University of British Columbia

Electrical and Computer Engineering

ELEC 291: Electrical Engineering Design Studio I 2023

Instructor’s name: Dr. Jesus Calvino-Fraga

Section: L2B

PROJECT 1 - REFLOW OVEN CONTROLLER

Date: March 2nd, 2023

Group number:

Student number	Student Name	% Points	Signature
22946222	Duc Thang Huynh	120	
48395651	Andrey Abushakhmanov	97	
46590261	Bevio Chen	102	
90300815	Pratham Goel	92	
88191713	Victor Lui	102	
63156905	Tomas Bang	87	

TABLE OF CONTENTS

1. Introduction

The reflow oven controller's purpose is to generate a reflow graph and control the oven for soldering the EFM8 board in assembly language. The controller communicates with the user through button input and output on the LCD display and speaker, and it measures temperatures between 25℃ and 240℃. The soaking temperature, soak time, reflow temperature and reflow time are selectable between 130℃ and 170℃ for the soak temperature, between 220℃ and 240℃ for the reflow temperature, and 0 to 60 seconds for soak and reflow time. The reflow oven controller's safety features comprise the emergency stop button and safety oven control when the oven temperature is not reaching 50℃ in 60 seconds. The controller speaker plays a sound of the current temperature every 5 seconds, state changes, and the start and finish of the procedure. Furthermore, a strip chart plot of the oven temperature concerning time is plotted during the reflow process on the user's computer. The first part of the report gives idea investigation, brainstorming, data collection, and result analysis. The reflow oven controller design, objectives and constraints, problems and solutions, and a detailed design during the development process are provided in the second part of the report. In the report's final part, a Live-long learning, project conclusion, recommendations, references and bibliography is given.

2.0 Investigation

To implement the design, there are multiple approaches available. However, before choosing a specific approach, investigation and identification of relevant parameters and materials listed to evolve the project understanding. This section highlights the methodology used to investigate idea generation, investigation design, and data collection and synthesis.

2.1 Idea Generation

A finite state machine (FSM) is the integration of a combinational and sequential logic that processes the information given and produces programmed output at some specific state and state transition [1], which can determine the current state and the user's choice temperature and time. A two-state finite state machine is built as a testing machine with inputs from buttons, temperature measurement, and time overflow. Identifying the list of requirements that needs to be fulfilled, the project is divided into smaller, more manageable parts. This approach made it easier to generate solutions and complete the project efficiently. Brainstorm various ideas that could potentially solve challenges. The most suitable design for each element is selected and implemented through discussion and filtering. After successfully testing every part separately, they combined to get a complete reflow oven controller. With this approach, the complete system might achieve all the requirements while optimizing its overall performance.

2.2 Investigation Design

The two-state finite state machine is built as a draft code to test the logic and LCD. The opamp gain is calculated to get the accurate temperature reading from the thermocouple while the temperature is displayed on the LCD and sent on Putty. Before implementing the design, a component identification process is conducted to ensure that necessary parts are identified, and the correct part numbers are selected to meet the project requirements. In the event of any missing components, a suitable functional replacement will be chosen that matches the original component's functionality while considering any potential differences in specifications or performance characteristics. In addition, datasheets and spec sheets are collected for all components from reputable sources before implementing the circuit design on the breadboard.

2.3 Data Collection and Synthesis

The data was collected and analyzed from consistent testing to get accurate findings about our soldering oven controller. A strict methodology is used for the Data Gathering procedure and comparing the findings to the expected values produced from the design documentation and circuit diagrams. Furthermore, the controller operation and user interface are tested and monitored by uploading the coded software to the controller, observing the states and values presented and operating on the LCD panel and Putty. The thermocouple is also tested with a Python code to determine the accurate temperature measurement and opamp voltage gain. In addition, the speaker is tested to produce numbering from 1 to 10, and some humans record sound. The deviation and challenges are identified and approached to be solved.

2.4 Analysis of Results

Testing and troubleshooting using a multimeter to compare our results with the expected outcomes. The data is collected and analyzed through testing at various stages of the design process to achieve predictable and precise outcomes. The temperature measured was given a source of error of approximately 5% compared to the temperature measured using a thermometer. The two-state FSM is tested with plenty of user input possibility at different temperatures with the timer to monitor the overall process to determine the result. In the final stages of the design process, many test cases are generated to fine-tune and iterate our design, ensuring accurate results. To further ensure safety and functionality, corner cases such as the erroneous installation of the temperature sensor or user cancellation during the reflow process are examined. Through these efforts, we verified that our soldering oven controller performed as expected in all cases.

3.0 Design

3.1 Use of Process

The code to measure temperature and time in the previous lab provides a basis for understanding the structure of the project code, serving as the foundation for developing our functionality for measuring temperature using the thermocouple and developing a timer. The voltage gain is calculated with the formula below:

Formula

$$V_{out} = 41\frac{\mu V}{°C} \text{ x} \frac{R_1}{R_2}$$

After professor Jesus's recommendation, the R2 value has to be greater than 330 Ohms to measure the voltage difference between two materials accurately, thus providing a correct temperature calculation. After some calculations and experiments, the voltage gain has to be over 341.5 to get an appropriate voltage difference. Therefore, 390 Ohm resistors and 133 kOhm resistors are selected for R2 and R1 to get the voltage gain desired.

3.2 Needs and Constraints

The reflow oven controller needs to stop the oven when the temperature is not over 50 degree Celsius in 1 minute, and the reflow process must be stopped when the stop button is pressed at any duration of the reflow process. In addition, the reflow oven should let the user know the oven temperature every 5 seconds and state transitions. The oven controller has two key constraints: an LCD and a thermocouple meter. The LCD constraint specifies a display of two lines and sixteen positions for representing letters and numbers. Another constraint is the thermocouple meter's 2-meter cable length, which uses twisted Chromel and Alumel wires that can introduce significant errors in the reading when there is an unconnected part compared to the temperature measured by the thermometer. Additionally, the solution must be achievable within a given time frame of two weeks and must not be too complex.

3.3 Problem Specification

The controller can be adjusted to some specific parameter to meet the needs of the microprocessor system by employing push buttons to alter parameters like soak temperature, soak time, reflow temperature, and reflow time. However, measuring temperature with a thermocouple greater than 156 degrees Celsius poses a challenge for the FSM state transition while the temperature is displayed correctly on the LCD and sending it to Putty but not operating as expected. Furthermore, the correct temperature can't be stored in a single bit, thus making the comparison condition in the FSM state transition faces difficulties. Another challenge is prioritizing user safety, a failsafe system has been designed to prevent ongoing heating in the event of incorrect thermocouple placement. The Pulse Width Modulation (PWM) is coded, but only keeping the oven on causes a problem with the reflow oven graph. The user interface consists of an LCD that shows temperature and time, data output to a PC for a more visually appealing process presentation, and sound effects that notify the user during state transitions and capture their attention as necessary.

