

Assignment 1

by Joseff Jones

FILE	2874_JOSEFF_JONES_ASSIGNMENT_1_139109_848125267.DOCX (7.49M)		
TIME SUBMITTED	01-FEB-2018 02:39PM (UTC+0000)	WORD COUNT	3420
SUBMISSION ID	82668183	CHARACTER COUNT	25085

HND Electrical and Electronic Engineering

PSDE5013 – Configuration and Programming of Embedded Systems Assignment 1 – Calendar Alarm Clock

Joseff Jones

Date of Submission: 1/2/18

Word count: 3420

Contents

1. Introduction	4
2. Research.....	5
3. System Design & Spec	7
4. Software Design	14
5. Testing and Analysis of Results	26
6. Conclusions and Recommendations	35
7. References	36

Table of Figures

Figure 1 - PIC16F877 Device Features From (Datasheet, 2018)	5
Figure 2 - Reading the Register	6
Figure 3 - PIC16F877 Pin Diagram According to Datasheet	7
Figure 4 - PIC16F877 on Proteus	7
Figure 5 - State Table	8
Figure 6 - State Diagram	9
Figure 7 - Designed System	10
Figure 8 - Practical Board 1/3	11
Figure 9 - Practical Board (2/3)	12
Figure 10 - Practical Board (3/3)	13
Figure 11 - Coding (Section 1)	14
Figure 12 - Coding (Section 2)	15
Figure 13 - Coding (Section 3)	15
Figure 14 - Coding (Section 4)	16
Figure 15 - Coding (Section 5)	17
Figure 16 - Coding (Section 6)	18
Figure 17 - Coding (Section 7)	19
Figure 18 - Coding (Section 8)	20
Figure 19 - Coding (Section 9)	21
Figure 20 - Coding (Section 10)	22
Figure 21 - Coding (Section 11)	23
Figure 22 - Coding (Section 12)	24
Figure 23 - Coding (Section 13)	25
Figure 24 - Setting the Alarm to 7:05	26
Figure 25 - Waiting for the Alarm	26
Figure 26 - Alarm Activated	27
Figure 27 - Turning off Alarm	27
Figure 28 - Paused Values	30
Figure 29 - Incrementing Hour	30
Figure 30 - Incrementing Minutes	30
Figure 31 - Incrementing Day	31
Figure 32 - Incrementing Date	31
Figure 33 - Incrementing Month	31
Figure 34 - Incrementing Year	31
Figure 35 - Showing that TMR0 = 63	32
Figure 36 - Proof that TMR0 = 10ms	32
Figure 37 - Changing the 10ms to 1s	33
Figure 38 - 1s Cycle	33
Figure 39 - Functioning Practical Board	34

1. Introduction

This assignment had a number of learning outcomes and assessment criteria to meet, the primary criteria to meet was to use a suitable embedded programming language to control and operate a range of on-board peripherals attached to a microcontroller-based embedded system. For this assignment, the target design to be developed is a calendar alarm clock - an appropriate design approach was utilized where an embedded programming language and a number of software packages were used in order to achieve this. To work as an effective calendar alarm clock, the system needed to have a number of functions, such as:

- A function for setting the time and date
- A function allowing the alarm to be enabled at a desired time in the future
- A function allowing the alarm to be turned off and on

The aim of this report is to highlight the findings of the research done towards the methodology of creating the microcontroller-based embedded system and also to highlight the process involved in designing the final product.

reasonable intro. A diagram would be useful here.

2. Research

Prior to tackling the main tasks in the assignment, information on the PIC16F was taken from the official datasheet of the chip; that datasheet provided details about the PIC such as operating frequencies, number of interrupts, size of data memory, size of flash program memory, number of timers, number of I/O ports, and available packages. The specific chip being used in the assignment would be the PIC16F877.

Key Features	PIC16F873A	PIC16F874A	PIC16F876A	PIC16F877A
Operating Frequency	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz
Resets (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
Flash Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory (bytes)	128	128	256	256
Interrupts	14	15	14	15
I/O Ports	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C	Ports A, B, C, D, E
Timers	3	3	3	3
Capture/Compare/PWM modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Analog Comparators	2	2	2	2
Instruction Set	35 Instructions	35 Instructions	35 Instructions	35 Instructions
Packages	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	40-pin PDIP 44-pin PLCC 44-pin TQFP 44-pin QFN	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	40-pin PDIP 44-pin PLCC 44-pin TQFP 44-pin QFN

Figure 1 - PIC16F877 Device Features From (Datasheet, 2018)

The above figure shows multiple features of the PIC16F877 (shown in the column on the right) such as the amount of interrupts and number of I/O ports. Important to note is the EEPROM data memory size, which is 256 bytes – this is a sufficient amount of memory for the type of program being designed; the amount of I/O ports is also apt.

Configuring the Timer

In order to get the system to time itself correctly, an interrupt in the code was utilized which uses the timer (whether that's internal or externally driven) to determine the period for 10ms; the timer in this case was called TIMERO. In order to get the correct preload count, two registers needed to be configured: OPTION and INTCON registers. An example of setting the register for the OPTION register would be OPTION_reg = 0b10100000, which can be deciphered using the table below which translated the register.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RBP	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
1	0	0	0	0	1	1	1

Figure 2 - Reading the Register

The table shows that bit 7, "RBP", is disabling pull-up resistors, bit 6, "INTEDG", interrupt on falling edge; bit 5, "T0CS", uses internal clock, bit 4, "T0SE", is showing the low to high transition, bit 3 "PSA" uses a prescaler and bit 0 "PS0" to 2 "PS2" shows the prescaler rate set to 1:256 (111) (4. Timers, 2018).

From that, it can be seen that 0b10100000, which was used in the actual coding, would disable the pull-up resistors (bit 7) and would use the internal clock (bit 5) and would leave everything else.

In order to configure the TMR0 correctly, the correct preload count for the timer needed to be calculated. Knowing that the oscillating frequency that is being used is 1966800Hz and that there are four pulses per instruction, the actual value can be calculated to be:

$$\frac{1966800}{4} = 4915200\text{HZ}$$

The timer's period can then be calculated using:

$$\frac{1}{4915200} \times 256 = 51.2\mu\text{s}$$

Note - 256 is the prescaler value. In order to get the desired 10ms count, the timer's period needs to be used:

$$\left(\frac{10\text{ms}}{51.2\mu\text{s}} \right) - 2 = 193$$

Note – 2 was subtracted from the value to account for error. The final step to determine the correct TMR0 preload value is to take away this value from the initial preload count value:

$$256 - 193 = 63$$

Therefore, in order to get the desired 10ms interrupt delay, the preload value needs to be 63.

3. System Design & Spec

The design process for the calendar alarm clock was broken up into three parts – the system design, the software design, and the practical. The first part of the process was the system design, where software called Proteus was used to design and to later simulate the system.

System Design

To begin designing the system, the first thing to be done was to decide upon a processor for the system – the PIC16F877 was chosen as it was both fast and economic enough with sufficient pin-outs to use in the design of the calendar alarm clock. The pin diagram for the PIC16F877 was obtained from the data sheet and is shown in the figure below.

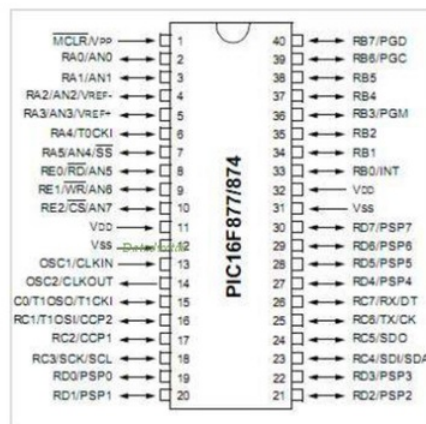


Figure 3 - PIC16F877 Pin Diagram According to Datasheet

While the pin diagram above showed 40 pins, the Proteus software which was being used to design the system showed the PIC16F877 as having fewer pins, and that's because the software didn't need to use those pins (such as VCC, VSS, VDD etc.) to correctly analyse/simulate the system. How the PIC is shown on Proteus is shown in the figure below.

