2017

Computer Engineering Design 2 Phase 2 Report

UNIVERSITY OF KWAZULU-NATAL ™

INYUVESI YAKWAZULU-NATALI

Keshav Jeewanlall

213508238

9/28/2017

Contents

Abstract 3

1. Project 3 – Digital Thermometer 4

1.1. Introduction 4

1.2. Problem Statement and Specifications 4

1.3. Design and Configuration 4

1.3.1. Components used 4

1.3.2. The LM35 Temperature Sensor 4

1.3.3. Analogue to Digital Conversions (ADC) 4

1.3.4. Display 5

1.3.5. Circuit Schematic 5

1.4. Code 6

1.4.1. Code Structure and Implementation 6

1.4.2. Testing 9

1.4.3. Performance Statistics 9

1.5. Duration of Project 3 10

2. Project 4 – Digital Thermometer with Audio Alarm for Temperature Monitoring 10

2.1. Introduction 10

2.2. Problem Statement and Specifications 10

2.3. Design and Configuration 11

2.3.1. Components Used 11

2.3.2. Switch Setup and Debouncing 11

2.3.3. Configuration of Timers 11

2.3.4. Retaining the Threshold Value Even After the System is Powered Down and Up 13

2.3.5. Circuit Schematic 14

2.4. Code 14

2.4.1. Code Structure and Implementation 14

2.4.2. Testing 20

2.4.3. Performance Statistics 20

2.5. Duration of Project 4 20

References 21

Appendix A – Project 3 Code 22

Appendix B – Project 4 Code 24

Abstract

This report explains the details of design, development and implementation of Project 3 and Project 4. Project 3 is a digital thermometer, which receives an analogue signal from an LM35 chip and converts this signal to a digital output. The system is programmed to accurately display the temperature on 7-Segment Displays. Project 4 is an extension of Project 3. This Project has two modes; “Run” and “Set Threshold. “Run”, continuously displays the current temperature. “Set Threshold”, allows for the user to set and adjust the threshold temperature. When the current temperature exceeds the threshold temperature, an audio alarm is activated. The threshold is displayed on 7-Segment displays when it is being set or adjusted.

The source codes, simulations and debugging of these projects was done using MPLAB X IDE and implemented on a PIC16F690 microcontroller.

1. Project 3 – Digital Thermometer

1.1. Introduction

This project is based on a PIC16F690 microcontroller and an LM35 temperature sensor. LM35 is an analogue temperature sensor that converts the surrounding temperature to a proportional analogue voltage. The output from the sensor is connected to one of the ADC channel inputs of the PIC16F690 microcontroller to derive the equivalent temperature value in digital format. The computed temperature is displayed on two 7-segment displays.

1.2. Problem Statement and Specifications

The objective of this project is to build a thermometer using a LM35 temperature sensor that will measure the ambient temperature in the range of 0°C to 99°C and display the ambient temperature on a 2 digit, seven segment display. The temperature measurement must be instantaneous and continuous.

1.3. Design and Configuration

1.3.1. Components used

- PIC16F690 microcontroller
- LM35 temperature sensor
- 2 x Common anode seven-segment displays
- 7 x Current limiting resistors (330Ω)
- 2 x 220Ω resistors

1.3.2. The LM35 Temperature Sensor

The LM35 temperature sensor is rated to operate over a -55 °C to 150°C temperature range. The sensor does not require any external calibration and the output voltage is linearly proportional to the centigrade temperature scale. It changes by 10mV per 1°C. The LM35 temperature sensor has zero offset voltage, which means that the output voltage is 0V, at 0 °C. For a maximum temperature value (150 °C), the maximum output voltage of the sensor would be 150 * 10 mV = 1.5V, therefore, if the supply voltage of 5V is used as the V_{ref} for the Analog to Digital Conversion, the result will be inaccurate as the input voltage will only go up to 1.5V and the power supply voltage variations may affect the ADC output. So it is better to use a stable low voltage above 1.5 as V_{ref}. For this project, the LM35 output was chosen to be connected to pin RA4/AN3.

1.3.3. Analogue to Digital Conversions (ADC)

Analog to Digital Converter (ADC) is a device that converts an analogue quantity (continuous voltage) to discrete digital values, therefore, to obtain and display the temperature measured by the LM35, the Analogue-to-Digital (ADC) module of the PIC16F690 microcontroller needed to be used. The conversion of analogue signal results in corresponding 10-bit digital result. The reference voltage of PIC ADC is software selectable, which can be VDD or the voltage applied by the V_{ref} pin RA1.

As stated above, if the supply voltage of 5V is used as the V_{ref} for the Analog to Digital Conversion, the result will be inaccurate, therefore an external V_{ref} was used in this design in order to produce more accurate results. The value of V_{ref} was chosen as follows:

The 10-bit result obtained from the conversion of analogue signal is stored in two file registers (ADRESH:ADRESL). This means that there are 2^{10} (1024) steps.

$$\text{Step size} = 2.56\text{V} / 1024$$
$$= 2.5\text{mV}$$

Therefore, to get an appropriate step size, V_{ref} should be 2.56V.

Since the LM35 changes by 10mV per 1°C, using this V_{ref} means that 1°C is 4(2.5) mV, which is 1°C is 4 steps. This means that the result obtained from the ADC will be divided by 4 to give the accurate temperature.

Examples:

TEMPERATURE (°C)	**LM35 OUTPUT (mV)**	**ADC RESULT (mV) R = Vin / 2.5mV**	**ADC result / 4**
10°C	100mV	0000101000_2 (40_{10})	00001010_2 (10_{10})
25°C	250mV	0001100100_2 (100_{10})	00011001_2 (25_{10})

In this design, the reference voltage of 2.56V will be produced using two 220Ω resistors and the voltage divider rule applied to pin RA1.

1.3.4. Display

The two SSDs will be connected to PORTC as this is the only port with 8 pins. Pin RC0 to RC6 will be connected to pin a-g on the SSD. Pin RC7 will be used as an enable pin for the tens SSD and pin RA5 will be used as an enable pin for the units SSD. Switching these two SSDs between opposite logic will allow for multiplexing of the SSDs.

1.3.5. Circuit Schematic

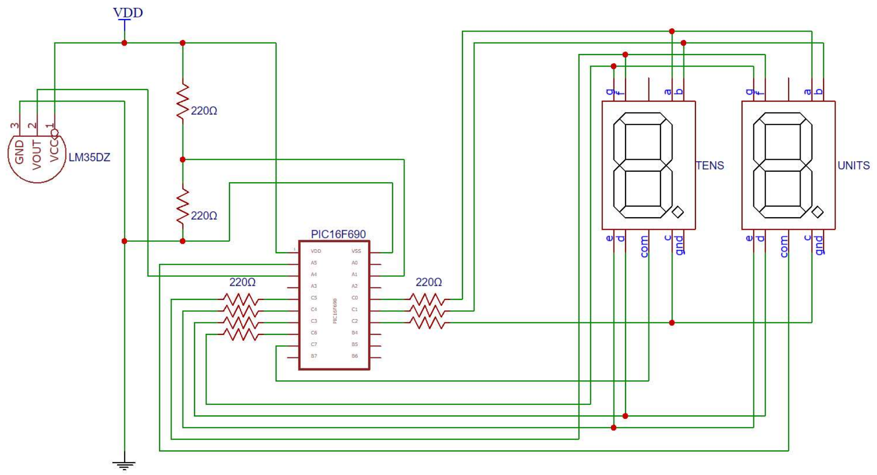

Figure 1: Project 3 Schematic

1.4. Code

1.4.1. Code Structure and Implementation

First the necessary registers and ports need to be setup in order to allow for input, output and to set up the ADC. This is all done in a setup phase which is executed as soon as the microcontroller is powered on. In order to display the correct temperature reading, the ADC needs to be configured correctly. The ADRESH:ADRESL registers can store results in two ways; “Right Justified” which will place the first two LSB in the ADRESL register and the remaining 8 bits in the ADRESH register and “Left Justified” which will place the first two MSB in the ADRESH register and the remaining 8 bits are in the ADRESL register. As stated previously, the ADC result needs to be divided by 4 in order to display the correct temperature.