3.4 Solution Generation

To meet these specifications, an integrated FSM is implemented to control the oven processes and user interface on the same clock frequency. However, the audio aspect of the user interface takes seconds to finish, thus some pipelining would also be employed to prevent these processes from blocking one another. In addition, we would also need to create a dormant state for which the soak and reflow times and temperatures could be customized via buttons and a user interface. For the oven control, we decided to use PWM to communicate with the oven's solid state relay during each state, using a thermocouple with an LM335 as its cold junction calculation to measure oven temperature. An addition of a manual stop push button is accessible to terminate the soldering process at any time during operation whenever needed, further enhancing the reflow oven controller cautiously. The control over pin is the measure before the connection with the negative wire of the SSR box is 5V, but when connected the voltage measured is approximately 3V. Therefore, an N-mosfet is added its gate is connected to the pin controlling PWM_OUTPUT from the code while the drain is connected to the negative wire of the oven. The audio aspect of the user interface would be integrated by uploading a .wav file to the speaker, which the finite state machine would control what part of the file to play.

3.5 Solution Evaluation

This generated solution is chosen because it offered a balance between performance and feasibility of implementation given the time constraint. Thus, most of the solution evaluations are not aimed at perfection, but at practicality, covering the project criteria. The designed concept user interface has the LCD and speaker running in tandem, with the former updating time, state, and temperature in real-time and the latter updating the state of the oven process and temperature every five seconds. This design option fully follows the project criteria as requested. The designed oven controller system would also run in tandem with the user interface design and update the PWM based on which state the customizable processes were in, determined by the thermocouple and timer. However, as a choice for the facility in implementation, the safety feature was designed as a dependency on the user interface updates as opposed to the oven. As a result, this design meets the project criteria in an alternative, faster method, but possibly sacrifices compatibility. In the end, the controller final design is chosen as it met the needs and requirements of the project while aiming to be cost-effective in terms of time allocated. More importantly, our solution needs to be practical and easy to run when we demonstrate it, so we dedicated our attention to this design as it valued the user interface more.

3.6 Detailed Design

(See Appendix for hardware parts used)

Hardware Block Diagram

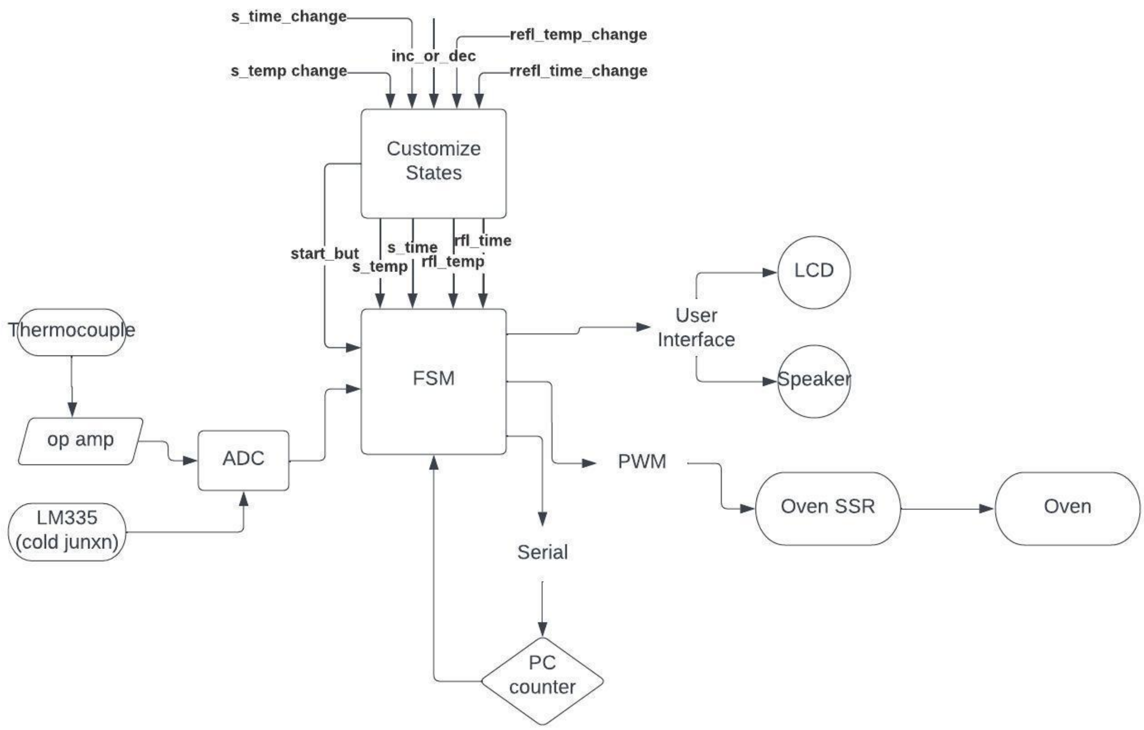

Figure 1a: Block Diagram of Reflow Oven Controller Circuit

The entire circuit is made up of five main sections. At the center is the FSM logic on the microcontroller, connected to all the processes of the circuit. The main FSM is preset with instructions by the user when the program is executed or if the circuit is reset. This is controlled by six buttons that let the user select which values to change and exit into the main FSM. During this process, the main FSM reads inputs of the temperature and timer and outputs changes to the user interface and pulse width modulation function. The user interface consists of the LCD and speaker functions as its outputs. The PWM has its output regulated by the circuit, which then communicates with the oven's SSR. The thermocouple is used to read the temperature in the oven as input. However, it needs to be connected to a cold junction, and these two voltages need to be put through an analog-to-digital converter to send back to the FSM. The timer is an independent input that loops through continuously. Finally, the serial port allows us to upload our software onto flash memory to the microcontroller. During the process of the second increment in the timer, the speaker function is called, thus has a chance to produce any sound at any period of the reflow process

Software:

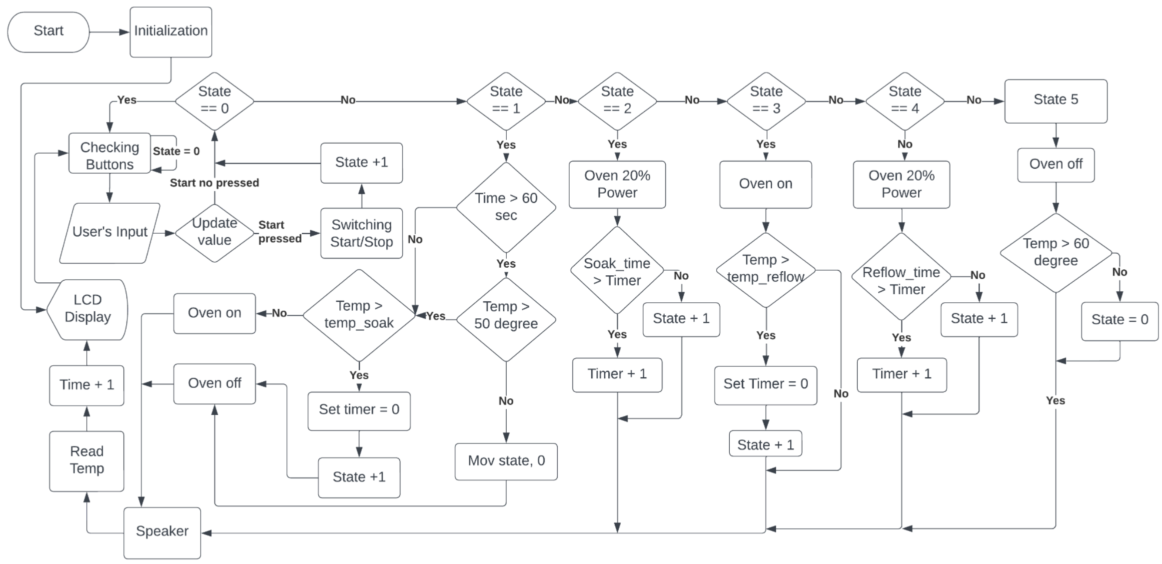

Figure 1b: The Block Digram of Reflow Oven Controller FSM

The reflow oven controller is coded in assembly language. The two-state FSM is built with only two adjustable values, soak temperatures and time by two buttons. Then the value is printed on the LCD to verify. An increment and decrement of temperature and time are developed and tested through printing on the LCD. Upon successfully configuring the buttons, a Start button is implemented with some default values added. The two-state FSM tested the state transition when the temperature reached 23 degrees Celsius, using the code from the previous lab and some adjustments with the button inputs and message printing. A better state message is developed by showing the temperature measured through LM335 and time and aligned at a specific position that eases users. After some successful tests with the previous adjustment, a stop button function is implemented to the Start button as a Stop button during the reflow process. The next step is to create a timer variable to be printed and show users the time in the soak period. This development required resetting the value and setting up the timer whenever state two is entered. In one of the lectures, professor Jesus shows the implementation of the user's selected memorized value for all the parameters. The code is then implemented in the two-state FSM. The safety feature is built and developed into the two-state FSM with a safe temperature to be 23 degrees for testing. Verification of state transition and LCD are as expected.

The final version of the reflow oven controller is developed by the similarity of state one and state three without the safety feature and different variables between state two and state 4. In state 5, the oven temperature is compared with 60 degrees as a safety temperature to touch and finish the reflow soldering process. Then the reflow oven controller is set to go back to the configured state for the next reflow process. The oven is controlled from PWM by coding the pin on/off or alternating. Then oven positive wire is connected to a 5V source with the negative wire is connected to the source of an N-mosfet to avoid the change in voltage. The PWM is tested successfully and controlled the oven as required. Until this point, the FSM is used to test with LM335 for temperature measurement. The thermocouple is installed with a voltage gain of 333.33 and the value for R2 is 390 Ohm. The temperature display on the LCD is accurate even with a temperature above 200 ℃ and sending the temperature to

Putty. However, with the thermocouple implementation, the FSM timer is not working properly, specifically with the state transition from state 1 to state 2. The final version of the code is provided in the appendix with the temperature measured by calculation of the voltage measured from LM335 (cold junction) and thermocouple (hot junction).

3.6 Solution Assessment

Constant testing of the developed code and FSM state transition is crucial to avoid challenges in debugging and, further, simply and breakdown sections of the FSM development. With the error in storing and comparing the value of the thermocouple wire, the reflow oven controller is not fully developed.

4. Live-Long Learning

During the reflow oven controller project, our team utilized various technical concepts from our prerequisite courses, including programming and hardware design skills. Specifically, CPEN 211 is particularly useful in resolving the circuit for designing and developing the finite state machine. We identified a knowledge gap when designing the FSM, and used this opportunity to gain a deeper understanding of shift registers, common cathode and common anode configurations for LEDs, and circuit design for affixing all of the LEDs to our prototypes.

In addition to technical skills, our team also learned the importance of working effectively in a team and communicating well. We found that open and frequent communication and collaboration were essential to ensure that everyone understood the project goals and tasks and to ensure that we were all working towards the same objectives. We delegated tasks based on individual strengths and availability, which allowed us to utilize each team member's strengths and maximize productivity.

Furthermore, we had to adapt to unexpected challenges and changes. One of our teammates had to attend to a family emergency in the middle of the project, which required us to adjust our schedules and redistribute tasks. We learned how to balance individual responsibilities with team goals, and how to work efficiently and effectively in a group setting.

Overall, this project provided valuable experience with embedded systems, circuit analysis, hardware design, construction, and teamwork, which will not only be beneficial for our future studies as electrical engineering students and our future careers but also for our personal growth as effective communicators.

5. Conclusion

Our soldering oven is specifically built to solder SMT components with user-defined parameters, such as an LCD to display the temperature. Our design also includes several safety mechanisms to ensure that the process is carried out properly, such as when the temperature sensor is not put within the oven or when the user needs to stop the reflow midway through.Despite various delays and errors during the project, the soldering oven controller was eventually successful. We worked on this project for 60 hours in total.

References

[1] Ziogou. C., Krinidis. S., ..“An intelligent decision making and notification system based on a knowledge-enabled supervisory monitoring platform”, Computer Aided Chemical Engineering. Vol(38). 2016.

