Table of Contents

1.0 Introduction

The Oven Reflow Controller produced by team A8 was only possible due to a combination of great teamwork along with a well executed plan. In the following report, we will outline our design process, including the project specifications, the design investigation, and the overall chosen design. We will also discuss how we evaluated our solution as well as what we learned.

1.1 Design Objectives and Specifications

We designed our Oven Reflow Controller with four main objectives in mind:

1. Safety
2. Functionality
3. Accuracy
4. Ease of Use

These design objectives, along with the requirements given in the lab manual resulted in the final version of the Oven Reflow Controller having the following specifications:

1. Programmed in 8051 assembly and MATLAB
2. Measures temperatures from 25^{o}C to 250^{o}C using a K-type thermocouple with cold junction compensation to an accuracy of ∓3^{o}C
3. Creates a real-time graph for temperature with respect to time, beginning and ending with the reflow process which also displays various statistics (minimum temperature, maximum temperature, mean temperature, etcetera) and emails a video of the graph during runtime and the final stripchart to the user

4. Can control a standard off-the-shelf 1500W toaster oven using a solid state relay (SSR), regulating power by Pulse Width Modulation (PWM)
5. Settable time and temperature for soaking and reflow using a set of pushbuttons
6. A 4-bit LCD that gives feedback based on the functionality selected by the user (reflow soldering or setting parameters)
7. Automatic reflow process abortion if the temperature does not reach 50^{o}C by 60 seconds
8. Start and Emergency Stop pushbuttons
9. Beeper Feedback based on what state of the reflow process the controller is in
    a. Short beep to indicate state change
    b. Long beep to indicate completed reflow process
    c. Short song to indicate PCB is cool enough to handle
10. Choice of English or French
11. Custom LCD character [1] displaying a personalized logo generated in photoshop
12. Progress Bar on a secondary LCD to indicate progress through process, controlled with secondary AT89LP52
13. State LEDs to indicate menu, reflow soldering and emergency abort (blue, green and red respectively)
14. Cardstock box to protect electrical components and to provide aesthetic appeal

1.2 Design Approach

Our design approach was to develop our device such that it was as close to our design objective as possible. This often meant refining block diagrams until they met said objectives. Figure 1 from the Appendix illustrates the reflow process in a state diagram, a design we closely followed to ensure an optimized and efficient program. Figure 1b shows how our final UI design flows, optimized for ease of use. Figure 7 shows how all of the hardware interacts with each other, as well as the flow of data.

2.0 Investigation

2.1 Idea Generation

Our group initially generated ideas based on what we had learned from previous labs, such as converting the output of the LM335 temperature sensor[2] to temperature using an analog to digital converter. We discovered that we had to convert the output voltage of the thermocouple into temperature using similar concepts. Our working hypothesis was that the thermocouple measures the potential difference between the reference junction and the hot junction[3]-- and therefore the corresponding temperature difference. Because of this, we realized we had to add the reference temperature calculated from the LM335 to get the actual temperature. We also hypothesized that the voltage from the thermocouple was too low to be read directly from the ADC, which is why an operational amplifier[4] was required. Bonus features were generated to improve functionality, usability and data presentation.

2.2 Investigation Design

Our group tested the design throughout the process, constantly enhancing and debugging the system to optimize functionality. We first minimized error by adjusting the operational amplifier offset using a potentiometer[5]. By shorting the circuit between the two thermocouple inputs we were able to adjust the potentiometer until the input offset was approximately 0.010 mV. To keep the amplification gain constant, we used resistor values as close together as possible and measured the amplification over a series of temperatures. We used the multimeter to ensure the resistance values were highly accurate according to our needs. Taking the average amplification, we found a gain of 337 (compared to the ideal value of 330) and used this value in our conversion calculations. To test the accuracy of the temperature values calculated, they were compared to the temperature values calculated by the python program. All the temperatures were found to be within 3 degrees of the python values, which met our desired specifications. The user interface was programmed independently of the oven controller program on a separate circuit board, allowing us to work in parallel and merge the two pieces of code later on.

2.3 Data Collection

The tools used for data collection included the multimeter, oscilloscope, thermocouple, and the temperature sensor from lab 3. These instruments provided a wide range of data to regulate the temperature allowing for the most accurate readings. The temperature was closely observed and recorded after each test, allowing for consistency in the readings. The oven was monitored closely during the early tests to ensure that it was turning on and off accordingly, according to the Pulse Width Modulation programmed for the microcontroller output pin.

2.4 Data Synthesis

Once the various components of the program were completed, the system was merged together to enable full testing of the oven. Additional hardware peripherals, including the second microcontroller and LCD, were also added during this stage. The user interface was tested in the beginning of the reflow as it processed the desired temperature, inputted by the user. The oven temperature was closely monitored with the strip chart as well as with the voltage readings using the multimeter. The design and code were adjusted appropriately for the efficiency of the program.

2.5 Analysis of Results

We performed multiple tests on the oven temperature, as well as the abort feature. Since the temperature statistics were recorded with a strip chart on MATLAB, and were sent via email to the user, temperature variations with each test were observed. Initially, the plots revealed a +/-10 degree error with our temperature output. As to correct this inaccuracy, our team added a potentiometer to the op-amp inputs and carefully adjusted the potentiometers' resistance as to null the offset voltage [5]. After performing multiple tests, and ensuring that the basic operations of the reflow oven operated smoothly, we tested the entire process and successfully soldered a PCB.

3.0 Design

3.1 Use of Process

Previous labs were referenced for a general idea of how to design certain features. We integrated and improved upon many functionalities of previous labs to program the reflow oven controller. Some of these improvements include wiring additional chips together for increased functionality, adding personalized characters for a more aesthetic interface. Additionally, we improved upon the MATLAB code from the temperature sensor lab, enhancing the strip chart graph to also display data such as the temperature and time elapsed for the procedure. Another feature of the MATLAB code was the email sent to the user containing the final graph of the strip chart and a video of the total run time of the reflow process. We also used knowledge from previous experiences and classes to help troubleshoot issues. For example, shortly after testing the output voltage of thermocouple, we realized the voltage was too low for the ADC to read [6]. From this, we were able to extrapolate that we needed to amplify the output voltage, knowing from ELEC 201 that this could be done using operational amplifiers. Once we knew all of this, it was just a matter of finding the correct combination of components to get the voltage output in the proper range. We employed similar techniques and processes to complete all the functionality for the controller.

3.2 Need and Constraint Identification

3.2.1 Customer, User and Enterprise Needs

Most of the features of the controller were adapted from requirements listed in the project manual. From this, we were also able to identify our target user. We identified the customer/user as electrical engineers, or people who have a technical interest in reflow soldering. Additional features of our design were based on our research on different types of reflow ovens. One of the most important needs of the project was to have a safety feature for aborting the process when a certain temperature is not reached in a given amount of time, or when the user decides to halt the process.

3.2.2 Constraints

As our group tested our program, we looked for ways optimize the system by testing the system from a user's perspective. Through this process, we were able to identify several constraints. The most prominent constraints we found were the small flash memory available on the AT89LP52 [7] board, and the fact that we were limited to using a bootloader instead of SPI to flash code onto the microcontroller. This meant that we needed to be selective with our functionality (i.e. only write code for the most important features) and efficiently program the functionality that we deemed necessary. Another key constraint was the time it took to test the system as we carried out the design process. This meant that testing of the system was slow, since a run-through of the entire reflow process was required to debug a feature present near the end of the reflow cycle.

These needs must be met and constraints must be overcome as the market demands a low cost and effective way to solder.

3.3 Problem Specification

From the specified design requirements above, the main focus of the project design became the safety feature of the system, and the way data is handles and transferred during run time. The abort function was specified to operate in any state, and at any time that the user chooses to stop the reflow process, or when the temperature does not reach the desired state in a given amount of time. A user friendly interface was also part of the additional design requirements as it would be more accessible to engineers and regular people. Lastly, since we strove to obtain the most accurate and precise readings from the oven, we wanted to display these results for the user to see. A visual representation of the temperature and current time of the reflow process was added to the design requirements to monitor the system more closely and to get the information to the user.

3.4 Solution Generation

The user interface required two microcontrollers for the extra language feature as one microcontroller did not have enough memory to store the programs that the system needed for the reflow oven to run. MATLAB was used to display a temperature strip chart, with additional details that conveyed the process to the user in a quantitative manner. The initial temperature, current temperature, run time, and highest temperature were all included in the email of the graph that was sent to the user after the process has finished.

3.5 Solution Evaluation

The final design was chosen for its dexterity and overall functionality. The accuracy of the temperature readings were calibrated each round according to data received on the graph. The graph was evaluated by checking its accuracy compared to the multimeter voltage readings, as well as its general shape, in comparison with the reflow graph model shown in the lecture slides.

3.6 Detailed Design

The user interface was first planned on paper by assessing the required features and added bonus features. Then a pseudo description was written of how the state machine was expected to function, this was done to reduce possible logic and planning errors. Finally the pseudo code was used as a reference and the state machine was developed in assembly. The bridge between user interface and the back end controlling code was the starting wait state that prompted the user to start the reflow process. Pressing the menu button navigated the user to the menu states used to set a variety of options such as, language, reflow time/temperature and soak time/temperature. The final state machine included twenty states total and was controlled by eight buttons. The circuit diagram showing this is found in Figure 6.