Using the following example: 0000101000_2 (40_{10})
Dividing by 4 gives us: 00001010_2 (10_{10})

We can see from the above example that by ignoring the first two LSB the result is the value divided by 4. For this reason, the ADCONO register is set up to be “Right Justified”, which will place the first two LSB in the ADRESL register and the remaining 8 bits in the ADRESH register. Hence, the final result will come from the ADRESH register. The ADCONO register is also set up to use an external reference voltage (pin RA1) and to use analogue channel 3 (pin RA4).

This project uses a function, Binary_To_BCD, from an external library which was created in Project 1 to convert the binary result to BCD.

The SSDs are multiplexed by using a short delay between enabling and disabling each SSD to allow the SSD to be on for a longer time thus increasing the brightness.

The code structure is as follows:

3 General Registers are used:

- tens - Stores the 'tens' value to be displayed
- units - Stores the 'units' value to be displayed
- temp - used as temporary variable

Setup:

- In Bank 0, PORTA, PORTB and PORTC are cleared. The ADCON0 register is loaded with the value 01001101, which sets the result to “left justified”, enables an external reference voltage, enable AN3 for analogue input and to enable the ADC.

- In Bank 1, PORTA<1,4> is configured to be used as inputs to read in an external reference voltage and to read the LM35 output. PORTC is cleared and is used to output the result onto the SSDs. PORTA<5> is cleared to be used as an enable pin for the units SSD. ADCON1 is loaded with 00000000 for the ADC to operate at FOSC / 2.

- In Bank 2, ANSEL and ANSELH is cleared to configure all ports as digital I/O. Pins RA1 and RA4 is set to high in order to be used as analogue inputs the LM35 output and the external reference voltage.

Subroutine - Get_Temperature:

- In this subroutine, the ADC is activated to get a reading from the LM35.
- Bit 1 in the ADCON0 register is polled to wait for the ADC to complete the conversion.
- The result from ADRESH is moved to the W register.
- The result is then displayed by calling the Display subroutine.

Subroutine - Display:

- The Convert_to_BCD subroutine is called.
- The Binary_To_BCD converts the binary value to BCD
- The subroutine, SSD_Table is called, which returns the required value in the W register to be displayed.
- A value of 128_{10} (10000000_2) is added to the W register, this enables the tens SSD by setting the MSB (PORT<7>) to 1. This is then moved to PORTC to display the tens. A similar method is used to display the units although by leaving bit 7 in PORTC as 0, this will disable the Tens SSD. By setting PORTA<5> to 1, this will enable the Units SSD so the units can be displayed.
- The SSDs are multiplexed by using a short delay subroutine called Multiplexing_Delay between enabling and disabling each SSD.

Subroutine - Convert_to_BCD:

- The Convert_to_BCD subroutine calls the external library subroutine Binary_To_BCD.
- The Binary_To_BCD converts the binary value to BCD.
- The Convert_to_BCD subroutine then takes this value and separates it into tens and units and then stores the tens and units values in their respective registers.

Subroutine – SSD_Table:

- This subroutine adds the value that is in the W register to the Program Counter.
- PC will skip to whichever value is needed to be displayed and returns the value in the WREG.

Subroutine – Multiplexing_Delay:

- This subroutine is a delay routine that lasts approximately 0.125ms at an FOSC of 4MHz
- A value of 250 is loaded into temp register
- A loop called Multiplexing_Delay_Loop is then used to decrease temp by 1 each time.
- The loop is then exited once temp is zero.

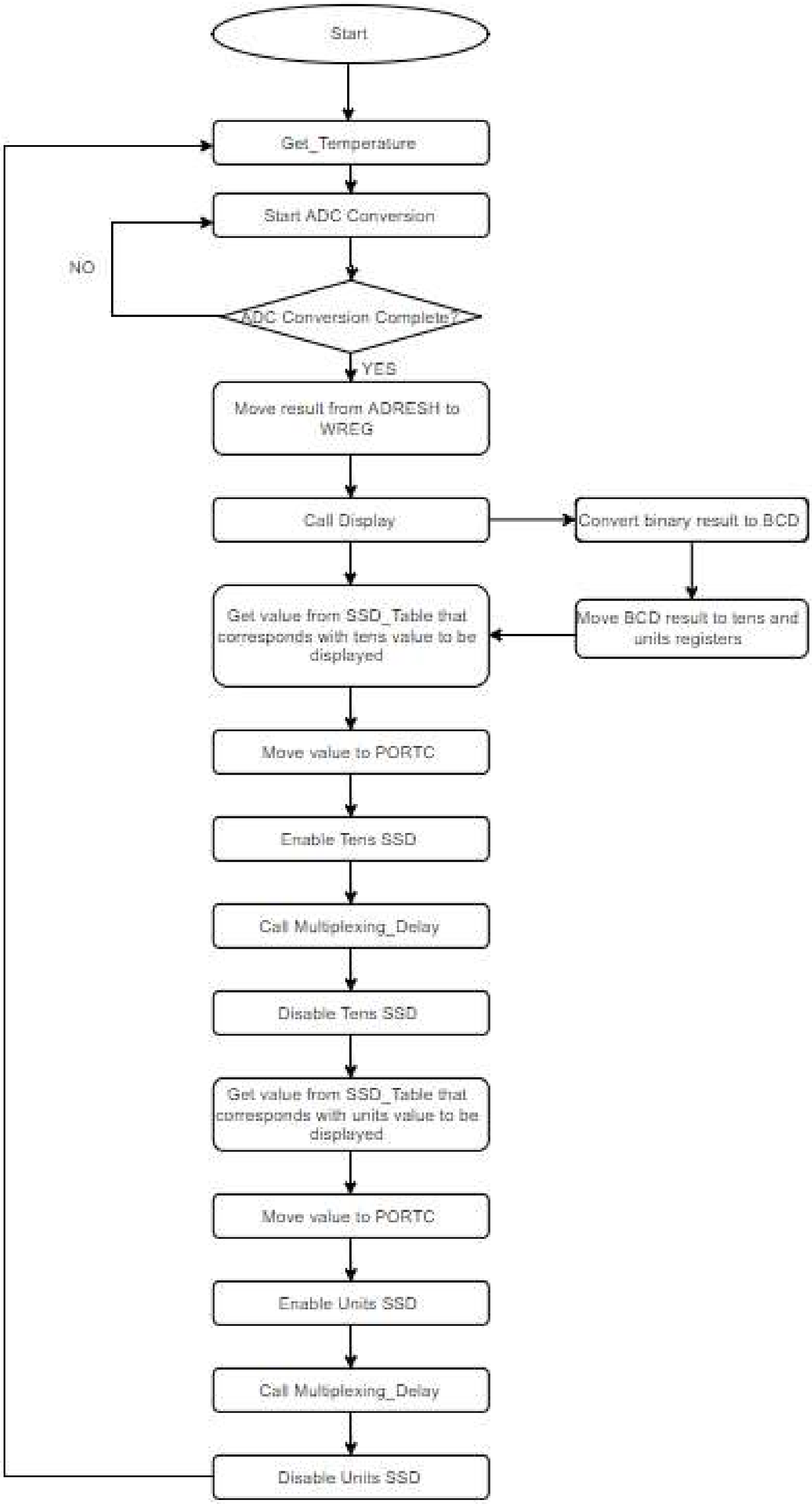
Start
Get_Temperature
Start ADC Conversion
NO
ADC Conversion Complete?
YES
Move result from ADRESH to WREG
Call Display
Convert binary result to BCD
Move BCD result to tens and units registers
Get value from SSD_Table that corresponds with tens value to be displayed
Move value to PORTC
Enable Tens SSD
Call Multiplexing_Delay
Disable Tens SSD
Get value from SSD_Table that corresponds with units value to be displayed
Move value to PORTC
Enable Units SSD
Call Multiplexing_Delay
Disable Units SSD

Figure 2: Project 3 Flowchart

1.4.2. Testing

All debugging and testing was done on a PIC16F690 microcontroller with all the components connected to it. Debugging using the MPLAB X simulator was not the most efficient way to debug this project as hardware interaction was involved. The code was programmed to the microcontroller using a PicKit3. All system specifications were met.