Bibliography

1. J.Calvino-Fraga. ELEC 291. Project, Topic: “ELEC291_Reflow_Oven_Controller_2023.” Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC
2. J.Calvino-Fraga. ELEC 291. Class Lecture, Topic: “February 3 lecture slides: Project1 - Reflow Oven Controller 2023.” Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC
3. J.Calvino-Fraga. ELEC 291. Class Lecture, Topic: “Lecture slides from 2023. (project1 - EFM8_ FSM_NVMEM_2023).” Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC

Appendix I: Hardware Components

    - Atmel AT89LP52 Microcontroller IC 80C51 MCU 8K FLASH 40-DIP
    - Microchip Technology Inc. MCP3008 8-Channel 10-Bit ADC with SPI Interface
- Associated Parts for Chipsets and Hardware
    - 22.1184MHz CTS ATS122B-E Quartz Crystal
    - 3 0.1 Microfarad Capacitors
    - 5mm Green LED
    - BO230XS serial USB Adaptor
    - 4 Push Button Switches
    - 2000 Ohm 5% tolerance Resistors
    - 330 Ohm 5% tolerance Resistors
    - STMicroelectronics 2N3904 NPN Transistor
- Team-Designed Hardware
    - Op Amp circuit
        - Texas Instruments OP07CP Op Amp chip
        - National Semiconductor LMC7660IN Switched Capacitor Voltage Converter
        - 4, 100 Ohm 5% tolerance Resistors
        - 2, 22000 Ohm 5% tolerance Resistors
        - 2, 2000 Ohm 5% tolerance Resistors
        - 2, 10 Microfarad 50 V Capacitors
- Temperature Monitoring Hardware
    - General Purpose Thermocouple
    - STMicroelectronics LM335 for use as cold junction

Appendix II: Source Code

```
$MODLP51RC2
org 0000H
   ljmp MainProgram

org 0x001B ; Timer/Counter 1 overflow interrupt vector. Used in this code to replay the
wave file.
       ljmp Timer1_ISR
; Timer/Counter 2 overflow interrupt vector
org 0x002B
       ljmp Timer2_ISR



CLK  EQU 22118400
BAUD equ 115200
BRG_VAL equ (0x100-(CLK/(16*BAUD)))
TIMER2_RATE   EQU 1000
TIMER2_RELOAD EQU ((65536-(CLK/TIMER2_RATE)))

;====SPEAKER CONFIG===================
TIMER1_RATE    EQU 22050     ; 22050Hz is the sampling rate of the wav file we are playing
TIMER1_RELOAD  EQU 0x10000-(CLK/TIMER1_RATE)

SPEAKER  EQU P2.6 ; Used with a MOSFET to turn off speaker when not in use

; The pins used for SPI
FLASH_CE  EQU  P2.5
SPEAKERMY_MOSI   EQU  P2.4
SPEAKERMY_MISO   EQU  P2.1
SPEAKERMY_SCLK   EQU  P2.0

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

DSEG at 0x30
       result: ds 4
       bcd: ds 5
       x: ds 4
       y: ds 4
       temp_soak: ds 1
       time_soak: ds 1
       temp_refl: ds 1
       time_refl: ds 1
       state: ds 1
       temp: ds 1
       time: ds 1
       fivesec_timer: ds 1
       sec: ds 1
       pwm_ratio:    ds 2
       Count1ms: ds 2
       pwm: ds 1
       mode: ds 1
       time_display: ds 1
       cold: ds 1

; Variables used in the program:
dseg at 30H
       w:   ds 3 ; 24-bit play counter.  Decremented in Timer 1 ISR.

BSEG
       mf: dbit 1
```

```
       configure_flag: dbit 1
    r2s_sound_flag: dbit 1
    s_sound_flag : dbit 1
    r2r_sound_flag: dbit 1
    reflow_sound_flag: dbit 1
    cool_sound_flag: dbit 1

CSEG
; These 'EQU' must match the wiring between the microcontroller and ADC
CE_ADC    EQU  P0.6
MY_MOSI   EQU  P0.5
MY_MISO   EQU  P0.3
MY_SCLK   EQU  P0.0
LCD_RS equ P3.2
LCD_E  equ P3.3
LCD_D4 equ P3.4
LCD_D5 equ P3.5
LCD_D6 equ P3.6
LCD_D7 equ P3.7

START_BUTTON equ P2.7
SHIFT_PB equ P4.5
SOAKTIME_BUTTON equ P0.4
SOAKTEMP_BUTTON equ P0.7
REFLTIME_BUTTON       equ P0.1
REFLTEMP_BUTTON equ P0.2

PWM_OUTPUT equ P1.0

$NOLIST
$include(LCD_4bit.inc)
$include(math32.inc)
$LIST

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

SendToLCD:
       mov b, #100
       div ab
       orl a, #0x30 ; Convert hundreds to ASCII
       lcall ?WriteData ; Send to LCD
       mov a, b    ; Remainder is in register b
       mov b, #10
       div ab
       orl a, #0x30 ; Convert tens to ASCII
       lcall ?WriteData; Send to LCD
       mov a, b
       orl a, #0x30 ; Convert units to ASCII
       lcall ?WriteData; Send to LCD
ret

Load_Configuration:
       mov dptr, #0x7f84 ; First key value location.
       getbyte(R0) ; 0x7f84 should contain 0x55
```

```
        cjne R0, #0x55, Load_Defaults
        getbyte(R0) ; 0x7f85 should contain 0xAA
        cjne R0, #0xAA, Load_Defaults
        ; Keys are good.  Get stored values.
        mov dptr, #0x7f80
        getbyte(temp_soak) ; 0x7f80
        getbyte(time_soak) ; 0x7f81
        getbyte(temp_refl) ; 0x7f82
        getbyte(time_refl) ; 0x7f83
ret

Save_Configuration:
        push IE ; Save the current state of bit EA in the stack
        clr EA ;Disable interrupts
        mov FCON, #0x08 ; Page Buffer Mapping Enabled (FPS = 1)
        mov dptr, #0x7f80 ; Last page of flash memory
        ; Save variables
        loadbyte(temp_soak) ; @0x7f80
        loadbyte(time_soak) ; @0x7f81
        loadbyte(temp_refl) ; @0x7f82
        loadbyte(time_refl) ; @0x7f83
        loadbyte(#0x55) ; First key value @0x7f84
        loadbyte(#0xAA) ; Second key value @0x7f85
        mov FCON, #0x00 ; Page Buffer Mapping Disabled (FPS = 0)
        orl EECON, #0b01000000 ; Enable auto-erase on next write sequence
        mov FCON, #0x50 ; Write trigger first byte
        mov FCON, #0xA0 ; Write trigger second byte
        ; CPU idles until writing of flash completes.
        mov FCON, #0x00 ; Page Buffer Mapping Disabled (FPS = 0)
        anl EECON, #0b10111111 ; Disable auto-erase
        pop IE
ret

;-----------------------;
;  Set up default values ;
;-----------------------;

Load_Defaults:
        mov temp_soak, #150
        mov time_soak, #45
        mov temp_refl, #25
        mov time_refl, #30
ret

; Configure the serial port and baud rate
InitSerialPort:
    ; Since the reset button bounces, we need to wait a bit before
    ; sending messages, otherwise we 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 proceed with the configuration
        orl     PCON,#0x80
        mov     SCON,#0x52
        mov     BDRCON,#0x00
        mov     BRL,#BRG_VAL
        mov     BDRCON,#0x1E ; BDRCON=BRR|TBCK|RBCK|SPD;
    ret

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

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

```
    ret

;---------------------------------;
; Routine to initialize the ISR   ;
; for timer 2                     ;
;---------------------------------;
Timer2_Init:
       mov T2CON, #0 ; Stop timer/counter.  Autoreload mode.
       mov TH2, #high(TIMER2_RELOAD)
       mov TL2, #low(TIMER2_RELOAD)
       ; Set the reload value
       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 TR2
    setb ET2  ; Enable timer 2 interrupt
       ret

Timer1_Init:
       ; Configure P2.0, P2.4, P2.5 as open drain outputs
       orl P2M0, #0b_0011_0001
       orl P2M1, #0b_0011_0001
       setb SPEAKERMY_MISO  ; Configured as input
       setb FLASH_CE ; CS=1 for SPI flash memory
       clr SPEAKERMY_SCLK   ; Rest state of SCLK=0
       clr SPEAKER   ; Turn off speaker.

