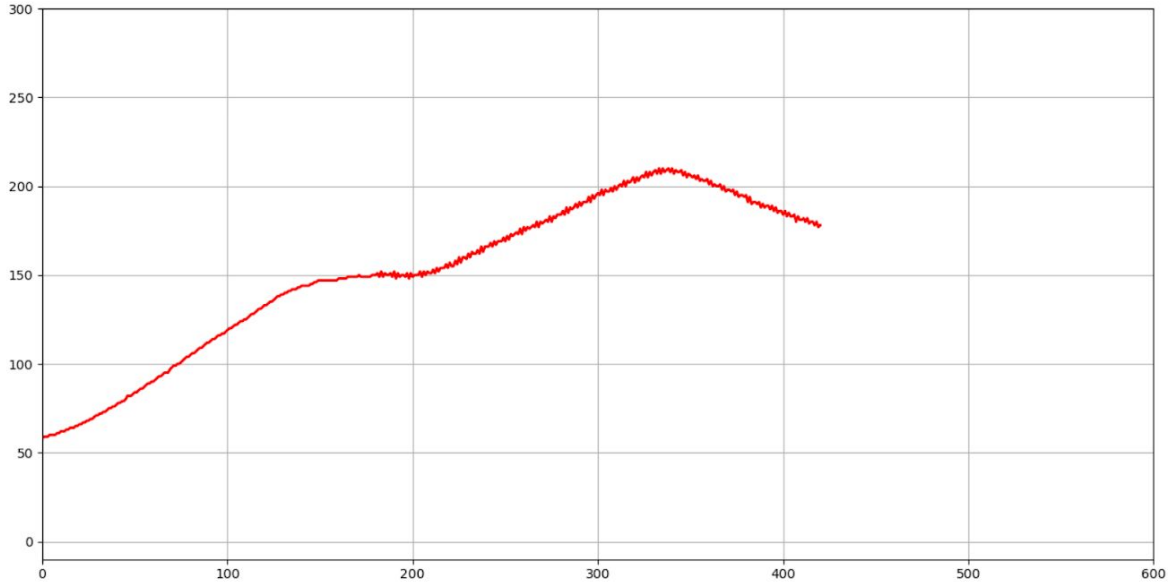


Figure 7.10

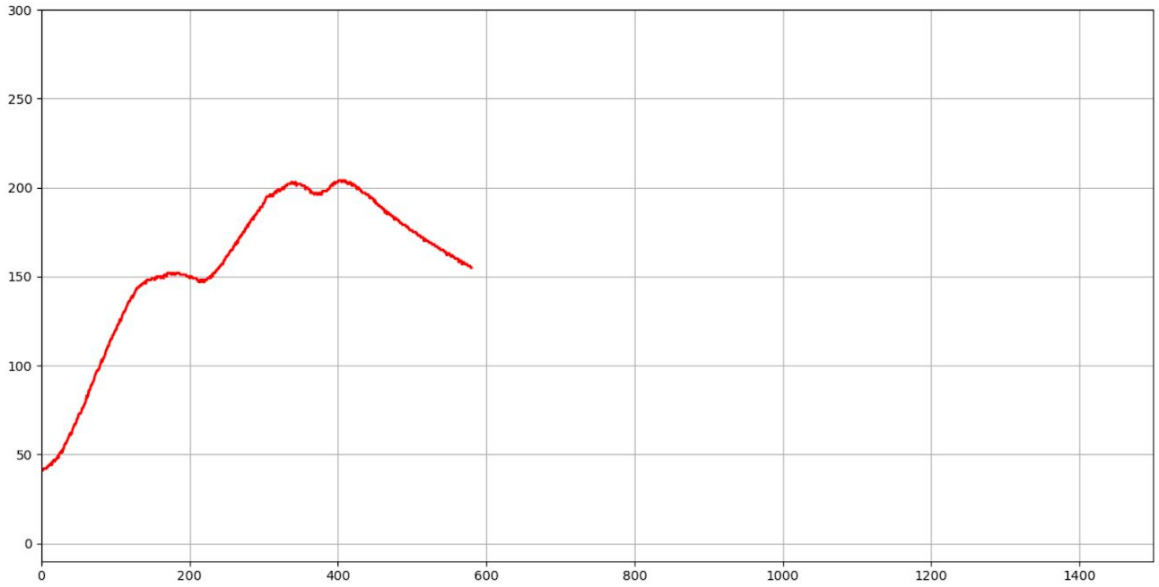


Figure 7.11

7.5 - Final Images

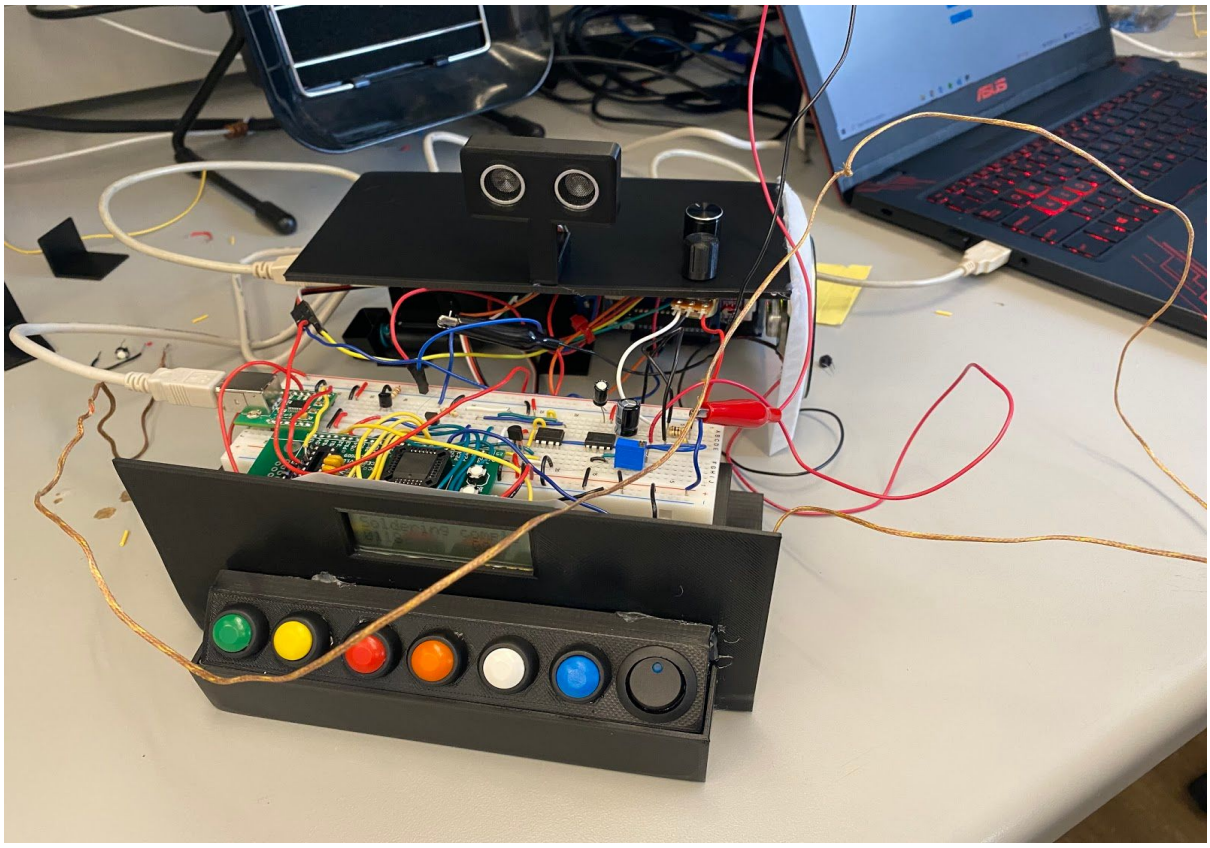


Figure 7.12

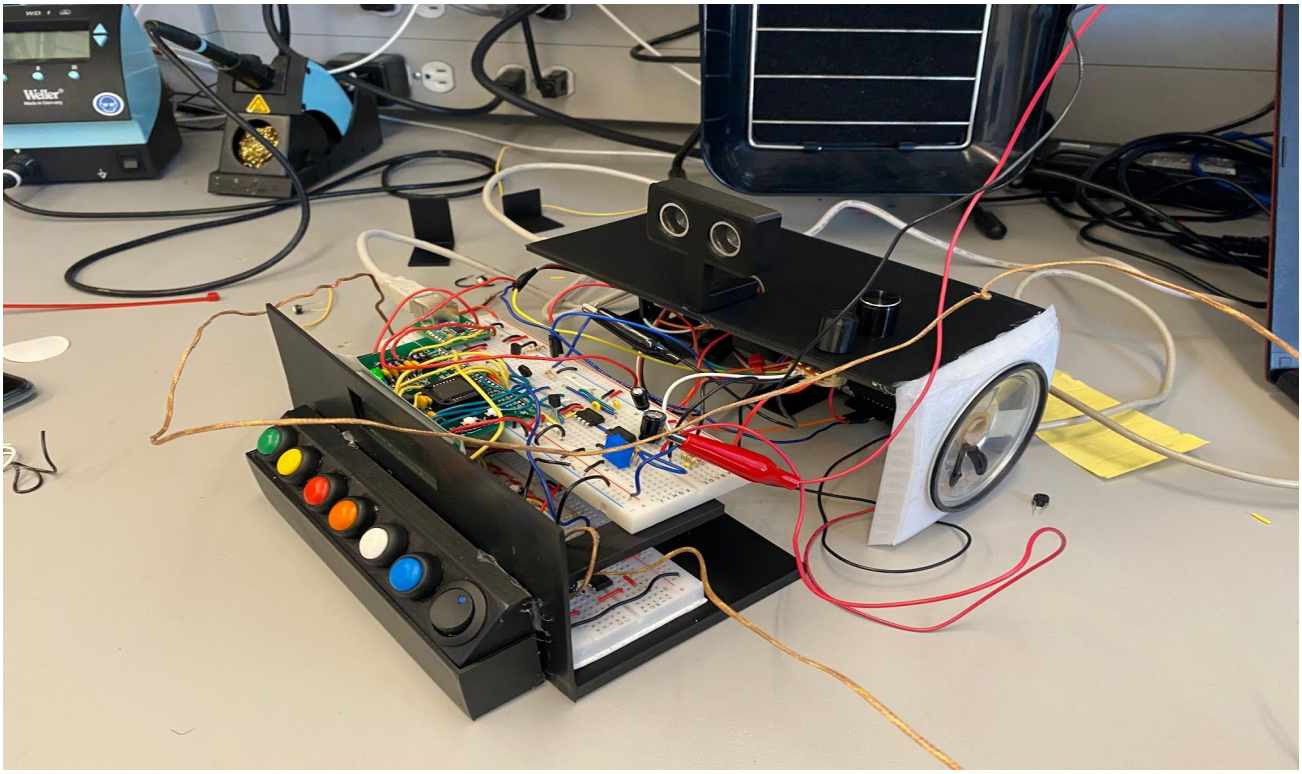
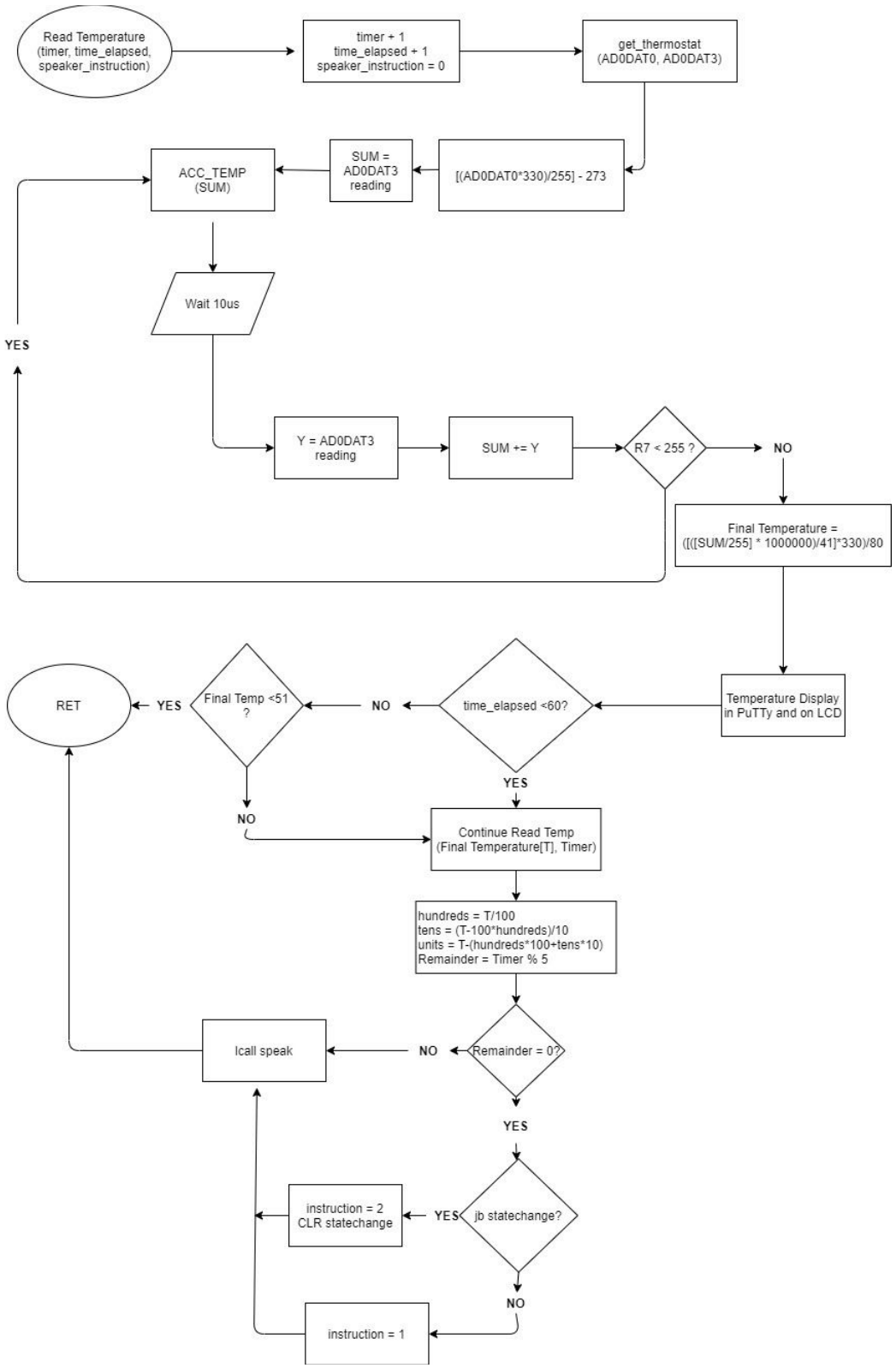