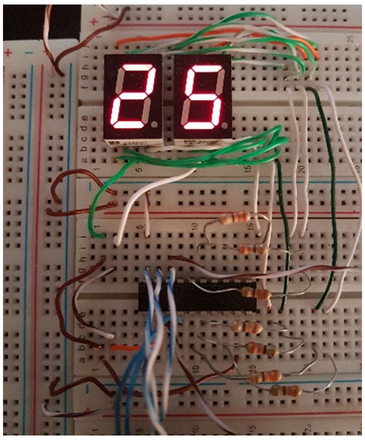

Figure 3: Circuit temperature reading of 25°C

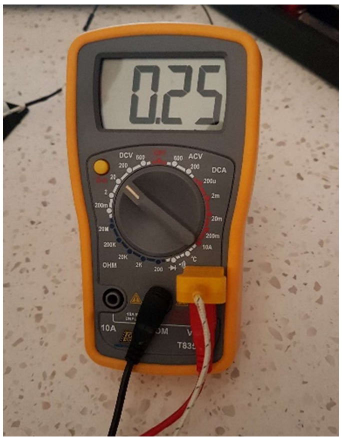

Figure 4: LM35 output voltage of 25V at 25°C

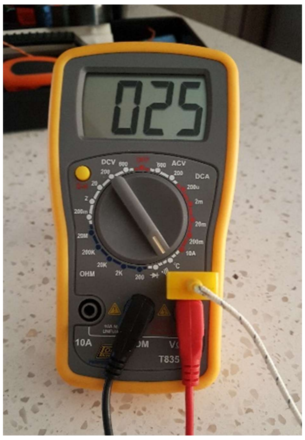

Figure 5: Multi-meter showing temperature is at 25°C

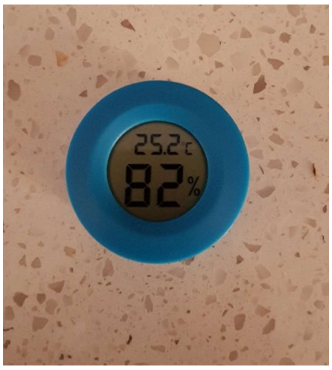

Figure 6: Thermometer showing temperature is at 25°C

1.4.3. Performance Statistics

Lines of code: 70 lines
Execution cycles: 650 (Including execution cycles from Binary_To_BCD library subroutine)

See Appendix A for the code.

1.5. Duration of Project 3

The amount of time taken to complete this project was approximately 8 to 10 hours spread over the course of two weeks. Research on how the ADC of the PIC16F690 operates needed to be done. Debugging of code consumed most of the time taken.

2. Project 4 – Digital Thermometer with Audio Alarm for Temperature Monitoring

2.1. Introduction

Project 4 is an extension of project 3, which adds an audio alarm and the ability to set and adjust a threshold temperature. It also requires the usage of the PIC16F690 timers.

2.2. Problem Statement and Specifications

The objective of this project is to build a digital thermometer which measures the ambient temperature, allows the user to set and adjust a threshold value and sounds an alarm if this threshold value is exceeded. This system uses a LM35 temperature sensor that will measure the ambient temperature in the range of 0°C to 99°C and display the ambient temperature on a 2 digit, seven segment display. The temperature measurement must be instantaneous and continuous.

The system must have two multi-functional buttons to select the system functional mode and adjust the threshold temperature. The buttons have to be debounced in software.
The system will have two modes:

- “run” which continuously displays the current temperature.
- “Set threshold” which allows the user to set the threshold temperature level.

In order to set the threshold, the user should press and hold either of the buttons. After 2 seconds the device will enter the “Set threshold” mode. If the switch is released before 2 seconds, the 2 second cycle restarts over the next time the switch is pressed. The “Set threshold” mode will be indicated by flashing the display on and off. The value that is displayed will be the current threshold temperature set point.

In the “Set threshold” mode, the buttons are used to adjust the threshold temperature level. One button is used to decrement the threshold temperature by 1°C and the other button to increment the threshold temperature by 1°C. If neither button is pressed for 3 seconds, the device will return to the normal “run” mode and display the current temperature and the display will stop flashing. The set threshold temperature value should be retained and later displayed in the appropriate mode even after the system is powered down and up.

If the threshold value is exceeded, an audio alarm will sound at a tone of 440Hz, in 1 second intervals and continue to sound in 1 second intervals as long as the temperature is above the threshold limit. The display will continue to display the current temperature while the alarm sounds. The 440Hz frequency for the audio alert will be generated within the microcontroller.

2.3. Design and Configuration

N.B. Temperature sensing, displaying and the ADC configuration for this project is the same as Project 3

2.3.1. Components Used

- PIC16F690 microcontroller
- LM35 temperature sensor
- 2 x Common anode seven-segment displays
- 7 x Current limiting resistors (330Ω)
- 2 x 220Ω resistors
- 1x Piezo buzzer
- 2x SPST push buttons

2.3.2. Switch Setup and Debouncing

The switches are set up as pull up switches, this sends a low signal to the microcontroller when the switch is pressed. The decrement button is connected pin RA0 and the increment is connected pin RA2. Both pins are polled to wait for a low signal. Since the system requires each button press to have a delay after the press to check if the user is entering the set threshold mode, this delay can also be used for switch debouncing. These delay loops produce enough time for the switch to debounce and then commence with the rest of the code.

2.3.3. Configuration of Timers

The PIC16F690 has 3 timers, which is, Timer0, Timer2 which are 8-bit timers and Timer1 which is a 16-bit timer. For this system, Timer0 is used to control the active and non-active states of the buzzer. Timer1 is used to for the 2 seconds and 3 seconds delay and Timer2 is used for generating the 440Hz buzzer tone. A slower internal oscillator speed of 500kHz is used to allow for the timers to wait for these periods.

Setup of the 2 second wait to enter the “Set Threshold” Mode

To enter the “Set Threshold” mode, the user must hold down either the decrement button or the increment button for 2 seconds. This is done by calling a 1 second delay after each button press. If a button is pressed, the system will delay for 1 second then check if the button is still pressed. If it is, the system will delay for another second, therefore waiting 2 seconds. After the 2 seconds, the system immediately checks if the button is still pressed, if so, the system will enter the “Set Threshold” mode.

Timer1 is used to generate this 1 second delay as it has a 16-bit counter and is therefore capable of holding much larger values thus spanning over longer periods of time when compared to the 8-bit timers.

Timer1 clock speed operates at $F_{OSC}/4$. To obtain the most accurate value to be loaded into the TMR1H:TMR1L registers, Timer1 prescaler is set as 1:8 and the value to be loaded was calculated as follows:

$$No.of\ steps = \frac{Time}{\frac{Prescaler\ x\ 4}{500x10^3}} = \frac{1}{\frac{8x4}{500x10^3}} = 15625$$

To start the 1 second delay, TMR1H:TMR1L = ((2^{16} - 1) – 15625) = 65535 – 15625 = 49911.

Therefore, TMR1L needs to be loaded with a hex value F7 and TMR1H needs to be loaded with a hex value C2.

Setup of the 3 second wait whilst in the “Set Threshold” mode

Whilst in the “Set Threshold” mode, the user is given 3 seconds to set a threshold value. If no button is pressed within this 3 second period, the system will exit the “Set Threshold” mode and enter the normal “Run” mode. However, if either of the two buttons is pressed whilst in the “Set Threshold” mode, the system will start the 3 second timer again. Timer1 is used to generate this 3 second delay. To obtain the most accurate value to be loaded into the TMR1H:TMR1L registers, Timer1 prescaler is set as 1:8 and the value to be loaded was calculated as follows:

$$No.of\ steps = \frac{Time}{\frac{Prescaler\ x\ 4}{500x10^3}} = \frac{3}{\frac{8x4}{500x10^3}} = 46875$$

TMR1H:TMR1L = ((2^{16} - 1) – 46875) = 65535 – 46875 = 18660.

Therefore, TMR1L needs to be loaded with a hex value E4 and TMR1H needs to be loaded with a hex value 48.

Setup of Timer2 to generate the 440Hz tone of the buzzer

In this system, the buzzer is connected to pin RB4. Toggling the state of this pin in 1 second intervals at a tone of 440Hz will produce the required sounding of the alarm.