When designing the control for SSR box during reflow soldering, it was important to us that the oven was always being supplied the correct amount of power, or run the risk of burning the PCBs. For this reason, we chose to design the reflow soldering process as a state machine (seen in the source code) with six main states. Those states were wait, ramp to soak, soaking, ramp to reflow, reflow, cooling and emergency stop. Power supplied to the oven and the sound output were therefore determined based on what state of the reflow process we were in, which made the system more reliable.

As for the MATLAB portion, the axes were adjusted to show the full temperature range of the strip chart. On the side of the graph, a description of the highest, starting, and current temperatures are displayed, along with the current run time. The main focus of the MATLAB portion was to convey accessible data for the user, as delivered by the email with the final statistics of the process. The data plot was tested for accuracy by comparing it with the reflow process reference graph, posted on the project files or the lab.

3.7 Solution Assessment

3.7.1 Design Testing

The design performance was tested and evaluated according to the lab manual specifying the basic requirements of the program. Each time the program was executed from start to finish, features such as UI, state changes, abort, beeping, strip chart, and data recording functions were evaluated for their performance. A UI test was performed each time a new program was added to the system, and as each reflow process started. We tested the UI by using pushbuttons to navigate to every UI state, and observing the stripchart being plotted while the reflow stages were being carried out. Figures 2 and 3 in the Appendix illustrate the consistent oven temperature results after each full reflow execution. The changes in the reflow states were closely monitored to guarantee that they were synced with the temperature and run time of the system. Additionally, another LCD was added to the circuit to show the user when the next state change will occur. We also recorded calculated temperatures in comparison to the actual temperatures to test the accuracy of our calculation over a variety of temperature values over four tests. The results of these tests are shown in Figure 5, demonstrating that the range of our temperatures was within 3 degrees Celsius of the actual temperatures for the range we were using the device. This is also where we found one of our design weaknesses. This weakness was that the while the temperature was always within 3 degrees of the actual temperature, the offset was not consistent.

3.7.2 Strengths

- Accurate temperature readings based on the voltage and temperature displayed
- Reflow oven process data is available to the user after use
- Friendly user interface, with French
- Safety features - indicating states and the bonus "loading" on LCD for getting an estimate of when the state changes will occur

3.7.3 Weaknesses and Options for Improvement

- Casing had a mostly aesthetic function and did not provide protection from motion, external forces, water damage, etc.
  - Casing could be made from sturdier material (wood/metal), using a CAD software to ensure a close fit with the circuit
- Connections were loose and unstable, since the circuit was built on a breadboard
  - PCB would be a more durable and portable option

- *After* the 60 second mark, the controller had no way of checking whether the thermocouple was still in the oven
    - Voltage change behaviour of hot thermocouple in room temperature could be characterized, and the corresponding voltage change could be actively sought out by the controller in order to detect this displacement of the thermocouple from the oven

4.0 Lifelong Learning

In order for the user to be able to set several variables, as required, the most reasonable approach was to create a state machine. The state machine was extensive to ensure user friendly functionality. While state machines were introduced in a previous course, we have had no exposure to creating state machines in assembly. It would have improved our process of planning and creating the state machine to have had some exposure in previous labs. Several unseen issues arose as we coded, such as constraints to comparing and jumping commands. This resulted in having to reorganize and reassemble the program several times and made adding any additional functions once completed, extremely difficult.

5.0 Conclusions

5.1 Design and Functionality

All the design requirements were satisfied according to the project files provided for the lab. The toaster oven was controlled using a PWM and SSR programmed in 8051 for state changes, temperature regulation, and additional features for greater user satisfaction. The states were shown in the stripchart and LCD monitors so that the user knows which stage of the reflow process the PCB is in. For reference of the state diagrams, see appendix (Figure 1a, 1b). Keeping in mind our original goals, the final system is both safe and functional. It allows the user full controllability while keeping accurate time and temperature values. The bonus features were used to enhance the user's ease of use with the project, while effectively communicating the data. The dual language, aesthetic case, and music to signal the end of the cooling process all add up to the user's satisfaction with the program. Information about the reflow process is conveyed through the three LEDs to indicate the menu, abort, and reflow states; a secondary LCD displayed the progress bar to indicate when a state change will occur; lastly, the MATLAB portion notifies the user with an email, with an attachment of the final temperature strip chart, and a video showing the progress of the strip chart during the run time.

5.2 Problems Encountered

Early on, we noticed that the oven experienced significant radiative heat losses through the sides of the glass. This meant that the temperature measured using the thermocouple was not representative of the temperatures everywhere in the oven. We reasoned that if the radiative losses were minimized by covering the glass with tinfoil, the temperature readings would be more consistent at various locations in the oven, making the thermocouple readings more reliable. When combine all respective parts of final code, the added lines created many errors in compare and jump statements as some jumps were no longer in range. Due to this, we were required to greatly re organize the code. This later caused issues with debugging code that was difficult to follow.

6.0 References

[1] Edwards, Scott. Getting Graphic: Defining Custom LCD Characters (n.d.): Seetron.com. SEETRON. Web.

[2] "LMx35, LMx35A Precision Temperature Sensors." Ti.com. Texas Instruments Inc, Jan. 2017. Web.

[3] Ada, Lady. "Thermocouple.". Adafruit.com. Adafruit Industries. Web. 9 Sept. 2016.

[4] "OP07 Ultralow Offset Voltage Operational Amplifier." Analog.com. Analog Devices Inc, Jan. 2011. Web.

[5] Nguyen, Bao and David Smith. "Nulling Input Offset Voltage of Operational

Amplifiers." Ti.com. Texas Instruments Inc, Aug. 2000. Web.

[6] "MCP3004/3008 Analog to Digital Converter." Microchip.com. Microchip Technology Inc, Jan. 2008. Web.

[7] "Atmel AT89LP51 Microcontroller." Atmel.com. Atmel Coorperation, Dec. 2011. Web.

7.0 Bibliography

"Custom Character Generatorfor HD44780 LCD Modules." *Custom Character Generator for HD44780 LCD Modules*. N.p., n.d. Web. 02 Mar. 2017.

Kellogg, Michael. "Dictionnaires De Langue En Ligne." *WordReference.com*. WordReference Ltd., n.d. Web. 02 Mar. 2017.

Saeed Yasin. "How to Display Custom Characters on LCD Using PIC16F877." *Saeed's Blog*. N.p., n.d. Web. 02 Mar. 2017.