       ; Configure timer 1
       anl    TMOD, #0x0F ; Clear the bits of timer 1 in TMOD
       orl    TMOD, #0x10 ; Set timer 1 in 16-bit timer mode.  Don't change the bits of
timer 0
       mov TH1, #high(TIMER1_RELOAD)
       mov TL1, #low(TIMER1_RELOAD)
       ; Set autoreload value
       mov RH1, #high(TIMER1_RELOAD)
       mov RL1, #low(TIMER1_RELOAD)

       ; Enable the timer and interrupts
    setb ET1  ; Enable timer 1 interrupt
       ; setb TR1 ; Timer 1 is only enabled to play stored sound

       ; Configure the DAC.  The DAC output we are using is P2.3, but P2.2 is also
reserved.
       mov DADI, #0b_1010_0000 ; ACON=1
       mov DADC, #0b_0011_1010 ; Enabled, DAC mode, Left adjusted, CLK/4
       mov DADH, #0x80 ; Middle of scale
       mov DADL, #0
       orl DADC, #0b_0100_0000 ; Start DAC by GO/BSY=1
ret

check_DAC_init:
       mov a, DADC
       jb acc.6, check_DAC_init ; Wait for DAC to finish

       setb EA ; Enable interrupts
ret

;---------------------------------;
; ISR for timer 2                 ;
;---------------------------------;
Timer2_ISR:
       clr TF2  ; Timer 2 doesn't clear TF2 automatically. Do it in ISR

       ; 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:
        ;Do the PWM thing
        clr c
        mov a, pwm_ratio+0
        subb a, Count1ms+0
        mov a, pwm_ratio+1
        subb a, Count1ms+1
        mov PWM_OUTPUT, c

        ; Check if a second has passed
        mov a, Count1ms+0
        cjne a, #low(1000), Timer2_ISR_done
        mov a, Count1ms+1
        cjne a, #high(1000), Timer2_ISR_done

        ; Reset to zero the milli-seconds counter, it is a 16-bit variable
        clr a
        mov Count1ms+0, a
        mov Count1ms+1, a

        ; Increment binary variable 'seconds'
        inc sec
        inc time
        inc fivesec_timer
        lcall Read_temp
Timer2_ISR_done:
        pop psw
        pop acc
        reti

;-------------------------------------;
; ISR for Timer 1.  Used to playback  ;
; the WAV file stored in the SPI      ;
; flash memory.                       ;
;-------------------------------------;
Timer1_ISR:
        ; The registers used in the ISR must be saved in the stack
        push acc
        push psw

        ; Check if the play counter is zero.  If so, stop playing sound.
        mov a, w+0
        orl a, w+1
        orl a, w+2
        jz stop_playing

        ; Decrement play counter 'w'.  In this implementation 'w' is a 24-bit counter.
        mov a, #0xff
        dec w+0
        cjne a, w+0, keep_playing
        dec w+1
        cjne a, w+1, keep_playing
        dec w+2

keep_playing:
        setb SPEAKER
        lcall Send_SPI ; Read the next byte from the SPI Flash...
        mov P0, a ; WARNING: Remove this if not using an external DAC to use the pins of P0
as GPIO
        add a, #0x80
        mov DADH, a ; Output to DAC. DAC output is pin P2.3
        orl DADC, #0b_0100_0000 ; Start DAC by setting GO/BSY=1
        sjmp Timer1_ISR_Done

stop_playing:
        clr TR1 ; Stop timer 1
        setb FLASH_CE  ; Disable SPI Flash
        clr SPEAKER ; Turn off speaker.  Removes hissing noise when not playing sound.
        mov DADH, #0x80 ; middle of range
        orl DADC, #0b_0100_0000 ; Start DAC by setting GO/BSY=1

Timer1_ISR_Done:
        pop psw
        pop acc
```

```
        reti

;--------------------------------;
; Sends AND receives a byte via   ;
; SPI.                            ;
;--------------------------------;
Send_SPI:
        SPIBIT MAC
            ; Send/Receive bit %0
                rlc a
                mov SPEAKERMY_MOSI, c
                setb SPEAKERMY_SCLK
                mov c, SPEAKERMY_MISO
                clr SPEAKERMY_SCLK
                mov acc.0, c
        ENDMAC

        SPIBIT(7)
        SPIBIT(6)
        SPIBIT(5)
        SPIBIT(4)
        SPIBIT(3)
        SPIBIT(2)
        SPIBIT(1)
        SPIBIT(0)

        ret

Read_temp:
clr a
        Read_ADC_Channel(0)
        ;Load_X
        mov x+0, R6
        mov x+1, R7
        mov x+2, #0
        mov x+3, #0
        ; Multiply by 410
        load_Y(410)
        lcall mul32
        ; Divide result by 1023
        load_Y(1023)
        lcall div32
        ; Subtract 273 from result
        load_Y(273)
        lcall sub32
        mov cold, x+0

        ;Thermocouple
        Read_ADC_Channel(1)
        ;Load_X
        mov x+0, R6
        mov x+1, R7
        mov x+2, #0
        mov x+3, #0
        Load_Y(283)
        lcall mul32
        Load_Y(1000)
        lcall div32

        ;put the cold temp into Y
        mov y+0, cold+0
        mov y+1, #0
        mov y+2, #0
        mov y+3, #0
        lcall add32
        ; the addtion will be in x

        lcall hex2bcd
        mov temp+1, bcd+1
        mov temp, bcd
        Send_BCD(bcd+1)
        Send_BCD(bcd+0)


        mov a , #'\r'
    lcall putChar
```

```
    mov a, #'\n'
    lcall putChar
ret


;=====FUCNTIONS RELATED TO SPEKAER==========
rst_stage_flgs:
    setb configure_flag
    setb r2s_sound_flag
    setb s_sound_flag
    setb r2r_sound_flag
    setb reflow_sound_flag
    setb cool_sound_flag
ret

play_sound:
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker.