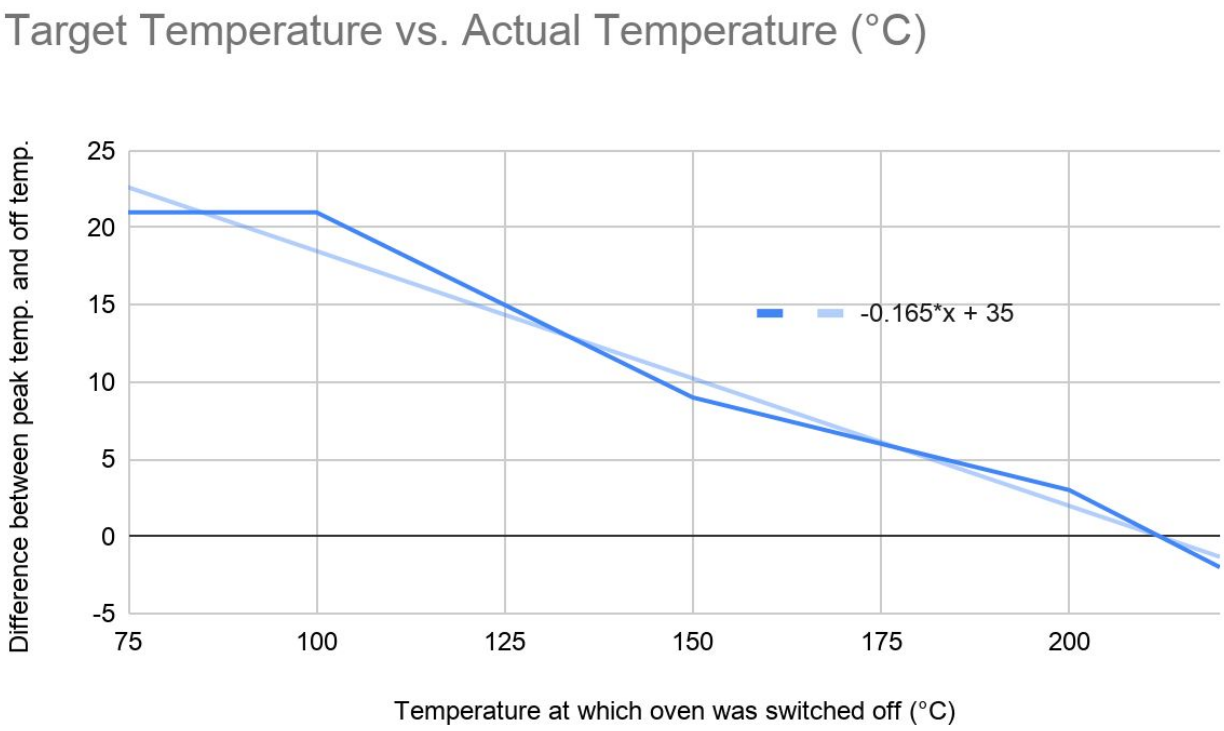
Figure 7.13

7.6 - ReadTemperature Flowchart



7.7 - Excel Sheet and Basic LED Test Code

a) Excel sheet



b) Basic LED Test Code

```
1.  startoven:
2.
3.          mov a, #0x77
4.          mov Result, a
5.          mov a, timesoak
6.          cjne a, Result, wait
7.
8.          cpl SSRpin
9.
10. now: sjmp now
```