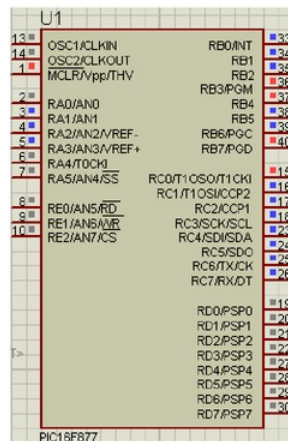


Figure 4 - PIC16F877 on Proteus

Research should extend further with more relevant sources.

State Diagram

The system that needed to be designed was built with the assistance of a state diagram. The state diagram was developed by first making a state table – a state table describes each possible state of the system as well as the result of applying inputs to each state; since there are four inputs, there are four or fewer possible outputs of each state.

State	Action	Result of Applying Input			
		RA1	RA2	RA3	RA4
0	Normal display of time and date	Go to State 8	Go to State 1	Toggle Alarm On/off	No output
1	Allows hours to be set	Go to State 7	Go to State 2	Increment hour by one	Decrement hour by one
2	Allows minutes to be set	Go to State 7	Go to State 3	Increment minutes by one	Decrement minutes by one
3	Allows day to be set	Go to State 7	Go to State 4	Increment day by one	Decrement day by one
4	Allows date to be set	Go to State 7	Go to State 5	Increment date by one	Decrement date by one
5	Allows month to be set	Go to State 7	Go to State 6	Increment month by one	Decrement month by one
6	Allows year to be set	Go to State 7	Go to State 0	Increment year by one	Decrement year by one
7	Saves the temporary variables and go to State 0	No output	No output	No output	No output
8	Displays the message "ALARM"	Go to State 0	Go to State 9	No output	No output
9	Allows alarm hours to be set	Go to State 7	Go to State 10	Increments alarm hours	No output
10	Allows alarm minutes to be set	Go to State 7	Go to State 8	Increments alarm minutes	No output

Figure 5 - State Table

From this state table, the state diagram could be developed. The state diagram is alike a flow-chart in that it describes the result of inputs with the use of a diagram – these are important because some people can read from these better than from tables. The figure on the following page shows the state diagram.

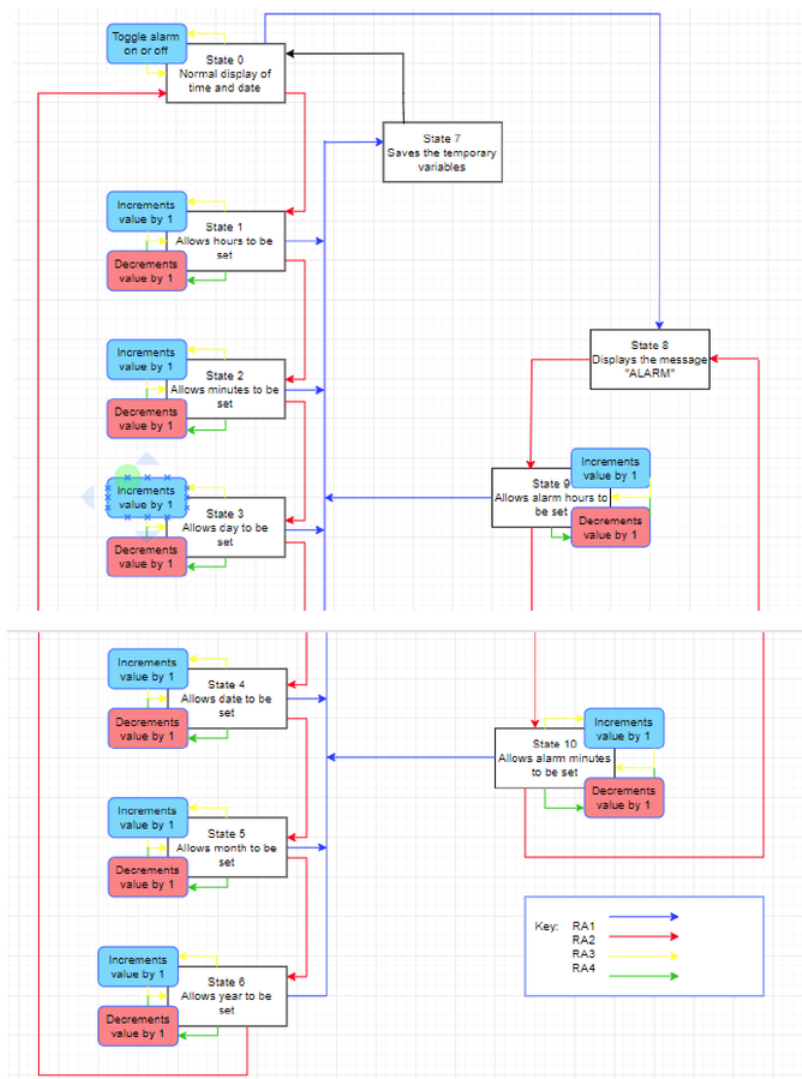


Figure 6 - State Diagram

The above state diagram was created using the <https://www.draw.io> website, and the state table for reference.

State diagram is clear and well presented.

The design of the system involved four switches, the PIC16F877, the LCD and four resistors connected between the switches and the PIC, all grounded. That system is shown in the figure below.

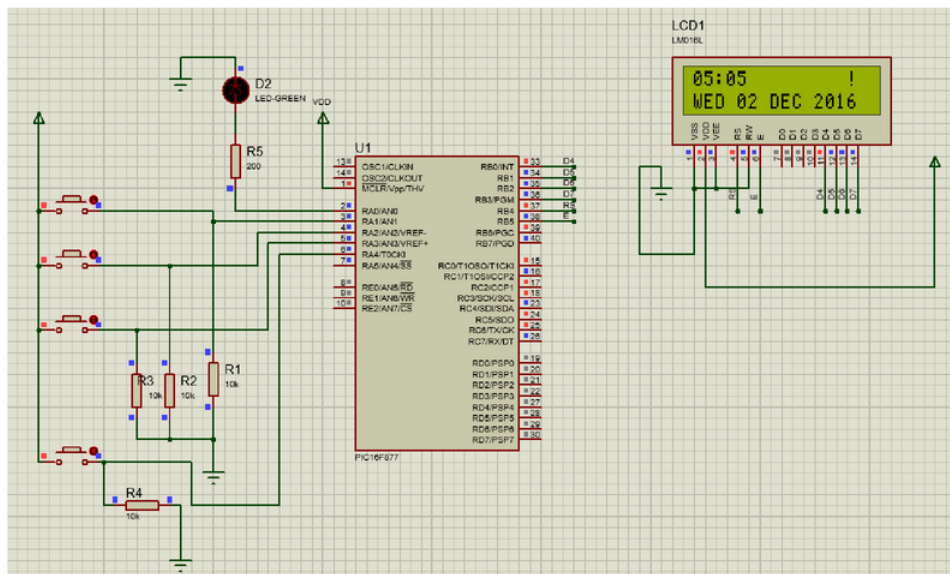


Figure 7 - Designed System

An LED was connected to RA0 in order to test the alarm system, as RA0 is the alarm clock output.

Practical

Once the system was designed and the software was developed, the program could then be uploaded to the board where it could be used as a functioning digital calendar alarm clock. The following figures show the functioning program display the correct information on the LED.