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set the initial position in memory where to start playing
    ;.
    ;.
    ;.

    ;How many bytes to play?
    ;.
    ;.
    ;.

    setb SPEAKER ; Turn on speaker.
    setb TR1 ; Start playback by enabling Timer 1

ret

;-------------;
; Start Here  ;
;-------------;
LCD_Menu:
        db 'TS  tS  TR  tR  ', 0

Clear_board:
       db '                 ', 0

Stage1:
       db 'Stage1           ', 0

Stage2:
       db 'Stage2 Timer:    ', 0

Stage3:
       db 'Stage3           ', 0

Stage4:
       db 'Stage4 Timer:    ', 0

Stage5:
       db 'Stage5           ', 0

Temp_dis:
       db 'Time:   Temp:    ', 0


MainProgram:
       ;Initialize
    mov SP, #7FH ; Set the stack pointer to the begining of idata
    mov P0M0, #0
    mov P0M1, #0
    setb CE_ADC
    setb EA
       setb configure_flag
    setb r2s_sound_flag
    setb s_sound_flag
```

```
    setb r2r_sound_flag
    setb reflow_sound_flag
    setb cool_sound_flag
    lcall INIT_SPI
    lcall InitSerialPort
    lcall Timer2_Init
    lcall LCD_4bit
    lcall Load_Configuration
    lcall Timer1_Init
    lcall check_DAC_init




forever:
       mov a, #0 ;Set all the variables to 0
    mov state, a ;state after reset is state 0 (waiting state)
    mov sec, a
    mov mf, a
    mov time, a
       mov fivesec_timer, a
    mov mode, a
    mov pwm_ratio+0, #low(200)
       mov pwm_ratio+1, #high(200)
    Set_Cursor(1,1)
    Send_Constant_String(#LCD_Menu)
    Set_Cursor(2,1)
    Send_Constant_String(#Clear_board)

    Set_Cursor(2,1)
    mov a, temp_soak
    lcall SendToLCD

    Set_Cursor(2,5)
    mov a, time_soak
    lcall SendToLCD

    Set_Cursor(2,9)
    mov a, temp_refl
    lcall SendToLCD

    Set_Cursor(2,13)
    mov a, time_refl
    lcall SendToLCD
       ; After initialized
loop:
       clr TR2;stop the clk
       Change_8bit_Variable(SOAKTEMP_BUTTON, temp_soak, loop_a)
       Set_Cursor(2, 1)
       mov a, temp_soak
       lcall SendToLCD
       lcall Save_Configuration
loop_a:
    Change_8bit_Variable(SOAKTIME_BUTTON, time_soak, loop_b)
       Set_Cursor(2, 5)
       mov a, time_soak
       lcall SendToLCD
       lcall Save_Configuration
loop_b:
    Change_8bit_Variable(REFLTEMP_BUTTON, temp_refl, loop_c)
       Set_Cursor(2, 9)
       mov a, temp_refl
       lcall SendToLCD
       lcall Save_Configuration
loop_c:
    Change_8bit_Variable(REFLTIME_BUTTON, time_refl, loop_d)
       Set_Cursor(2, 13)
       mov a, time_refl
       lcall SendToLCD
       lcall Save_Configuration
loop_d:
       clr a
       jnb START_BUTTON, start
       ljmp fsm_machine      ;update the temp variable
```

```
start: ; when the start is pressed
       jnb START_BUTTON, $   ;wait for the button to release
       setb TR2
       mov a, mode
       cjne a, #0, stopfunc  ;jmp if != 0
       cpl a ;set the start button to become a stop button
       mov mode, a
       mov a, state
       add a, #1
       da a
       mov state, a   ;configure to state 1
       lcall loop_d
stopfunc:      ;when the oven is working and the start is pressed
       clr TR2
       mov a, mode
       clr a
       mov mode, a
       mov a, state
       mov a, #0
       da a
       mov state, a
    lcall rst_stage_flgs
       ljmp forever


;--------------;
; fsm machine  ;  the oven turn on/off in the fsm
;--------------;
fsm_machine:
       mov a, state
state0:
       cjne a, #0, state1    ;if != 0 jmp to state 1
       mov pwm_ratio+0, #low(0)
       mov pwm_ratio+1, #high(0)
       ljmp loop      ;jump to Menu and allow user to configure time, temp at each state
state1:;ramp to soak
       cjne a, #1, jumpstate2
    mov pwm_ratio+0, #low(1000)
       mov pwm_ratio+1, #high(1000)
       setb PWM_OUTPUT
    ;;============SAFETY FEATURE abort to stage 0 if hasnt reached 50 degrees by 60
seconds==================
    mov a, time
    clr c
    subb a, #60
    jnc check_safety ;;if time >= 60 go to check_safety
    ljmp continuestate1
jumpstate2:
    ljmp state2
check_safety:
    mov a, temp
    clr c
    subb a, #50
    jnc continuestate1 ;;if temp >= 50 continuestate1
    ;;else change state to 0
    mov a, state
    mov a, #0
    da a
    mov state, a
    mov a, #0
    mov time, a
    ljmp loop_d
    ;;====
continuestate1:
       mov a, temp_soak
       clr c
       subb a, temp
       jnc LCD_Stage1 ; if temp < temp_soak, go to LCD_Stage1-Process
       ;if temp > soak temp, mov to next state
       mov a, state   ;inc state
       add a, #1
       da a
       mov state, a
       mov a, time           ;reset up a timer
       mov a, #0
       mov time, a
```

```
        mov fivesec_timer, a ;;reset fivesecond timer
LCD_Stage1:
        Set_Cursor(1,1)
        Send_Constant_String(#Stage1)
        jb r2s_sound_flag, play_r2s
        ljmp LCD_Process
play_r2s:
    ;;play r2s
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x19
    lcall Send_SPI
    mov a, #0xb8
    lcall Send_SPI
    mov a, #0xc7
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xd0
    mov w+0, #0xf5
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    cpl r2s_sound_flag
    ljmp LCD_Stage1
state2: ;soak period
        cjne a, #2, state3
        mov pwm_ratio+0, #low(200)
        mov pwm_ratio+1, #high(200)
        mov a, time_soak
        clr c
        subb a, time   ; compare with time_soak
        jnc LCD_Stage2 ;if time_soak > timer, jump to display LCD_Stage2/Process
        ;if time_soak < timer
        mov a, state   ;inc state
        add a, #1
        da a
        mov state, a
LCD_Stage2:
        Set_Cursor(1,1)
        Send_Constant_String(#Stage2)
        Set_Cursor(1,14)
        jb s_sound_flag, play_soak
        mov a, time
        lcall SendToLCD
        ljmp LCD_Process
play_soak:
    ;;play soak
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x1a
    lcall Send_SPI
    mov a, #0x89
    lcall Send_SPI
    mov a, #0xbc
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xf8
    mov w+0, #0x2f
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    cpl s_sound_flag
    ljmp LCD_Stage2
state3:;ramp to peak
```