7.8 - Assembly Source Code (Final Assembly.asm)

```
1.  ;----------------;
2.  ; INITIALIZATION ;
3.  ;----------------;
4.
5.  ; PLEASE READ:

6.  ; While the program itself is identical to the final submitted version on Canvas, comments
    and format have been changed for readability.
7.
8.  $NOLIST
9.  $MOD9351
10. $LIST
11.
12. CLK         EQU 14746000  ; Microcontroller system clock frequency in Hz
13. CCU_RATE    EQU 22050     ; 22050Hz is the sampling rate of the wav file we are playing
14. CCU_RELOAD  EQU ((65536-((CLK/(2*CCU_RATE)))))
15.
16. XTAL           EQU 7373000
17. BAUD        EQU 115200
18. NEWBRVAL    EQU ((CLK/BAUD)-16)
19. BRVAL          EQU ((XTAL/BAUD)-16)
20.
21. ; OUR PINS ;
22.
23. SSRpin         EQU P2.7
24. FLASH_CE    EQU P2.4
25. SOUND       EQU P2.1
26.
27. ; Commands supported by the SPI flash memory according to the datasheet
28. WRITE_ENABLE     EQU 0x06  ; Address:0 Dummy:0 Num:0
29. WRITE_DISABLE    EQU 0x04  ; Address:0 Dummy:0 Num:0
30. READ_STATUS      EQU 0x05  ; Address:0 Dummy:0 Num:1 to infinite
31. READ_BYTES       EQU 0x03  ; Address:3 Dummy:0 Num:1 to infinite
32. READ_SILICON_ID  EQU 0xab  ; Address:0 Dummy:3 Num:1 to infinite
33. FAST_READ        EQU 0x0b  ; Address:3 Dummy:1 Num:1 to infinite
34. WRITE_STATUS     EQU 0x01  ; Address:0 Dummy:0 Num:1
35. WRITE_BYTES      EQU 0x02  ; Address:3 Dummy:0 Num:1 to 256
36. ERASE_ALL        EQU 0xc7  ; Address:0 Dummy:0 Num:0
37. ERASE_BLOCK      EQU 0xd8  ; Address:3 Dummy:0 Num:0
38. READ_DEVICE_ID   EQU 0x9f  ; Address:0 Dummy:2 Num:1 to infinite
39.
40. ; OUR PINS ;
41.        CSEG
42. org 0x0000 ; Reset vector
43.     ljmp MainProgram
44.
45. org 0x0003 ; External interrupt 0 vector (not used in this code)
46.        reti
47.
48. org 0x000B ; Timer/Counter 0 overflow interrupt vector (not used in this code)
49.        reti
50.
51. org 0x0013 ; External interrupt 1 vector (not used in this code)
52.        reti
53.
54. org 0x001B ; Timer/Counter 1 overflow interrupt vector (not used in this code
55.        reti
56.
57. org 0x0023 ; Serial port receive/transmit interrupt vector (not used in this code)
58.        reti
59.
60. org 0x005b ; CCU interrupt vector.  Used in this code to replay the wave file.
61.        ljmp CCU_ISR
62.
63. bseg
64. PB0: dbit 1 ; Variable to store the state of pushbutton 0 after calling ADC_to_PB below
```

```
65. PB1: dbit 1 ; Variable to store the state of pushbutton 1 after calling ADC_to_PB below
66. PB2: dbit 1 ; Variable to store the state of pushbutton 2 after calling ADC_to_PB below
67. PB3: dbit 1 ; Variable to store the state of pushbutton 3 after calling ADC_to_PB below
68. PB4: dbit 1 ; Variable to store the state of pushbutton 4 after calling ADC_to_PB below
69. PB5: dbit 1 ; Variable to store the state of pushbutton 5 after calling ADC_to_PB below
70. PB6: dbit 1 ; Variable to store the state of pushbutton 6 after calling ADC_to_PB below
71. mf: dbit 1
72. dog: dbit 1 ;
73. statechangeflag: dbit 1
74.
75. dseg at 0x30
76.
77. ; OVEN CONTROL VARIABLES:
78.
79. time: ds 2
80. timesoak: ds 2
81. tempsoak: ds 2
82. timereflow: ds 2
83. tempreflow: ds 2
84. Result: ds 2
85. soaktimer: ds 2
86. reflowtimer: ds 2
87. x: ds 4
88. y: ds 4
89. bcd: ds 5
90.
91. cat: ds  4 ; represents oven temp
92. sheep: ds 4 ; represent reference temp
93.
94. ; VARIABLES FOR USE BY THE SPEAKER:
95.
96. soakramptimer: ds 2 ; Used to keep time in the whole circuit for the speaker - initially used
    for soakramp, hence the name
97. reflowramptimer:  ds 2 ; This timer was not used in the final implementation
98.
99. statecounter: ds 1
100.    instruction: ds 1
101.    hundredsj: ds 1
102.    tensj: ds 1
103.    unitsj: ds 1
104.    unitsresult: ds 1
105.    tensresult: ds 1
106.    hundredsresult: ds 1
107.    cooltimer: ds 1
108.
109.    temp: ds 1
110.    speak_num: ds 1
111.    tmp_count: ds 1
112.    last: ds 1
113.
114.    ; VARIABLES FOR TESTING:
115.
116.    other_dog: ds 2 ; this variable is not used in the current iteration of the code
117.
118.    ; GUI AND SPEAKER INTEGRATION VARIABLES:
119.
120.    w:   ds 3 ; 24-bit play counter.  Decremented in CCU ISR.
121.    somecounter: ds 1
122.    textvar: ds 4
123.
124.    cseg
125.
126.    ; These 'equ' must match the wiring between the microcontroller and the LCD!
127.    LCD_RS equ P0.5
128.    LCD_RW equ P0.6
129.    LCD_E  equ P0.7
130.    LCD_D4 equ P1.2
131.    LCD_D5 equ P1.3
132.    LCD_D6 equ P1.4
```

```
133.    LCD_D7 equ P1.6
134.
135.    $NOLIST
136.    $include(LCD_4bit_LPC9351.inc) ; A library of LCD related functions and utility macros
137.    $include(math32.inc)
138.    $LIST
139.
140.    ;---------------------------------;
141.    ; Routine to initialize the CCU.  ;
142.    ; We are using the CCU timer in a ;
143.    ; manner similar to the timer 2   ;
144.    ; available in other 8051s        ;
145.    ;---------------------------------;
146.
147.    CCU_Init:
148.        mov TH2, #high(CCU_RELOAD)
149.        mov TL2, #low(CCU_RELOAD)
150.        mov TOR2H, #high(CCU_RELOAD)
151.        mov TOR2L, #low(CCU_RELOAD)
152.        mov TCR21, #10000000b ; Latch the reload value
153.        mov TICR2, #10000000b ; Enable CCU Timer Overflow Interrupt
154.        setb ECCU ; Enable CCU interrupt
155.        setb TMOD20 ; Start CCU timer
156.        ret
157.
158.    ;---------------------------------;
159.    ; ISR for CCU.  Used to playback  ;
160.    ; the WAV file stored in the SPI  ;
161.    ; flash memory.                   ;
162.    ;---------------------------------;
163.
164.    CCU_ISR:
165.        mov TIFR2, #0 ; Clear CCU Timer Overflow Interrupt Flag bit. Actually, it clears all
    the bits!
166.        setb P2.6 ; To check the interrupt rate with oscilloscope.
167.
168.        ; The registers used in the ISR must be saved in the stack
169.        push acc
170.        push psw
171.
172.        ; Check if the play counter is zero.  If so, stop playing sound.
173.        mov a, w+0
174.        orl a, w+1
175.        orl a, w+2
176.        jz stop_playing
177.
178.        ; Decrement play counter 'w'.  In this implementation 'w' is a 24-bit counter.
179.        mov a, #0xff
180.        dec w+0
181.        cjne a, w+0, keep_playing
182.        dec w+1
183.        cjne a, w+1, keep_playing
184.        dec w+2
185.
186.    keep_playing:
187.
188.        lcall Send_SPI ; Read the next byte from the SPI Flash...
189.        mov AD1DAT3, a ; and send it to the DAC
190.
191.        sjmp CCU_ISR_Done
192.
193.    stop_playing:
194.        clr TMOD20 ; Stop CCU timer
195.        setb FLASH_CE  ; Disable SPI Flash
196.        clr SOUND ; Turn speaker off
197.
198.    CCU_ISR_Done:
199.        pop psw
200.        pop acc
```

```
201.        clr P2.6
202.        reti
203.
204.    putchar:
205.        jbc     TI,putchar_L1
206.        sjmp putchar
207.    putchar_L1:
208.        mov     SBUF,a
209.        ret
210.
211.    getchar:
212.        jbc     RI,getchar_L1
213.        sjmp getchar
214.    getchar_L1:
215.        mov     a,SBUF
216.        ret
217.
218.    Wait1S:
219.        mov R2, #40
220.    M3: mov R1, #250
221.    M2: mov R0, #184
222.    M1: djnz R0, M1 ; 2 machine cycles-> 2*0.27126us*184=100us
223.        djnz R1, M2 ; 100us*250=0.025s
224.        djnz R2, M3 ; 0.025s*40=1s
225.        ret
226.
227.    Wait200mS:
228.        mov R2, #8
229.    M7: mov R1, #250
230.    M6: mov R0, #184
231.    M5: djnz R0, M5 ; 2 machine cycles-> 2*0.27126us*184=100us
232.        djnz R1, M6 ; 100us*250=0.025s
233.        djnz R2, M7 ; 0.025s*40=1s
234.        ret
235.
236.    Wait10mS:
237.
238.        mov R0, #184
239.    M4: djnz R0, M4 ; 2 machine cycles-> 2*0.27126us*184=100us
240.        ret
241.
242.    InitSerialPort:
243.
244.        mov     BRGCON,#0x00
245.        mov     BRGR1,#high(BRVAL)
246.        mov     BRGR0,#low(BRVAL)
247.        mov     BRGCON,#0x03 ; Turn-on the baud rate generator
248.        mov     SCON,#0x52 ; Serial port in mode 1, ren, txrdy, rxempty
249.        anl     P1M1,#11111100B
250.        anl     P1M2,#11111100B
251.        ret
252.
253.    DoubleInitSerialPort:
254.
255.        mov     BRGCON,#0x00
256.        mov     BRGR1,#high(NEWBRVAL)
257.        mov     BRGR0,#low(NEWBRVAL)
258.        mov     BRGCON,#0x03 ; Turn-on the baud rate generator
259.        mov     SCON,#0x52 ; Serial port in mode 1, ren, txrdy, rxempty
260.        anl     P1M1,#11111100B
261.        anl     P1M2,#11111100B
262.        ret
263.
264.    ;---------------------------------;
265.    ; Initialize ADC1/DAC1 as DAC1.   ;
266.    ; Warning, the ADC1/DAC1 can work ;
267.    ; only as ADC or DAC, not both.   ;
268.    ; The P89LPC9351 has two ADC/DAC  ;
269.    ; interfaces.  One can be used as ;
```

```
270.    ; ADC and the other can be used   ;
271.    ; as DAC.  Also configures the    ;
272.    ; pin associated with the DAC, in ;
273.    ; this case P0.4 as 'Open Drain'. ;
274.    ;----------------------------------;
275.
276.    InitDAC1:
277.        ; Configure pin P0.4 (DAC1 output pin) as open drain
278.        orl     P0M1,   #00010000B
279.        orl     P0M2,   #00010000B
280.        mov ADMODB, #00101000B ; Select main clock/2 for ADC/DAC.  Also enable DAC1 output
    (Table 25 of reference manual)
281.        mov     ADCON1, #00000100B ; Enable the converter
282.        mov AD1DAT3, #0x80     ; Start value is 3.3V/2 (zero reference for AC WAV file)
283.        ret
284.
285.    ;----------------------------------;
286.    ; Initialize ADC0/DAC0 as ADC0.   ;
287.    ;----------------------------------;
288.
289.    InitADC0:
290.        ; ADC0_0 is connected to P1.7
291.        ; ADC0_1 is connected to P0.0
292.        ; ADC0_2 is connected to P2.1
293.        ; ADC0_3 is connected to P2.0
294.        ; Configure pins P1.7, P0.0, P2.1, and P2.0 as inputs
295.        orl P0M1, #00000001b
296.        anl P0M2, #11111110b
297.        orl P1M1, #10000000b
298.        anl P1M2, #01111111b
299.        orl P2M1, #00000011b
300.        anl P2M2, #11111100b
301.        ; Setup ADC0
302.        setb BURST0 ; Autoscan continuos conversion mode
303.        mov     ADMODB,#0x20 ;ADC0 clock is 7.3728MHz/2
304.        mov     ADINS,#0x0f ; Select the four channels of ADC0 for conversion
305.        mov     ADCON0,#0x05 ; Enable the converter and start immediately
306.        ; Wait for first conversion to complete
307.    InitADC0_L1:
308.        mov     a,ADCON0
309.        jnb     acc.3,InitADC0_L1
310.        ret
311.    ;----------------------------------;
312.    ; Change the internal RC osc. clk ;
313.    ; from 7.373MHz to 14.746MHz.     ;
314.    ;----------------------------------;
315.
316.    Double_Clk:
317.        mov dptr, #CLKCON
318.        movx a, @dptr
319.        orl a, #00001000B ; double the clock speed to 14.746MHz
320.        movx @dptr,a
321.        ret
322.
323.    Single_Clk:
324.        mov dptr, #CLKCON
325.        movx a, @dptr
326.        anl a, #11110111B ; double the clock speed to 14.746MHz
327.        movx @dptr,a
328.        ret
329.
330.    ;----------------------------------;
331.    ; Initialize the SPI interface    ;
332.    ; and the pins associated to SPI. ;
333.    ;----------------------------------;
334.
335.    Init_SPI:
336.        ; Configure MOSI (P2.2), CS* (P2.4), and SPICLK (P2.5) as push-pull outputs (see table
    42, page 51)
```

```
337.        anl P2M1, #low(not(00110100B))
338.        orl P2M2, #00110100B
339.        ; Configure MISO (P2.3) as input (see table 42, page 51)
340.        orl P2M1, #00001000B
341.        anl P2M2, #low(not(00001000B))
342.        ; Configure SPI
343.        mov SPCTL, #11010000B ; Ignore /SS, Enable SPI, DORD=0, Master=1, CPOL=0, CPHA=0,
    clk/4
344.        ret
345.
346.    ;---------------------------------;
347.    ; Sends AND receives a byte via   ;
348.    ; SPI.                           ;
349.    ;---------------------------------;
350.
351.    Send_SPI:
352.        mov SPDAT, a
353.    Send_SPI_1:
354.        mov a, SPSTAT
355.        jnb acc.7, Send_SPI_1 ; Check SPI Transfer Completion Flag
356.        mov SPSTAT, a ; Clear SPI Transfer Completion Flag
357.        mov a, SPDAT ; return received byte via accumulator
358.        ret
359.
360.    ;---------------------------------;
361.    ; SPI flash 'write enable'        ;
362.    ; instruction.                   ;
363.    ;---------------------------------;
364.
365.    Enable_Write:
366.        clr FLASH_CE
367.        mov a, #WRITE_ENABLE
368.        lcall Send_SPI
369.        setb FLASH_CE
370.        ret
371.
372.    ;---------------------------------;
373.    ; This function checks the 'write ;
374.    ; in progress' bit of the SPI     ;
375.    ; flash memory.                   ;
376.    ;---------------------------------;
377.
378.    Check_WIP:
379.        clr FLASH_CE
380.        mov a, #READ_STATUS
381.        lcall Send_SPI
382.        mov a, #0x55
383.        lcall Send_SPI
384.        setb FLASH_CE
385.        jb acc.0, Check_WIP ;  Check the Write in Progress bit
386.        ret
387.
388.    ;---------------------------------;
389.    ; CRC-CCITT (XModem) Polynomial:  ;
390.    ; x^16 + x^12 + x^5 + 1 (0x1021)  ;
391.    ; CRC in [R7,R6].                ;
392.    ; Converted to a macro to remove  ;
393.    ; the overhead of 'lcall' and     ;
394.    ; 'ret' instructions, since this  ;
395.    ; 'routine' may be executed over  ;
396.    ; 4 million times!                ;
397.    ;---------------------------------;
398.    ;crc16:
399.
400.    crc16 mac
401.        xrl     a, r7                  ; XOR high of CRC with byte
402.        mov r0, a                      ; Save for later use
403.        mov     dptr, #CRC16_TH ; dptr points to table high
404.        movc a, @a+dptr        ; Get high part from table
```

```
405.        xrl     a, r6                   ; XOR With low byte of CRC
406.        mov     r7, a                   ; Store to high byte of CRC
407.        mov a, r0                       ; Retrieve saved accumulator
408.        mov     dptr, #CRC16_TL; dptr points to table low
409.        movc a, @a+dptr         ; Get Low from table
410.        mov     r6, a                   ; Store to low byte of CRC
411.        ;ret
412.    endmac
413.
414.    ;---------------------------------;
415.    ; High constants for CRC-CCITT    ;
416.    ; (XModem) Polynomial:            ;
417.    ; x^16 + x^12 + x^5 + 1 (0x1021)  ;
418.    ;---------------------------------;
419.
420.    CRC16_TH:
421.        db      000h, 010h, 020h, 030h, 040h, 050h, 060h, 070h
422.        db      081h, 091h, 0A1h, 0B1h, 0C1h, 0D1h, 0E1h, 0F1h
423.        db      012h, 002h, 032h, 022h, 052h, 042h, 072h, 062h
424.        db      093h, 083h, 0B3h, 0A3h, 0D3h, 0C3h, 0F3h, 0E3h