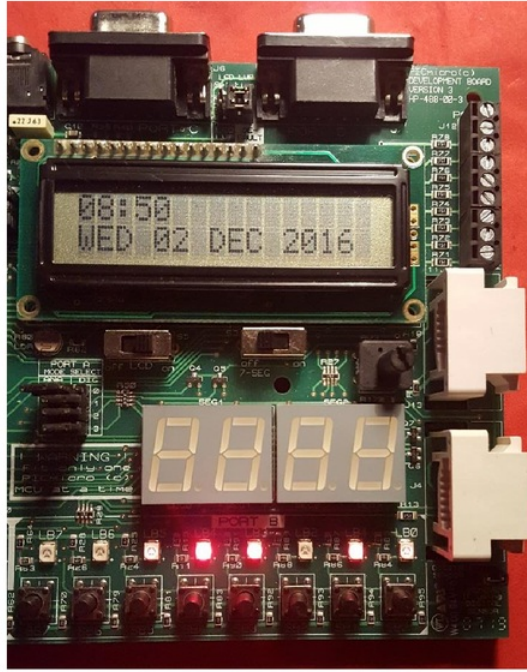


Figure 8 - Practical Board 1/3

The above figure shows the LED screen displaying a functioning clock with hours, minutes, day, date, month and year. The clock was ticking continuously and in time, and the minutes looped to 0 from 59 and the hour incremented correctly.

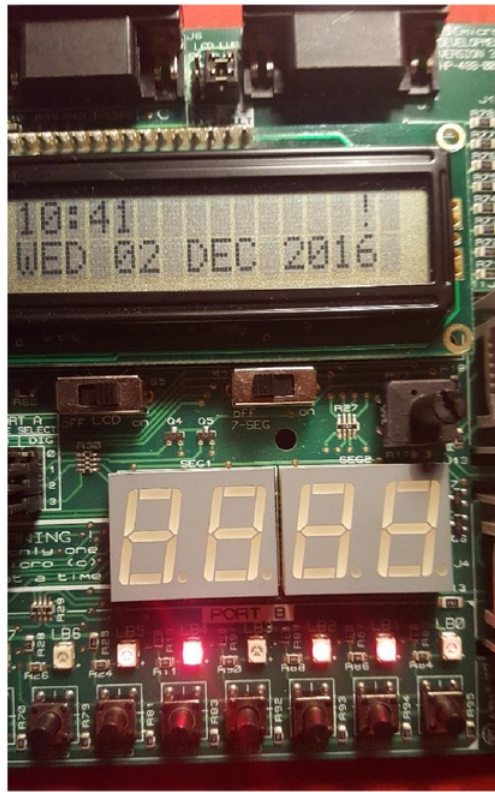


Figure 9 - Practical Board (2/3)

The above figure shows the exclamation point (!) on the LCD, which indicates that the alarm is set.



Figure 10 - Practical Board (3/3)

The above figure shows the state in which the alarm can be set – as indicated by the word “ALARM” on the top right of the LCD.

The configuration is explained

4. Software Design

The next part of the process to be done was the coding – the following section shows the process behind the coding for the calendar alarm clock; all coding was done on MPLAB.

Coding

The coding for the calendar alarm clock was all done in one long program - the following figures contain the complete program, with comments within the actual coding and comments within this report after each figure.

```
#include <pic.h> //Header file to include
#include "LCD.h" //Header file to include

void DisplayTime(int line); //Function prototype for displaying time
void DisplaySetTime(int line); //Function prototype for displaying time to set
void DisplayState(int line);
void DisplayDate(int line); //Function prototype for displaying date
void DisplaySetDate(int line); //Function prototype for display date to set
void DisplayAlarmTime(int line); //Function prototype for displaying alarm time
void DisplaySetAlarmTime(int line); //Function prototype for displaying alarm time to set

void Delay(int t); //Function prototype

int milli=0, secs=0, mins=60, hours=24, day_code=2, days=1, month=12,
year=2016, state=0, alarm_hours=0, alarm_mins=0, timeout=0, alarm=0,
temp_hours=0, temp_mins=0, temp_day_code=0, temp_days=0, temp_month=0,
temp_year=0, temp_al_hours=0, temp_al_mins=0, temp_state=0; //global variables

void interrupt ISR (void)
{
    if(TOIF==1)
    {
        TOIF = 0; //Set interrupt flag to zero
        TMRO = 255; //Reload preload time value
        milli += 10; //Increment milliseconds by 10
        if(milli==100) //Opening loop when milliseconds equals 100
        {
            secs++; //Increment seconds by 1
            milli = 0; //Reset milliseconds to 0
            if(secs==60) //Opening loop when seconds equals 60
            {
                mins++; //Increment mins by 1
                secs = 0; //Set secs to 0
                if(mins==60) //Opening loop when minutes equals 60
                {
                    hours++; //Increment hours by 1
                    mins = 0; //Set mins to 0
                    if(hours==24) //Opening loop when hours equals 24
                    {
                        days++; //Increment days by 1
                        day_code++; //Increment day_code by 1
                        hours = 0;

                        if(days>31)
                        {
                            month++; //Increment month by 1
                            days = 1; //Set days to 0
                            if(month>12) //If month greater than 12
                            {
                                year++; //Increment year by 1
                                month = 1; //Set month to 1
                            }
                        }
                    }
                    if(day_code==7) day_code=0; //If day_code equals 7 then day_code=zero
                }
            }
        }
    }
}
```

Figure 11 - Coding (Section 1)

The above figure is the first part of the code. This part of the code includes the header files, which are used to fetch .h library files, function prototypes, which are used to declare the display values for the LCD as int values; and global variables, which are used to declare the number of variables within the program. The next part of the program begins an interrupt which uses the timer and then describes when the clock should cycle from milliseconds to seconds to minutes to hours to days and finally to months. The "TMRO = 255" preload value is for simulation purposes only, as when the program is used practically, that value is changed to 63.

```

////////////////////////////////////
void main(void)                                //Main Function
{
    TRISA = 0x1E;                               //Configure port A = 0001 1110 in hex
    TRISB = 0x00;                               //Configure port B = 0000 0000 in hex
    TRISC = 0x00;                               //Configure port C = 0000 0000 in hex
    ADCON1 = 0x07;                              //Configure ADCON1 = 0000 0111 in hex
    OPTION_REG = 0x87;                          //Configure OPTION_REG = 1000 0111 in hex
    INTCON = 0xA0;                              //Configure INTCON = 1010 0000 in hex
    TMRO=255;                                   //Pre-loading TMRO

    LCD_Startup();                             //Function header, runs function in LCD.c

    while(1)                                    //Start super-loop
    {
        if(((month==1)||(month==3)||(month==5)||(month==7)||(month==8)||(month==10))&&(days>31))
            /*If month equals 1 or 3 or 5 or 7 or 8 or 10 and days > 31*/
            {
                month++;                          //Increment month by 1
                days = 1;                        //Set day to 1
            }

        if((month==2)&&(days>28)&&(year%4 !=0))
            /*If month equals 2 and days > 28 and year does not
            divide by 4 (not a leap year)*/
            {
                month++;                          //Increment month by 1
                days = 1;                        //Set days to 1
            }

        if((month==12)&&(days>31))              //If month equals 12 and days > 31
            {
                year++;                          //Increment year by 1
                month = 1;                       //Set month to 1
                days = 1;                       //Set days to 1
            }
        if(day_code==7) day_code = 0;           //Sets day_code to 0 if day_code is equal to 7
    }
}

```

Figure 12 - Coding (Section 2)

The next section of the code begins the main function of the program, this includes the configuration of multiple ports and registers, the startup of the function of the LCD, and a super-loop. The super-loop contains how the program manages the amount of days in each month, how the program manages leap years, and how the program manages the amount of days in a week.

```

DisplayState(1);                               // displays the state on line 1 of LCD

if(alarm==1)                                   //If alarm equals 1 step in
{
    if((hours==alarm_hours)&&(mins==alarm_mins))&&(retained_alarm==1)
    /*If hours equals alarm_hours and mins equals alarm_mins or retained)alarm equals 1 step in*/
    {
        RA0 = 1;                               //Set output RA0 to 1
    }
}
if(alarm==0)                                   //If alarm equals zero
{
    RA0 = 0;                                     //Set output RA0 to zero
}