Timer2 clock speed operates at $F_{OSC}/4$. To obtain the most accurate value to be loaded into the TMR2 register, Timer2 prescaler is set as 1:4 and the value to be loaded was calculated as follows:

$$No.of\ steps = \frac{Time}{\frac{Prescaler\ x\ 4}{500x10^3}} = \frac{\frac{1}{440}}{\frac{4x4}{500x10^3}} = 72$$

To generate the 440Hz tone, TMR2 = ((2^8 - 1) – 72) = 255 – 72 = 183.

Therefore, TMR2 needs to be loaded with a hex value B7

Setup of Timer0 to generate the 2Hz pulse

The buzzer needs to be switched on for 1 second and off for the other second. To achieve this, a general register called *buzzer_state* is incremented after every second. By doing this, bit 0 in this register is toggled after every second. Therefore, when bit 0 = 1, the buzzer is turned off and when bit 0 = 1, the buzzer will be turned on.

A timer needs to be continuously running in order to count every second. Timer0 is used for this operation as Timer1 and Timer2 is used for other purposes. Since Timer0 is an 8-bit timer, it will not produce a 1 second delay. This was solved by setting the timer to run for 500ms and when overflow has occurred twice, it means 1 second has occurred. A *count* register is used to keep track of how many times the overflow has occurred. When *count* = 2, it means 1 second has occurred and therefore the buzzer*_state register will increment*.

To obtain the most accurate value to be loaded into the TMR0 register, Timer0 prescaler is set as 1:256 and the value to be loaded was calculated as follows:

$$No.of\ steps = \frac{Time}{\frac{Prescaler\ x\ 4}{500x10^{3}}} = \frac{500x10^{-3}}{\frac{256x4}{500x10^{3}}} = 244$$

TMR0 = 255 – 244 = 11 for 500ms

2.3.4. Retaining the Threshold Value Even After the System is Powered Down and Up

In order to retain the threshold value even after the system is powered down and up, the PIC16F690 EEPROM is used. At the end of Set_Threshold subroutine, the threshold value is stored in the EEPROM register. The EEPROM stores the data from runtime when the system is powered down. This allows the threshold value be retained when system is powered down. When system starts up, the threshold value is read from the EEPROM registers and is stored in the threshold register and displayed when the user enters the “Set Threshold” mode.

2.3.5. Circuit Schematic

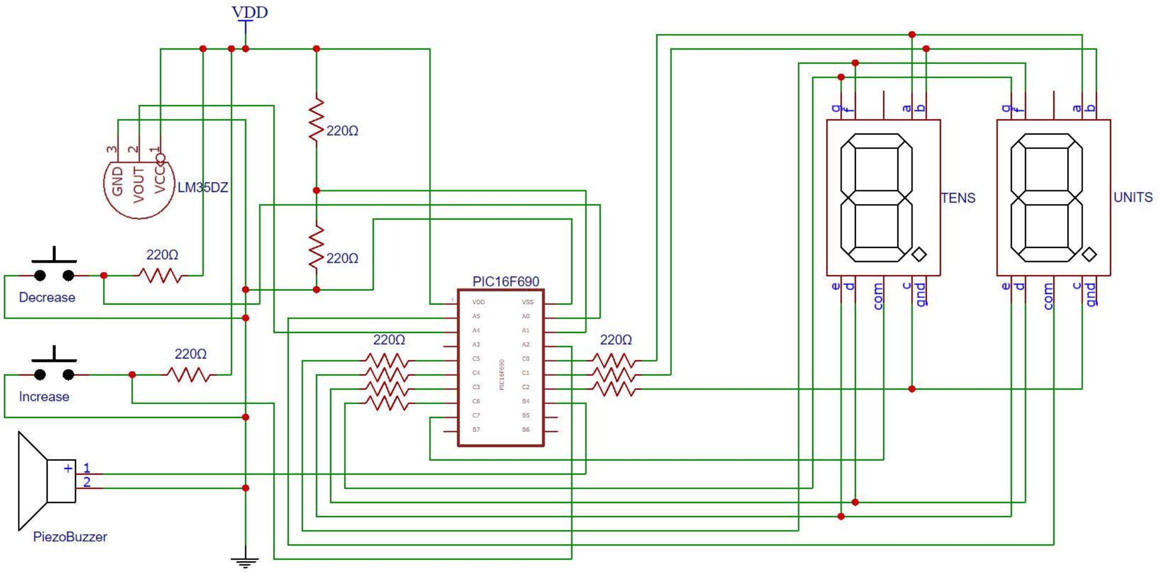

Figure 7: Project 4 Schematic

2.4. Code

2.4.1. Code Structure and Implementation

The code structure is as follows:

9 general file registers were used:

- tens - Stores the 'tens' value to be displayed
- units - Stores the 'units' value to be displayed
- temperature - Stores the current temperature
- threshold - Stores the threshold temperature during runtime
- buzzer_state - Used to control the state of the buzzer (on or off)
- count - Used to control Timer0
- eeprom_address - Stores the EEPROM address for Threshold value
- temp - Used for the delay subroutines
- temp2 - Used for the delay subroutines

Setup:

- In Bank 0, the same setup as Project 3 is used with the addition of loading the value of 00110000 to T1CON register which sets the Timer pre-scalar to 1:8. Also Timer2 interrupt flag (TMR2IF) is cleared and T2CON register is cleared

- In Bank 1, the clock speed is set to 500kHz. The necessary pins are set as inputs and outputs. OPTION_REG is loaded with value 00000111 to set Timer0 pre-scalar to 1:256

- In Bank 2, the setup is the same as Project 3. The Threshold value is taken from EEPROM.

- Going back to Bank 0, the threshold value is loaded into the threshold register. All other general registers are cleared. Timer0 is configured for 500ms and then starts.

Subroutine – Trigger_Alarm:

- Checks if bit 0 in *the buzzer_state register* is 0. If so, enable the buzzer
- Call Set_440Hz subroutine to sound the buzzer at a 440Hz tone
- Disable buzzer.

Subroutine - Get_Temperature:

- In this subroutine, the ADC is activated to get a reading from the LM35.
- Bit 1 in the ADCON0 register is polled to wait for the ADC to complete the conversion.
- The result from ADRESH is moved to the W register.
- The result is then displayed by calling the Display subroutine.

Subroutine - Display:

- The Convert_to_BCD subroutine is called.
- The Binary_To_BCD converts the binary value to BCD
- The subroutine, SSD_Table is called, which returns the required value in the W register to be displayed.
- A value of 128_{10} (10000000_2) is added to the W register, this enables the tens SSD by setting the MSB (PORT<7>) to 1. This is then moved to PORTC to display the tens. A similar method is used to display the units although by leaving bit 7 in PORTC as 0, this will disable the Tens SSD. By setting PORTA<5> to 1, this will enable the Units SSD so the units can be displayed.
- The SSDs are multiplexed by using a short delay subroutine called Multiplexing_Delay between enabling and disabling each SSD.

Subroutine - Convert_to_BCD:

- The Convert_to_BCD subroutine calls the external library subroutine Binary_To_BCD.
- The Binary_To_BCD converts the binary value to BCD.
- The Convert_to_BCD subroutine then takes this value and separates it into tens and units and then stores the tens and units values in their respective registers.

Subroutine – SSD_Table:

- This subroutine adds the value that is in the W register to the Program Counter.
- PC will skip to whichever value is needed to be displayed and returns the value in the WREG.

Subroutine – Set_Threshold:

- Call Three_Second_Delay subroutine to start Timer1 to run for 3 seconds.
- Displays Threshold value.
- Call Flash_SSD subroutine in order to flash the SSDs off and on.
- Checks if Increment button pressed, if so, call Increase_Threshold.
- Checks if Decrement button pressed, if so, call Decrease_Threshold.

- If 3 seconds has occurred, stop Timer1 and exit the routine.
- Call Write_To_EEPROM to store the threshold value in EEPROM.

Subroutine – Increase_Threshold:

- Increase threshold value by 1.
- Reset threshold value to 0 if threshold value exceeds 99.
- Waits till button gets released.
- Displays Threshold value while waiting.
- Call Three_Second_Delay to restart Timer1 to run for 3 seconds. Routine then exits.