8.0 Appendix

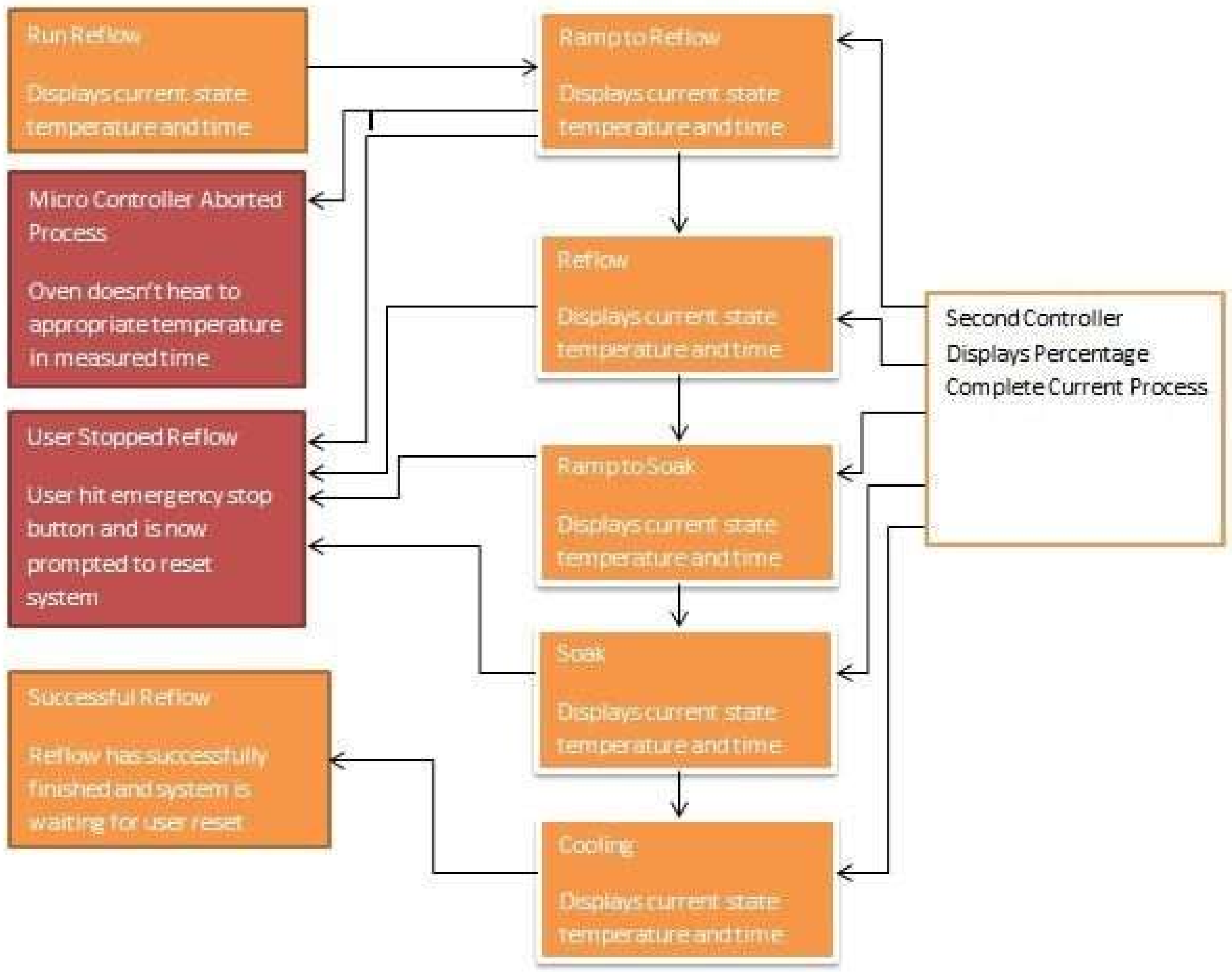

Run Reflow
Displays current state temperature and time
Micro Controller Aborted Process
Oven doesn't heat to appropriate temperature in measured time
User Stopped Reflow
User hit emergency stop button and is now prompted to reset system
Successful Reflow
Reflow has successfully finished and system is waiting for user reset
Ramp to Reflow
Displays current state temperature and time
Reflow
Displays current state temperature and time
Ramp to Soak
Displays current state temperature and time
Soak
Displays current state temperature and time
Cooling
Displays current state temperature and time
Second Controller Displays Percentage Complete Current Process

Figure 1a: The state-flow of our reflow oven controller during runtime (the reflow process).

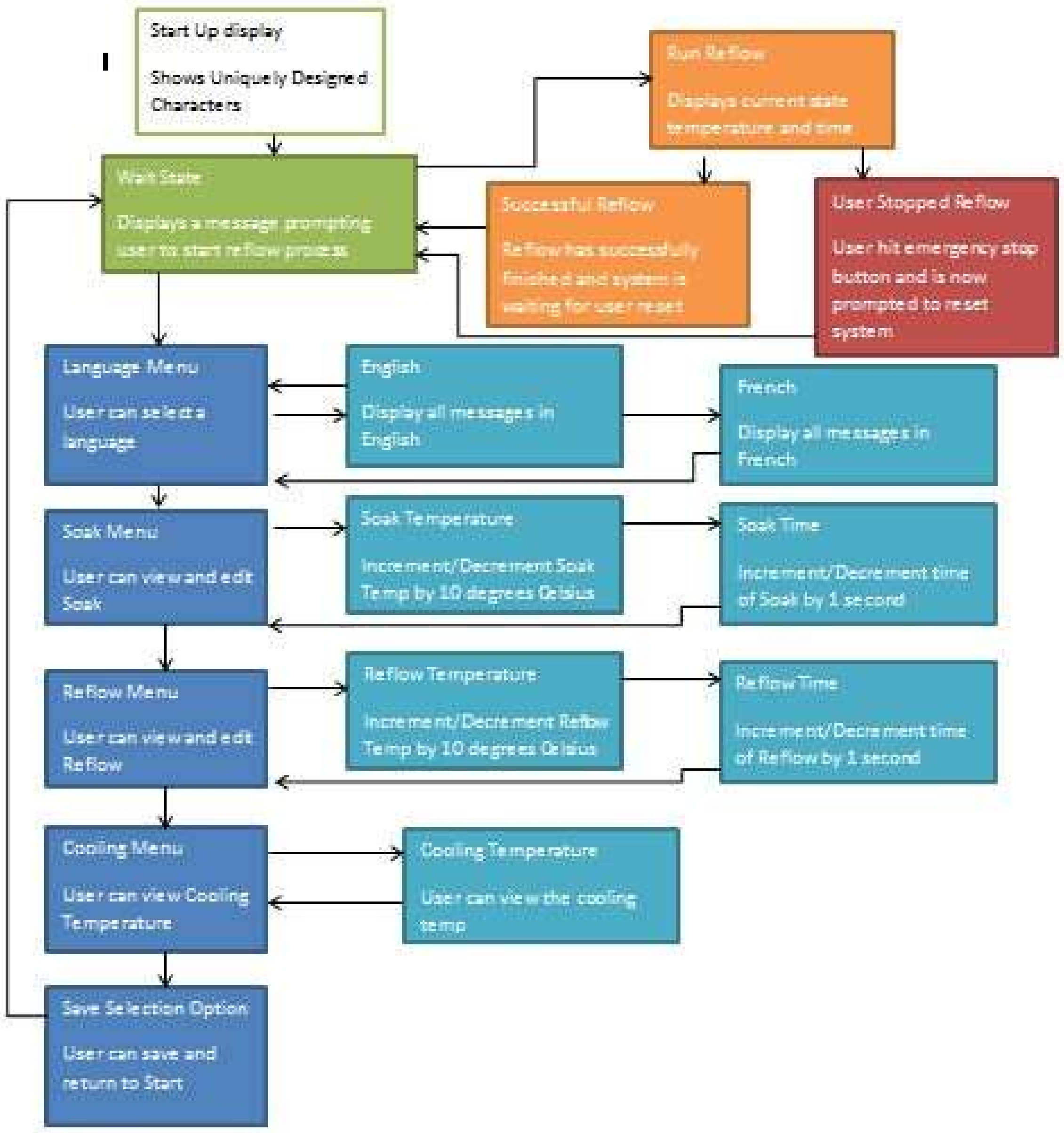

Start Up display
Shows Uniquely Designed Characters
Run Reflow
Displays current state temperature and time
Wait State
Displays a message prompting user to start reflow process
Successful Reflow
Reflow has successfully finished and system is waiting for user reset
User Stopped Reflow
User hit emergency stop button and is now prompted to reset system
Language Menu
User can select a language
English
Display all messages in English
French
Display all messages in French
Soak Menu
User can view and edit Soak
Soak Temperature
Increment/Decrement Soak Temp by 10 degrees Celsius
Soak Time
Increment/Decrement time of Soak by 1 second
Reflow Menu
User can view and edit Reflow
Reflow Temperature
Increment/Decrement Reflow Temp by 10 degrees Celsius
Reflow Time
Increment/Decrement time of Reflow by 1 second
Cooling Menu
User can view Cooling Temperature
Cooling Temperature
User can view the cooling temp
Save Selection Option
User can save and return to Start

Figure 1b. The state-flow of our reflow oven controller UI during the initial interface process.

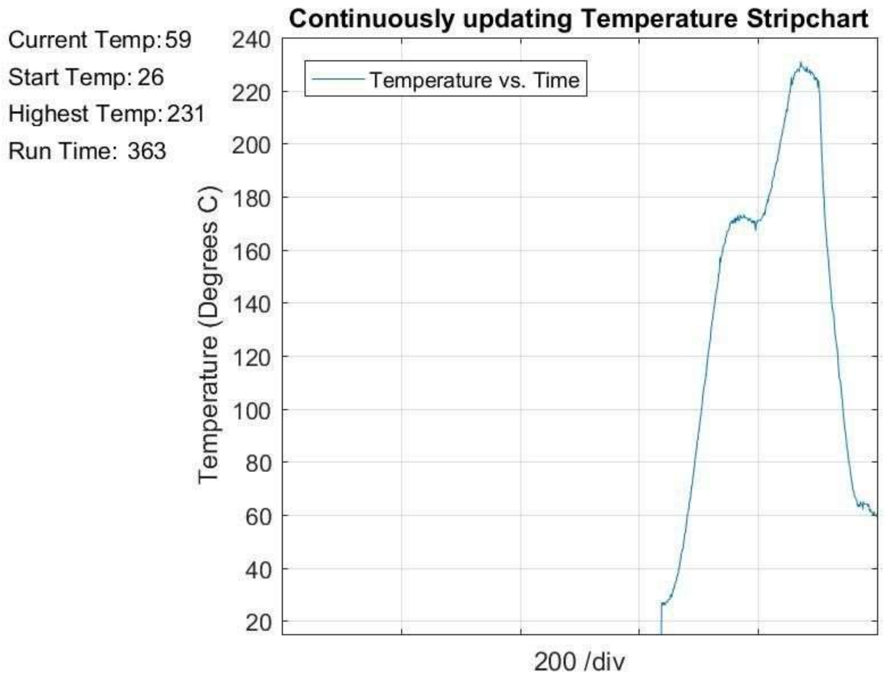
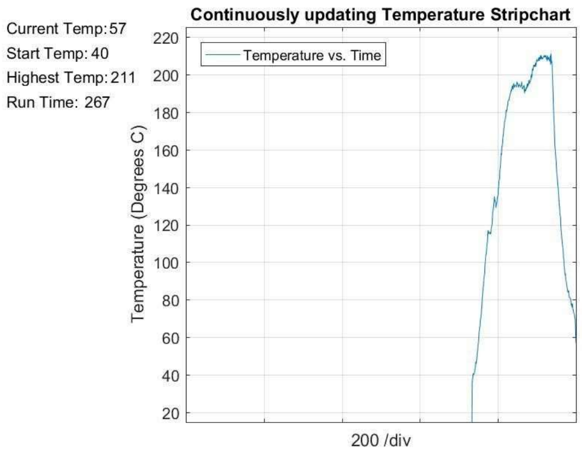


Figure 2. A sample Temperature vs. Time stripchart plot of the reflow process from start to finish.