425.        db      024h, 034h, 004h, 014h, 064h, 074h, 044h, 054h
426.        db      0A5h, 0B5h, 085h, 095h, 0E5h, 0F5h, 0C5h, 0D5h
427.        db      036h, 026h, 016h, 006h, 076h, 066h, 056h, 046h
428.        db      0B7h, 0A7h, 097h, 087h, 0F7h, 0E7h, 0D7h, 0C7h
429.        db      048h, 058h, 068h, 078h, 008h, 018h, 028h, 038h
430.        db      0C9h, 0D9h, 0E9h, 0F9h, 089h, 099h, 0A9h, 0B9h
431.        db      05Ah, 04Ah, 07Ah, 06Ah, 01Ah, 00Ah, 03Ah, 02Ah
432.        db      0DBh, 0CBh, 0FBh, 0EBh, 09Bh, 08Bh, 0BBh, 0ABh
433.        db      06Ch, 07Ch, 04Ch, 05Ch, 02Ch, 03Ch, 00Ch, 01Ch
434.        db      0EDh, 0FDh, 0CDh, 0DDh, 0ADh, 0BDh, 08Dh, 09Dh
435.        db      07Eh, 06Eh, 05Eh, 04Eh, 03Eh, 02Eh, 01Eh, 00Eh
436.        db      0FFh, 0EFh, 0DFh, 0CFh, 0BFh, 0AFh, 09Fh, 08Fh
437.        db      091h, 081h, 0B1h, 0A1h, 0D1h, 0C1h, 0F1h, 0E1h
438.        db      010h, 000h, 030h, 020h, 050h, 040h, 070h, 060h
439.        db      083h, 093h, 0A3h, 0B3h, 0C3h, 0D3h, 0E3h, 0F3h
440.        db      002h, 012h, 022h, 032h, 042h, 052h, 062h, 072h
441.        db      0B5h, 0A5h, 095h, 085h, 0F5h, 0E5h, 0D5h, 0C5h
442.        db      034h, 024h, 014h, 004h, 074h, 064h, 054h, 044h
443.        db      0A7h, 0B7h, 087h, 097h, 0E7h, 0F7h, 0C7h, 0D7h
444.        db      026h, 036h, 006h, 016h, 066h, 076h, 046h, 056h
445.        db      0D9h, 0C9h, 0F9h, 0E9h, 099h, 089h, 0B9h, 0A9h
446.        db      058h, 048h, 078h, 068h, 018h, 008h, 038h, 028h
447.        db      0CBh, 0DBh, 0EBh, 0FBh, 08Bh, 09Bh, 0ABh, 0BBh
448.        db      04Ah, 05Ah, 06Ah, 07Ah, 00Ah, 01Ah, 02Ah, 03Ah
449.        db      0FDh, 0EDh, 0DDh, 0CDh, 0BDh, 0ADh, 09Dh, 08Dh
450.        db      07Ch, 06Ch, 05Ch, 04Ch, 03Ch, 02Ch, 01Ch, 00Ch
451.        db      0EFh, 0FFh, 0CFh, 0DFh, 0AFh, 0BFh, 08Fh, 09Fh
452.        db      06Eh, 07Eh, 04Eh, 05Eh, 02Eh, 03Eh, 00Eh, 01Eh
453.
454.    ;---------------------------------;
455.    ; Low constants for CRC-CCITT     ;
456.    ; (XModem) Polynomial:            ;
457.    ; x^16 + x^12 + x^5 + 1 (0x1021)  ;
458.    ;---------------------------------;
459.
460.    CRC16_TL:
461.        db      000h, 021h, 042h, 063h, 084h, 0A5h, 0C6h, 0E7h
462.        db      008h, 029h, 04Ah, 06Bh, 08Ch, 0ADh, 0CEh, 0EFh
463.        db      031h, 010h, 073h, 052h, 0B5h, 094h, 0F7h, 0D6h
464.        db      039h, 018h, 07Bh, 05Ah, 0BDh, 09Ch, 0FFh, 0DEh
465.        db      062h, 043h, 020h, 001h, 0E6h, 0C7h, 0A4h, 085h
466.        db      06Ah, 04Bh, 028h, 009h, 0EEh, 0CFh, 0ACh, 08Dh
467.        db      053h, 072h, 011h, 030h, 0D7h, 0F6h, 095h, 0B4h
468.        db      05Bh, 07Ah, 019h, 038h, 0DFh, 0FEh, 09Dh, 0BCh
469.        db      0C4h, 0E5h, 086h, 0A7h, 040h, 061h, 002h, 023h
470.        db      0CCh, 0EDh, 08Eh, 0AFh, 048h, 069h, 00Ah, 02Bh
471.        db      0F5h, 0D4h, 0B7h, 096h, 071h, 050h, 033h, 012h
472.        db      0FDh, 0DCh, 0BFh, 09Eh, 079h, 058h, 03Bh, 01Ah
473.        db      0A6h, 087h, 0E4h, 0C5h, 022h, 003h, 060h, 041h
```

```
474.        db      0AEh, 08Fh, 0ECh, 0CDh, 02Ah, 00Bh, 068h, 049h
475.        db      097h, 0B6h, 0D5h, 0F4h, 013h, 032h, 051h, 070h
476.        db      09Fh, 0BEh, 0DDh, 0FCh, 01Bh, 03Ah, 059h, 078h
477.        db      088h, 0A9h, 0CAh, 0EBh, 00Ch, 02Dh, 04Eh, 06Fh
478.        db      080h, 0A1h, 0C2h, 0E3h, 004h, 025h, 046h, 067h
479.        db      0B9h, 098h, 0FBh, 0DAh, 03Dh, 01Ch, 07Fh, 05Eh
480.        db      0B1h, 090h, 0F3h, 0D2h, 035h, 014h, 077h, 056h
481.        db      0EAh, 0CBh, 0A8h, 089h, 06Eh, 04Fh, 02Ch, 00Dh
482.        db      0E2h, 0C3h, 0A0h, 081h, 066h, 047h, 024h, 005h
483.        db      0DBh, 0FAh, 099h, 0B8h, 05Fh, 07Eh, 01Dh, 03Ch
484.        db      0D3h, 0F2h, 091h, 0B0h, 057h, 076h, 015h, 034h
485.        db      04Ch, 06Dh, 00Eh, 02Fh, 0C8h, 0E9h, 08Ah, 0ABh
486.        db      044h, 065h, 006h, 027h, 0C0h, 0E1h, 082h, 0A3h
487.        db      07Dh, 05Ch, 03Fh, 01Eh, 0F9h, 0D8h, 0BBh, 09Ah
488.        db      075h, 054h, 037h, 016h, 0F1h, 0D0h, 0B3h, 092h
489.        db      02Eh, 00Fh, 06Ch, 04Dh, 0AAh, 08Bh, 0E8h, 0C9h
490.        db      026h, 007h, 064h, 045h, 0A2h, 083h, 0E0h, 0C1h
491.        db      01Fh, 03Eh, 05Dh, 07Ch, 09Bh, 0BAh, 0D9h, 0F8h
492.        db      017h, 036h, 055h, 074h, 093h, 0B2h, 0D1h, 0F0h
493.
494.    sound_index:
495.        db 0x00, 0x0d, 0x50 ; 0
496.        db 0x00, 0xbb, 0x53 ; 1
497.        db 0x01, 0x3c, 0x1c ; 2
498.        db 0x01, 0x9f, 0x96 ; 3
499.        db 0x02, 0x2d, 0xd9 ; 4
500.        db 0x02, 0x99, 0xe6 ; 5
501.        db 0x03, 0x20, 0xe6 ; 6
502.        db 0x03, 0xa0, 0xe6 ; 7
503.        db 0x04, 0x10, 0x09 ; 8
504.        db 0x04, 0x7c, 0x62 ; 9
505.        db 0x04, 0xfd, 0x5d ; 10
506.        db 0x05, 0x70, 0x00 ; 11
507.        db 0x05, 0xcc, 0x5d ; 12
508.        db 0x06, 0x65, 0x11 ; 13
509.        db 0x06, 0xc2, 0xa6 ; 14
510.        db 0x07, 0x4a, 0xd2 ; 15
511.        db 0x07, 0xc8, 0xd2 ; 16
512.        db 0x08, 0x67, 0xc8 ; 17
513.        db 0x08, 0xe3, 0xfe ; 18
514.        db 0x09, 0x71, 0x4a ; 19
515.        db 0x09, 0xf5, 0x1b ; 20
516.        db 0x0a, 0x60, 0xe4 ; 30 (21)
517.        db 0x0a, 0xdf, 0xc4 ; 40 (22)
518.        db 0x0b, 0x55, 0xbb ; 50 (23)
519.        db 0x0b, 0xdc, 0xb7 ; 60 (24)
520.        db 0x0c, 0x5d, 0xe0 ; 70 (25)
521.        db 0x0c, 0xdd, 0x05 ; 80 (26)
522.        db 0x0d, 0x5e, 0x12 ; 90 (27)
523.        db 0x0e, 0x55, 0xc4 ; Welcome to the oven (28)
524.        db 0x0f, 0xdd, 0x00 ; Heating Oven (29)
525.        db 0x10, 0x22, 0xd7 ; Beginning Soak (30)
526.        db 0x11, 0x22, 0x9d ; Beginning Reflow (31)
527.        db 0x11, 0xa0, 0x81 ; Reflow Complete (32)
528.        db 0x12, 0x4c, 0x17 ; Soldering Complete (33)
529.        db 0x13, 0xcc, 0x43 ; one hundred (34)
530.        db 0x14, 0x77, 0x04 ; two hundreds (35)
531.
532.    ; Size of each sound in 'sound_index'
533.    size_length:
534.        db 0x00, 0x00, 0x00 ; 0
535.        db 0x00, 0x45, 0x13 ; 1
536.        db 0x00, 0x33, 0x02 ; 2
537.        db 0x00, 0x35, 0x43 ; 3
538.        db 0x00, 0x45, 0x43 ; 4
539.        db 0x00, 0x4f, 0x23 ; 5
540.        db 0x00, 0x4f, 0x23 ; 6
541.        db 0x00, 0x42, 0x59 ; 7
542.        db 0x00, 0x42, 0x59 ; 8
```

```
543.        db 0x00, 0x4e, 0x94 ; 9
544.        db 0x00, 0x32, 0x67 ; 10
545.        db 0x00, 0x48, 0x0c ; 11
546.        db 0x00, 0x48, 0x0c ; 12
547.        db 0x00, 0x60, 0xa8 ; 13
548.        db 0x00, 0x6d, 0x95 ; 14
549.        db 0x00, 0x5c, 0x2c ; 15
550.        db 0x00, 0x61, 0xe7 ; 16
551.        db 0x00, 0x60, 0x0f ; 17
552.        db 0x00, 0x60, 0x0f ; 18
553.        db 0x00, 0x60, 0x0f ; 19
554.        db 0x00, 0x60, 0x0f ; 20
555.        db 0x00, 0x55, 0x0f ; 30 (21)
556.        db 0x00, 0x4d, 0xbb ; 40 (22)
557.        db 0x00, 0x7e, 0x3b ; 50 (23)
558.        db 0x00, 0x70, 0xe0 ; 60 (24)
559.        db 0x00, 0x60, 0xf7 ; 70 (25)
560.        db 0x00, 0x50, 0xf7 ; 80 (26)
561.        db 0x00, 0x44, 0x0d ; 90 (27)
562.        db 0x00, 0xaa, 0x67 ; Welcome to the oven (28) 0
563.        db 0x00, 0x77, 0x67 ; Heating Oven (29) 1
564.        db 0x00, 0xbb, 0x72 ; Beginnig Soak (30) 2
565.        db 0x00, 0x88, 0x73 ; Beginning Reflow (31) 3
566.        db 0x00, 0xbb, 0x3a ; Reflow Complete (32) 4
567.        db 0x00, 0xa8, 0x66 ; Soldering Complete (33) 5
568.        db 0x00, 0x50, 0xd1 ; One hundred (34)
569.        db 0x00, 0x4d, 0x96 ; Two hundreds (35)
570.
571.    HexAscii: db '0123456789ABCDEF'
572.
573.    SendTemp:
574.        mov dptr, #HexAscii
575.
576.        mov a, bcd+1
577.        swap a
578.        anl a, #0xf
579.        movc a, @a+dptr
580.        lcall putchar
581.        mov a, bcd+1
582.        anl a, #0xf
583.        movc a, @a+dptr
584.        lcall putchar
585.        mov a, bcd+0
586.        swap a
587.        anl a, #0xf
588.        movc a, @a+dptr
589.        lcall putchar
590.        mov a, bcd+0
591.        anl a, #0xf
592.        movc a, @a+dptr
593.        lcall putchar
594.        ret
595.
596.    SendHex:
597.        mov a, #'0'
598.        lcall putchar
599.        mov a, #'x'
600.        lcall putchar
601.        mov dptr, #HexAscii
602.        mov a, b
603.        swap a
604.        anl a, #0xf
605.        movc a, @a+dptr
606.        lcall putchar
607.        mov a, b
608.        anl a, #0xf
609.        movc a, @a+dptr
610.        lcall putchar
611.        mov a, #' '
```

```
612.        lcall putchar
613.        ret
614.
615.    SendString:
616.        clr a
617.        movc a, @a+dptr
618.        jz SendString_L1
619.        lcall putchar
620.        inc dptr
621.        sjmp SendString
622.    SendString_L1:
623.        ret
624.
625.    ADC_to_PB:
626.        setb PB6
627.        setb PB5
628.        setb PB4
629.        setb PB3
630.        setb PB2
631.        setb PB1
632.        setb PB0
633.        ; Check PB6
634.        clr c
635.        mov a, AD0DAT1
636.        subb a, #(206-10) ; 2.8V=216*(3.3/255); the -10 is to prevent false readings
637.        jc ADC_to_PB_L6
638.        clr PB6
639.        ret
640.    ADC_to_PB_L6:
641.        ; Check PB5
642.        clr c
643.        mov a, AD0DAT1
644.        subb a, #(185-10) ; 2.4V=185*(3.3/255); the -10 is to prevent false readings
645.        jc ADC_to_PB_L5
646.        clr PB5
647.        ret
648.    ADC_to_PB_L5:
649.        ; Check PB4
650.        clr c
651.        mov a, AD0DAT1
652.        subb a, #(154-10) ; 2.0V=154*(3.3/255); the -10 is to prevent false readings
653.        jc ADC_to_PB_L4
654.        clr PB4
655.        ret
656.    ADC_to_PB_L4:
657.        ; Check PB3
658.        clr c
659.        mov a, AD0DAT1
660.        subb a, #(123-10) ; 1.6V=123*(3.3/255); the -10 is to prevent false readings
661.        jc ADC_to_PB_L3
662.        clr PB3
663.        ret
664.    ADC_to_PB_L3:
665.        ; Check PB2
666.        clr c
667.        mov a, AD0DAT1
668.        subb a, #(92-10) ; 1.2V=92*(3.3/255); the -10 is to prevent false readings
669.        jc ADC_to_PB_L2
670.        clr PB2
671.        ret
672.    ADC_to_PB_L2:
673.        ; Check PB1
674.        clr c
675.        mov a, AD0DAT1
676.        subb a, #(61-10) ; 0.8V=61*(3.3/255); the -10 is to prevent false readings
677.        jc ADC_to_PB_L1
678.        clr PB1
679.        ret
680.    ADC_to_PB_L1:
```

```
681.        ; Check PB1
682.        clr c
683.        mov a, AD0DAT1
684.        subb a, #(30-10) ; 0.4V=30*(3.3/255); the -10 is to prevent false readings
685.        jc ADC_to_PB_L0
686.        clr PB0
687.        ret
688.    ADC_to_PB_L0:
689.        ; No pusbutton pressed
690.        ret
691.
692.    ;-----------;
693.    ; FUNCTIONS ;
694.    ;-----------;
695.
696.    ; SEND_BCD ;
697.
698.    Send_BCD mac
699.        push ar0
700.        mov r0, %0
701.        lcall ?Send_BCD
702.        pop ar0
703.    endmac
704.
705.    ?Send_BCD:
706.        push acc
707.        ; Write most sig digit
708.        mov a, r0
709.        swap a
710.        anl a, #0fh
711.        orl a, #30h
712.        lcall putchar
713.        ; Write least sig digit
714.        mov a, r0
715.        anl a, #0fh
716.        orl a, #30h
717.        lcall putchar
718.        pop acc
719.        ret
720.
721.    ; Set_Value:
722.    ; Sets a particular reflow parameter based on the user's input
723.
724.    ; Inputs:
725.    ; PB0: hundreds
726.    ; PB1: tens
727.    ; PB2: units
728.    ; PB4: clear
729.    ; PB5: cycle back through program
730.    ; PB6: cycle forward through program
731.
732.    Set_Value:
733.
734.        mov r2, #0
735.
736.    Set_Value_loop:
737.
738.        mov a, #'\r' ; move cursor all the way to the left
739.        lcall putchar
740.
741.        mov a, r2
742.
743.        mov x, a
744.        mov x+1, #0
745.        mov x+2, #0
746.        mov x+3, #0
747.
748.        lcall hex2bcd
749.        Send_BCD(bcd+1)
```

```
750.        Send_BCD(bcd)
751.
752.        Set_Cursor(2,1) ; (1,2)
753.
754.        Display_BCD(bcd+1)
755.        Display_BCD(bcd)
756.
757.        lcall ADC_to_PB
758.
759.                        jb PB0, tens_check ; hundreds check - if hundreds button pressed, add
    100 to r2
760.
761.                        mov a, somecounter
762.                        add a, #1
763.                        mov somecounter, a
764.
765.                        cjne a, #3, add_hundred
766.
767.                        mov r2, #0
768.                        mov somecounter, #0
769.
770.                        sjmp hundredsloop
771.
772.                        add_hundred:
773.                        mov a, r2
774.                        add a, #0x64
775.                        mov r2, a
776.
777.                                        hundredsloop: ; for bounce
778.                                        lcall ADC_to_PB
779.                                        jnb PB0, hundredsloop
780.
781.                tens_check:
782.
783.                        lcall ADC_to_PB
784.                        jb PB1, units_check ; tens check - if tens button pressed, add 10 to r2
785.
786.                        mov a, r2
787.                        add a, #0xA
788.                        mov r2, a
789.
790.                                        tensloop: ; for bounce
791.                                        lcall ADC_to_PB
792.                                        jnb PB1, tensloop
793.
794.                units_check:
795.
796.                        lcall ADC_to_PB
797.                        jb PB2, buttoncheck ; units check - if units button pressed, add 1 to
    r2
798.
799.                        mov a, r2
800.                        add a, #0x1
801.                        mov r2, a
802.
803.                        mov a, r0
804.                        add a, #0x1
805.                        mov r0, a
806.
807.                                        unitsloop: ; for bounce
808.                                        lcall ADC_to_PB
809.                                        jnb PB2, unitsloop
810.
811.        buttoncheck: ; if any of these buttons are pressed, return to the original function
812.                lcall ADC_to_PB
813.                jnb PB4, GOHOME
814.                jnb PB5, GOHOME
815.                jnb PB6, GOHOME
816.
```

```
817.        retval: ljmp Set_Value_loop
818.
819.        GOHOME: ret
820.
821.    ; Wait 10us
822.
823.    Wait10us:
824.        mov R0, #18
825.        djnz R0, $ ; 2 machine cycles-> 2*0.27126us*18=10us
826.        ret
827.
828.    ;----------;
829.    ; Messages ;
830.    ;----------;
831.
832.    Title: db 'Oven Controller!\r\n', 0
833.    TimeSoakMessage: db 'Enter soak time: \r\n', 0
834.    TempSoakMessage: db 'Enter soak temp: \r\n', 0
835.    TimeReflowMessage: db 'Enter reflow time: \r\n', 0
836.    TempReflowMessage: db 'Enter reflow temp: \r\n', 0
837.    Blank: db '      ',0
838.    Weback: db 'We are back\r\n', 0