Subroutine – Decrease_Threshold:

- Decrease threshold value by 1.
- Reset threshold value to 99 if threshold value goes below 0.
- Waits till button gets released.
- Displays Threshold value while waiting.
- Call Three_Second_Delay to restart Timer1 to run for 3 seconds. Routine then exits.

Subroutine – Write_To_EEPROM:

- Obtain data memory address to write
- Loads the threshold value needed to be stored in EEPROM from threshold register to WREG
- Store Value in EEPROM

Subroutine – Multiplexing_Delay:

- A HEX value of 30 is loaded into temp register
- A loop called Multiplexing_Delay_Loop is then used to decrease temp by 1 each time.
- The loop is then exited once temp is zero.

Subroutine – Flash_SSD:

- This delay is to flash the SSDs off and on. It runs for approximately 17ms.

Subroutine – One_Second_Delay:

- Loads 449910_{10} ($C2F7_{16}$) to TMR1H:TMR1L. This configures Timer1 to run for 1 second.
- Timer1 is then started.
- While waiting, Get_Temperature is called to display the temperature.
- Overflow flag polled. When set, Timer1 will be stopped and subroutine will exit.

Subroutine – Three_Second_Delay:

- Stops Timer1.
- Loads 18660_{10} ($48E4_{16}$) to TMR1H:TMR1L. This configures Timer1 to run for 3 seconds.
- Timer1 then restarted.

Subroutine – Timer0_Control

- Gets called after every 500ms.
- Increment *count*.
- If *count* = 2, it means 1 second has occurred.

- Increment *buzzer_state*
- Reset *count* to zero.
- Reset Timer0.

Subroutine – Set_440Hz

- Clear Timer2 interrupt flag (TMR2IF)
- Load TMR2 with Hex value B9 to generate a 440Hz tone
- Configure Timer2 prescaler to 1:4 and start Timer2
- Stop Timer2

Subroutine – Main_loop:

- Call Get_Temperature.
- Checks if Decrement button is pressed, if so, wait for 1 second.
- Checks if Decrement button is still pressed, if so, wait for another second.
- Checks if Decrement button is pressed, if so, call Set_Threshold
- Checks if Increment button is pressed, if so, wait for 1 second.
- Checks if Increment button is still pressed, if so, wait for another second.
- Checks if Increment button is pressed, if so, call Set_Threshold
- Check if 500ms occurred by checking if Timer0 overflow flag is set, if so, call Timer0_Control.
- Checks if Temperature is greater than Threshold, if so, call Trigger_Alarm. This is done by subtracting the threshold value from the temperature value. If a negative result occurs, bit 7 will be set, this means that the threshold is greater than the temperature and therefore should not sound the alarm.