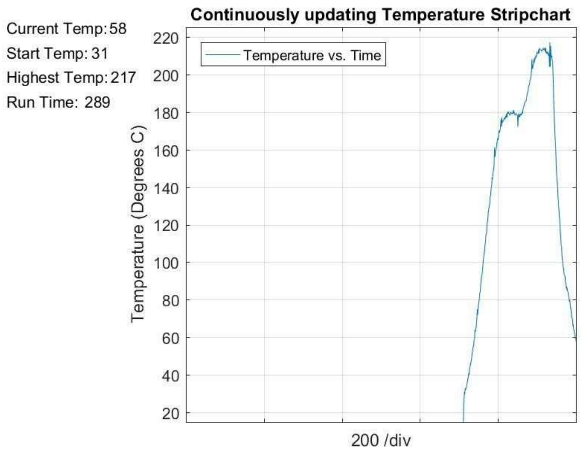


Figure 3: Consistency of the Reflow Oven temperature shown as a stripchart in MATLAB.

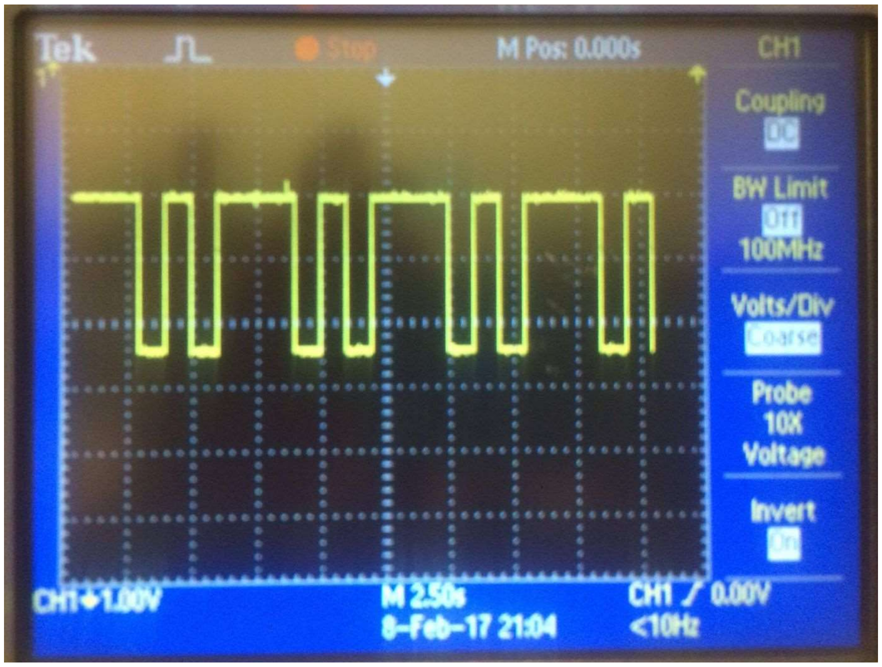


Figure 4: An oscilloscope capture of the voltage supplied to the oven SSR box while testing the programmed PWM.

Temperature (ºC)	Test 1	Test 2	Test 3	Test 4
20	-2.5	-2.2	-2.1	-2.7
30	-2.6	-2.8	-2.9	-2.2
40	-2.1	-2.0	-2.5	-2.3
50	-1.9	-1.6	-2.2	-2.5
60	-1.9	-2.9	-2.3	-2.4
70	-2.0	-1.8	-1.9	-2.2
80	-1.9	-1.9	-1.8	-2.0
90	-1.8	-2.0	-2.1	-1.1
100	-1.9	-1.4	-2.6	-1.8
110	-0.4	-1.1	-2.0	-1.3
120	-2.4	-0.3	-1.3	-1.2
130	-1.1	-0.4	-1.1	0.4
140	-1.9	-2.2	-0.2	-1.0
150	-2.0	-0.3	1.0	-1.5
160	0.1	-1.9	-2.3	-1.1
170	0.3	2.3	-2.1	1.2
180	-0.4	-0.9	1.3	0.5
190	-1.6	0.8	0.7	0.9
200	0.0	1.2	0.5	-0.3
210	1.8	0.2	0.8	2.8
220	2.5	1.1	1.3	0.2
230	0.7	1.9	1.2	-0.0
240	1.0	2.1	2.6	0.0

Figure 5: Differences between calculated temperature from thermocouple to temperature from multimeter for 4 different tests

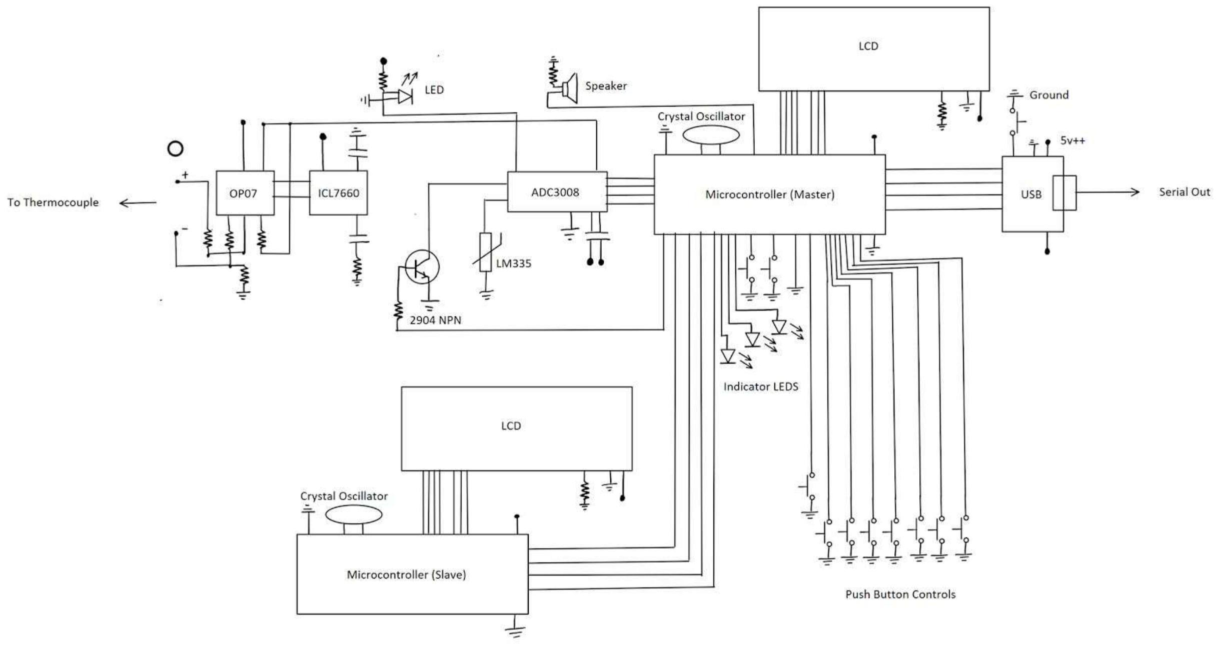


Figure 6: Circuit Diagram.

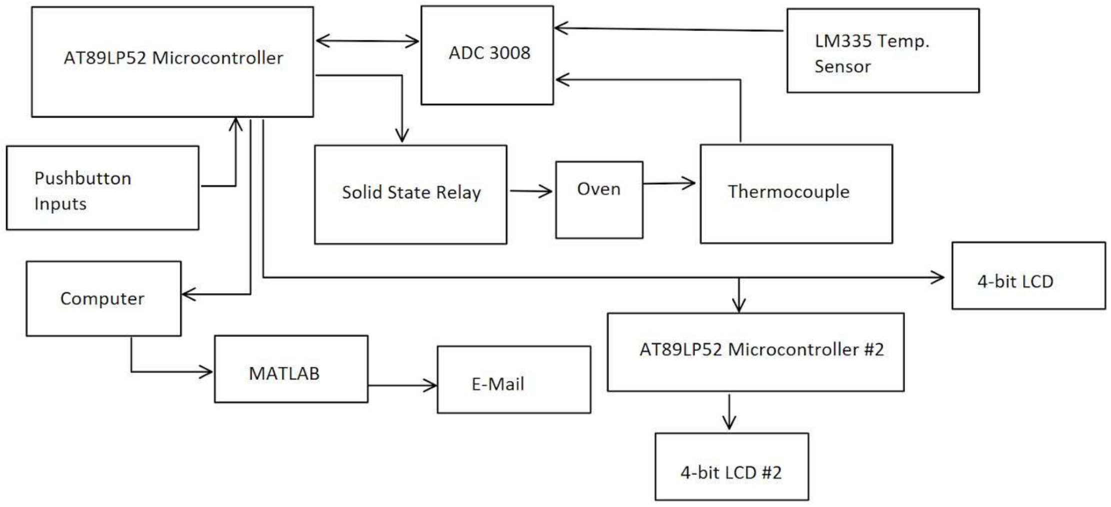


Figure 7: Hardware Block Diagram.