```

Figure 13 - Coding (Section 3)

The next section of the code determines what happens when the alarm is on the on state and on the off state – when the alarm is on the on state, the alarm will play when the clock's time is equal to the alarm's time.

```

switch(state)                                //Controls flow depending on state value
{
    case 0:                                  //Case 0, run if state equals 0
        DisplayTime(0);                      //Function header, runs DisplayTime function prototype
        DisplayDate(1);

        temp_hours = hours;                  //Sets temp_hours to equal hours
        temp_mins = mins;                    //Sets temp_mins to equal mins
        temp_day_code = day_code;            //Sets temp_day_code to equal day_code
        temp_days = days;                    //Sets temp_days to equal days
        temp_month = month;                  //Sets temp_month to equal month
        temp_year = year;                    //Sets temp_year to equal year

        if(RA1) state = 8;                   //If RA1 is equal to 1 set state to equal 8
        if(RA2) state = 1;                   //If RA2 is equal to 1 set state to equal 1

        if(RA3==1)                           //If RA3 is equal to 1 step in
        {
            if(alarm==0)                     //If alarm equals zero
            {
                alarm = 1;                   //Sets alarm to equal 1
            }
            else                               //Else command, if alarm is not equal to zero step in
            {
                alarm = 0;                   //Sets alarm to zero
            }
        }

        break;                               //This breaks out from each case

    case 1:                                  //Case 1, run if state equals 1
        DisplaySetTime(0);                   //Function header, runs DisplayTime function prototype

        if(RA1) state = 7;                   //If RA1 is equal to 1 set state to equal 7
        if(RA2) state = 2;                   //If RA2 is equal to 1 set state to equal 2
        if(RA3)                               //If RA3 is equal to 1
        {
            temp_hours++;                     //Increment temp_hours by 1
            if(temp_hours>23) temp_hours=0;    //If temp_hours is greater than 23 set to zero
        }

        if(RA4)                               //If RA4 = 1
        {
            while(RA4);                       //While RA4 is pressed
            Delay(100);                       //Delay 100ms

            temp_hours--;                     //decrement temp_hours by 1
            if(temp_hours<0) temp_hours=23;    //If temp_hours is greater than 23 set to zero
        }

        break;                               //This breaks out from each case
}

```

Figure 14 - Coding (Section 4)

The next section of the code introduces a switch case where states and their functions are determined. In state 0, temp values are set to actual values, this is there in order to alter the actual values using the buttons on the system. The state also scans for when either RA1 or RA2 buttons are activated, and then sets the program to either state 8 or state 0 respectively. It also scans for RA3; when RA3 is pressed, the alarm turns on if it's already off, or off if it's already on. In state 1, when RA1 is pressed the program goes to state 7, and when RA2 is pressed the program goes to state 2. RA3 increments the hours by one, and RA4 decrements the hours by one.

```

case 2: //Case 2, run if state equals 2
    DisplaySetTime(0); //Function header, runs DisplaySetTime function prototype

    if(RA1) state = 7; //If RA1 is equal to 1 set state to equal 7
    if(RA2) state = 3; //If RA2 is equal to 1 set state to equal 3
    if(RA3) //If RA3 is equal to 1 step in
    {
        temp_mins++; //Increment temp_mins by 1
        if(temp_mins>59) temp_mins=0; //If temp_mins is greater than 59 set to zero
    }

    if(RA4) //If RA4 is = 1
    {
        while(RA4);
        Delay(100);
        temp_mins--; //decrement temp_mins by 1
        if(temp_mins<0) temp_mins=59; //If temp_mins is greater than 59 set to zero
    }
    break; //This breaks out from each case

case 3: //Case 3, run if state equals 3
    DisplaySetDate(1); //Function header, runs DisplaySetDate function prototype

    if(RA1) state = 7; //If RA1 is equal to 1 set state to equal 7
    if(RA2) state = 4; //If RA2 is equal to 1 set state to equal 4
    if(RA3) //If RA3 is equal to 1 step in
    {
        temp_day_code++; //Increment temp_day_code by 1
        if(temp_day_code>6) temp_day_code=0; //If temp_day_code is greater than 6 set to zero
    }
    if(RA4) //If RA4 = 1
    {
        while(RA4);
        Delay(100);
        temp_day_code--; //decrement temp_day_code by 1
        if(temp_day_code<0) temp_day_code=6; //If temp_day_code is less than 0 set to 6
    }
    break; //This breaks out from each case

case 4: //Case 4, run if state equals 4
    DisplaySetDate(1); //Function header, runs DisplaySetDate function prototype

    if(RA1) state = 7; //If RA1 is equal to 1 set state to equal 7
    if(RA2) state = 5; //If RA2 is equal to 1 set state to equal 5
    if(RA3) //If RA3 is equal to 1 step in
    {
        temp_days++; //Increment temp_days by 1
        if(((temp_month==1)||((temp_month==3)||((temp_month==5)||((temp_month==7)||((temp_month==8)
        ||(temp_month==10)||((temp_month==12)||((temp_days>31))))))
        /*If temp_month equals 1 or 3 or 5 or 7 or 8 or 10 and temp_days greater than 31 step in*/
        {
            temp_days = 1; //Set days to 1
        }
        if(((temp_month==2)&&(temp_days>28)&&(temp_year% 4 != 0))
        /*If temp_month equals 2 and temp_days greater than 29 and
        4 does not divide into year step in (not leap year)*/
        {
            temp_days = 1; //Set days to 1
        }
        if(((temp_month==2)&&(temp_days>29)&&(temp_year% 4 == 0))
        {
            temp_days = 1; //Set days to 1
        }
        if(((temp_month==4)||((temp_month==6)||((temp_month==9)||((temp_month==11)||((temp_days>30))))
        /*If temp_month equals 4 or 6 or 9 or 11 and temp_days greater than 30 step in*/
        {
            temp_days = 1; //Set days to 1
        }
    }
}

```

Figure 15 - Coding (Section 5)

The next section of code covers the next three states – state 2 is for incrementing and decrementing the minutes and resets the minutes to 0 every 60 minutes, and state 3 is for incrementing and decrementing the days of the week and resets the day to Sunday whenever it surpasses Saturday. State 4 is for incrementing and decrementing the date and required coding in the amount of days of each month of the year; the figure above only shows the incrementing coding, the next figure shows the decrementing.

```

if(RA4)                                     //If RA4 = 1
{
    while(RA4);

        Delay(100);

        temp_days--;                       //decrement temp_days by 1

        if(((temp_month==1)|| (temp_month==3)|| (temp_month==5)|| (temp_month==7)|| (temp_month==8))
        || (temp_month==10)|| (temp_month==12)) && (temp_days<1))
        /*If temp_month equals 1 or 3 or 5 or 7 or 8 or 10 and temp_days greater than 31 step in*/
        {
            temp_days = 31;                //Set days to 31
        }
        if((temp_month==2)&&(temp_days<1)&&(temp_year% 4 != 0))
        /*If temp_month equals 2 and temp_days greater than 29 and
        4 does not divide into year step in (not leap year)*/
        {
            temp_days = 28;                //Set days to 29
        }
        if((temp_month==2)&&(temp_days<1)&&(temp_year% 4 == 0))
        /*If temp_month equals 2 and temp_days greater than 29 and
        4 divides into year step in (leap year)*/
        {
            temp_days = 29;                //Set days to 29
        }
        if(((temp_month==4)|| (temp_month==6)|| (temp_month==9)|| (temp_month==11))&&(temp_days<1))
        /*If temp_month equals 4 or 6 or 9 or 11 and temp_days greater than 30 step in*/
        {
            temp_days = 30;                //Set days to 30
        }

    }

    break;                                //This breaks out from each case

case 5:
    DisplaySetDate(1);                    //Case 5, run if state equals 5
                                          //Function header, runs DisplaySetDate function prototype

    if(RA1) state = 7;                     //If RA1 is equal to 1 set state to equal 7
    if(RA2) state = 6;                     //If RA2 is equal to 1 set state to equal 6
    if(RA3)                                //If RA3 is equal to 1 step in
    {
        temp_month++;                     //Increment temp_month by 1
        if(temp_month>12) temp_month=1; //If temp_month is greater than 12 set to 1
    }

    if(RA4)                                //If RA4 = 1
    {

        while(RA4);

            Delay(100);

            temp_month--;                 //decrement temp_day_code by 1
            if(temp_month<1) temp_month=12; //If temp_month is less than 0 set to 12

    }

    break;                                //This breaks out from each case