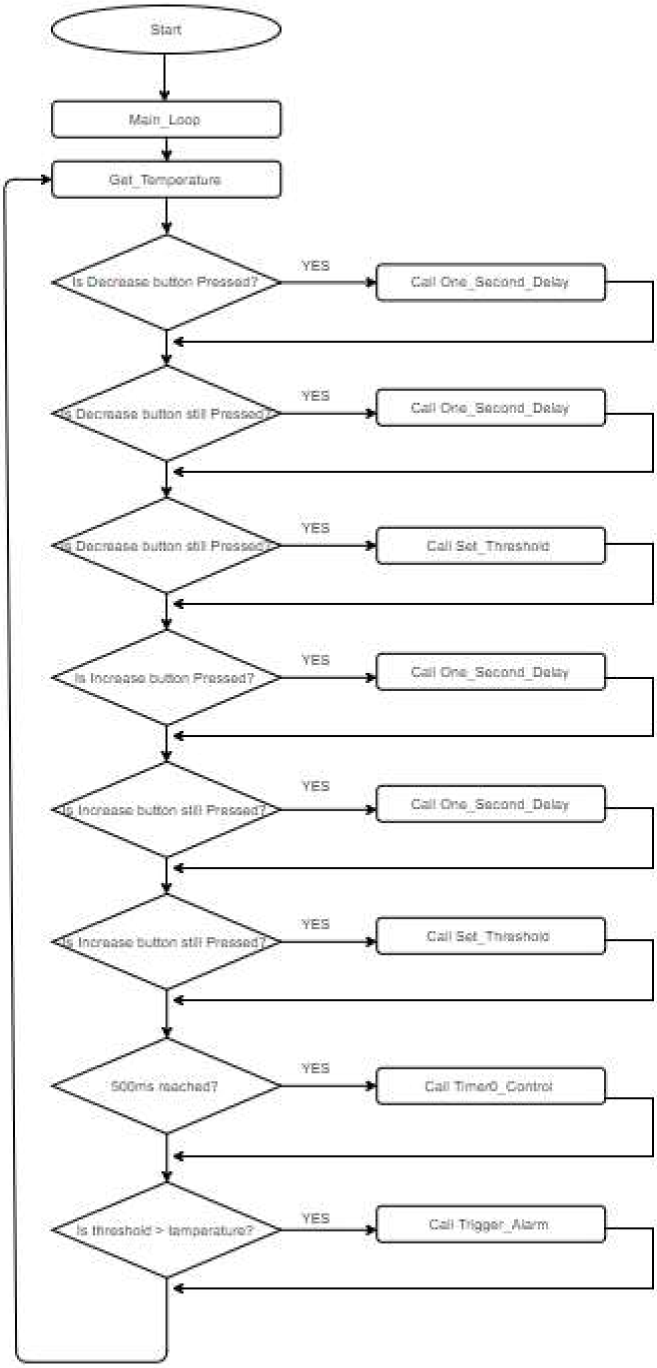

Figure 8: : Main_Loop flowchart of Project 4

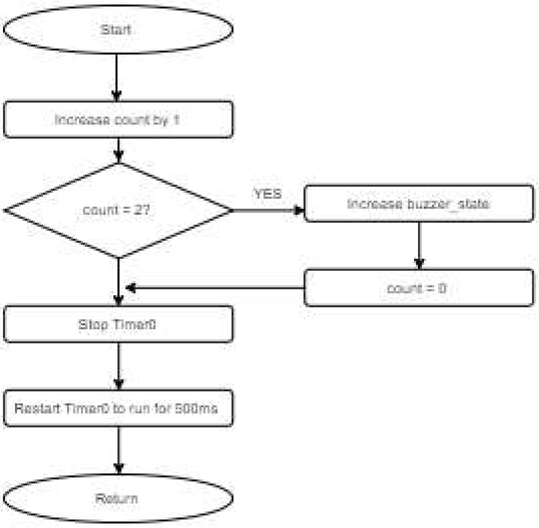

Figure 9: "Timer0_Control" subroutine flowchart

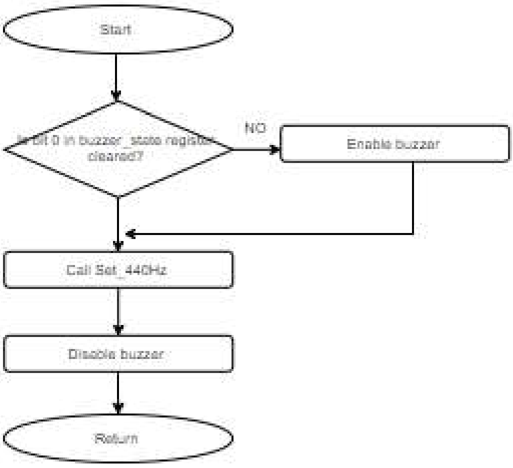

Figure 10: Trigger_Alarm" subroutine flowchart

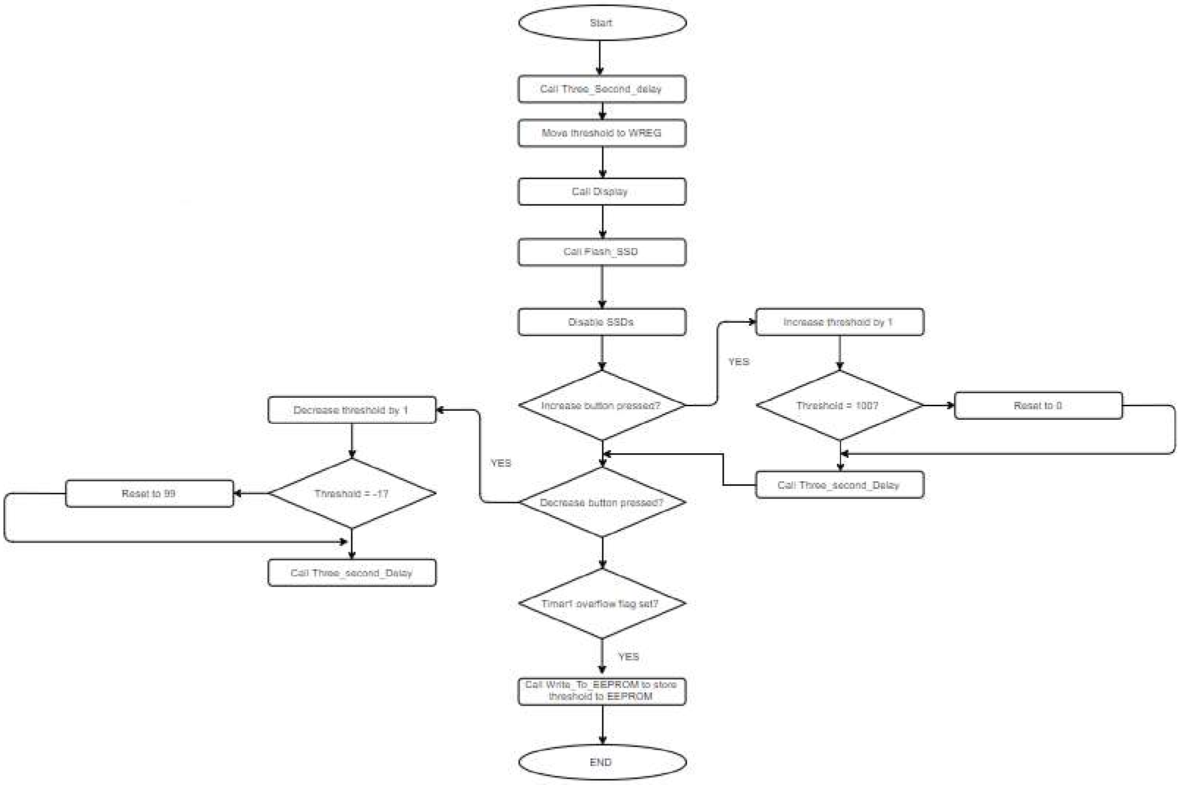

Figure 11: "Set_Threshold" subroutine flowchart

2.4.2. Testing

All debugging and testing was done on a PIC16F690 microcontroller with all the components connected to it. Debugging using the MPLAB X simulator was not the most efficient way to debug this project as hardware interaction was involved. The code was programmed to the microcontroller using a PicKit3. During "Set Threshold" mode, the SSDs flash off and on at a high frequency but are still visible enough for the human eye. The 2 seconds and 3 seconds delays were tested manually using a stopwatch and the system performed as expected.

2.4.3. Performance Statistics

- 213 lines of code were used in this project (excludes the Binary_to_BCD library subroutine, labels and comments).
- The system uses 9 general file registers, 1 register from EEPROM and the W register.

See Appendix B for the code.

2.5. Duration of Project 4

The amount of time taken to complete this project was approximately 8 to 10 hours spread over the course of two weeks. Debugging of code consumed most of the time taken.

References

[1] George, L., George, L. and George, L. (2017). *Digital Thermometer using PIC Microcontroller and LM35*. [online] electroSome. Available at: https://electrosome.com/thermometer-pic-microcontroller-lm35/ [Accessed 18 Sep. 2017].

[2] Embedded Lab. (2017). *A Digital temperature meter using an LM35 temperature sensor - Embedded Lab*. [online] Available at: http://embedded-lab.com/blog/digital-temperature-meter-using-an-lm35-temperature-sensor/ [Accessed 18 Sep. 2017].

Appendix A – Project 3 Code

```
;*******************************************************************************

;    Student Name         : Keshav Jeewanlall
;    Student Number       : 213508238
;    Date                 : 05 / 09 / 2017
;    Description          : A digital thermometer using a LM35 chip
;                             and two SSDs.
;
;   This code processes an analogue input from a LM35 chip and displays the
;   result on two multiplexed SSDs.
;*******************************************************************************

    List p=16f690
#include <p16F690.inc>
errorlevel  -302
    __CONFIG    _CP_OFF & _CPD_OFF & _BOR_OFF & _MCLRE_ON & _WDT_OFF & _PWRTE_ON &
_INTRC_OSC_NOCLKOUT & _FCMEN_OFF & _IESO_OFF

;************************VARIABLE DEFINITIONS & VECTORS*************************
 UDATA
tens             RES 1     ;stores tens digit
units            RES 1     ;stores units digit
temp              RES 1

EXTERN Binary_To_BCD       ;library to convert binary to BCD

RESET ORG 0x00             ;Reset vector, PIC starts here on power up and reset
GOTO Setup

 ;*****************************SETUP OF PIC16F690*******************************
Setup
                              ;Use Bank0
    BCF STATUS,5
    BCF STATUS,6

     CLRF PORTA                       ;Initialise Port A
     CLRF PORTB                       ;Initialise Port B
     CLRF PORTC                       ;Initialise Port C

                              ;Load 01001101 into ADCONO to Adjust left,
                              ;use external Vref, enable AN3 and enable ADC
     MOVLW 0x4D
     MOVWF ADCON0
                              ;Use Bank 1
     BSF STATUS,5

     CLRF TRISC                       ;Set PORTC as output
     BSF TRISA,4              ;Set RA4 as input for the LM35
     BSF TRISA,1              ;Set RA1 as input. Used to read Vref
     BCF TRISA,5              ;RA5 used to control the Units SSD
     CLRF ADCON1              ;Conversion cloack set at FOSC/2

                              ;Use Bank 2
    BCF STATUS,5
    BSF STATUS,6

     CLRF ANSEL                       ;Initialize all ports as digital I/O
     CLRF ANSELH
     BSF ANSEL,3              ;Set RA4/AN3 to be analog input
     BSF ANSEL,1              ;Set Vref to be analog input

                              ;Set back to Bank 0
     BCF STATUS,6

     GOTO Get_Temperature

     CODE
 ;********************CODE TO GET TEMPERATURE FROM LM35***********************

Get_Temperature
                           ;Conversion is initiated by setting the GO/DONE
                           ;bit ADCON0<1>

    BSF ADCON0,1           ;Start ADC conversion

Get_Temperature_Loop
    BTFSC ADCON0,1         ;Checks if conversion done, if so, exit loop
    GOTO Get_Temperature_Loop
    MOVFW ADRESH           ;Move conversion result to WREG
    Call Display
    GOTO Get_Temperature

;************************CODE FOR DISPLAYING ON SSDs****************************

Display
    CALL Convert_to_BCD           ;subroutine to convert count to BCD
    CALL SSD_Table         ;gets code for displaying the number (Tens)
    ADDLW 0x80             ;setting the MSB (Bit 7) will enable the Tens SSD
    MOVWF PORTC                   ;display Tens value
    CALL Multiplexing_Delay ;delay for multiplexing SSDs
    BCF PORTC,7                   ;Disable tens SSD
    MOVFW units
    CALL SSD_Table         ;gets code for displaying the number (Units)
    BSF PORTA,5                   ;Set RA5 to enable units SSD
    MOVWF PORTC                   ;displays units value
    CALL Multiplexing_Delay
    BCF PORTA,5                   ;Disable the Units SSD
    RETURN

Convert_to_BCD           ;converts count to BCD
    Call Binary_To_BCD   ;uses library subroutine to get BCD value of number
    MOVWF tens
    ANDLW 0x0F           ;b'00001111 , clears upper nibble of BCD number
    MOVWF units                 ;stores the value as the units
    SWAPF tens,1         ;swaps the nibbles of the BCD number
    MOVFW tens
    ANDLW 0x0F           ;b'00001111, clears the high nibble to get tens value
    MOVWF tens           ;stores value in tens register
    RETURN

;This subroutine adds the value that is in the W register to the Program
;Counter. PC will skip to whichever value is needed to be displayed and returns
;the value in the WREG.


SSD_Table
                         ;These HEX values are required because common anode SSDs
                         ;are being used
    ADDWF PCL,1
    RETLW 0x40           ;displays number 0 on SSD
    RETLW 0x79           ;displays number 1 on SSD
    RETLW 0x24           ;displays number 2 on SSD
    RETLW 0x30           ;displays number 3 on SSD
    RETLW 0x19           ;displays number 4 on SSD
    RETLW 0x12           ;displays number 5 on SSD
    RETLW 0x02           ;displays number 6 on SSD
    RETLW 0x78           ;displays number 7 on SSD
    RETLW 0x00           ;displays number 8 on SSD
    RETLW 0x10           ;displays number 9 on SSD

Multiplexing_Delay      ;A delay subroutine. Runs for approx. 0.125ms
    MOVLW 0xFA          ;Loads a value of 250 and stores it in temp
    MOVWF temp
Multiplexing_Delay_Loop
    DECFSZ temp,1       ;When temp = 0, exit loop
    GOTO Multiplexing_Delay_Loop
    RETURN

    END
```