8.1 Source Code

```
$MODLP52

CLK  equ 22118400
BAUD equ 115200
T1LOAD equ (0x100-(CLK/(16*BAUD)))
TIMER0_RATE   EQU 4096     ; 2048Hz squarewave (peak amplitude of CEM-1203 speaker)
TIMER0_RELOAD EQU ((65536-(CLK/TIMER0_RATE)))
TIMER2_RATE   EQU 1000     ; 1000Hz, for a timer tick of 1ms
TIMER2_RELOAD EQU ((65536-(CLK/TIMER2_RATE)))


div1023 equ 1023
sub273 equ 273
VLED equ 292
OpAmp_Ratio equ 337   ;might be 337-338 (re-measure later) -->test with multimeter thermometer
ThermConst equ 24390
OneHun equ 100

Abort equ #0x59
AbortTemp equ #0x32
coolTemp equ #0x3C

beepTime equ #0x05
songLength equ #18



; These 'EQU' must match the wiring between the microcontroller and ADC
CE_ADC EQU P2.0
MY_MOSI EQU P2.1
MY_MISO EQU P2.2
MY_SCLK EQU P2.3
LCD_RS equ P1.2
LCD_RW equ P1.3
LCD_E  equ P1.4
LCD_D4 equ P3.2
LCD_D5 equ P3.3
LCD_D6 equ P3.4
LCD_D7 equ P3.5
OVENCOUNT equ P2.4
SOUND equ P3.7
PULSE_PIN equ P4.4

BOOT_BUTTON   equ P4.5
START_BUTTON  equ P0.7 ;Starts oven once selected time
STOP_BUTTON   equ P0.6 ;Stops oven if running
MENU_BUTTON   equ P0.4 ;Allows user to navigate selection menu for timing and temperature
UP_BUTTON     equ P0.2 ;Changes timing or temperature, up x degrees or x seconds/minutes
DOWN_BUTTON   equ P0.1 ;Changes timeing or temp. down
LEFT_BUTTON   equ P0.5 ;Navigate left through menu
RIGHT_BUTTON  equ P0.0 ;Navigate right through menu
SELECT_BUTTON equ P0.3 ;Extra button with no purpose yet, could be a save time and temp or
something we can add on later
RED_RGB equ P2.5
GREEN_RGB equ P2.6
BLUE_RGB equ P2.7



; Reset vector
org 0x0000
```

```
    ljmp MainProgram

; External interrupt 0 vector (not used in this code)
org 0x0003
       reti
; Timer/Counter 0 overflow interrupt vector
org 0x000B
       ljmp Timer0_ISR

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

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

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

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

DSEG at 30H
x:   ds 4                     ;defines that x stores 4 bytes of data
y:   ds 4
bcd: ds 5
Vcc: ds 2
ThermCup: ds 4
TempRef:  ds 2

tempHolder: ds 2
timeHolder: ds 2

Count1ms:     ds 2 ; Used to determine when half second has passed
state: ds 1 ;
temperature: ds 2

power: ds 1

powerSecs:      ds 1
soakTempBCD:   ds 2
soakTemp:      ds 1 ;
soakTime:      ds 2 ;
reflowTempBCD: ds 2
reflowTemp:    ds 1 ;
reflowTime:    ds 2 ;
loop_count:    ds 1 ;

ovenONOFF:     ds 1
timeCounter:   ds 2
powerFlag:     ds 1

beepLength:    ds 1 ; 0 for short, 1 for long
numberOfBeeps: ds 1 ;
beepFlag:      ds 1 ;
beepTimer:     ds 1 ;

UICURRSTATENUM: ds 2
UILANGFLAG:     ds 1

songFlag: ds 1
nextNote: ds 1
```

```
BSEG
mf: dbit 1
half_seconds_flag: dbit 1 ; Set to one in the ISR every time 500 ms had passed

$NOLIST
$include(workingLCD_4bit.inc) ; A library of LCD related functions and utility macros
$include(math32.asm)
$include(menu_bonusV3.asm)
$LIST


CSEG
;                      1234567890123456    <- This helps determine the location of the counter
Fr_Time:               db      'TMPS:', 0
State_Time:            db      'TIME:', 0
State_Temp:            db      'T:', 0
DEG_C:                 db      'C', 0
SEC:                   db      's', 0

_SPACE:                db      ' ', 0
RAMPTOSOAK:            db      ' -RAMP TO SOAK- ', 0
SOAKING:               db      '   -SOAKING!-   ', 0
RAMPTOPEAK:            db     ' -RAMP TO PEAK- ', 0
REFLOW:                db      '    -REFLOW-    ', 0
COOLING:               db      '   -COOLING!-   ', 0

FR_RAMPSOAK:           db      'RAMPE DE TREMPER', 0
FR_SOAK:               db      '    TREMPAGE    ', 0
FR_RAMPPEAK:           db      ' RAMPE A SOMMET ', 0
FR_REFLOW:             db      '    REFUSION    ', 0
FR_COOL:               db     'REFROIDISSEMENT!', 0

UIPRESSSTART:   db ' Press Start To ', 0  ;part 1 of start menu, cursor at 1,1
UIPRESSSTART2:  db '  Begin Reflow  ', 0  ;part 2 of start menu, cursor at 2,1
UISTOPPED:      db 'Process Complete', 0  ;part 1 of stop menu, cursor at 1,1
UIOKTORESET:    db ' Press To Reset ', 0  ;part 2 of stop menu, cursor at 2,1
UISOAK:         db '      Soak      ', 0  ;for next three shows current state or the next
state to be selected
UIREFLOW:       db '     Reflow     ', 0  ;for next three shows current state or the next
state to be selected
UICOOLING:      db '     Cooling    ', 0  ;for the selection menu, cursor will be at 1,1 or
2,1 depending
UIBACK:         db ' Save Selection ', 0
UILANGUAGESELECT:db '   >Language<   ', 0  ;select your fav langauge buds
UISOAKSELECT:   db '     >Soak<     ', 0  ;for next three shows current state or the next
state to be selected
UIREFLOWSELECT: db '    >Reflow<    ', 0  ;for next three shows current state or the next
state to be selected
UICOOLINGSELECT:db '    >Cooling<   ', 0  ;for the selection menu, cursor will be at 1,1 or
2,1 depending
UIBACKSELECT:   db '>Save Selection<', 0
UIBLANK:        db '                ', 0
UIENGLISHSELECT:db '   >English<    ', 0
UIFRENCH:       db '     French     ', 0
UIFRENCHSELECT: db '    >French<    ', 0
UISOAKTEMP:     db 'Soak Temp:', 0  ;selecting temp for soak, cursor at 1,1
UIREFLOWTEMP:   db 'RF Temp:', 0  ;selecting temp for reflow, cursor at 1,1
UISOAKTIME:     db 'Soak Time:', 0  ;selecting temp for soak, cursor at 1,1
UIREFLOWTIME:   db 'RF Time:', 0  ;selecting temp for reflow, cursor at 1,1
UICOOLINGTEMP:  db 'Cooling:     60C', 0  ;selecting temp for coolinh, cursor at 1,1
UIEMERG:        db ' Reflow Aborted ', 0  ;part 1 of stop menu, cursor at 1,1
UIOKEMERG:      db 'Press Reset Btn.', 0  ;part 2 of stop menu, cursor at 2,1

FR_UIEMERG:     db ' Refusion term. ', 0  ;part 1 of stop menu, cursor at 1,1
FR_UIOKEMERG:   db ' Reinitialiser! ', 0  ;part 2 of stop menu, cursor at 2,1
```

```
CLEAR: db                '                ',0
;french;;;;;;;;;;;;;;;;;;;;;;
UIPRESSSTART_FR:    db '     Appuie     ', 0  ;part 1 of start menu, cursor at 1,1
UIPRESSSTART2_FR:   db '    Demarrer    ', 0  ;part 2 of start menu, cursor at 2,1
UISTOPPED_FR:       db ' Procs. Termine ', 0  ;part 1 of stop menu, cursor at 1,1
UIOKTORESET_FR:     db ' Reinitialiser  ', 0  ;part 2 of stop menu, cursor at 2,1
UILANGUAGE_FR:      db '    Langage     ', 0  ;select your fav langauge buds
UISOAK_FR:          db '    Tremper     ', 0  ;for next three shows current state or the next
state to be selected
UIREFLOW_FR:        db '    Refusion    ', 0  ;for next three shows current state or the next
state to be selected
UICOOLING_FR:       db ' Refroidissment ', 0  ;for the selection menu, cursor will be at 1,1
or 2,1 depending
UIBACK_FR:          db '  Enregistrer   ', 0
UILANGUAGESELECT_FR:db '   >Langage<    ', 0  ;select your fav langauge buds
UISOAKSELECT_FR:    db '   >Tremper<    ', 0  ;for next three shows current state or the next
state to be selected
UIREFLOWSELECT_FR:  db '   >Refusion<   ', 0  ;for next three shows current state or the next
state to be selected
UICOOLINGSELECT_FR: db '>Refroidissment<', 0  ;for the selection menu, cursor will be at 1,1
or 2,1 depending
UIBACKSELECT_FR:    db ' >Enregistrer<  ', 0
UISOAKTEMP_FR:      db 'Tremper:', 0  ;selecting temp for soak, cursor at 1,1
UIREFLOWTEMP_FR:    db 'Refusion:', 0  ;selecting temp for reflow, cursor at 1,1
UISOAKTIME_FR:      db 'Restant:', 0  ;selecting temp for soak, cursor at 1,1
UIREFLOWTIME_FR:    db 'Restant:', 0  ;selecting temp for reflow, cursor at 1,1
UICOOLINGTEMP_FR:   db 'Refroid.:', 0  ;selecting temp for coolinh, cursor at 1,1
UIEMERG_FR:         db '     Arrete     ', 0  ;part 1 of stop menu, cursor at 1,1
UIOKEMERG_FR:       db ' Reinitialiser  ', 0  ;part 2 of stop menu, cursor at 2,1

Space:
    DB  '\r', '\n', 0