```
        cjne a, #3, state4
        mov pwm_ratio+0, #low(1000)
        mov pwm_ratio+1, #high(1000)
        setb PWM_OUTPUT
        mov a, temp_refl
        clr c
        subb a, temp
        jnc LCD_Stage3 ; if temp < temp_refl, go to LCD_Stage1-Process
        ;if temp > temp_reflo, mov to next state
        mov a, state   ;inc state
        add a, #1
        da a
        mov state, a
        mov a, time           ;reset up a timer
        mov a, #0
        da a
        mov time, a
LCD_Stage3:
        Set_Cursor(1,1)
        Send_Constant_String(#Stage3)
        jb r2r_sound_flag, play_r2r
        ljmp LCD_Process
play_r2r:
    ;;playr2r
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x1b
    lcall Send_SPI
    mov a, #0x81
    lcall Send_SPI
    mov a, #0xeb
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xeb
    mov w+0, #0xda
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    cpl r2r_sound_flag
    ljmp LCD_Stage3
state4:;reflow period
        cjne a, #4, state5
        mov pwm_ratio+0, #low(200)
        mov pwm_ratio+1, #high(200)
        mov a, time_refl
        clr c
        subb a, time
        jnc LCD_Stage4 ;if time_refl > timer, jump to display LCD_Stage2/Process
        ;if time_refl < timer
        mov a, state   ;inc state
        add a, #1
        da a
        mov state, a
LCD_Stage4:
        Set_Cursor(1,1)
        Send_Constant_String(#Stage4)
        jb reflow_sound_flag, play_reflow
        Set_Cursor(1,14)
        mov a, time
        lcall SendToLCD
        ljmp LCD_Process
play_reflow:
    ;;play reflow
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
```

```
    ; Set initial position in memoery where to start playing
    mov a, #0x1c
    lcall Send_SPI
    mov a, #0x6d
    lcall Send_SPI
    mov a, #0xc5
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xe5
    mov w+0, #0x31
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    cpl reflow_sound_flag
    ljmp LCD_Stage4
state5: ;cooling period
       mov pwm_ratio+0, #low(0)
       mov pwm_ratio+1, #high(0)
       Set_Cursor(1,1)
       Send_Constant_String(#Stage5)
       jb cool_sound_flag, play_cool
       mov a, temp
       clr c
       subb a, #22                              ;Edit when thermometer is ready
       jnc LCD_Process ;if 60 C < temp, Wait
       ljmp forever ; basically reset
play_cool:
    ;;play sound
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x1d
    lcall Send_SPI
    mov a, #0x52
    lcall Send_SPI
    mov a, #0xf6
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xd0
    mov w+0, #0xf5
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    cpl cool_sound_flag
    ljmp state5
LCD_Process: ;when the oven is on
       Set_Cursor(2,1)
       Send_Constant_String(#Temp_dis)
       Set_Cursor(2,6)
       mov a, sec
       lcall SendToLCD
       Set_Cursor(2,14)
       mov a, temp
       lcall SendToLCD
       mov a, #5
    cjne a, fivesec_timer, jumpd
    ;;play temperature sound
    lcall play_temperature
    mov a, #0
    mov fivesec_timer, a
    ljmp loop_d
jumpd:
    ljmp  loop_d

play_temperature:
    mov a, temp
    clr c
    subb a, #100
    jnc sound_two ;if temp >= 100 go to stage 2
    ljmp sound_five
```

```
sound_two:
    clr c
    subb a, #100
    jnc play200;;if temp >= 100 + 100
    ljmp play100

play200:
    ;;play 200
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x17
    lcall Send_SPI
    mov a, #0xcb
    lcall Send_SPI
    mov a, #0xdd
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xed
    mov w+0, #0x14
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_three

play100:
    ;;play 100
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x16
    lcall Send_SPI
    mov a, #0xe2
    lcall Send_SPI
    mov a, #0xfa
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xe8
    mov w+0, #0xe3
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_three

sound_three:
    jb TR1, sound_three
    ljmp sound_five

sound_five:
    clr c
    mov a, temp
    mov r3, a    ;save value of a in r3
    mov b, #100
    div ab
    mov r4, #20
    clr c
    mov a, b
    subb a, r4      ;;b shows remainder of a/100
    jnc sound_eight   ;;if b is over 20
    ljmp sound_six ;;if under 20 20
sound_eight: ;;decades twenty and over
    mov a, b
    mov b, #10
    div ab
    mov r4, #2
    mov r3, a
```

```
    mov r5, b
    clr c
    mov a, r3
    subb a, r4
    mov a, r3
    jnc sound_eight30
    ;;play20
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x0f
    lcall Send_SPI
    mov a, #0xe1
    lcall Send_SPI
    mov a, #0xe7
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xd3
    mov w+0, #0x38
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_nine
sound_eight30:
    mov r3, a
    mov r4, #3
    clr c
    subb a, r4
    mov a, r3
    jnc sound_eight40
    ;;play 30
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x10
    lcall Send_SPI
    mov a, #0xb5
    lcall Send_SPI
    mov a, #0x1f
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xd8
    mov w+0, #0x5c
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_nine
sound_eight40:
    mov r3, a
    mov r4, #4
    clr c
    subb a, r4
    mov a, r3
    jnc sound_eight_50
    ;;play 40
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x11
    lcall Send_SPI
    mov a, #0x8d
```

```
    lcall Send_SPI
    mov a, #0x7b
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xd7
    mov w+0, #0x87
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_nine
sound_eight_50:
    mov r3, a
    mov r4, #5
    clr c
    subb a, r4
    mov a, r3
    jnc sound_eight_60
    ;;play 50
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x12
    lcall Send_SPI
    mov a, #0x65
    lcall Send_SPI
    mov a, #0x02
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xe4
    mov w+0, #0x07
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_nine
sound_eight_60:
    mov r3, a
    mov r4, #6
    clr c
    subb a, r4
    mov a, r3
    jnc sound_eight_70
    ;;play 60
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x13
    lcall Send_SPI
    mov a, #0x49
    lcall Send_SPI
    mov a, #0x09
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xe8
    mov w+0, #0xe7
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_nine
sound_eight_70:
    mov r3, a
    mov r4, #7
    clr c
    subb a, r4
    mov a, r3
    jnc sound_eight_80
    ;;play 70
```