```

Figure 16 - Coding (Section 6)

The next section starts in state 4 and determines that RA4 decrements the date. State 5 increments and decrements the months.

```

case 6:                                //Case 6, run if state equals 6
    DisplaySetDate(1);                 //Function header, runs DisplaySetDate function prototype

    if(RA1) state = 7;                 //If RA1 is equal to 1 set state to equal 7
    if(RA2) state = 0;                 //If RA2 is equal to 1 set state to equal 0
    if(RA3)                             //If RA3 is equal to 1 step in
    {
        temp_year++;                   //Increment temp_year by 1
    }

    if(RA4)                             //If RA4 = 1
    {

        while(RA4);
            Delay(100);
            temp_year--;                //decrement temp_year by 1

    }

    break;                             //This breaks out from each case

case 7:                                //Case 7, run if state equals 7
    DisplayTime(0);                    //Function header, runs DisplayTime function prototype
    DisplayDate(1);

    hours = temp_hours;                //Sets hours to equal temp_hours
    mins = temp_mins;                  //Sets mins to equal temp_mins
    day_code = temp_day_code;          //Sets day_code to equal temp_day_code
    days = temp_days;                  //Sets days to equal temp_days
    month = temp_month;                //Sets month to equal temp_month
    year = temp_year;                  //Sets year to equal temp_year
    alarm_hours = temp_al_hours;        //Sets alarm hours to equal temp_al_hours
    alarm_mins = temp_al_mins;          //Sets alarm_mins to equal temp_al_mins

    Delay(100);                        //Function header, runs Delay function prototype
    state = 0;                         //Sets state to equal zero
    break;                             //This breaks out from each case

case 8:                                //Case 8, run if state equals 8

    DisplayDate(1);                    //Function header, runs DisplayAlarmDate function prototype
    DisplayAlarmTime(0);
    temp_al_hours = alarm_hours;        //Sets temp_al_hours to equal alarm_hours
    temp_al_mins = alarm_mins;          //Sets temp_al_mins to equal alarm_mins

    if(RA1) state = 0;                 //If RA1 is equal to 1 set state to equal 0
    if(RA2) state = 9;                 //If RA2 is equal to 1 set state to equal 9

    break;                             //This breaks out from each case

case 9:                                //Case 9, run if state equals 9
    DisplaySetAlarmTime(0);
    if(RA1) state = 7;                 //If RA1 is equal to 1 set state to equal 7
    if(RA2) state = 10;                //If RA2 is equal to 1 set state to equal 10
    if(RA3)                             //If RA3 is equal to 1 step in
    {
        temp_al_hours++;               //Increment temp_al_hours by 1
        if(temp_al_hours>23) temp_al_hours=0; //If temp_al_hours is greater than 23 set to 0
    }

    break;                             //This breaks out from each case

```

Figure 17 - Coding (Section 7)

The next section of code covers the next four states – state 6 increments and decrements the year. State 7 sets the actual values of the clock to the temporary values which were set using RA1, RA2, RA3 and RA4; state 8 sets the alarm hours and minutes to the same values there set as the temporary alarm hours and minutes. State 9 increments the alarm hours.

Wh
at
abo
ut
stat
e
10.

```

    case 10:
        DisplaySetAlarmTime(0);
        //Case 10, run if state equals 10
        //Function header, runs DisplaySetAlarmTime function prototype

        if(RA1) state = 7;
        if(RA2) state = 8;
        //If RA1 is equal to 1 set state to equal 7
        //If RA2 is equal to 1 set state to equal 8

        if(RA3)
        {
            temp_al_mins++;
            if(temp_al_mins>59) temp_al_mins=0;
            //Increment temp_al_mins by 1
            //If temp_al_mins is greater than 59 set to 0
        }

        break;
        //This breaks out from each case
    }
    while(RA1 || RA2 || RA3);
    //While RA1 or RA2 or RA3 are active do nothing
}

///////////////////////////////////////////////////
void DisplayTime(int line)
{
    char time_string[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //Local string declared with 16 elements

    if(alarm==1)
    {
        //If alarm is equal to 1 step in
        time_string[0] = (char) hours / 10 + 48; //Displays the first hour digit. 48 equals the ascii code for 0
        time_string[1] = (char) hours % 10 + 48; //Displays the second hour digit. 48 equals the ascii code for 0
        time_string[2] = ':'; //Displays the : digit.
        time_string[3] = (char) mins / 10 + 48; //Displays the first minute digit. 48 equals the ascii code for 0
        time_string[4] = (char) mins % 10 + 48; //Displays the second minute digit. 48 equals the ascii code for 0
        time_string[5] = ' '; //Creates space
        time_string[6] = ' '; //Creates space
        time_string[7] = ' '; //Creates space
        time_string[8] = ' '; //Creates space
        time_string[9] = ' '; //Creates space
        time_string[10] = ' '; //Creates space
        time_string[11] = ' '; //Creates space
        time_string[12] = ' '; //Creates space
        time_string[13] = ' '; //Creates space
        time_string[14] = (char) 33; //Ascii code 33 = !
        time_string[15] = 0; //Ends the string by placing 0 in the last string
    }

    if(alarm==0)
    {
        //If alarm is equal to 0 step in
        time_string[0] = (char) hours / 10 + 48; //Displays the first hour digit. 48 equals the ascii code for 0
        time_string[1] = (char) hours % 10 + 48; //Displays the second hour digit. 48 equals the ascii code for 0
        time_string[2] = ':'; //Displays the : digit.
        time_string[3] = (char) mins / 10 + 48; //Displays the first minute digit. 48 equals the ascii code for 0
        time_string[4] = (char) mins % 10 + 48; //Displays the second minute digit. 48 equals the ascii code for 0
        time_string[5] = ' '; //Creates space
        time_string[6] = ' '; //Creates space
        time_string[7] = ' '; //Creates space
        time_string[8] = ' '; //Creates space
        time_string[9] = ' '; //Creates space
        time_string[10] = ' '; //Creates space
        time_string[11] = ' '; //Creates space
        time_string[12] = ' '; //Creates space
        time_string[13] = ' '; //Creates space
        time_string[14] = ' '; //Creates space
        time_string[15] = 0; //Ends the string by placing 0 in the last string
    }
    DisplayString(line, time_string);
    //Sends the string to the LCD on line 0
}

```

State code is explained fairly well. What about the while statement at the end. This is to counter switch bounce.

Figure 18 - Coding (Section 8)

The next section of code covers the final state of the switch-case, state 10; which decrements the alarm hours. Following the switch-case is a loop which is used to display the current time on the LED; when the alarm is turned on an exclamation point (!) is shown on the LED, denoted by (char) 33 in the ASCII code and in the loop above.