Temp:
       DB '', 0

Tetris_Timer_Reload_Array:
                                                   db     0xD4, 0x44
                                                   db     0xD6, 0xB8
                                                   db     0xDB, 0x39
                                                   db     0xD6, 0xB8
                                                   db     0xD4, 0x44
                                                   db     0xCE, 0xE9
                                                   db     0xCE, 0xE9
                                                   db     0xD6, 0xB8
                                                   db     0xDF, 0x3C
                                                   db     0xDB, 0x39
                                                   db     0xD6, 0xB8
                                                   db     0xD4, 0x44
                                                   db     0xD6, 0xB8
                                                   db     0xDB, 0x39
                                                   db     0xDF, 0x3C
                                                   db     0xD6, 0xB8
                                                   db     0xCE, 0xE9
                                                   db     0xCE, 0xE9

;---------------------------------;
; Routine to initialize the ISR   ;
; for timer 0                     ;
;---------------------------------;
Timer0_Init:
       mov a, TMOD
       anl a, #0xf0 ; Clear the bits for timer 0
       orl a, #0x01 ; Configure timer 0 as 16-timer
       mov TMOD, a
       mov TH0, #high(TIMER0_RELOAD)
       mov TL0, #low(TIMER0_RELOAD)
       ; Enable the timer and interrupts
    clr ET0  ; Disable Timer 1 Interrupt
    setb TR0  ; Start timer 0
       reti
;---------------------------------;
; ISR for timer 0.  Set to execute;
; every 1/4096Hz to generate a    ;
; 2048 Hz square wave at pin P3.7 ;
;---------------------------------;
Timer0_ISR:
       ;clr TF0  ; According to the data sheet this is done for us already.
       ; In mode 1 we need to reload the timer.
       clr TR0
       push acc
       mov a, numberOfBeeps
       cjne a, #0x00, setNormalFreq
       mov a, nextNote
       add a, #0x99
       da a
       lcall loadNote
       sjmp endT0
setNormalFreq:
       mov TH0, #high(TIMER0_RELOAD)
       mov TL0, #low(TIMER0_RELOAD)
       ; Enable the timer and interrupts
endT0:
    setb TR0
       cpl SOUND ; Connect speaker to P3.7!
       pop acc
       reti

loadNote:
       mov b, #2
       mul ab

       push acc

       mov dptr, #Tetris_Timer_Reload_Array
       movc a,@a+dptr

       mov TH0,a
       inc dptr

       pop acc

       movc a,@a+dptr
       mov TL0, a
       clr a

       ret

;---------------------------------;
; Routine to initialize the ISR   ;
; for timer 2                     ;
;---------------------------------;
Timer2_Init:
       mov T2CON, #0 ; Stop timer/counter.  Autoreload mode.
       mov RCAP2H, #high(TIMER2_RELOAD)
       mov RCAP2L, #low(TIMER2_RELOAD)
       ; Init One millisecond interrupt counter.  It is a 16-bit variable made with two 8-bit
parts
       clr a
       mov Count1ms+0, a
       mov Count1ms+1, a
       ; Enable the timer and interrupts
    setb ET2  ; Enable timer 2 interrupt
    setb TR2  ; Enable timer 2
        ret

;---------------------------------;
; ISR for timer 2                 ;
;---------------------------------;
Timer2_ISR:
        clr TF2  ; Timer 2 doesn't clear TF2 automatically. Do it in ISR
        cpl P3.6 ; To check the interrupt rate with oscilloscope. It must be precisely a 1 ms
pulse.

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

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

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

        ; 500 milliseconds have passed.  Set a flag so the main program knows
        setb half_seconds_flag ; Let the main program know half second had passed
        ;cpl TR0 ; Enable/disable timer/counter 0. This line creates a
beep-silence-beep-silence sound.
        ; Reset to zero the milli-seconds counter, it is a 16-bit variable
        clr a
        mov Count1ms+0, a
        mov Count1ms+1, a

;___________________________Clock Incrementation______________________________
        mov a, timeCounter
        cjne a, #0x99, continueADD
        mov a, timeCounter+1
        add a, #0x01
        da a
        mov timeCounter+1, a
        mov timeCounter, #0x00
        sjmp goPower
continueADD:
        add a, #0x01
        da a
        mov timeCounter, a

goPower:
        lcall sendSPI
        Send_BCD(bcd+1)
        Send_BCD(bcd)
        mov DPTR, #Space
    lcall SendString
        lcall powerCalc
        lcall beepOn
        lcall displayData
Timer2_ISR_done:
        pop psw
        pop acc
        reti

INIT_SPI:
        setb MY_MISO ; Make MISO an input pin
        clr MY_SCLK ; For mode (0,0) SCLK is zero
        ret

DO_SPI_G:
        push acc
        mov R1, #0 ; Received byte stored in R1
        mov R2, #8 ; Loop counter (8-bits)
DO_SPI_G_LOOP:
        mov a, R0 ; Byte to write is in R0
        rlc a ; Carry flag has bit to write
        mov R0, a
        mov MY_MOSI, c
        setb MY_SCLK ; Transmit
        mov c, MY_MISO ; Read received bit
        mov a, R1 ; Save received bit in R1
        rlc a
        mov R1, a
        clr MY_SCLK
        djnz R2, DO_SPI_G_LOOP
        pop acc
        ret
testDisplay:
       Set_Cursor(1,1)
       Display_BCD(state)
       ret

print_toPuTTY:

       lcall convert_Temp_ThermCoup
       lcall convert_Temp_Reference
       lcall calc_realTemp

       lcall hex2bcd
       mov a, bcd+1
       cjne a, #0x00, continue_bcd
       mov a, bcd
       cjne a, #0x00, continue_bcd
       lcall print_ToPutty
continue_bcd:
       ret


convert_Temp_Reference:
       lcall Calc_Vcc

       mov y, TempRef
       mov y+1, TempRef+1
       mov y+2, #0
       mov y+3, #0

       ; New result stored in x
       lcall mul32

       Load_Y(div1023)
       lcall div32

       Load_Y(sub273)
       lcall sub32

       mov TempRef+1, x+1
       mov TempRef, x

       ret

convert_Temp_ThermCoup:
        ;___________Temp conversion from opAmp Vout_________
        mov a, ThermCup
        cjne a, #0x00, not_zero
        sjmp cont1
cont1:
        mov a, ThermCup+1
        cjne a, #0x00, not_zero
        sjmp skip_n

not_zero:
        lcall Calc_Vcc

        mov y, vcc
        mov y+1, vcc+1
        mov y+2, #0
        mov y+3, #0

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

        ; New result stored in x
        lcall mul32

        Load_Y(div1023)
        lcall div32

        Load_Y(ThermConst)
        lcall mul32

        Load_Y(OpAmp_Ratio)
        lcall div32

        Load_Y(OneHun)
        lcall div32

        mov ThermCup+1, x+1
        mov ThermCup, x


skip_n:
        ret

calc_realTemp:
        mov y, TempRef
        mov y+1, TempRef+1
        mov y+2, #0
        mov y+3, #0

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

        lcall add32
        mov temperature, x
        mov temperature+1, x+1
        ret


Calc_Vcc:
        ; Measure the LED voltage. Used as reference to find VCC.
        Read_ADC_Channel(7) ; Read voltage, returns 10-bits in [R6-R7]
        mov y+3, #0 ; Load 32-bit "y" with value from ADC
        mov y+2, #0
        mov y+1, R7
        mov y+0, R6
        Load_X(VLED*1023) ; Macro to load "x" with constant
        lcall div32 ; Divide "x" by "y", the result is VCC in "x"
        mov Vcc+1, x+1 ; Save calculated VCC high byte
        mov Vcc+0, x+0 ; Save calculated VCC low byte

        mov x+2, #0
        mov x+3, #0
        ret


SendString:
        ; Send a constant-zero-terminated string using the serial port