839.    forwardsuccess: db 'Onward!\n', 0
840.    backwardsuccess: db 'Backward!\r\n', 0
841.    Success: db 'We made it\r\n', 0
842.    Blank1: db '                ', 0
843.    Blank2: db '                ', 0
844.    Welcome: db 'Welcome!', 0
845.    StartingOven: db 'Starting Oven', 0
846.    EnteringSoak: db 'Entering soak', 0 ; 0vvvvv
847.    EnteringReflow: db 'Entering reflow', 0
848.    SolderingComplete: db 'Soldering complete', 0
849.    ReflowComplete: db 'Reflow complete', 0
850.
851.    MainProgram:
852.        mov SP, #0x7F
853.
854.    ;---------------------------------;
855.    ; Initial configuration of ports. ;
856.    ; After reset the default for the ;
857.    ; pins is 'Open Drain'.  This     ;
858.    ; routine changes them pins to    ;
859.    ; Quasi-bidirectional like in the ;
860.    ; original 8051.                  ;
861.    ; Notice that P1.2 and P1.3 are   ;
862.    ; always 'Open Drain'. If those   ;
863.    ; pins are to be used as output   ;
864.    ; they need a pull-up resistor.   ;
865.    ;---------------------------------;
866.
867.        mov P0M1, #00H
868.        mov P0M2, #00H
869.        mov P1M1, #00H
870.        mov P1M2, #00H ; WARNING: P1.2 and P1.3 need 1kohm pull-up resistors!
871.        mov P2M1, #00H
872.        mov P2M2, #00H
873.        mov P3M1, #00H
874.        mov P3M2, #00H
875.
876.        lcall InitADC0
877.        lcall InitDAC1 ; Call after 'Ports_Init'
878.        lcall CCU_Init
879.        lcall Init_SPI
880.        lcall LCD_4bit
881.
882.        setb EA ; Enable global interrupts.
883.
884.    ;-------------------------------------------;
885.    ;                     Input program                              ;
```

```
886.    ;-------------------------------------------;
887.
888.    ;  1. Main menu
889.    ;  2. Soak time input mode
890.    ;  3. Soak temperature input mode
891.    ;  4. Reflow time input mode
892.    ;  5. Reflow temperature input mode
893.    ;  6. Wait for user to hit start
894.
895.    start:
896.
897.        mov timesoak, #0 ; initialize all reflow parameter variables to 0
898.        mov tempsoak, #0
899.        mov timereflow, #0
900.        mov tempreflow, #0
901.
902.        clr SSRpin
903.
904.        lcall DoubleInitSerialPort
905.        lcall Double_Clk
906.
907.        ; Speaker state #1: "Welcome to the oven!"
908.
909.        mov instruction, #2
910.        mov statecounter, #0
911.        lcall speak
912.        mov instruction, #0
913.
914.        mov somecounter, #0
915.
916.        Set_Cursor(1,1)
917.        Send_Constant_String(#Welcome)
918.
919.        lcall Wait1s
920.        lcall Wait1s
921.        lcall Wait1s
922.
923.        lcall InitSerialPort
924.        lcall Single_Clk
925.
926.        sjmp mainmenu
927.
928.    SoaktimeB: ; for bounce
929.    lcall ADC_to_PB
930.    jnb PB4, SoaktimeB
931.
932.    ; 1. Main menu
933.
934.    mainmenu:
935.        mov a, #'\r' ; move cursor all the way to the left
936.        lcall putchar
937.
938.        mov a, #'\n' ; move cursor all the way to the left
939.        lcall putchar
940.
941.        mov dptr, #Title
942.        lcall SendString
943.
944.    mainmenu2:
945.        lcall ADC_to_PB
946.        jnb PB6, mainmenuwaitloop ; jumps to soak time input if forward is pressed
947.        sjmp mainmenu2
948.
949.    mainmenuwaitloop:
950.    lcall ADC_to_PB
951.    jnb PB6, mainmenuwaitloop
952.
953.    lcall Wait10ms
954.    lcall Wait10ms
```

```
955.    lcall Wait10ms
956.    lcall Wait10ms
957.    lcall Wait10ms
958.    lcall Wait10ms
959.    lcall Wait10ms
960.    lcall Wait10ms
961.    lcall Wait10ms
962.    lcall Wait10ms
963.
964.    ; 2. Soak time input mode
965.
966.    SoakTempB:
967.    lcall ADC_to_PB
968.    jnb PB4, SoakTempB
969.
970.    soaktime:
971.
972.        mov a, #'\r' ; move cursor all the way to the left
973.        lcall putchar
974.
975.        mov DPTR, #TimeSoakMessage
976.        lcall SendString
977.
978.        Set_Cursor(1,1)
979.        Send_Constant_String(#TimeSoakMessage)
980.
981.        lcall Set_Value
982.        mov timesoak, r2
983.
984.        mov a, #'\n' ; move cursor all the way to the left
985.        lcall putchar
986.
987.
988.    soaktimeforward: jnb PB6, SoakTimeA ; next routine
989.
990.    back1:
991.        jnb PB4, SoakTimeBinter ; goes back to main menu
992.        sjmp soaktimeclear
993.        SoakTimeBinter:  ljmp SoakTimeB
994.
995.    soaktimeclear:
996.        jnb PB5, SoakTimeC
997.        ljmp soaktime
998.
999.        SoakTimeC:
1000.       lcall ADC_to_PB
1001.       jnb PB5, SoakTimeC
1002.
1003.       mov timesoak, #0
1004.       ljmp soaktime
1005.
1006.   ; 3. Soak temperature input mode
1007.
1008.   SoakTimeA:
1009.   lcall ADC_to_PB
1010.   jnb PB6, SoaktimeA
1011.
1012.   lcall Wait10ms
1013.   lcall Wait10ms
1014.   lcall Wait10ms
1015.   lcall Wait10ms
1016.   lcall Wait10ms
1017.   lcall Wait10ms
1018.   lcall Wait10ms
1019.   lcall Wait10ms
1020.   lcall Wait10ms
1021.   lcall Wait10ms
1022.
1023.   ReflowTimeB:
```

```
1024.   lcall ADC_to_PB
1025.   jnb PB4, ReflowTimeB
1026.
1027.   soaktemp:
1028.
1029.       mov a, #'\r' ; move cursor all the way to the left
1030.       lcall putchar
1031.
1032.       mov dptr, #TempSoakMessage
1033.       lcall SendString
1034.
1035.       Set_Cursor(1,1)
1036.       Send_Constant_String(#TempSoakMessage)
1037.
1038.       lcall Set_Value
1039.       mov tempsoak, r2
1040.
1041.       mov a, #'\n' ; move cursor all the way to the left
1042.       lcall putchar
1043.
1044.   soaktempforward: jnb PB6, SoakTempA ; next routine
1045.
1046.   back2: jnb PB4, back2inter ; goes back to soak time
1047.
1048.   sjmp soaktempclear
1049.
1050.   back2inter: ljmp SoakTempB
1051.
1052.   soaktempclear:
1053.       jnb PB5, SoakTempC
1054.       ljmp soaktemp
1055.
1056.       SoakTempC:
1057.       lcall ADC_to_PB
1058.       jnb PB5, SoakTempC
1059.
1060.       mov timesoak, #0
1061.       ljmp soaktemp
1062.
1063.   ;  4. Reflow time input mode
1064.
1065.   SoakTempA:
1066.   lcall ADC_to_PB
1067.   jnb PB6, SoakTempA
1068.
1069.   ReflowTempB:
1070.   lcall ADC_to_PB
1071.   jnb PB4, ReflowTempB
1072.
1073.   reflowtime:
1074.       mov a, #'\r' ; move cursor all the way to the left
1075.       lcall putchar
1076.
1077.       mov dptr, #TimeReflowMessage
1078.       lcall SendString
1079.
1080.       Set_Cursor(1,1)
1081.       Send_Constant_String(#TimeReflowMessage)
1082.
1083.       lcall Set_Value
1084.       mov timereflow, r2
1085.
1086.       mov a, #'\n' ; move cursor all the way to the left
1087.       lcall putchar
1088.
1089.   reflowtimeforward: jnb PB6, ReflowTimeA ; next routine
1090.
1091.   back3: jnb PB4, ReflowTimeBinter ; goes back to soak time
1092.   sjmp reflowtimeclear
```

```
ReflowTimeBinter: ljmp ReflowTimeB

reflowtimeclear:
    jnb PB5, ReflowTimeC
    ljmp reflowtime

    ReflowTimeC:
    lcall ADC_to_PB
    jnb PB5, ReflowTimeC

    mov timereflow, #0
    ljmp reflowtime

; Reflow temp input mode

ReflowTimeA:
lcall ADC_to_PB
jnb PB6, ReflowTimeA

Waitreturn:
lcall ADC_to_PB
jnb PB4, Waitreturn

reflowtemp:
    mov a, #'\r' ; move cursor all the way to the left
     lcall putchar

    mov dptr, #TempReflowMessage
    lcall SendString

    Set_Cursor(1,1)
    Send_Constant_String(#TempReflowMessage)

    lcall Set_Value
    mov tempreflow, r2

    mov a, #'\n' ; move cursor all the way to the left
     lcall putchar

reflowtempforward: jnb PB6, ReflowTempA ; next routine

back4: jnb PB4, ReflowTempBinter ; goes back to soak time
sjmp reflowtempclear
ReflowTempBinter: ljmp ReflowTempB

reflowtempclear:
    jnb PB5, ReflowTempC
    ljmp reflowtemp

    ReflowTempC:
    lcall ADC_to_PB
    jnb PB5, ReflowTempC

    mov tempreflow, #0
    ljmp reflowtemp

; 5. Wait for user to hit start

ReflowTempA:
lcall ADC_to_PB
jnb PB6, ReflowTempA

mov a, #'\r' ; move cursor all the way to the left
lcall putchar

mov dptr, #Success
lcall SendString

mov a, #'\n' ; move cursor all the way to the left
```

```
1162.   lcall putchar
1163.
1164.   wait:
1165.   lcall ADC_to_PB
1166.   jnb PB4, Waitreturn
1167.   jnb PB6, startoven
1168.
1169.       mov a, #'\r' ; move cursor all the way to the left
1170.       lcall putchar
1171.
1172.   sjmp wait
1173.
1174.   ;------------;
1175.   ; START OVEN ;
1176.   ;------------;
1177.
1178.   ; Algorithm:
1179.
1180.   ; At this point, input values have been set, available in variables:
1181.   ; timesoak, tempsoak, timereflow, tempreflow
1182.   ; Step 1: Switch on SSR, switching on oven
1183.   ; Step 2: Check 1 - if tempsoak achieved:
1184.       ; a: Switch off SSR
1185.       ; b: Switch on soaktimer
1186.   ; Step 3:
1187.       ; Check 2.1 - if soaktimer = timesoak, switch on oven and don't switch off
1188.       ; Check 2.2 - if oven temp < tempsoak - 3 degrees, switch on oven / else if oven temp
    > tempsoak + 3 degrees, switch off oven / else, chill
1189.   ; Step 4: Check 3 - if tempreflow achieved:
1190.       ; a. Switch off SSR
1191.       ; b. switch on reflowtimer
1192.   ; Step 5:
1193.       ; Check 4.1  - if reflowtimer = timereflow, switch on oven and don't switch off
1194.       ; Check 4.2 - if oven temp < reflowtemp - 3 degrees, switch on oven / else if oven
    temp > reflowtemp + 3 degrees, switch off oven/ else, chill
1195.
1196.   ; NOTES:
1197.   ; 1. 'theoatmeal' refers to a state that is 'just right'. In an earlier version of the
    code, this was the plateau stage. It now refers to
1198.   ; the transition stage immediately before the reflow ramp begins.
1199.   ; 2. 'Result' is the temperature input from the oven, updated in the readtemperature code.
1200.
1201.   startoven:
1202.
1203.   ; Step 1: Switch on SSR, switching on oven
1204.
1205.       lcall Double_Clk
1206.       lcall DoubleInitSerialPort
1207.
1208.       clr a
1209.       setb SSRpin
1210.
1211.       mov soaktimer, #0
1212.       mov reflowtimer, #0
1213.       mov soakramptimer, #0
1214.       mov time, #0
1215.
1216.       mov statecounter, #1
1217.       setb statechangeflag
1218.
1219.       Set_Cursor(1,1)
1220.       Send_Constant_String(#Blank1)
1221.
1222.       Set_Cursor(2,1)
1223.       Send_Constant_String(#Blank2)
1224.
1225.       Set_Cursor(1,1)
1226.       Send_Constant_String(#StartingOven)
1227.
```

```
1228.   ; Step 2: Check 1 - if Result > tempsoak:
1229.       ; a: Switch off SSR
1230.       ; b: Switch on soaktimer
1231.
1232.   ;- SOAK RAMP -;
1233.
1234.   soakramploop:
1235.
1236.       lcall readtemperature
1237.       clr c
1238.       mov a, tempsoak
1239.       subb a, #15 ; jumps to pwm 40 degrees below required temperature
1240.       subb a, Result
1241.       jnc soakramploop
1242.
1243.   ;   clr SSRpin ; Step 2a: switch off SSR
1244.
1245.   pwmloop:
1246.       setb SSRpin
1247.       lcall readtemperature
1248.       lcall soakcompletecheck
1249.       lcall readtemperature
1250.       lcall soakcompletecheck
1251.       lcall readtemperature
1252.       lcall soakcompletecheck
1253.       lcall readtemperature
1254.       lcall soakcompletecheck
1255.       clr SSRpin
1256.       lcall readtemperature
1257.       lcall soakcompletecheck
1258.       lcall readtemperature
1259.       lcall soakcompletecheck
1260.       lcall readtemperature
1261.       lcall soakcompletecheck
1262.       lcall readtemperature
1263.       lcall soakcompletecheck
1264.       lcall readtemperature
1265.       lcall soakcompletecheck
1266.       lcall readtemperature
1267.       lcall soakcompletecheck
1268.       lcall readtemperature
1269.       lcall soakcompletecheck
1270.
1271.       ljmp pwmloop
1272.
1273.       ; if Result > tempsoak, jump to soakmesa - else, jump back and switch on oven again
1274.
1275.   soakcompletecheck:
1276.
1277.       clr c
1278.       mov a, tempsoak
1279.       subb a, Result
1280.       jc soakmesapreloop
1281.
1282.       ret
1283.
1284.   ;- SOAK MESA -;
1285.
1286.   soakmesapreloop:
1287.
1288.       mov statecounter, #2
1289.       setb statechangeflag
1290.
1291.       Set_Cursor(1,1)
1292.       Send_Constant_String(#EnteringSoak)
1293.
1294.   soakmesaloop: ;
1295.       lcall readtemperature ; gets temperature once a second - this is functionally a one
    second wait
```

```
1296.       mov a, soaktimer
1297.       add a, #0x01
1298.       mov soaktimer, a; increments timer by one
1299.       cjne a, timesoak, soakmesaloop1 ;  Check 2.1 - soak time
1300.
1301.       ljmp theoatmeal
1302.
1303.       ; Check 2.2 - soak temperature +-3 (Result +-3)
1304.
1305.   soakmesaloop1:
1306.       ; Result > tempsoak + 3
1307.
1308.       clr c
1309.       mov a, tempsoak
1310.       add a, #0x01 ; 3 degree tolerance
1311.       subb a, Result
1312.       jnc soakmesaloop2
1313.
1314.       clr SSRpin
1315.       lcall soakcontrolroutine1
1316.
1317.   soakmesaloop2:
1318.       ; Result < tempsoak - 3 == Result + 3 < tempsoak
1319.       clr c
1320.       mov a, Result
1321.       add a, #0x01 ; 3 degree tolerance
1322.       subb a, tempsoak
1323.       jnc soakmesaloop
1324.
1325.       setb SSRpin
1326.       lcall soakcontrolroutine2
1327.
1328.   sjmp soakmesaloop
1329.
1330.   ; SOAKCONTROLROUTINE1 ;
1331.
1332.   soakcontrolroutine1:
1333.
1334.       lcall readtemperature ; reads temperature once per second
1335.       mov a, soaktimer
1336.       add a, #0x01
1337.       mov soaktimer, a; increments timer by one
1338.       cjne a, timesoak, soakcontrolroutinecheck1 ;  Check 2.1 - soak time
1339.
1340.       ljmp theoatmeal
1341.
1342.   ; Result > tempsoak + 2 control loop
1343.   ; oven starts in off state in this loop
1344.   ; while Result > tempsoak + 2: stay here
1345.       ; return
1346.
1347.   soakcontrolroutinecheck1:
1348.
1349.       clr c
1350.       mov a, tempsoak
1351.       add a, #0x01 ; 2 degree tolerance
1352.       subb a, Result
1353.       jc soakcontrolroutine1
1354.
1355.       ret ; if Result drops below tempsoak - 2, return to original loop
1356.
1357.   ; SOAKCONTROLROUTINE2 ;
1358.
1359.   soakcontrolroutine2:
1360.
1361.       lcall readtemperature ; reads temperature once per second
1362.       mov a, soaktimer
1363.       add a, #0x01
1364.       mov soaktimer, a; increments timer by one
```

```
1365.       cjne a, timesoak, soakcontrolroutinecheck2 ;  Check 2.1 - soak time
1366.
1367.       ljmp theoatmeal
1368.
1369.   ; Result < tempsoak + 2 control loop
1370.   ; oven starts in off state in this loop
1371.   ; pseudocode:
1372.   ; while Result > tempsoak + 2: stay here
1373.       ; return
1374.
1375.   soakcontrolroutinecheck2:
1376.
1377.       clr c
1378.       mov a, Result
1379.       add a, #0x01 ; 2 degree tolerance
1380.       subb a, tempsoak
1381.       jc soakcontrolroutine2
1382.
1383.       ret
1384.