Appendix B – Project 4 Code

```
;*******************************************************************************
;  Student Name            : Keshav Jeewanlall
;  Student Number    : 213508238
;  Date             : 26 / 09 / 2017
;
;  Description:
;
;  This code processes an analogue input from a LM35 chip and displays the
;  result on two multiplexed SSDs. There are two push buttons, one for
;  incrementing the threshold and the other for decrementing it. Holding either
;  button down for 2s will enter the "Set Threshold" mode. Set Threshold mode
;  will exit after 3s if no button is pressed. An alarm of 440Hz is sounded
;  when the temperature exceeds the threshold value. This alarm sounds on and
;  and off in 1s intervals.

;*******************************************************************************

    List p=16f690
    #include <p16F690.inc>

    errorlevel  -302             ;Configuration bits setup
    __CONFIG   _CP_OFF & _CPD_OFF & _BOR_OFF & _MCLRE_ON & _WDT_OFF & _PWRTE_OFF &
_INTRC_OSC_NOCLKOUT & _FCMEN_OFF & _IESO_OFF

;************************VARIABLE DEFINITIONS & VECTORS*************************
 GPR_VAR UDATA
tens           RES 1          ;;stores tens digit
units          RES 1          ;stores units digit
temperature    RES 1          ;store current temperature value
threshold      RES 1          ;stores threshold value
buzzer_state   RES 1          ;used to control duty cycle of the buzzer
count          RES 1          ;used to control Timer0
eeprom_address RES 1          ;stores the EEPROM address for threshold value
temp            RES 1
temp2           RES 1

 EXTERN Binary_To_BCD      ;library to convert binary to BCD

RESET ORG 0x00
    GOTO Setup

;*****************************SETUP OF PIC16F690*******************************
Setup

                          ;Use Bank 0
    BCF STATUS,5
    BCF STATUS,6

    CLRF PORTA            ;Initialise Port A
    CLRF PORTB            ;Initialise Port B
    CLRF PORTC            ;Initialise Port C

    MOVLW b'01001101'
    MOVWF ADCON0          ;Load 01001101 into ADCONO to Adjust left,
                          ;use external Vref, enable AN3 and enable ADC

    BSF T1CON,4
    BSF T1CON,5                   ;Set T1CKPS1 & T1CKPS0 bits for 1:8 Prescaler

    BCF PIR1,1            ;clear TMR2IF
    CLRF T2CON


    MOVLW 0x02
    MOVWF eeprom_address    ;Use address 0x02 to store Threshold value in EEPROM

                          ;Use Bank 1
    BSF STATUS,5

    BCF OSCCON,6
    BSF OSCCON,5
    BSF OSCCON,4           ;Set IRCF1 & IRCF0 to select 500kHz FOSC

    CLRF TRISC             ;Set PORTC as output

    MOVLW b'00010111'
    MOVWF TRISA                    ;set RA1 for as input for decrement button
                           ;Set RA1 as input. Used to read Vref
                           ;Set RA2 as input for increment button
                           ;Set RA4 as input for the LM35

    BCF TRISB,4                    ;RB4 used to sound alarm

    CLRF ADCON1                    ;Conversion clock set at FOSC/2

    MOVLW b'00000111'
    MOVWF OPTION_REG       ;Timer0 used in Timer mode with Prescaler 1:256

                           ;Use Bank 2
    BCF STATUS,5
    BSF STATUS,6

    CLRF ANSEL             ;Initialize all ports as digital I/O
    CLRF ANSELH
    BSF ANSEL,3                    ;Set RA4/AN3 to be analog input
    BSF ANSEL,1                    ;Set Vref to be analog input

    BANKSEL EEADR
    MOVFW eeprom_address
    MOVWF EEADR                    ;Load address to EEADR

                           ;Use Bank 3
    BSF STATUS,5
    BSF STATUS,6

    BANKSEL EECON1
    BCF EECON1,7           ;Clear EEPGD bit to Access Data memory
    BSF EECON1,0           ;Set RD bit to read from EEPROM

                           ;Use Bank 2
    BCF STATUS,5

    MOVFW EEDAT                    ;Move the data from EEPROM to WREG

                           ;Use Bank 0
    BCF STATUS,6

    MOVWF threshold        ;Move WREG to Threshold register
    CLRF temperature       ;Clear variables
    CLRF buzzer_state
    CLRF count

    MOVLW 0x0B             ;Initial value for Timer0, will run for 500ms
    MOVWF TMR0
    BCF INTCON,2           ;Start Timer0

    GOTO Main_Loop

;********************************MAIN LOOP*************************************
    CODE
Main_Loop
    CALL Get_Temperature    ;Sample the Temperature and Display it

    BTFSS PORTA,0          ;Check if Decrement button is pressed
    CALL One_Second_Delay   ;If yes, wait for 1 second
    BTFSS PORTA,0          ;Check if Decrement button is still pressed
    CALL One_Second_Delay   ;If yes, wait another second.
    BTFSS PORTA,0          ;Check if button is still pressed after 2 seconds
    CALL Set_Threshold     ;If yes, call Set_Threshold

    BTFSS PORTA,2          ;Check if Increment button is pressed
    CALL One_Second_Delay   ;If yes, wait for 1 seconds
    BTFSS PORTA,2          ;Check if Increment button is still pressed
    CALL One_Second_Delay   ;If yes, wait another second.
    BTFSS PORTA,2         ;Check if button is pressed after 2 seconds
    CALL Set_Threshold    ;If yes, call Set_Threshold

    BTFSC INTCON,2        ;Check if 500ms reached by checking timer overflow
    CALL Timer0_Control           ;If set, call Timer0_Control

    MOVFW threshold
    SUBWF temperature,0           ;temp = temperature - threshold
    MOVWF temp            ;If threshold > temperature, negative occurs,
                          ;bit 7 will be set
    BTFSS temp,7          ;If bit 7 set, don't sound alarm because temperature
                          ;is under the threshold
    CALL Trigger_Alarm

    GOTO Main_Loop

;***************************CODE FOR SOUNDING ALARM*****************************

Trigger_Alarm


  BTFSS buzzer_state,0    ;Used to control active state of the buzzer
                          ;(When bit 0 is 1, the buzzer is off)
  BSF PORTB,4
  CALL Set_440Hz
  BCF PORTB,4
    RETURN

;**********************CODE TO GET TEMPERATURE FROM LM35************************

Get_Temperature
                          ;Conversion is initiated by setting the GO/DONE
                          ;bit ADCON0<1>

    BSF ADCON0,1          ;Start ADC conversion

Wait_Loop
    BTFSC ADCON0,1        ;Checks if conversion done, if so, exit loop
    GOTO Wait_Loop
    MOVFW ADRESH          ;Move conversion result to WREG
    MOVWF temperature
    CALL Display          ;Displays the Temperature
    RETURN

;************************CODE FOR DISPLAYING ON SSDs****************************

Display

    CALL Convert_to_BCD           ;subroutine to convert count to BCD
    CALL SSD_Table        ;gets code for displaying the number (Tens)
    ADDLW 0x80            ;setting the MSB (Bit 7) will enable the Tens SSD
    MOVWF PORTC                   ;display Tens value
    CALL Multiplexing_Delay ;delay for multiplexing SSDs
    BCF PORTC,7                   ;Disable tens SSD
    MOVFW units
    CALL SSD_Table        ;gets code for displaying the number (Units)
    BSF PORTA,5                   ;Set RA5 to enable units SSD
    MOVWF PORTC                   ;displays units value