    clr A
    movc A, @A+DPTR
    jz SendStringDone
    lcall putchar
    inc DPTR
    sjmp SendString
SendStringDone:
    ret





; Configure the serial port and baud rate using timer 1
InitSerialPort:
    ; Since the reset button bounces, we need to wait a bit before
    ; sending messages, or risk displaying gibberish!
    mov R1, #222
    mov R0, #166
    djnz R0, $   ; 3 cycles->3*45.21123ns*166=22.51519us
    djnz R1, $-4 ; 22.51519us*222=4.998ms
    ; Now we can safely proceed with the configuration
        clr     TR1
        anl     TMOD, #0x0f
        orl     TMOD, #0x20
        orl     PCON,#0x80
        mov     TH1,#T1LOAD
        mov     TL1,#T1LOAD
        setb TR1
        mov     SCON,#0x52
    ret

; Send a character using the serial port
putchar:
    jnb TI, putchar
    clr TI
    mov SBUF, a
    ret

MainProgram:
    mov SP, #7FH ; Set the stack pointer to the begining of idata
    mov PMOD, #0 ; Configure all ports in bidirectional mode
    cpl PULSE_PIN
    lcall LCD_4BIT
        WriteCommand(#0x40)

    lcall UIBEGIN
    Wait_Milli_Seconds (#255)
    Wait_Milli_Seconds (#255)
    Wait_Milli_Seconds (#255)
    Wait_Milli_Seconds (#255)
    lcall InitSerialPort
    lcall INIT_SPI
    lcall LCD_4bit
    lcall Timer0_Init
    lcall Timer2_Init
    setb EA

    mov UILANGFLAG, #0x0
    mov UICURRSTATENUM, #0x00


    cpl OVENCOUNT
    cpl BLUE_RGB

    mov soakTemp, #0x96 ; 0x96 is 150
    mov state, #0x00

    mov beepFlag, #0x00
    mov beepLength, #0x00
    mov numberOfBeeps, #0x00
    mov beepTimer, #0x00

    mov songFlag, #0x00
    mov nextNote, #0x00

    mov ovenONOFF, #0x01

    mov soakTime, #0x60
    mov soakTime+1, #0x00

    mov power, #0x00
    mov powerSecs, #0x00
    mov powerFlag, #0x00

    mov reflowTempBCD+1, #0x02
    mov reflowTempBCD, #0x20

    mov reflowTemp, #0xDC ;220C is DC
    mov reflowTime+1, #0x00
    mov reflowTime, #0x45

    mov timeCounter+1, #0x00
    mov timeCounter, #0x00

    mov soakTempBCD+1, #0x01
    mov soakTempBCD, #0x50

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

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

Forever:

        ljmp stateMachine
        ljmp Forever

;___________________________Reflow State Machine_________________________________
stateMachine:
        mov a, state
        cjne a, #0x00,checkAbort
        ljmp state0
checkAbort:

        jb STOP_BUTTON, checkS1
        Wait_Milli_Seconds(#50)
        jb STOP_BUTTON, checkS1
        jnb STOP_BUTTON, $
        mov bcd+1, #0x00
        mov bcd, #0x05
        Send_BCD(bcd+1)
        Send_BCD(bcd)
        cpl RED_RGB
        cpl GREEN_RGB
        mov state, #0x06
        ljmp Forever
checkS1:
        cjne a, #0x01, checkS2
        ljmp state1
checkS2:
        cjne a, #0x02, checkS3
        ljmp state2
checkS3:
        cjne a, #0x03, checkS4
        ljmp state3
checkS4:
        cjne a, #0x04, checkS5
        ljmp state4
checkS5:
        cjne a, #0x05, checkS6
        ljmp state5
checkS6:
        cjne a, #0x06, checkS1
        ljmp resetState

state0:
        mov power, #0
        ljmp Forever
state1:
        ;Ramp to Soak
        mov power, #5 ;100% power
        mov a, Abort
        clr c ; clears the carry flag
        subb a, timeCounter ;Compares the current runtime to 60 seconds
        jnc continuePreheat ;If timer<60 then continue preheating

        mov a, AbortTemp
        clr c
        subb a, temperature; Compares current temp to 50C
        jnc abortPreheat
continuePreheat:
        clr c
        mov a, soakTemp
        subb a, #0x01
        subb a, temperature
        jnc goForever
        ;Goes to next state
        mov timeCounter, #0x00
        mov timeCounter+1, #0x00
        mov state, #0x02
        lcall pulsePin
        mov numberOfBeeps, #0x01
        mov beepLength, #0x00

        ljmp Forever
```

```
abortPreheat:
        mov bcd+1, #0x00
        mov bcd, #0x05
        Send_BCD(bcd+1)
        Send_BCD(bcd)
        cpl RED_RGB    ;Turn on Red LED
        cpl GREEN_RGB  ;Turn off Green LED
        mov UICURRSTATENUM, #0x00
        mov state, #0x06
        ljmp Forever
state2:
        ;Preheat/Soak
        mov power, #1 ; 20% power

        mov a, soakTime
        clr c
        subb a, #0x01
        subb a, timeCounter
        jnc goForever

        mov a, soakTime+1
        cjne a, #0x00, continueCheck
        sjmp changeState3
continueCheck:
        clr c
        subb a, #0x01
        subb a, timeCounter+1
        jnc goForever
changeState3:
        lcall pulsePin
        mov numberOfBeeps, #0x01
        mov beepLength, #0x00

        mov state, #0x03
        mov timeCounter, #0x00
        mov timeCounter+1, #0x00
goForever:
        ljmp Forever

state3:
        ;Ramp to Peak
        mov power, #5 ; 100% power
        clr c
        mov a, reflowTemp
        subb a, #0x01
        subb a, temperature
        jnc goForever
        mov timeCounter, #0x00
        mov timeCounter+1, #0x00

        mov numberOfBeeps, #0x01
        mov beepLength, #0x00
        lcall pulsePin
        mov state, #0x04
        ljmp Forever

state4:
        ;Reflow
        mov power, #1 ; 20% power

        mov a, reflowTime
        clr c
        subb a, #0x01
        subb a, timeCounter
        jnc goForever
        mov a, reflowTime+1
        cjne a, #0x00, continueCheckB
        sjmp changeState5
continueCheckB:
        clr c
        subb a, #0x01
        subb a, timeCounter+1
        jnc goForever
changeState5:
        lcall pulsePin
        mov numberOfBeeps, #0x01
        mov beepLength, #0x01

        mov timeCounter, #0x00
        mov timeCounter+1, #0x00
        mov state, #0x05
        ljmp Forever
continueReflow:
        ljmp Forever
state5:
        ;Cooling
        mov power, #0
        mov a, temperature
        clr c
        subb a, coolTemp
        jnc goForever

        mov timeCounter, #0x00
        mov timeCounter+1, #0x00

        mov bcd+1, #0x05
        mov bcd, #0x00
        Send_BCD(bcd+1)
        Send_BCD(bcd)
        mov numberOfBeeps, #0x00
        mov songFlag, #0x01
        mov state, #0x05

        mov UICURRSTATENUM, #0x00
        mov state, #0x00

        cpl BLUE_RGB
        cpl GREEN_RGB

        ljmp Forever
resetState:
        ;Loops until Reset is hit
        mov power, #0

        mov numberOfBeeps, #0x01
        mov beepLength, #0x00

        sjmp resetState


retState:
        ret

;_______________________________PWM Calculations________________________________________
powerCalc:
        mov a, power
        cjne a, #0x00, continue_power
        mov a, powerFlag
        cjne a, #0x01, retPower
        lcall turnOffOven
        sjmp retPower
```

```
continue_power:
        mov a, powerSecs
        cjne a, #0x00, leaveOn
        mov a, powerFlag
        cjne a, #0x00, leaveOn
        lcall turnOnOven
leaveOn:
        mov a, powerSecs
        add a, #0x01
        da a
        cjne a, #0x05, doNormal
        mov a, powerFlag
        cjne a, #0x00, skip_on
        lcall turnOnOven
skip_on:
        mov a, #0x00
doNormal:
        mov powerSecs, a
        cjne a, power, retPower
        mov a, powerFlag
        cjne a, #0x01, retPower
        lcall turnOffOven
retPower:
        ret

turnOnOven:
        mov powerFlag, #0x01
        cpl OVENCOUNT
        ret
turnOffOven:
        mov powerFlag, #0x00
        cpl OVENCOUNT
        ret
;_______________________________________DisplayData___________________________________________
displayData:

        mov a, state
        cjne a, #0x00, dispUpper
        mov a, UICURRSTATENUM
        cjne a, #0x00, leaveThisState
        mov a, UILANGFLAG
        cjne a, #0x00, initialFrench
        Set_Cursor(1,1)
        Send_Constant_String(#UIPRESSSTART)