1385.   theoatmeal:
1386.   setb SSRpin
1387.
1388.   ;- REFLOWRAMP -;
1389.
1390.   reflowramploop: ;
1391.
1392.       lcall readtemperature
1393.       clr c
1394.       mov a, tempreflow
1395.       subb a, #15 ; jumps to pwm 40 degrees below required temperature
1396.       subb a, Result
1397.       jnc reflowramploop
1398.
1399.       ljmp pwmloop2
1400.
1401.       ;setb P2.5
1402.
1403.   ;   clr SSRpin ; Step 2a: switch off SSR
1404.
1405.   pwmloop2:
1406.       setb SSRpin
1407.       lcall readtemperature
1408.       lcall reflowcompletecheck
1409.       lcall readtemperature
1410.       lcall reflowcompletecheck
1411.       lcall readtemperature
1412.       lcall reflowcompletecheck
1413.       lcall readtemperature
1414.       lcall reflowcompletecheck
1415.       lcall readtemperature
1416.       lcall reflowcompletecheck
1417.       lcall readtemperature
1418.       lcall reflowcompletecheck
1419.       lcall readtemperature
1420.       lcall reflowcompletecheck
1421.       lcall readtemperature
1422.       lcall reflowcompletecheck
1423.       lcall readtemperature
1424.       lcall reflowcompletecheck
1425.       clr SSRpin
1426.       lcall readtemperature
1427.       lcall reflowcompletecheck
1428.       lcall readtemperature
1429.       lcall reflowcompletecheck
1430.       lcall readtemperature
1431.       lcall reflowcompletecheck
1432.       lcall readtemperature
1433.       lcall reflowcompletecheck
```

```
1434.       lcall readtemperature ; if Result > tempsoak, jump to soakmesa - else, jump back and
    switch on oven again
1435.
1436.       ljmp pwmloop2
1437.
1438.   reflowcompletecheck:
1439.
1440.       clr c
1441.       mov a, tempreflow
1442.       subb a, Result
1443.       jc reflowmesaloopppre
1444.
1445.       ret
1446.
1447.   ;- REFLOWMESA -;
1448.
1449.   reflowmesaloopppre:
1450.
1451.       mov statecounter, #3
1452.       setb statechangeflag
1453.
1454.       Set_Cursor(1,1)
1455.       Send_Constant_String(#EnteringReflow)
1456.
1457.   reflowmesaloop:
1458.       lcall readtemperature ; gets temperature once a second - this is functionally a one
    second wait
1459.       mov a, reflowtimer
1460.       add a, #0x01
1461.       mov reflowtimer, a; increments timer by one
1462.       cjne a, timereflow, reflowmesaloop1 ;  Check 2.1 - reflow time
1463.
1464.       ljmp cool
1465.
1466.       ; Check 2.2 - reflow temperature +-3 (Result +-3)
1467.
1468.   reflowmesaloop1:
1469.       ; Result > tempreflow + 3
1470.       clr c
1471.       mov a, tempreflow
1472.       add a, #0x01 ; 3 degree tolerance
1473.       subb a, Result
1474.       jnc reflowmesaloop2
1475.
1476.       clr SSRpin
1477.       lcall reflowcontrolroutine1
1478.
1479.   reflowmesaloop2:
1480.       ; Result < tempreflow - 3 == Result + 3 < tempreflow
1481.       clr c
1482.       mov a, Result
1483.       add a, #0x01 ; 3 degree tolerance
1484.       subb a, tempreflow
1485.       jnc reflowmesaloop
1486.
1487.       setb SSRpin
1488.       lcall reflowcontrolroutine2
1489.
1490.   sjmp reflowmesaloop
1491.
1492.   ; REFLOWCONTROLROUTINE1 ;
1493.
1494.   reflowcontrolroutine1:
1495.
1496.       lcall readtemperature ; reads temperature once per second
1497.       mov a, reflowtimer
1498.       add a, #0x01
1499.       mov reflowtimer, a; increments timer by one
1500.       cjne a, timereflow, reflowcontrolroutinecheck1 ;  Check 2.1 - reflow time
```

```
1501.
1502.       ljmp cool
1503.
1504.   ; Result > tempreflow + 2 control loop
1505.   ; oven starts in off state in this loop
1506.   ; while Result > tempreflow + 2: stay here
1507.       ; return
1508.
1509.   reflowcontrolroutinecheck1:
1510.
1511.       clr c
1512.       mov a, tempreflow
1513.       add a, #0x01 ; 2 degree tolerance
1514.       subb a, Result
1515.       jc reflowcontrolroutine1
1516.
1517.       ret ; if Result drops below tempreflow - 2, return to original loop
1518.
1519.   ; REFLOWCONTROLROUTINE2 ;
1520.
1521.   reflowcontrolroutine2:
1522.
1523.       lcall readtemperature ; reads temperature once per second
1524.       mov a, reflowtimer
1525.       add a, #0x01
1526.       mov reflowtimer, a; increments timer by one
1527.       cjne a, timereflow, reflowcontrolroutinecheck2 ;  Check 2.1 - reflow time
1528.
1529.       ljmp cool
1530.
1531.   ; Result < tempreflow + 2 control loop
1532.   ; oven starts in off state in this loop
1533.   ; while Result > tempreflow + 2: stay here
1534.       ; return
1535.
1536.   reflowcontrolroutinecheck2:
1537.
1538.       clr c
1539.       mov a, Result
1540.       add a, #0x01 ; 2 degree tolerance
1541.       subb a, tempreflow
1542.       jc reflowcontrolroutine2
1543.
1544.       ret
1545.
1546.   cool:
1547.   clr SSRpin
1548.   mov statecounter, #4
1549.
1550.   setb statechangeflag
1551.
1552.   Set_Cursor(1,1)
1553.   Send_Constant_String(#ReflowComplete)
1554.
1555.   lcall readtemperature
1556.
1557.   ;--------------;
1558.   ; TEXT MESSAGE ;
1559.   ;--------------;
1560.
1561.   ; Sends a 01 flag that indicates that a text message should be sent
1562.
1563.   mov textvar, #1
1564.
1565.   mov x, textvar
1566.   mov x+1, #0
1567.   mov x+2, #0
1568.   mov x+3, #0
1569.
```

```
1570.   lcall hex2bcd
1571.
1572.   lcall SendTemp
1573.
1574.   lcall Wait1s
1575.   lcall Wait1s
1576.
1577.   ; COOL DOWN
1578.
1579.   cooldown:
1580.   lcall readtemperature
1581.
1582.   clr c
1583.   mov a, Result
1584.   subb a, #50
1585.   jnc cooldown
1586.
1587.   lcall Wait1s
1588.   lcall Wait1s
1589.   lcall Wait1s
1590.   lcall Wait1s
1591.
1592.   mov instruction, #2
1593.   mov statecounter, #5
1594.   lcall speak
1595.   mov soakramptimer, #0
1596.
1597.   Set_Cursor(1,1)
1598.   Send_Constant_String(#SolderingComplete)
1599.
1600.   done:
1601.   clr SSRpin
1602.   sjmp done
1603.
1604.   ;-----------------;
1605.   ; READTEMPERATURE ;
1606.   ;-----------------;
1607.
1608.   ; Functionality:
1609.   ;   1. Reads room temperature
1610.   ;   2. Reads relative oven temperature
1611.   ;   3. Calculates the final oven temperature using an average function, and stores it in
    variable 'Result'
1612.   ;   4. Sends a signal to the SPEAK function to 'speak' the temperature every 5 seconds
1613.   ;   5. 60 second abort check: if temperature < 50 Celsius after the oven has heated for 60
    seconds, ABORT (shut down the oven)
1614.
1615.   readtemperature:
1616.       clr dog ; variable initalization, flag used for read temp
1617.       mov a, soakramptimer ; soakramptimer stores time elapsed since the start of the reflow
    cycle
1618.       add a, #0x01
1619.       mov soakramptimer, a
1620.
1621.       mov a, time
1622.       add a, #0x01
1623.       mov time, a
1624.
1625.       mov x, time
1626.       mov x+1, #0
1627.       mov x+2, #0
1628.       mov x+3, #0
1629.
1630.       lcall hex2bcd
1631.
1632.       Set_Cursor(2,1)
1633.       Display_BCD(bcd+1)
1634.       Display_BCD(bcd) ; displays continuous timer
1635.
```

```
1636.       mov instruction, #0
1637.
1638.   check_inputs:
1639.
1640.     lcall get_thermostat
1641.     lcall oven_temp
1642.
1643.     mov x, sheep ; reference temperature
1644.     mov x+1, #0
1645.     mov x+2, #0
1646.     mov x+3, #0
1647.
1648.     mov y, cat ; oven temperature
1649.     mov y+1, #0
1650.     mov y+2, #0
1651.     mov y+3, #0
1652.
1653.     lcall add32
1654.     mov Result, x ; final oven temperature value stored in 'Result'
1655.     mov Result+1, x+1
1656.
1657.     lcall hex2bcd
1658.
1659.     Set_Cursor(2,12)
1660.     Display_BCD(bcd+1)
1661.     Display_BCD(bcd)
1662.
1663.     lcall SendTemp
1664.
1665.   ;----------------------------;
1666.   ;   SAFETY CRITICAL:               ;
1667.   ;   60 SECOND ABORT CHECK    ;
1668.
1669.     mov a, time
1670.     cjne a, #60, continuereadtemp
1671.
1672.     clr c
1673.     mov a, Result
1674.     subb a, #51
1675.     jnc continuereadtemp
1676.
1677.     ljmp done
1678.
1679.   ; CHECK ENDS HERE ;
1680.   ;-----------------;
1681.
1682.   ; The following code sends the 'Result' value, as well as its hundreds, tens and units
    values to the 'SPEAK' function every 5 seconds
1683.
1684.   continuereadtemp:
1685.
1686.     mov a, Result
1687.     mov b, #100
1688.     div ab
1689.     mov hundredsresult, a
1690.
1691.     mov a, b
1692.     mov b, #10
1693.     div ab
1694.     mov tensresult, a
1695.
1696.     mov unitsresult, b
1697.
1698.       mov a, soakramptimer
1699.       mov b, #5
1700.       div ab
1701.       mov a, b
1702.
1703.     cjne a, #0, returnsequence
```

```
1704.     jb statechangeflag, returnstatesequence
1705.
1706.       mov hundredsj, hundredsresult
1707.       mov tensj, tensresult
1708.       mov unitsj, unitsresult
1709.
1710.       mov temp, Result
1711.
1712.       mov instruction, #1
1713.
1714.   returnsequence:
1715.
1716.       lcall speak
1717.
1718.       lcall Wait1s
1719.       lcall Wait1s
1720.
1721.       ret
1722.
1723.   returnstatesequence:
1724.
1725.       mov instruction, #2
1726.       clr statechangeflag
1727.       lcall speak
1728.
1729.       lcall Wait1s
1730.       lcall Wait1s
1731.
1732.       ret
1733.
1734.   ; get_thermostat: gets the room temperature value from an LM335 temperature sensor to use
    as a reference
1735.
1736.   get_thermostat:
1737.       mov x+0, AD0DAT0
1738.       mov x+1, #0
1739.       mov x+2, #0
1740.       mov x+3, #0
1741.
1742.       ; Convert to temperature (C)
1743.       Load_Y(330) ; Vref is 3.3V
1744.       lcall mul32
1745.       Load_Y(255)
1746.       lcall div32
1747.       Load_Y(273)
1748.       lcall sub32
1749.
1750.       mov cat, x
1751.
1752.       ret
1753.
1754.   ; oven_temp: gets 255 oven temperature values and averages them (average = sum/255)
1755.
1756.   oven_temp:
1757.       Load_x(0)
1758.       mov x+0, AD0DAT3
1759.       mov x+1, #0
1760.       mov x+2, #0
1761.       mov x+3, #0
1762.       mov R7, #255
1763.       lcall Wait10us
1764.   acc_temp:
1765.       mov y+0, AD0DAT3
1766.       mov y+1, #0
1767.       mov y+2, #0
1768.       mov y+3, #0
1769.
1770.       lcall add32
1771.       lcall Wait10us
```

```
1772.       djnz R7, acc_temp
1773.
1774.       Load_y(255)
1775.       lcall div32
1776.
1777.       Load_Y(1000000)
1778.       lcall mul32
1779.
1780.       Load_Y(41)
1781.       lcall div32
1782.
1783.   ;R2 value
1784.       Load_Y(330)
1785.       lcall mul32
1786.
1787.       Load_Y(80)
1788.       lcall div32
1789.
1790.   ;R1 value
1791.       Load_Y(100000)
1792.       lcall div32
1793.
1794.       mov sheep, x
1795.
1796.       ret
1797.
1798.   ;-------;
1799.   ; SPEAK ;
1800.   ;-------;
1801.
1802.   ; Inputs:
1803.   ; 1. instruction: 0 = do nothing, 1 = read temperature, 2 = state transition
1804.   ; 2. statecounter: indicates which state transition is currently occurring - only used
      when instruction = 2
1805.   ; 3. temp, hundredsj, tensj, unitsj: the temperature of the oven, as well as its hundreds,
      tens and units value to be read out, received
1806.   ;    every 5 seconds
1807.
1808.   ; Outputs:
1809.   ; - If instruction = 0, nothing.
1810.   ; - If instruction = 1, reads out the current oven temperature.
1811.   ; - If instruction = 2, reads out the current state message.
1812.
1813.   speak:
1814.
1815.       mov a, instruction
1816.       cjne a, #0, start_routine_1
1817.
1818.       mov a, tmp_count
1819.       cjne a, #0, check_tens
1820.       ret
1821.
1822.   start_routine_1:
1823.       cjne a, #1, start_routine_2
1824.       mov tmp_count, #0
1825.       ljmp check_tens
1826.
1827.   start_routine_2:
1828.       mov a, statecounter
1829.       add a, #28
1830.       mov b, #3
1831.       mul ab
1832.
1833.       mov speak_num, a
1834.       ljmp play_temp
1835.
1836.   check_tens:
1837.       clr c
1838.       mov a, tensj
```

```
1839.       subb a, #2
1840.       jnc check_tmp_counter_2 ; if tens < 2, there will be a carry, so jump
1841.
1842.   define_last:
1843.
1844.       mov a, temp
1845.       mov b, #100
1846.       div ab
1847.       mov last, b  ;last = temp % 100
1848.
1849.   check_tmp_counter_1: ;tens < 2
1850.       mov a, tmp_count
1851.
1852.       cjne a, #1, check_hundreds ;if temp_count is 1, then speak last
1853.
1854.       mov tmp_count, #0
1855.       mov a, last
1856.       mov b, #3
1857.       mul ab
1858.       mov speak_num, a
1859.
1860.       ljmp play_temp
1861.
1862.   check_hundreds:
1863.       mov a, hundredsj
1864.       cjne a, #0, G1
1865.
1866.       mov tmp_count, #0 ;no hundreds, so just play last
1867.       mov a, last
1868.       mov b, #3
1869.       mul ab
1870.       mov speak_num, a
1871.
1872.       ljmp play_temp
1873.
1874.   G1: cjne a, #1, G2
1875.       mov speak_num, #102 ;say one hundred
1876.
1877.       mov tmp_count, #1
1878.       ljmp play_temp
1879.
1880.   G2: mov speak_num, #105 ;say two hundreds
1881.       mov tmp_count, #1
1882.
1883.       ljmp play_temp
1884.
1885.
1886.   check_tmp_counter_2: ;tens > 2
1887.
1888.       mov a, tmp_count
1889.
1890.       cjne a, #0, C1
1891.       mov tmp_count, #1 ;say hundreds
1892.       mov a, hundredsj
1893.       mov b, #3
1894.       mul ab
1895.
1896.       mov speak_num, a
1897.       ljmp play_temp
1898.
1899.   C1: cjne a, #1, C2
1900.       mov tmp_count, #2 ;say tens
1901.       mov a, tensj
1902.       add a, #18
1903.       mov b, #3
1904.       mul ab          ;index = 3(18 + tens)
1905.       mov speak_num, a
1906.       ljmp play_temp
1907.
```

```
1908.   C2: mov tmp_count, #0
1909.       mov a, unitsj ;say units
1910.       mov b, #3
1911.       mul ab
1912.       mov speak_num, a
1913.       ljmp play_temp
1914.
1915.   play_temp:
1916.
1917.       mov instruction, #0
1918.
1919.       clr TMOD20 ; Stop the CCU from playing previous request
1920.       setb FLASH_CE
1921.
1922.       clr FLASH_CE ; Enable SPI Flash
1923.       mov a, #READ_BYTES
1924.       lcall Send_SPI
1925.
1926.       mov dptr, #sound_index
1927.
1928.       mov a, speak_num
1929.       movc a, @dptr+a
1930.       lcall Send_SPI
1931.       inc dptr
1932.
1933.       mov a, speak_num
1934.       movc a, @dptr+a
1935.       lcall Send_SPI
1936.       inc dptr
1937.
1938.       mov a, speak_num
1939.       movc a, @dptr+a
1940.       lcall Send_SPI
1941.
1942.       mov dptr, #size_length
1943.
1944.       mov a, speak_num
1945.       movc a, @dptr+a
1946.       mov w+2, a
1947.       inc dptr
1948.
1949.       mov a, speak_num
1950.       movc a, @dptr+a
1951.       mov w+1, a
1952.       inc dptr
1953.
1954.       mov a, speak_num
1955.       movc a, @dptr+a
1956.       mov w+0, a
1957.
1958.       mov a, #0x00
1959.       lcall Send_SPI
1960.
1961.       setb TMOD20
1962.
1963.       ret
1964.
1965.   end
1966.
```