Must include the 'DisplayString' function call at the end here because it is the function used in the LCD display.

```

void DisplaySetTime(int line)
{
    char time_string[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //Local string declared with 16 elements

    if(alarm==1) //If alarm is equal to 1 step in
    {
        time_string[0] = (char) temp_hours / 10 + 48; //Displays the first hour digit. 48 equals the ascii code for 0
        time_string[1] = (char) temp_hours % 10 + 48; //Displays the second hour digit. 48 equals the ascii code for 0
        time_string[2] = ':'; //Displays the : digit.
        time_string[3] = (char) temp_mins / 10 + 48; //Displays the first minute digit. 48 equals the ascii code for 0
        time_string[4] = (char) temp_mins % 10 + 48; //Displays the second minute digit. 48 equals the ascii code for 0
        time_string[5] = ' '; //Creates space
        time_string[6] = ' '; //Creates space
        time_string[7] = ' '; //Creates space
        time_string[8] = ' '; //Creates space
        time_string[9] = ' '; //Creates space
        time_string[10] = ' '; //Creates space
        time_string[11] = ' '; //Creates space
        time_string[12] = ' '; //Creates space
        time_string[13] = ' '; //Creates space
        time_string[14] = (char) 33; //Ascii code 33 = !
        time_string[15] = 0; //Ends the string by placing 0 in the last string
    }

    if(alarm==0) //If alarm is equal to 0 step in
    {
        time_string[0] = (char) temp_hours / 10 + 48; //Displays the first hour digit. 48 equals the ascii code for 0
        time_string[1] = (char) temp_hours % 10 + 48; //Displays the second hour digit. 48 equals the ascii code for 0
        time_string[2] = ':'; //Displays the : digit.
        time_string[3] = (char) temp_mins / 10 + 48; //Displays the first minute digit. 48 equals the ascii code for 0
        time_string[4] = (char) temp_mins % 10 + 48; //Displays the second minute digit. 48 equals the ascii code for 0
        time_string[5] = ' '; //Creates space
        time_string[6] = ' '; //Creates space
        time_string[7] = ' '; //Creates space
        time_string[8] = ' '; //Creates space
        time_string[9] = ' '; //Creates space
        time_string[10] = ' '; //Creates space
        time_string[11] = ' '; //Creates space
        time_string[12] = ' '; //Creates space
        time_string[13] = ' '; //Creates space
        time_string[14] = ' '; //Creates space
        time_string[15] = 0; //Ends the string by placing 0 in the last string
    }
    DisplayString(line, time_string); //Sends the string to the LCD on line 0
}

void DisplayDate(int line)
{
    char day_code_string[5] = {0, 0, 0, 0, 0}; //Local string declared with 5 elements
    char month_string[5] = {0, 0, 0, 0, 0}; //Local string declared with 5 elements
    char date_string[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //Local string declared with 16 elements

    if(day_code==0) //If day_code is equal to zero
    {
        day_code_string[0] = (char) 83; //Ascii code 83 = S
        day_code_string[1] = (char) 85; //Ascii code 85 = U
        day_code_string[2] = (char) 78; //Ascii code 78 = N
        day_code_string[3] = ' '; //creates space
        day_code_string[4] = 0; //Ends the string by placing 0 in the last string
    }
    if(day_code==1) //If day_code is equal to 1
    {
        day_code_string[0] = (char) 77;
        day_code_string[1] = (char) 79;
        day_code_string[2] = (char) 78;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if(day_code==2) //If day_code is equal to 2
    {
        day_code_string[0] = (char) 84;
        day_code_string[1] = (char) 85;
        day_code_string[2] = (char) 69;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
}

```

Figure 19 - Coding (Section 9)

The next section of code includes a loop called DisplaySetTime for displaying the LCD values when the time is being set for when the alarm is on and when the alarm is off. There is also another loop called DisplayDate for displaying the days of the week on the LCD.

```

if(month==1)                //If month is equal to 1
{
    month_string[0] = (char) 74; //Ascii code 74 = J
    month_string[1] = (char) 65; //Ascii code 65 = A
    month_string[2] = (char) 78; //Ascii code 78 = N
    month_string[3] = ' '; //Creates space
    month_string[4] = 0; //Ends the string by placing 0 in the last string
}
if(month==2)                //If month is equal to 2
{
    month_string[0] = (char) 70;
    month_string[1] = (char) 69;
    month_string[2] = (char) 66;
    month_string[3] = ' ';
    month_string[4] = 0;
}

if(month==3)                //If month is equal to 3
{
    month_string[0] = (char) 77;
    month_string[1] = (char) 65;
    month_string[2] = (char) 82;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if(month==4)                //If month is equal to 4
{
    month_string[0] = (char) 65;
    month_string[1] = (char) 80;
    month_string[2] = (char) 82;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if(month==5)                //If month is equal to 5
{
    month_string[0] = (char) 77;
    month_string[1] = (char) 65;
    month_string[2] = (char) 89;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if(month==6)                //If month is equal to 6
{
    month_string[0] = (char) 74;
    month_string[1] = (char) 85;
    month_string[2] = (char) 78;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if(month==7)                //If month is equal to 7
{
    month_string[0] = (char) 74;
    month_string[1] = (char) 85;
    month_string[2] = (char) 76;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if(month==8)                //If month is equal to 8
{
    month_string[0] = (char) 65;
    month_string[1] = (char) 85;
    month_string[2] = (char) 71;
    month_string[3] = ' ';
    month_string[4] = 0;
}

```

Figure 20 - Coding (Section 10)

Similarly to the last section of the code, this section of the code focuses on what to display on the LCD – specifically, this part of the code displays the months on the LCD.

```

    /*Places day_code, days, month, and year into the date_string*/
    date_string[0] = day_code_string[0];
    date_string[1] = day_code_string[1];
    date_string[2] = day_code_string[2];
    date_string[3] = day_code_string[3];
    date_string[4] = days / 10 + 48;
    date_string[5] = days % 10 + 48;
    date_string[6] = ' '; //Creates space
    date_string[7] = month_string[0];
    date_string[8] = month_string[1];
    date_string[9] = month_string[2];
    date_string[10] = month_string[3];
    date_string[11] = year / 1000 + 48;
    date_string[12] = year % 1000 / 100 + 48;
    date_string[13] = year % 100 / 10 + 48;
    date_string[14] = year % 10 + 48;
    date_string[15] = 0; //Ends the string by placing 0 in the last string

    DisplayString(line, date_string); //Sends the string to the LCD on line 1

d DisplaySetDate(int line)

    char day_code_string[5] = {0, 0, 0, 0, 0}; //Local string declared with 5 elements
    char month_string[5] = {0, 0, 0, 0, 0}; //Local string declared with 5 elements
    char date_string[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //Local string declared with 16 elements

    if(temp_day_code==0) //If temp_day_code is equal to zero step in
    {
        day_code_string[0] = (char) 83; //Ascii code 83 = S
        day_code_string[1] = (char) 85; //Ascii code 85 = U
        day_code_string[2] = (char) 78; //Ascii code 78 = N
        day_code_string[3] = ' '; //creates space
        day_code_string[4] = 0; //Ends the string by placing 0 in the last string
    }
    if(temp_day_code==1) //If temp_day_code is equal to 1 step in
    {
        day_code_string[0] = (char) 77; //Ascii code 77 = M
        day_code_string[1] = (char) 79; //Ascii code 79 = O
        day_code_string[2] = (char) 78; //Ascii code 78 = N
        day_code_string[3] = ' '; //creates space
        day_code_string[4] = 0; //Ends the string by placing 0 in the last string
    }
    if(temp_day_code==2) //If temp_day_code is equal to 2 step in
    {
        day_code_string[0] = (char) 84; //Ascii code 84 = T
        day_code_string[1] = (char) 85; //Ascii code 85 = U
        day_code_string[2] = (char) 69; //Ascii code 69 = E
        day_code_string[3] = ' '; //creates space
        day_code_string[4] = 0; //Ends the string by placing 0 in the last string
    }
    if(temp_day_code==3) //If temp_day_code is equal to 3 step in
    {
        day_code_string[0] = (char) 87; //Ascii code 87 = W
        day_code_string[1] = (char) 69; //Ascii code 69 = E
        day_code_string[2] = (char) 68; //Ascii code 68 = D
        day_code_string[3] = ' '; //creates space
        day_code_string[4] = 0; //Ends the string by placing 0 in the last string
    }
    if(temp_day_code==4) //If temp_day_code is equal to 4 step in
    {
        day_code_string[0] = (char) 84; //Ascii code 84 = T
        day_code_string[1] = (char) 72; //Ascii code 72 = H
        day_code_string[2] = (char) 85; //Ascii code 85 = U
        day_code_string[3] = ' '; //creates space
        day_code_string[4] = 0; //Ends the string by placing 0 in the last string
    }
}