        Set_Cursor(2,1)
        Send_Constant_String(#UIPRESSSTART2)
        ret
initialFrench:
        Set_Cursor(1,1)
        Send_Constant_STring(#UIPRESSSTART_FR)
        Set_Cursor(2,1)
        Send_Constant_String(#UIPRESSSTART2_FR)
leaveThisState:
        ret
dispUpper:
        cjne a, #0x06, continueDispN
        Set_Cursor(2,1)
        mov a, UILANGFLAG
        cjne a, #0x01, DispEngABORT
        Send_Constant_String(#FR_UIOKEMERG)
        Set_Cursor(1,1)
        Send_Constant_String(#FR_UIEMERG)
        ret
DispEngABORT:
        Send_Constant_String(#UIOKEMERG)
        Set_Cursor(1,1)
        Send_Constant_String(#UIEMERG)
        ret
continueDispN:
        Set_Cursor(1,1)
        Send_Constant_String(#CLEAR)
        Set_Cursor(2,1)
        Send_Constant_String(#CLEAR)
        Set_Cursor(1,4)
        Display_BCD(bcd)

        mov a, bcd+1
        cjne a, #0x00, dispUppperTempByte
        Set_Cursor(1,3)
        Send_Constant_String(#_SPACE)
        sjmp  dispTime

dispUppperTempByte:
        Set_Cursor(1,3)
        Display_BCD_low(bcd+1)

dispTime:
        Set_Cursor(1,14)
        Display_BCD(timeCounter)

        mov a, timeCounter+1
        cjne a, #0x00, dispUppperTimeByte
        Set_Cursor(1,13)
        Send_Constant_String(#_SPACE)
        sjmp dispVals

dispUppperTimeByte:
        Set_Cursor(1,13)
        Display_BCD_low(timeCounter+1)

dispVals:
        Set_Cursor(1,1)
        Send_Constant_String(#STATE_TEMP)
        Set_Cursor(1,6)
        Send_Constant_String(#DEG_C)
        Set_Cursor(1,16)
        Send_Constant_String(#SEC)
        Set_Cursor(1,8)

        mov a, UILANGFLAG
        cjne a, #0x01, continueDisplayEngl
        Send_Constant_String(#Fr_Time)
        sjmp contDispVals

continueDisplayEngl:
        Send_Constant_String(#STATE_TIME)

;______________________French Version of Display Outputs____________________________
contDispVals:
        Set_Cursor(2,1)
        mov a, state
        cjne a, #0x01, tryS2
        mov a, UILANGFLAG
        cjne a, #0x01, DisplayEngl_1
        Send_Constant_String(#FR_RAMPSOAK)
        ret
DisplayEngl_1:
        Send_Constant_String(#RAMPTOSOAK)
        ret
tryS2:
        cjne a, #0x02, tryS3
        mov a, UILANGFLAG
        cjne a, #0x01, DisplayEngl_2
        Send_Constant_String(#FR_SOAK)
        ret
DisplayEngl_2:
        Send_Constant_String(#SOAKING)
        ret
tryS3:
        cjne a, #0x03, tryS4
        mov a, UILANGFLAG
        cjne a, #0x01, DisplayEngl_3
        Send_Constant_String(#FR_RAMPPEAK)
        ret
DisplayEngl_3:
        Send_Constant_String(#RAMPTOPEAK)
        ret
tryS4:
        cjne a, #0x04, tryS5
        mov a, UILANGFLAG
        cjne a, #0x01, DisplayEngl_4
        Send_Constant_String(#FR_REFLOW)
        ret
DisplayEngl_4:
        Send_Constant_String(#REFLOW)
        ret
tryS5:
        cjne a, #0x05, exitStateDisp
        mov a, UILANGFLAG
        cjne a, #0x01, DisplayEngl_5
        Send_Constant_String(#FR_COOL)
        ret
DisplayEngl_5:
        Send_Constant_String(#COOLING)

exitStateDisp:
        ret


;___________________________Play Sound______________________________________;
;A function that will determine whether to play a short beep, long beep or song
beepOn:
        mov a, numberOfBeeps
        cjne a, #0x00, playNoise
        mov a, songFlag
        cjne a, #0x00, playSong
        ret
playNoise:
        mov a, beepLength
        cjne a, #0x00, playLong
playShort:
        mov a, beepFlag
        cjne a, #0x00,stopBeep
        setb ET0
        mov beepFlag,#0x01
        ret
stopBeep:
        clr ET0
        mov beepFlag, #0x00
        mov a, numberOfBeeps
        add a, #0x99
        da a
        mov numberOfBeeps, a
        ret
playLong:
        mov a, beepTimer
        cjne a, #0x04, keepPlaying
        mov a, beepFlag
        cjne a, #0x01, maintainState
        ;mov numberOfBeeps, #0x00
        mov beepFlag, #0x00
        clr ET0
        ret
keepPlaying:
        add a, #0x01
        da a
        mov beepTimer, a
        mov a, beepFlag
        cjne a, #0x00, maintainState
        mov beepFlag, #0x01
        setb ET0
        ret

playSong:
        mov a, nextNote
        cjne a, songLength, continuePlayingSong
        mov a, beepFlag
        cjne a, #0x01, maintainState
        mov songFlag, #0x00
        mov beepFlag, #0x00
        clr ET0
maintainState:
        ret
continuePlayingSong:
        add a, #0x01
        da a
        mov nextNote, a
        mov a, beepFlag
        cjne a, #0x00, maintainState
        mov beepFlag, #0x01
        setb ET0
        ret


UIBEGIN:
    ;_______________________Creates the "ECE" Character________________________________
    ; char 1

    WriteData(#0x00)
        WriteData(#0x00)
        WriteData(#0x00)
        WriteData(#0x01)
        WriteData(#0x01)
        WriteData(#0x03)
        WriteData(#0x00)
        WriteData(#0x00)

    ;char 2

    WriteData(#0x1f)
        WriteData(#0x1e)
        WriteData(#0x1c)
        WriteData(#0x18)
        WriteData(#0x1f)
        WriteData(#0x1f)
        WriteData(#0x03)
        WriteData(#0x07)


        ;char 3
    WriteData(#0x00)
        WriteData(#0x00)
        WriteData(#0x00)
        WriteData(#0x00)
        WriteData(#0x18)
        WriteData(#0x10)
        WriteData(#0x00)
        WriteData(#0x00)


        ;char 4

    WriteData(#0x00)
        WriteData(#0x00)
        WriteData(#0x00)
        WriteData(#0x0f)
        WriteData(#0x0c)
        WriteData(#0x0e)
        WriteData(#0x0c)
        WriteData(#0x0f)

        ;char 5
    WriteData(#0x0C)
        WriteData(#0x08)
        WriteData(#0x00)
        WriteData(#0x1F)
        WriteData(#0x1C)
        WriteData(#0x1C)
        WriteData(#0x1C)
        WriteData(#0x1F)


        ;char 6
    WriteData(#0x00)
        WriteData(#0x00)
        WriteData(#0x00)
        WriteData(#0x1E)
        WriteData(#0x18)
        WriteData(#0x1C)
        WriteData(#0x18)
        WriteData(#0x1E)



    WriteCommand(#0x87)
    WriteData(#0x00)

        WriteCommand(#0x88)
    WriteData(#0x01)

    WriteCommand(#0x89)
    WriteData(#0x02)

    WriteCommand(#0xC7)
    WriteData(#0x03)

    WriteCommand(#0xC8)
    WriteData(#0x04)

    WriteCommand(#0xC9)
    WriteData(#0x05)

    ret

sendSPI:
        clr CE_ADC
         ;_______________________Thermal Couple Output_______________________
         mov R0, #00000001B ; Start bit:1
         lcall DO_SPI_G
        mov R0, #10100000B ; Single ended, read channel 2
        lcall DO_SPI_G
        mov a, R1 ; R1 contains bits 8 and 9
        anl a, #00000011B ; We need only the two least significant bits
        mov ThermCup+1, a ; Save result high. (in the upper bits-->result is first byte,
result+1 is second byte, etc.)

        mov R0, #55H ; It doesn't matter what we transmit...
        lcall DO_SPI_G
        mov ThermCup, R1 ; R1 contains bits 0 to 7. Save result low.
        setb CE_ADC

        mov ThermCup+2, #0
        mov ThermCup+3, #0



        clr CE_ADC
        ;_______________________For Temperature Reference_______________________

        mov R0, #00000001B ; Start bit:1
        lcall DO_SPI_G

        mov R0, #10010000B ; Single ended, read channel 1
        lcall DO_SPI_G
        mov a, R1 ; R1 contains bits 8 and 9
        anl a, #00000011B ; We need only the two least significant bits
        mov TempRef+1, a ; Save result high. (in the upper bits-->result is first byte,
result+1 is second byte, etc.)

        mov R0, #55H ; It doesn't matter what we transmit...
        lcall DO_SPI_G
        mov TempRef, R1 ; R1 contains bits 0 to 7. Save result low.
        setb CE_ADC

        lcall print_toPuTTY
        ret

pulsePin:
       cpl PULSE_PIN
       Wait_Milli_Seconds(#50)
       cpl PULSE_PIN
       ret
END
```