7.9 - Python Source Code

```
1.  import tkinter
2.  import pygame
3.  import serial
4.  import pygame
5.  import numpy as np
6.  import matplotlib.pyplot as plt
7.  import matplotlib.animation as animation
8.  from matplotlib.figure import Figure
9.  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
10. import sys, time, math
11. import twilio
12.
13. #Global Variables:
14. GRAPHLINECOLOUR ='red'
15. MAINSCREEN = 'oven.gif'
16. OPTIONSSCREEN = 'Oven2.gif'
17. USERNAME = 'test'
18. NOTIFICATION = 'not yet'
19. USERNUMBER = 121
20. GRAPHSIZE = 1
21. TESTMESSAGE = 'not yet'
22. GRAPHSCALE = 100
23.
24. #Com ports:
25. ARDUINOPORT = 'COM9' #Port of arduino
26. PUTTYPORT = 'COM8' #Port of Putty
27. PLACEBOARDUINOPORT = 'COM1' #Placebo port of arduino
28. PLACEBOPUTTYPORT = 'COM3'
29.
30. #test
31. i = 1
32.
33. #Global button Colours
34. BUTTONCOLOUR = '#33AAFF'
35. BUTTONCOLOURFLASH = 'grey'
36. BUTTONFOREGROUND = 'black'
37. OPTIONSBUTTONCOLOUR = '#3379FF'
38. GRAPHBUTTONCOLOUR = 'black'
39. TEXTMESSAGEBUTTONCOLOUR = 'red'
40.
41.
42.
43. ##########################################################################################################
    ########################################
44. def checkPuttyPort():
45.     global PLACEBOPUTTYPORT
46.     PLACEBOPUTTYPORT = PLACEBOPUTTYPORT.get()
47.
48.     showPutty = Label(window, text = 'Putty: ' + PLACEBOPUTTYPORT, font=("Comic Sans MS",
    16), background = 'white', fg = 'black')
49.     showPutty.place(x = 790, y = 500)
50.
51.     PLACEBOPUTTYPORT = "'" + PLACEBOPUTTYPORT + "'"
52.     #print(PUTTYPORT)
53.
54. ##########################################################################################################
    ########################################
55. def checkArduinoPort():
56.     global PLACEBOARDUINOPORT
57.     PLACEBOARDUINOPORT = PLACEBOARDUINOPORT.get()
58.
59.     showArduino = Label(window, text = 'Arduino: ' +    PLACEBOARDUINOPORT, font=("Comic Sans
    MS", 16), background = 'white', fg = 'black')
60.     showArduino.place(x = 790, y = 600)
61.
62.     PLACEBOARDUINOPORT = "'" + ARDUINOPORT + "'"
63.     PLACEBOARDUINOPORT = str(ARDUINOPORT)
64.     #print(ARDUINOPORT)
65.
```

```
66. ##########################################################################################################
    ######################################
67. def comSetter():
68.     global PLACEBOARDUINOPORT
69.     global PLACEBOPUTTYPORT
70.
71.     # Clearing everything:
72.     mainScreenCoverup = Label(window,width = 1200, height = 1000, background = 'white')
73.     mainScreenCoverup.place(x = 0,y = 450)
74.
75.     main2ScreenCoverup = Label(window,width = 1000, height = 1000, background = 'white')
76.     main2ScreenCoverup.place(x = 240, y = 500)
77.
78.     #Putty Port
79.     PLACEBOPUTTYPORT = Entry(window, background = 'grey')
80.     PLACEBOPUTTYPORT.place(x  = 790, y = 465)
81.
82.     enterPuttyButton = Button(window,text="Enter Putty port",width = 16, background =
    'yellow', fg = 'black', activebackground = 'grey',  command = checkPuttyPort)
83.     enterPuttyButton.place(x=650,y=460)
84.
85.     #Arduino port
86.     PLACEBOARDUINOPORT = Entry(window, background = 'grey')
87.     PLACEBOARDUINOPORT.place(x = 790, y = 565)
88.
89.     enterNumberButton = Button(window,text="Enter Arduino port",width = 16, background =
    'yellow', fg = 'black', activebackground = 'grey',command = checkArduinoPort)
90.     enterNumberButton.place(x=650,y=560)
91.
92.     doneButton = Button(window,text="Done",width = 16, background = 'yellow', fg = 'black',
    activebackground = 'grey', command = mainMenu)
93.     doneButton.place(x='650',y=660)
94.
95.
96. ##########################################################################################################
    ######################################
97.
98. def pythonGraph():
99.     global GRAPHLINECOLOUR
100.        print(GRAPHLINECOLOUR)
101.
102.        #Opening the port
103.        Putty = serial.Serial(port=
    PUTTYPORT,baudrate=115200,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_TWO)
104.        plt.close('all')
105.
106.        #previous putty
107.        previousPuttyNumber = 0 #
108.        puttyNumber = 0 #
109.
110.        def data_gen():
111.            t = data_gen.t
112.            puttyNumber = 0 #
113.
114.            while True:
115.                t+=1
116.                #Read output from putty serial port
117.                previousPuttyNumber = puttyNumber #
118.                getLine = Putty.read(4)
119.                puttyNumber = int(getLine)
120.                if (puttyNumber == 1):
121.                    x = sendRealSMS()
122.                    puttyNumber = previousPuttyNumber #
123.                    print('TEST') #
124.                #Convert the number into format readable by python
125.                print(puttyNumber)
126.                yield t, puttyNumber
127.
128.        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):
            plt.close('all')
            Putty.close()
            x = mainMenu()
            #sys.exit(0)

        #plt.ion(blocking)
        fig = plt.figure()

        try:
            while 1 :
                xsize=600
                data_gen.t = -1
                plt.pause(0.01)
                #fig = plt.figure() # uncommetning this makes infinite windows
                fig.canvas.mpl_connect('close_event', on_close_figure)
                ax = fig.add_subplot(111)
                line, = ax.plot([], [], lw=2,color = GRAPHLINECOLOUR)
                ax.set_ylim(-10, GRAPHSCALE)
                ax.set_xlim(0, xsize)
                ax.grid()
                xdata, ydata = [], []

                #fig.canvas.draw() # updates figure
                # 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=1, repeat=False)
                plt.show()
                #window.update()
        except:
            Putty.close()

    ##########################################################################################################################
    def setGraphMessage():
            #clearing everything
            mainScreenCoverup = Label(window,width = 1200, height = 1000, background = 'white')
            mainScreenCoverup.place(x = 0,y = 450)

            main2ScreenCoverup = Label(window,width = 1000, height = 1000, background = 'white')
            main2ScreenCoverup.place(x = 240, y = 500)

            finalLabel = Label(window,text = 'Done! Graph Scale set to:' + GRAPHSIZE, width = 2,height = 1, background = 'white', fg = 'black', font = ("MS Sans Serif", 20) )
            finalLabel.place(x = 880, y = 710)

            yellowButton = Button(window,text="Yellow",width = 16, background = GRAPHBUTTONCOLOUR, activebackground = BUTTONCOLOURFLASH,  fg = 'yellow', command = yellowLine)
            yellowButton.place(x=650,y=560)
            x =mainMenu()
```

```
    ##########################################################################################################################################
    #Arduino Graph
    def graphScaling():
        global GRAPHSIZE
        global GRAPHSCALE

        ratio = 100
        #cleraing data
        mainScreenCoverup = Label(window,width = 1200, height = 1000, background = 'white')
        mainScreenCoverup.place(x = 0,y = 450)

        main2ScreenCoverup = Label(window,width = 1000, height = 1000, background = 'white')
        main2ScreenCoverup.place(x = 240, y = 500)

        #Setting up distance and timers
        distanceHeaderLabel = Label(window,text = 'Distance', width = 7,height = 1, background = 'white', fg = 'black', font = ("MS Sans Serif", 20,UNDERLINE) )
        distanceHeaderLabel.place(x = 880, y = 470)

        cmLabel = Label(window,text = 'Cm', width = 2,height = 1, background = 'white', fg = 'black', font = ("MS Sans Serif", 20,UNDERLINE) )
        cmLabel.place(x = 930, y = 510)

        scaleHeaderLabel = Label(window, text = 'Scale' , width = 4,height = 1, background = 'white', fg = 'black', font = ("MS Sans Serif", 20,UNDERLINE))
        scaleHeaderLabel.place(x = 1175, y = 470)


        scaleLabel = Label(window,text = ' X 600', width = 4,height = 1, background = 'white', fg = 'black', font = ("MS Sans Serif", 20,UNDERLINE) )
        scaleLabel.place(x = 1200, y = 510)

        window.update()

        try:
            #Opening the port
            arduinoSerial = serial.Serial(port = ARDUINOPORT)
            arduinoOutput = arduinoSerial.readline()
            GRAPHSIZE = arduinoOutput.decode("utf-8").replace("\r\n", "\n")
            GRAPHSIZE = int(GRAPHSIZE)
            #print(GRAPHSIZE)
            displayColour = 'green'

            def on_close_figure(event):
                            plt.close('all')
                            #x = mainMenu()
                            #sys.exit(0)

            plt.ion()
            fig = plt.figure()

            #setting graphsize as variable
            a = StringVar()
            a.set(GRAPHSIZE)

            #getting base time
            baseSeconds = time.time()

            #setting scale as variable
            scaler = StringVar()
            scaler.set(GRAPHSIZE * ratio)

            while GRAPHSIZE > 1:
                #Read output from arduino serial port
                #while counter > 0:
                    arduinoOutput = arduinoSerial.readline()
                    GRAPHSIZE = arduinoOutput.decode("utf-8").replace("\r\n", "\n")
                    GRAPHSIZE = int(GRAPHSIZE)
```

```
                    xsize=600
                    fig.canvas.mpl_connect('close_event', on_close_figure)
                    ax = fig.add_subplot(111)
                    line, = ax.plot([], [], lw=2,color = 'red')
                    ax.set_ylim(0, GRAPHSIZE * ratio)
                    ax.set_xlim(0, xsize)
                    ax.grid()
                    plt.pause(0.1)
                    fig.canvas.draw()

                    #Setting variables
                    a.set(GRAPHSIZE)
                    #scaler.set(GRAPHSIZE * ratio)
                    #print('test')
                    #printing variables
                    #distance
                    distanceLabel = Label(window,textvariable = a, width = 2,height = 1, background = 'white', fg = 'black', font = ("MS Sans Serif", 20) )
                    distanceLabel.place(x = 880, y = 510)

                    distanceLabel.config(text=str(a))
                    aboutToSetLabel = Label(window, width = 4,height = 2, background = 'green', fg = 'black')
                    aboutToSetLabel.place(x = 1000, y = 510)

                    #scale
                    b = GRAPHSIZE * ratio
                    scaler.set(b)
                    scaleLabel = Label(window,textvariable = scaler , width = 3,height = 1, background = 'white', fg = 'black', font = ("MS Sans Serif", 20) )
                    scaleLabel.place(x = 1150, y = 510)
                    scaleLabel.config(text=str(a))


                    #Comparing seconds
                    compareSeconds =time.time()
                    testSeconds1 = compareSeconds - 5
                    testSeconds2 = compareSeconds - 8
                    testSeconds3 = compareSeconds - 11
                    count = 0

                    window.update()
                    if  (testSeconds1 > baseSeconds) and count == 0 :
                        aboutToSetLabel = Label(window, width = 4,height = 2, background = 'yellow', fg = 'black')
                        aboutToSetLabel.place(x = 1000, y = 510)
                        count += 1

                        window.update()

                    if  testSeconds2 > baseSeconds:
                        aboutToSetLabel = Label(window, width = 4,height = 2, background = 'red', fg = 'black')
                        aboutToSetLabel.place(x = 1000, y = 510)
                        count = 0

                        #scaleNumberLabel = Label(window,text = '100' + 'X 600' , width = 3,height = 1, background = 'white', fg = 'black', font = ("MS Sans Serif", 20) )
                        #scaleNumberLabel.place(x = 1200, y = 810)
                        window.update()
                        while count < 10000:
                            count+=1
                            print(count)
                        break

            window.update()
            plt.close('all')

            print('test')
```

```
            GRAPHSIZE = GRAPHSIZE
            GRAPHSCALE = GRAPHSIZE * ratio
            #print (GRAPHSCALE)
            arduinoSerial.close()
            x = mainMenu()


        except:
            errorLabel = Label(window, text = "Oooops!", font=("Comic Sans MS", 32), background = 'white', fg = 'black')
            errorLabel.place(x = 610, y = 460)

            error2Label = Label(window, text = "Looks like your COM port is wrong, please reset your ports", font=("Comic Sans MS", 32), background = 'white', fg = 'black')
            error2Label.place(x = 130, y = 540)

            #Button
            setComButton = Button(window,text="Set COM",width = 24, background ='yellow', fg = 'black', activebackground = 'grey', command = comSetter)
            setComButton.place(x = 620, y = 710)

            mainButton = Button(window,text="Go back",width = 24, background = 'yellow', fg = 'black', activebackground = 'grey', command = mainMenu)
            mainButton.place(x = 620, y = 760)


        #arduinoOutputLabel = Label(window,textvariable = a, width = 24, background = 'black', fg = 'white')
        #arduinoOutputLabel.place(x = 340, y = 710)


    ##########################################################################################################################################
    def introToOven():
        pygame.init()
        from pygame import mixer
        mixer.init()
        mixer.music.load("C:/Users/Marko/Desktop/Work/Elec291/project 1/Code/Python GUI/harshproper.mp3")
        mixer.music.play()

        #Clearing everyhing
        mainScreenCoverup = Label(window,width = 1200, height = 1000, background = 'white')
        mainScreenCoverup.place(x = 0,y = 450)

        main2ScreenCoverup = Label(window,width = 1000, height = 1000, background = 'white')
        main2ScreenCoverup.place(x = 240, y = 500)


        #intro
        first = Label(window, text = "Welcome to our Oven", font=("Comic Sans MS", 32), background = 'white', fg = 'black')
        first.place(x = 505, y = 460)

        #next line
        first2 = Label(window, text = "Where all soldering is possible", font=("Comic Sans MS", 32), background = 'white', fg = 'black')
        first2.place(x = 415, y = 560)


        #Button
        mainButton = Button(window,text="Proceed to Main Menu",width = 24, background = BUTTONCOLOUR, fg = 'black', activebackground = 'grey', command = mainMenu)
        mainButton.place(x = 640, y = 710)
        window.update()
        #x = motionProceed() #Change forms here
        #x = mainMenu()
```

```
375.    ##########################################################################################################################################
376.    def motionProceed():
377.        arduinoSerial = serial.Serial(port = ARDUINOPORT)
378.        arduinoSerial.isOpen()
379.
380.        arduinoOutput = arduinoSerial.readline()
381.        displayNumber = arduinoOutput.decode("utf-8").replace("\r\n", "\n")
382.        displayNumber = int(displayNumber)
383.
384.
385.        while displayNumber > 10:
386.            #print(displayNumber)
387.            arduinoOutput = arduinoSerial.readline()
388.            displayNumber = arduinoOutput.decode("utf-8").replace("\r\n", "\n")
389.            displayNumber = int(displayNumber)
390.
391.        arduinoSerial.close()
392.        x =mainMenu()
393.    ##########################################################################################################################################
394.
395.    ##########################################################################################################################################
396.    def startProcess():
397.
398.        # Clearing everything
399.        mainScreenCoverup = Label(window,width = 1000, height = 1000, background = 'white')
400.        mainScreenCoverup.place(x = 100,y = 450)
401.
402.        main2ScreenCoverup = Label(window,width = 1000, height = 1000, background = 'white')
403.        main2ScreenCoverup.place(x = 240, y = 500)
404.
405.        #Putty = serial.Serial(port=
    PUTTYPORT,baudrate=115200,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_TWO)
406.        #Putty.isOpen()
407.
408.        #Reprinting back button
409.        #Go back
410.        goBackButton = Button(window,text="Go back",width = 16, background = BUTTONCOLOUR, fg
    = 'black',  activebackground = BUTTONCOLOURFLASH, command = mainMenu())
411.        goBackButton.place(x=2,y=4)
412.
413.        soakTemperatureLabel = Label(window,text = 'Oven Temperature:', width = 15,height = 1,
    background = 'white', fg = 'black', font = ("MS Sans Serif", 20,UNDERLINE) )
414.        soakTemperatureLabel.place(x = 600, y = 470)
415.
416.        #Setting up the temp label
417.        #scaleLabel = Label(window,text = '28' , width = 3,height = 1, background = 'white',
    fg = 'black', font = ("MS Sans Serif", 80) )
418.        #scaleLabel.place(x = 640, y =550)
419.
420.        window.update()
421.    ##########################################################################################################################################
422.        #Opening the port
423.
424.        Putty = serial.Serial(port=
    PUTTYPORT,baudrate=115200,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_TWO)
425.        Putty.isOpen()
426.
427.        #Get line
428.        getLine = Putty.read(4)
429.        puttyNumber = int(getLine.decode())
430.
431.        a = StringVar()
432.        a.set(puttyNumber)
433.
434.        while 1:
```

```
            getLine = Putty.read(4)
            puttyNumber = int(getLine.decode())
            print(puttyNumber)
            a.set(puttyNumber)
            scaleLabel = Label(window,textvariable = a , width = 3,height = 1, background =
    'white', fg = 'black', font = ("MS Sans Serif", 80) )
            scaleLabel.place(x = 640, y = 550)
            scaleLabel.config(text=str(a))
            window.update()
            if (puttyNumber == 80):
                Putty.close()
                break


    ##########################################################################################
    ##############################################

    ##########################################################################################
    ######################################################################

    def add():
        global i
        i += 1
        a = StringVar()
        a.set(i)
        print("new"+ str(a))
        testLabel = Label(window,textvariable = a, width = 24, background = 'black', fg =
    'white')
        testLabel.place(x = 340, y = 710)
        testLabel.config(text=str(a))
        #window.update_idletasks()

    ##########################################################################################
    ######################################################################

    def credits():

        #Clearing screen
        mainScreenCoverup = Label(window,width = 1000, height = 1000, background = 'white')
        mainScreenCoverup.place(x = 100,y = 450)

        main2ScreenCoverup = Label(window,width = 1000, height = 1000, background = 'white')
        main2ScreenCoverup.place(x = 240, y = 500)

        #Reprinting back button
        #Go back
        goBackButton = Button(window,text="Go back",width = 16, background = BUTTONCOLOUR, fg
    = 'black',  activebackground = BUTTONCOLOURFLASH, command = mainMenu)
        goBackButton.place(x=2,y=4)

        # Printing names

        HarshLabel = Label(window,text = 'Harsh Rajoria', width = 15,height = 1, background =
    'white', fg = 'black', font = ("MS Sans Serif", 20) )
        HarshLabel.place(x = 584, y = 500)

        YemiLabel = Label(window,text = 'Yemi ' + 'DSEG ' + 'Oke', width = 15,height = 1,
    background = 'white', fg = 'black', font = ("MS Sans Serif", 20) )
        YemiLabel.place(x = 584, y = 550)

        JinLabel = Label(window,text = 'Jin Hee Yun', width = 15,height = 1, background =
    'white', fg = 'black', font = ("MS Sans Serif", 20) )
        JinLabel.place(x = 584, y = 600)

        MarkoLabel = Label(window,text = 'Marko Jurisic', width = 15,height = 1, background =
    'white', fg = 'black', font = ("MS Sans Serif", 20) )
        MarkoLabel.place(x = 570, y = 650)