```

Figure 21 - Coding (Section 11)

The next section of code is similar to the second loop in section 9 of the code, except it's used to display the days of the week when *setting* the date on the LCD.

```

if(temp_day_code==5)           //If temp_day_code is equal to 5 step in
{
    day_code_string[0] = (char) 70; //Ascii code 70 = F
    day_code_string[1] = (char) 82; //Ascii code 82 = R
    day_code_string[2] = (char) 73; //Ascii code 73 = I
    day_code_string[3] = ' ';      //creates space
    day_code_string[4] = 0;        //Ends the string by placing 0 in the last string
}
if(temp_day_code==6)           //If temp_day_code is equal to 6 step in
{
    day_code_string[0] = (char) 83; //Ascii code 83 = S
    day_code_string[1] = (char) 65; //Ascii code 65 = A
    day_code_string[2] = (char) 84; //Ascii code 84 = T
    day_code_string[3] = ' ';      //creates space
    day_code_string[4] = 0;        //Ends the string by placing 0 in the last string
}
if(temp_month==1)              //If temp_month is equal to 1 step in
{
    month_string[0] = (char) 74;   //Ascii code 74 = J
    month_string[1] = (char) 65;   //Ascii code 65 = A
    month_string[2] = (char) 78;   //Ascii code 78 = N
    month_string[3] = ' ';        //Creates space
    month_string[4] = 0;          //Ends the string by placing 0 in the last string
}
if(temp_month==2)              //If temp_month is equal to 2 step in
{
    month_string[0] = (char) 70;   //Ascii code 70 = F
    month_string[1] = (char) 69;   //Ascii code 69 = E
    month_string[2] = (char) 66;   //Ascii code 66 = B
    month_string[3] = ' ';        //Creates space
    month_string[4] = 0;          //Ends the string by placing 0 in the last string
}
if(temp_month==3)              //If temp_month is equal to 3 step in
{
    month_string[0] = (char) 77;   //Ascii code 77 = M
    month_string[1] = (char) 65;   //Ascii code 65 = A
    month_string[2] = (char) 82;   //Ascii code 82 = R
    month_string[3] = ' ';        //Creates space
    month_string[4] = 0;          //Ends the string by placing 0 in the last string
}
if(temp_month==4)              //If temp_month is equal to 4 step in
{
    month_string[0] = (char) 65;   //Ascii code 65 = A
    month_string[1] = (char) 80;   //Ascii code 80 = P
    month_string[2] = (char) 82;   //Ascii code 82 = R
    month_string[3] = ' ';        //Creates space
    month_string[4] = 0;          //Ends the string by placing 0 in the last string
}

```

Figure 22 - Coding (Section 12)

Section 12 of the code is used to display the months of the year when *setting* the date on the LCD.

```

void DisplayAlarmTime(int line)
{
    char alarm_string[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //Local string declared with 16 elements

    alarm_string[0] = (char) alarm_hours / 10 + 48; //Displays the first hour digit. 48 equals the ascii code for 0
    alarm_string[1] = (char) alarm_hours % 10 + 48; //Displays the second hour digit. 48 equals the ascii code for 0
    alarm_string[2] = ':'; //Displays the : digit.
    alarm_string[3] = (char) alarm_mins / 10 + 48; //Displays the first minute digit. 48 equals the ascii code for 0
    alarm_string[4] = (char) alarm_mins % 10 + 48; //Displays the second minute digit. 48 equals the ascii code for 0
    alarm_string[5] = ' '; //creates space
    alarm_string[6] = ' '; //creates space
    alarm_string[7] = ' '; //creates space
    alarm_string[8] = ' '; //creates space
    alarm_string[9] = ' '; //creates space
    alarm_string[10] = (char) 65; //Ascii code 65 = A
    alarm_string[11] = (char) 76; //Ascii code 76 = L
    alarm_string[12] = (char) 65; //Ascii code 65 = A
    alarm_string[13] = (char) 82; //Ascii code 82 = R
    alarm_string[14] = (char) 77; //Ascii code 77 = M
    alarm_string[15] = 0; //Ends the string by placing 0 in the last string

    DisplayString(line, alarm_string); //Sends the string to the LCD on line 0
}

void DisplaySetAlarmTime(int line)
{
    char alarm_string[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //Local string declared with 16 elements

    alarm_string[0] = (char) temp_al_hours / 10 + 48; //Displays the first hour digit. 48 equals the ascii code for 0
    alarm_string[1] = (char) temp_al_hours % 10 + 48; //Displays the second hour digit. 48 equals the ascii code for 0
    alarm_string[2] = ':'; //Displays the : digit.
    alarm_string[3] = (char) temp_al_mins / 10 + 48; //Displays the first minute digit. 48 equals the ascii code for 0
    alarm_string[4] = (char) temp_al_mins % 10 + 48; //Displays the second minute digit. 48 equals the ascii code for 0
    alarm_string[5] = ' '; //creates space
    alarm_string[6] = ' '; //creates space
    alarm_string[7] = ' '; //creates space
    alarm_string[8] = ' '; //creates space
    alarm_string[9] = ' '; //creates space
    alarm_string[10] = (char) 65; //Ascii code 65 = A
    alarm_string[11] = (char) 76; //Ascii code 76 = L
    alarm_string[12] = (char) 65; //Ascii code 65 = A
    alarm_string[13] = (char) 82; //Ascii code 82 = R
    alarm_string[14] = (char) 77; //Ascii code 77 = M
    alarm_string[15] = 0; //Ends the string by placing 0 in the last string

    DisplayString(line, alarm_string); //Sends the string to the LCD on line 0
}

}

void Delay(int t) // Start of function
{
    int i; /*(A local variable)This declares an int variable, used to store whole numbers*/
    for(i=0;i<t;i++); /* for statement creates a loop as long as inside the brackets remain true,
    i=0("setup"), i<t("as long as"), i++("do this")increment by 1,
    loop will continue to do nothing creating a delay*/
}

void DisplayState(int line)
{
    char state_string[8] = {0, 0, 0, 0, 0, 0, 0, 0};
    state_string[0] = ' ';
    state_string[1] = ' ';
    state_string[2] = ' ';
    state_string[3] = ' ';
    state_string[4] = ' ';
    state_string[5] = ' ';
    state_string[6] = (char) state + 48;
    state_string[7] = 0;

    DisplayString(line, state_string);
}

```

Figure 23 - Coding (Section 13)

In this final section of the code the DisplayAlarmTime and DisplaySetAlarmTime loops are included – these are there to put the word “ALARM” on the LCD when the alarm is on when the clock is counting normally and when setting the alarm. At the bottom of the program is a void delay, which is used to constantly loop, and a DisplayState loop, which is used as a template of how many LCD segments are used.

5. Testing and Analysis of Results

The testing process for the system was split up into two section – simulation and practical. The majority of the testing and evidence was gathered using simulations, and the practical was carried out to prove that the program and system design would actually work.

Simulation

The first to be tested was the simulation, which was done by testing the system that was designed on Proteus along with the code that was developed on MPLAB. The figures and captions below explain the testing process for the simulated alarm.