    CALL Multiplexing_Delay
    BCF PORTA,5                   ;Disable the Units SSD
    RETURN

Convert_to_BCD          ;converts count to BCD
    Call Binary_To_BCD  ;uses library subroutine to get BCD value of number
    MOVWF tens
    ANDLW 0x0F          ;b'00001111 , clears upper nibble of BCD number
    MOVWF units                 ;stores the value as the units
    SWAPF tens,1        ;swaps the nibbles of the BCD number
    MOVFW tens
    ANDLW 0x0F          ;b'00001111, clears the high nibble to get tens value
    MOVWF tens          ;stores value in tens register
    RETURN

;This code adds the value that is in the W register to the Program Counter,
;PC will skip to whichever code is needed and returns the code in the WREG.

SSD_Table
```

```
                         ;These HEX values are required because common anode SSDs
                         ;are being used
    ADDWF PCL,1
    RETLW 0x40           ;displays number 0 on SSD
    RETLW 0x79           ;displays number 1 on SSD
    RETLW 0x24           ;displays number 2 on SSD
    RETLW 0x30           ;displays number 3 on SSD
    RETLW 0x19           ;displays number 4 on SSD
    RETLW 0x12           ;displays number 5 on SSD
    RETLW 0x02           ;displays number 6 on SSD
    RETLW 0x78           ;displays number 7 on SSD
    RETLW 0x00           ;displays number 8 on SSD
    RETLW 0x10           ;displays number 9 on SSD

;**********************CODE FOR SETTING THRESHOLD VALUE************************
Set_Threshold
    CALL Three_Second_Delay ;Calls routine to start Timer1 to run for 3s
Set_Threshold_Loop
    MOVFW threshold
    CALL Display           ;Displays the Threshold value
    CALL Flash_SSD         ;This flashes the SSDs off and on


    BCF PORTA,5                   ;Disables Units SSD
    BCF PORTC,7                   ;Disables Tens SSD

    BTFSS PORTA,2          ;Checks if Increment button pressed
    DECF threshold,1       ;Prevents immediate increase of value when entering
                           ;the "Set Threshold" mode
    CALL Increase_Threshold ;If so, call Increae_Threshold
    BTFSS PORTA,0          ;Checks if Decrement button pressed
    INCF threshold,1       ;Prevents immediate increase of value when entering
                           ;the "Set Threshold" mode
    CALL Decrease_Threshold ;If so, call Decrease_Threshold

    BTFSS PIR1,0           ;If 3 seconds occured, exit routine
    GOTO Set_Threshold_Loop
    CALL Write_To_EEPROM    ;Subroutine to write to EEPROM
    BCF PIR1,0             ;Re-enable the timer flag
    BCF T1CON,0                   ;Stop Timer1

    RETURN

Increase_Threshold
    INCF threshold,1       ;Increment the Threshold value
    MOVLW 0x1C             ;adds 28 to threshold
    ADDWF threshold,0      ;if threshold is 128 or greater, bit 7 will set
    MOVWF temp
    BTFSC temp,7           ;if bit 7 is clear, number is under 100
    CLRF threshold         ;reset the threshold
Wait_For_Increase
    MOVFW threshold
    CALL Display           ;Displays the Threshold value while waiting
    BTFSS PORTA,2          ;Waits till button released
    GOTO Wait_For_Increase
    CALL Three_Second_Delay ;Starts Timer1 again. Configure to run for 3s.
    RETURN

Decrease_Threshold
    DECF threshold,1       ;Decrement Threshold value
    BTFSS threshold,7      ;If -1 occurs(0xFF), bit 7 will be set. Reset to 99
    GOTO Wait_For_Decrease  ;else skip
    MOVLW 0x63
    MOVWF threshold
Wait_For_Decrease
    MOVFW threshold
    CALL Display           ;Displays the Threshold value while waiting
    BTFSS PORTA,0          ;Waits till button released
    GOTO Wait_For_Decrease
    CALL Three_Second_Delay ;Run 3s delay.
    RETURN

 ;**********************CODE FOR WRITTING TO EEPROM***************************

Write_To_EEPROM
    BANKSEL EEADR
    MOVFW eeprom_address
    MOVWF EEADR                    ;Data memory address to write
    BANKSEL 0x00
    MOVFW threshold        ;Load threshold value to be written to EEPROM
    BANKSEL EEDAT
    MOVWF EEDAT                    ;Data memory value to write
    BANKSEL EECON1
    BCF EECON1,7           ;Clear EEPGD bit to Access Data memory
    BSF EECON1,2           ;Set WREN to allow write cycle

    MOVLW 0x55
    MOVWF EECON2
    MOVLW 0xAA
    MOVWF EECON2
    BSF EECON1,1           ;Initiate write to EEPROM
    BCF EECON1,2           ;Clear WREN to disable write to EEPROM
    BANKSEL 0x00
    RETURN

;*************************CODE FOR ALL DELAYS REQUIRED*************************

Multiplexing_Delay
    MOVLW 0x30           ;Loads a value of 255 and stores it in temp
    MOVWF temp
Multiplexing_Delay_Loop
    DECFSZ temp,1        ;When temp = 0, exit loop
    GOTO Multiplexing_Delay_Loop

    RETURN


Flash_SSD                ;This delay is used to flash the SSDs off and on during
                         ;Set_Threshold mode.

    MOVLW 0x0C
   MOVWF temp2
    MOVWF temp
Wait_Flash
    DECFSZ temp
    GOTO Wait_Flash
    DECFSZ temp2
    GOTO Wait_Flash
    RETURN

One_Second_Delay           ;Uses Timer1 for a 1 second Delay
    MOVLW 0xF7
    MOVWF TMR1L
    MOVLW 0xC2
    MOVWF TMR1H                    ;Initial value loaded to TMR1H:TMR1L
                           ;to allow a 1sec overflow
    BSF T1CON,0                    ;Start Timer1
One_Second_Loop
    CALL Get_Temperature    ;Displays Temperature while waiting for overflow flag
    BTFSS PIR1,0           ;If flag not set, loop again
    GOTO One_Second_Loop
    BCF PIR1,0             ;Clear Timer1 interrupt flag
    BCF T1CON,0                    ;Stop Timer1
   RETURN

Three_Second_Delay
    BCF PIR1,0             ;Re-enable the timer flag
    BCF T1CON,0                    ;Stops Timer1
    MOVLW 0xE4
    MOVWF TMR1L
    MOVLW 0x48
    MOVWF TMR1H                    ;Initial value loaded to TMR1H:TMR1L to allow a
                           ;3s overflow
    BSF T1CON,0                    ;Start Timer1
    RETURN

;*****************************CODE FOR TIMER0 CONTROL**************************
Timer0_Control
;Timer0 set up for 500ms. When timer flag is set twice, it means 1s has reached.

    INCF count             ;Increment count each time 500ms reached.

                           ;Check if count = 2. Done by moving this to temp
                           ;and decrementing temp twice.
    MOVFW count
    MOVWF temp
    DECF temp,1
    DECFSZ temp,1          ;If temp = 2, zero should occur here
    GOTO Reset_Timer0      ;If not true, reset Timer0
    INCF buzzer_state,1           ;Everytime 1s occurs, buzzer_state is incremented.
                           ;This means every second, Bit 0 is changing value,
                           ;therefore the buzzer is activated for 1 second and
                           ;then deactivated for the next.
    CLRF count             ;Resets the count
Reset_Timer0
    BCF INTCON,2           ;Clear Timer0 overflow flag
    MOVLW 0x0B             ;Initiate Timer0 for 500ms
    MOVWF TMR0
    RETURN

 Set_440Hz

  BCF PIR1,1
  MOVLW 0xB9
  MOVWF TMR2
  MOVLW b'00000101'
  MOVWF T2CON
  BCF T2CON,2

    RETURN

    END
```