```
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x14
    lcall Send_SPI
    mov a, #0x31
    lcall Send_SPI
    mov a, #0xf0
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xe0
    mov w+0, #0xfe
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_nine
sound_eight_80:
    mov r3, a
    mov r4, #8
    clr c
    subb a, r4
    mov a, r3
    jnc sound_eight_90
    ;;play 80
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x15
    lcall Send_SPI
    mov a, #0x12
    lcall Send_SPI
    mov a, #0xee
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xe6
    mov w+0, #0xea
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_nine
sound_eight_90:
    ;;play 90
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x15
    lcall Send_SPI
    mov a, #0xf9
    lcall Send_SPI
    mov a, #0xd8
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xe9
    mov w+0, #0x22
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_nine
sound_nine:
    jb TR1, sound_nine
    ljmp sound_ten
```

```
sound_ten: ;;b in r5
    ;;play sound in b  (1,2,3,...9)
    mov r5, b
    mov r4, #0
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_1
    ljmp sound_seven
sound_ten_1:
    mov r5, b
    mov r4, #1
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_2
    ;;play 1
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x00
    lcall Send_SPI
    mov a, #0x16
    lcall Send_SPI
    mov a, #0xb8
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xd9
    mov w+0, #0x7f
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_2:
    mov r5, b
    mov r4, #2
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_3
    ;;play 2
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x00
    lcall Send_SPI
    mov a, #0xf0
    lcall Send_SPI
    mov a, #0x37
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xc3
    mov w+0, #0x82
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_3:
    mov r5, b
    mov r4, #3
```

```
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3

    jnc sound_ten_4
    ;;play 3
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x01
    lcall Send_SPI
    mov a, #0xb3
    lcall Send_SPI
    mov a, #0xb9
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xcb
    mov w+0, #0x80
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_4:
    mov r5, b
    mov r4, #4
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_5
    ;;play 4
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x02
    lcall Send_SPI
    mov a, #0x7f
    lcall Send_SPI
    mov a, #0x39
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xce
    mov w+0, #0x85
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_5:
    mov r5, b
    mov r4, #5
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_6
    ;;play 5
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
```

```
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x03
    lcall Send_SPI
    mov a, #0x4d
    lcall Send_SPI
    mov a, #0xbe
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xcd
    mov w+0, #0x01
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven

sound_ten_6:
    mov r5, b
    mov r4, #6
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_7
    ;;play 6
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x04
    lcall Send_SPI
    mov a, #0x1a
    lcall Send_SPI
    mov a, #0xbf
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xd6
    mov w+0, #0x24
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_7:
    mov r5, b
    mov r4, #7
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_8
    ;;play 7
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x04
    lcall Send_SPI
    mov a, #0xf0
    lcall Send_SPI
    mov a, #0xe3
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xc3
    mov w+0, #0x44
    setb SPEAKER ; turn on speaker
```

```
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_8:
    mov r5, b
    mov r4, #8
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_9
    ;;play 8
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x05
    lcall Send_SPI
    mov a, #0xb4
    lcall Send_SPI
    mov a, #0x27
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xce
    mov w+0, #0x1e
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_9:
    mov r5, b
    mov r4, #9
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_10
    ;;play 9
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x06
    lcall Send_SPI
    mov a, #0x82
    lcall Send_SPI
    mov a, #0x45
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xd6
    mov w+0, #0x6b
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_10:
    mov r5, b
    mov r4, #10
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_11
    ;;play 10
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
```

```
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x07
    lcall Send_SPI
    mov a, #0x58
    lcall Send_SPI
    mov a, #0xb0
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xe8
    mov w+0, #0xdb
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_11:
    mov r5, b
    mov r4, #11
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_12
    ;;play 11
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x08
    lcall Send_SPI
    mov a, #0x41
    lcall Send_SPI
    mov a, #0x8b
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xce
    mov w+0, #0x7d
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_12:
    mov r5, b
    mov r4, #12
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_13
    ;;play 12
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x09
    lcall Send_SPI
    mov a, #0x10
    lcall Send_SPI
    mov a, #0x08
    lcall Send_SPI

    mov w+2, #0x00
```

```
    mov w+1, #0xd5
    mov w+0, #0xa5
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_13:
    mov r5, b
    mov r4, #13
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_14
    ;;play 13
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x09
    lcall Send_SPI
    mov a, #0xe5
    lcall Send_SPI
    mov a, #0xad
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xd8
    mov w+0, #0x1d
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_14:
    mov r5, b
    mov r4, #14
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_15
    ;;play 14
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x0a
    lcall Send_SPI
    mov a, #0xbd
    lcall Send_SPI
    mov a, #0xca
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xdc
    mov w+0, #0x4e
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_15:
    mov r5, b
    mov r4, #15
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_16
```

```
    ;;play 15
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x0b
    lcall Send_SPI
    mov a, #0x9a
    lcall Send_SPI
    mov a, #0x18
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xdf
    mov w+0, #0x44
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_16:
    mov r5, b
    mov r4, #16
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_17
    ;;play 16
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x0c
    lcall Send_SPI
    mov a, #0x79
    lcall Send_SPI
    mov a, #0x5c
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xe5
    mov w+0, #0xed
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_17:
    mov r5, b
    mov r4, #17
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_18
    ;;play 17
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x0d
    lcall Send_SPI
    mov a, #0x5f
    lcall Send_SPI
    mov a, #0x49
```

```
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xd7
    mov w+0, #0xdd
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_18:
    mov r5, b
    mov r4, #18
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_19
    ;;play 18
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x0e
    lcall Send_SPI
    mov a, #0x37
    lcall Send_SPI
    mov a, #0x26
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xe2
    mov w+0, #0x3a
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_ten_19:
    ;;play 19
    clr TR1 ; Stop Timer 1 ISR from playing previous request
    setb FLASH_CE
    clr SPEAKER ; Turn off speaker

    clr FLASH_CE ; Enable SPI Flash
    mov a, #READ_BYTES
    lcall Send_SPI
    ; Set initial position in memoery where to start playing
    mov a, #0x0f
    lcall Send_SPI
    mov a, #0x19
    lcall Send_SPI
    mov a, #0x60
    lcall Send_SPI

    mov w+2, #0x00
    mov w+1, #0xc8
    mov w+0, #0x87
    setb SPEAKER ; turn on speaker
    setb TR1 ; start playback by enabling timer 1
    ljmp sound_seven
sound_six:
    ;;play sound in b
    mov r5, b
    mov r4, #0
    clr c
    mov r3, a
    mov a, b
    subb a, r4
    mov a, r3
    jnc sound_ten_1_jump
    ljmp sound_seven
sound_ten_1_jump:
    ljmp sound_ten_1
sound_seven:
```

```
    jb TR1, sound_seven
    clr a
    mov r3, a
    mov r4, a
    mov r5, a
    ;;then done
ret

END
```