```

```
494.        TaherLabel = Label(window,text = 'Taher Kathawala', width = 15,height = 1, background
    = 'white', fg = 'black', font = ("MS Sans Serif", 20) )
495.        TaherLabel.place(x = 570, y = 700)
496.
497.        TaherLabel = Label(window,text = 'Janith Wijekoon', width = 15,height = 1, background
    = 'white', fg = 'black', font = ("MS Sans Serif", 20) )
498.        TaherLabel.place(x = 570, y = 750)
499.
500.    ##########################################################################################
    ##########################################################################
501.
502.    # Change graph colours
503.    def redLine():
504.        global GRAPHLINECOLOUR
505.        #Clearing the screen
506.        clear = Label(window, background = 'white', width = 8)
507.        clear.place(x=800,y=460)
508.
509.        clear2 = Label(window, background = 'white', width = 8)
510.        clear2.place(x=800,y=510)
511.
512.        clear3 = Label(window, background = 'white', width = 8)
513.        clear3.place(x=800,y=570)
514.
515.        clear4 = Label(window, background = 'white', width = 8)
516.        clear4.place(x=800,y=630)
517.
518.        GRAPHLINECOLOUR = 'red'
519.        redBox = Label(window, background = 'red', width = 8)
520.        redBox.place(x=800,y=460)
521.
522.
523.
524.    def blueLine():
525.        global GRAPHLINECOLOUR
526.        #Clearing the screen
527.        clear = Label(window, background = 'white', width = 8)
528.        clear.place(x=800,y=460)
529.
530.        clear2 = Label(window, background = 'white', width = 8)
531.        clear2.place(x=800,y=510)
532.
533.        clear3 = Label(window, background = 'white', width = 8)
534.        clear3.place(x=800,y=570)
535.
536.        clear4 = Label(window, background = 'white', width = 8)
537.        clear4.place(x=800,y=630)
538.
539.        GRAPHLINECOLOUR = 'blue'
540.
541.        blueBox = Label(window, background = 'blue', width = 8)
542.        blueBox.place(x=800,y=510)
543.
544.    def yellowLine():
545.        global GRAPHLINECOLOUR
546.        #Clearing the screen
547.        clear = Label(window, background = 'white', width = 8)
548.        clear.place(x=800,y=460)
549.
550.        clear2 = Label(window, background = 'white', width = 8)
551.        clear2.place(x=800,y=510)
552.
553.        clear3 = Label(window, background = 'white', width = 8)
554.        clear3.place(x=800,y=570)
555.
556.        clear4 = Label(window, background = 'white', width = 8)
557.        clear4.place(x=800,y=630)
558.
559.        GRAPHLINECOLOUR = 'yellow'
```

```
560.
561.        yellowBox = Label(window, background = 'yellow', width = 8)
562.        yellowBox.place(x=800,y=570)
563.
564.    def orangeLine():
565.        global GRAPHLINECOLOUR
566.        #Clearing the screen
567.        clear = Label(window, background = 'white', width = 8)
568.        clear.place(x=800,y=460)
569.
570.        clear2 = Label(window, background = 'white', width = 8)
571.        clear2.place(x=800,y=510)
572.
573.        clear3 = Label(window, background = 'white', width = 8)
574.        clear3.place(x=800,y=570)
575.
576.        clear4 = Label(window, background = 'white', width = 8)
577.        clear4.place(x=800,y=630)
578.
579.        GRAPHLINECOLOUR = 'orange'
580.
581.        orangeBox = Label(window, background = 'orange', width = 8)
582.        orangeBox.place(x=800,y=630)
583.
584.
585.    #Function taken from:
    https://stackoverflow.com/questions/7966119/display-fullscreen-mode-on-tkinter
586.    class FullScreenApp(object):
587.        def __init__(self, master, **kwargs):
588.            self.master=master
589.            pad=3
590.            self._geom='200x200+0+0'
591.            master.geometry("{0}x{1}+0+0".format(
592.                master.winfo_screenwidth()-pad, master.winfo_screenheight()-pad))
593.            master.bind('<Escape>',self.toggle_geom)
594.        def toggle_geom(self,event):
595.            geom=self.master.winfo_geometry()
596.            print(geom,self._geom)
597.            self.master.geometry(self._geom)
598.            self._geom=geom
599.    ##########################################################################################
    ######################################
600.
601.    #Open the Ardunio serial port and read the data:
602.    def arduino():
603.            #Opening the port
604.            while 1:
605.                arduinoSerial = serial.Serial(port = PORT)
606.                arduinoOutput = arduinoSerial.readline()
607.                displayNumber =int(arduinoOutput.decode())
608.                print(displayNumber)
609.                #arduinoData = Label(window, textvariable = arduinoOutput, width = 25,
    background = 'black', fg = 'blue').place(x = 10, y =70)
610.
611.
612.    ##########################################################################################
    ######################################
613.    def testMessageCreate():
614.        global TESTMESSAGE
615.        Message = "\nHello " + USERNAME + "\nthis is a test message from your oven"
616.
617.        TESTMESSAGE = Message
618.
619.
620.    ##########################################################################################
    ######################################
621.
622.    def ovenNotification():
623.        global USERNAME
```

```
624.        global NOTIFICATION
625.        Message1 = "Hello " + USERNAME
626.        Message2 = ".\nYour oven has finished soldering!"
627.        Message3 = Message1 + Message2
628.
629.        NOTIFICATION = Message3
630.        print(NOTIFICATION)
631.
632.
633.    ##########################################################################################
    ########################################
634.
635.    def changeGraphColour():
636.        #Clearing screen
637.        mainScreenCoverup = Label(window,width = 1200, height = 1000, background = 'white')
638.        mainScreenCoverup.place(x = 0,y = 450)
639.
640.        main2ScreenCoverup = Label(window,width = 1000, height = 1000, background = 'white')
641.        main2ScreenCoverup.place(x = 240, y = 500)
642.
643.        redButton = Button(window,text="Red",width = 16, background = GRAPHBUTTONCOLOUR,
    activebackground = BUTTONCOLOURFLASH, fg = 'red', command = redLine)
644.        redButton.place(x=650,y=460)
645.
646.        blueButton = Button(window,text="Blue",width = 16, background = GRAPHBUTTONCOLOUR,
    activebackground = BUTTONCOLOURFLASH,  fg = 'blue', command = blueLine)
647.        blueButton.place(x=650,y=510)
648.
649.        yellowButton = Button(window,text="Yellow",width = 16, background = GRAPHBUTTONCOLOUR,
    activebackground = BUTTONCOLOURFLASH,  fg = 'yellow', command = yellowLine)
650.        yellowButton.place(x=650,y=560)
651.
652.        orangeButton = Button(window,text="Orange",width = 16, background = GRAPHBUTTONCOLOUR,
    activebackground = BUTTONCOLOURFLASH,  fg = 'orange',command = orangeLine)
653.        orangeButton.place(x=650,y=610)
654.    ##########################################################################################
    ########################################
655.
656.    def sendTestSMS():
657.        global TESTMESSAGE
658.        global USERNUMBER
659.        x = testMessageCreate()
660.
661.        # we import the Twilio client from the dependency we just installed
662.        from twilio.rest import Client
663.
664.        # the following line needs your Twilio Account SID and Auth Token
665.        client = Client("AC99e9e01f137d4f28043bba819a9c8cc3",
    "3ccbf1dbb4fcadbf8d2923957d33f824")
666.
667.        # change the "from_" number to your Twilio number and the "to" number
668.        # to the phone number you signed up for Twilio with, or upgrade your
669.        # account to send SMS to any phone number
670.        client.messages.create(to=USERNUMBER,
671.                               from_="+17316024317",
672.                               body=TESTMESSAGE)
673.
674.    ##########################################################################################
    ############################################
675.    def sendRealSMS():
676.        global NOTIFICATION
677.        global USERNUMBER
678.        x = testMessageCreate()
679.
680.        # we import the Twilio client from the dependency we just installed
681.        from twilio.rest import Client
682.
683.        # the following line needs your Twilio Account SID and Auth Token
```

```
684.        client = Client("AC99e9e01f137d4f28043bba819a9c8cc3",
    "3ccbf1dbb4fcadbf8d2923957d33f824")
685.
686.        # change the "from_" number to your Twilio number and the "to" number
687.        # to the phone number you signed up for Twilio with, or upgrade your
688.        # account to send SMS to any phone number
689.        client.messages.create(to=USERNUMBER,
690.                               from_="+17316024317",
691.                               body=NOTIFICATION)
692.
693.    ##########################################################################################
    ##########################################
694.
695.    def checkUserName():
696.        global USERNAME
697.        USERNAME = userName.get()
698.        #print(user)
699.        showUserName = Label(window, text = "Hello " + USERNAME + "!", font=("Comic Sans MS",
    16), background = 'white', fg = 'black')
700.        showUserName.place(x = 790, y = 500)
701.
702.    ##########################################################################################
    ########################################
703.
704.    def checkUserNumber():
705.        global USERNUMBER
706.        USERNUMBER = '+1' + userNum.get()
707.        #print(userNum)
708.        showUserName = Label(window, text = "Cool Number: " + USERNUMBER, font=("Comic Sans
    MS", 16), background = 'white', fg = 'black')
709.        showUserName.place(x = 790, y = 600)
710.
711.
712.    ##########################################################################################
    ########################################
713.
714.    def doneData():
715.        global USERNAME
716.        global USERNUMBER
717.
718.        #Clearing screen
719.        mainScreenCoverup = Label(window,width = 1200, height = 1000, background = 'white')
720.        mainScreenCoverup.place(x = 0,y = 450)
721.
722.        main2ScreenCoverup = Label(window,width = 1000, height = 1000, background = 'white')
723.        main2ScreenCoverup.place(x = 240, y = 500)
724.
725.        #Print text set message
726.        Message = Label(window,width = 60,  background = 'white', text = "Thank you " +
    USERNAME + ",you will be sent a text message at: ", font=("Comic Sans MS", 20), fg = 'black')
727.        Message.place(x = 220,y = 460)
728.
729.        Message2 = Label(window,width = 40,  background = 'white', text = USERNUMBER ,
    font=("Comic Sans MS", 20), fg = 'black')
730.        Message2.place(x = 395,y = 560)
731.
732.        x = ovenNotification()
733.
734.        SMSTestButton = Button(window,text="Send Test SMS",width = 16, background = 'green',
    fg = 'black', activebackground = 'grey', command = sendTestSMS)
735.        SMSTestButton.place(x = 650, y = 740)
736.
737.        homeButton = Button(window,text="Return",width = 16, background = 'green', fg =
    'black', activebackground = 'grey', command = mainMenu)
738.        homeButton.place(x= 650 ,y = 690)
739.
740.        redoButton = Button(window,text="Redo",width = 16, background = 'green', fg = 'black',
    activebackground = 'grey', command = textMessageCreator)
741.        redoButton.place(x = 650, y = 640)
```

```
742.
743.    ##########################################################################################################################################
744.
745.    def textMessageCreator():
746.        global userName
747.        global USERNUMBER
748.        global userNum
749.
750.        #Clearing screen
751.        mainScreenCoverup = Label(window,width = 1200, height = 1000, background = 'white')
752.        mainScreenCoverup.place(x = 0,y = 450)
753.
754.        main2ScreenCoverup = Label(window,width = 1000, height = 1000, background = 'white')
755.        main2ScreenCoverup.place(x = 240, y = 500)
756.
757.        #Username
758.        userName = Entry(window, background = 'grey')
759.        userName.place(x  = 790, y = 465)
760.
761.        enterNameButton = Button(window,text="Enter name",width = 16, background =
    TEXTMESSAGEBUTTONCOLOUR, fg = 'black', activebackground = 'grey',  command = checkUserName)
762.        enterNameButton.place(x=650,y=460)
763.
764.        #User Number
765.        userNum = Entry(window, background = 'grey')
766.        userNum.place(x = 790, y = 565)
767.
768.        enterNumberButton = Button(window,text="Enter number",width = 16, background =
    TEXTMESSAGEBUTTONCOLOUR, fg = 'black', activebackground = 'grey',command = checkUserNumber)
769.        enterNumberButton.place(x=650,y=560)
770.
771.        doneButton = Button(window,text="Done",width = 16, background =
    TEXTMESSAGEBUTTONCOLOUR, fg = 'black', activebackground = 'grey', command = doneData)
772.        doneButton.place(x='650',y=660)
773.
774.    ##########################################################################################################################################
775.
776.    def changeMainScreen():
777.        global MAINSCREEN
778.        MAINSCREEN = 'homer.gif'
779.
780.
781.    ##########################################################################################################################################
782.
783.    def optionsMenu():
784.        #graph colour
785.        graphColourButton = Button(window,text="Graph Colour",width = 16, background =
    OPTIONSBUTTONCOLOUR, fg = 'black', activebackground = BUTTONCOLOURFLASH,  command =
    changeGraphColour)
786.        graphColourButton.place(x=650,y=460)
787.
788.        #Change Units, no command yet, MUST ADD!
789.        #changeUnitsButton = Button(window,text="Change Units",width = 16, background =
    OPTIONSBUTTONCOLOUR, fg = 'black', activebackground = BUTTONCOLOURFLASH)
790.        #changeUnitsButton.place(x=650,y=510)
791.
792.        #Text Message
793.        textMessageButton = Button(window,text="Text Message",width = 16, background =
    OPTIONSBUTTONCOLOUR, fg = 'black',  activebackground = BUTTONCOLOURFLASH, command =
    textMessageCreator)
794.        textMessageButton.place(x=650,y=560)
795.
796.        #Oven
797.        #mainScreenButton = Button(window,text="Don't click the oven",width = 16, background =
    OPTIONSBUTTONCOLOUR, fg = 'black',  activebackground = BUTTONCOLOURFLASH, command =
    changeMainScreen)
```

```
798.        #mainScreenButton.place(x=650,y=610)
799.
800.        #Size graph
801.        graphSizeButton = Button(window,text="Graph size",width = 16, background =
    OPTIONSBUTTONCOLOUR, fg = 'black',  activebackground = BUTTONCOLOURFLASH, command =
    graphScaling) # Change the graphs scale here
802.        graphSizeButton.place(x=650,y=610)
803.
804.        #Set COM
805.        setCOMButton = Button(window,text="Set COMS",width = 16, background =
    OPTIONSBUTTONCOLOUR, fg = 'black',  activebackground = BUTTONCOLOURFLASH, command =
    comSetter)
806.        setCOMButton.place(x=650,y=510)
807.
808.        #Go back
809.        goBackButton = Button(window,text="Go back",width = 16, background = BUTTONCOLOUR, fg
    = 'black',  activebackground = BUTTONCOLOURFLASH, command = mainMenu)
810.        goBackButton.place(x=2,y=4)
811.
812.
813.
814.    ##########################################################################################
    ########################################
815.
816.    def mainMenu():
817.
818.        # Clearing everything:
819.        mainScreenCoverup = Label(window,width = 1200, height = 1000, background = 'white')
820.        mainScreenCoverup.place(x = 0,y = 450)
821.
822.        main2ScreenCoverup = Label(window,width = 1000, height = 1000, background = 'white')
823.        main2ScreenCoverup.place(x = 240, y = 500)
824.
825.
826.        #startbutton
827.        startButton = Button(window,text="Start", width = 16, background = BUTTONCOLOUR ,
    activebackground = BUTTONCOLOURFLASH  ,  fg = 'black',command = startProcess)
828.        startButton.place(x=650,y=460)
829.
830.
831.        #Options button
832.        optionsButton = Button(window,text="Options",width = 16, background = BUTTONCOLOUR, fg
    = 'black', activebackground = BUTTONCOLOURFLASH, command = optionsMenu)
833.        optionsButton.place(x=650,y=510)
834.
835.        #Print Graph button
836.        printGraphButton = Button(window,text="Print Graph",width = 16, background =
    BUTTONCOLOUR, fg = 'black', activebackground = BUTTONCOLOURFLASH, command = pythonGraph)
837.        printGraphButton.place(x=650,y=560)
838.
839.        #Go back
840.        goBackButton = Button(window,text="Go back",width = 16, background = BUTTONCOLOUR, fg
    = 'black',  activebackground = BUTTONCOLOURFLASH, command = introToOven)
841.        goBackButton.place(x=2,y=4)
842.
843.        #Credits
844.        creditsButton = Button(window,text="Credits",width = 16, background = BUTTONCOLOUR, fg
    = 'black', activebackground = BUTTONCOLOURFLASH, command = credits)
845.        creditsButton.place(x=650,y=610)
846.
847.
848.
849.        window.mainloop()
850.
851.    ##########################################################################################
    ########################################
852.
853.
854.    from tkinter import *
```

```
855.    #Creating window
856.    window = Tk()
857.    window.title("Oven Controller")
858.    window.configure(background="white")
859.    full =FullScreenApp(window)
860.
861.    #Adding gif to the window
862.    mainImage = PhotoImage(file = MAINSCREEN)
863.    Label(window, image = mainImage, bg = "white") .grid(row = 0, column = 0, padx  =480, pady
    = 0)
864.
865.    #Calling functions:
866.    intro = introToOven()
867.    #mainmenu = mainMenu()
868.
869.
870.    #Loop the Window
871.    window.mainloop()
```