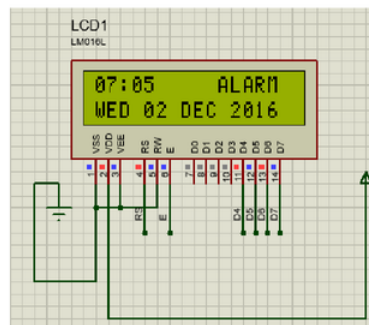


Figure 24 - Setting the Alarm to 7:05

The alarm was tested by first setting the alarm to 07:05. It was set to that value by pressing the RA1 button to enter the alarm altering mode, then pressing RA2 to choose the alarm hours, pressing RA3 to increment to the desired alarm hours, pressing RA2 to choose the alarm minutes, then pressing RA3 to increment to the desired alarm minutes.

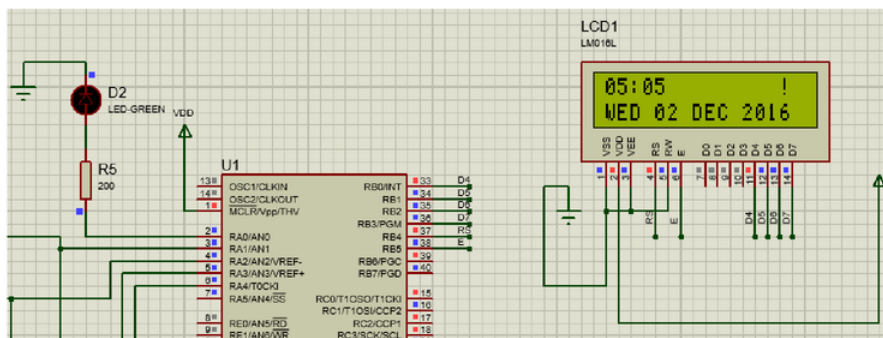


Figure 25 - Waiting for the Alarm

The above figure shows is a screenshot of the alarm clock counting up to the alarm clock's set hours/minutes, the alarm is turned on using RA3 while the clock is counting and is indicated by an exclamation point (!) on the interface. It can be seen that the LED has not yet been triggered, as the clock hasn't reached the alarm time.

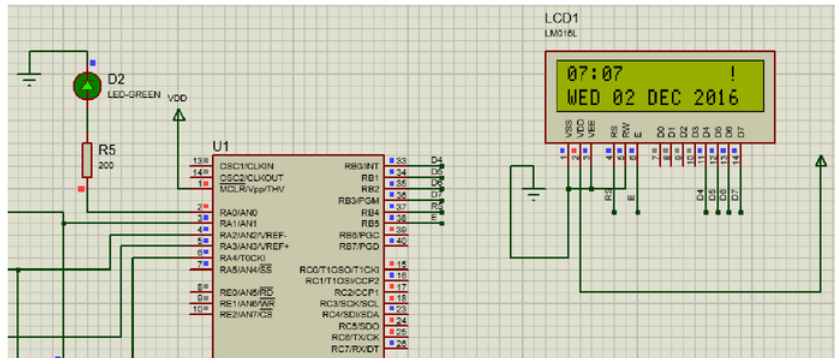


Figure 26 - Alarm Activated

As soon as the clock reached the same value as the alarm clock's value, the LED connected to RA0 lit up, indicating that the alarm had been activated.

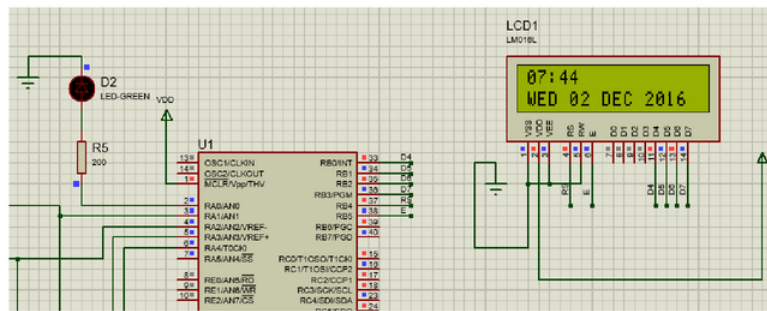


Figure 27 - Turning off Alarm

The alarm is turned off using RA3.

A testing table was drawn up to test each of the specific coded functions of the alarm, that testing table is shown in the following two pages.

Current state	Test procedure	Expected result	Actual result	Test pass
0	Display normal time and date on screen	Display normal time and date on screen	Displayed normal time and date on screen	Yes
1	Increment hours by 1	Increment hours by 1 when Ra3 is pressed (Loops from 23 to 00)	Incremented hours by 1 when Ra3 is used (Loops from 23 to 00)	Yes
1	Decrement hours by 1	Decrement hours by 1 when RA4 is pressed (Loops from 00 to 23)	Decrement hours by 1 when RA4 is pressed (Loops from 00 to 23)	Yes
2	Increment minutes by 1	Increment minutes by 1 when Ra3 is pressed (Loops from 59 to 00)	Incremented minutes by 1 when Ra3 is pressed (Loops from 59 to 00)	Yes
2	Decrement minutes by 1	Decrement minutes by 1 when RA4 is pressed (Loops from 00 to 59)	Decrement minutes by 1 when RA4 is used (Loops from 00 to 59)	Yes
3	Increment day by 1	Increment day by 1 when Ra3 is pressed (Loops from SUN to MON)	Incremented day by 1 when Ra3 is used (Loops from SUN to MON)	Yes
3	Decrement day by 1	Decrement day by 1 when RA4 is pressed (Loops from MON to SUN)	Decrement day by 1 when RA4 is pressed (Loops from MON to SUN)	Yes
4	Increment date by 1	Increment date by 1 when Ra3 is pressed (loops from end of month to beginning)	Incremented date by 1 when Ra3 is pressed (loops from end of month to beginning)	Yes
4	Decrement date by 1	Decrement date by 1 when RA4 is pressed (loops from beginning of month to end)	Decrement date by 1 when RA4 is pressed (loops from beginning of month to end)	Yes

5	Increment month by 1	Increment month by 1 when Ra3 is pressed (Loops from DEC to JAN)	Incremented month by 1 when Ra3 is pressed (Loops from DEC to JAN)	Yes
5	Decrement month by 1	Decrement month by 1 when RA4 is pressed (Loops from JAN to DEC)	Decrement month by 1 when RA4 is pressed (Loops from JAN to DEC)	Yes
6	Increment year by 1	Increment year by 1 when Ra3 is pressed	Incremented year by 1 when Ra3 is pressed	Yes
6	Decrement year by 1	Decrement year by 1 when RA4 is pressed	Decrement year by 1 when RA4 is pressed	Yes
7	Save the modified values to the current 'normal' display by pressing RA1 from states 1 to 6 and 9 to 10	Save the modified values to the current 'normal' display, returning to the 'normal' display screen after a short delay. All states working correctly.	Save the modified values to the current 'normal' display, returning to the 'normal' display screen after a short delay. All states worked correctly.	Yes
8	Displays the alarm time and date	When RA1 is pressed from state 0 the alarm time display should be shown	When RA1 was pressed from state 0 the alarm time display was shown	Yes
9	Increment alarm hours by 1	Increment hours by 1 when Ra3 is pressed (Loops from 23 to 00)	Incremented hours by 1 when Ra3 is used (Loops from 23 to 00)	Yes
9	Decrement alarm hours by 1	Decrement hours by 1 when RA4 is pressed (Loops from 00 to 23)	Decrement hours by 1 when RA4 is used (Loops from 00 to 23)	Yes
10	Increment alarm minutes by 1	Increment minutes by 1 when Ra3 is pressed (Loops from 59 to 00)	Incremented hours by 1 when Ra3 is used (Loops from 59 to 00)	Yes
10	Decrement alarm minutes by 1	Decrement minutes by 1 when RA4 is pressed (Loops from 00 to 59)	Decrement hours by 1 when RA4 is used (Loops from 00 to 59)	Yes

Evidence

Evidence of incrementing and decrementing values is shown in the following figures.

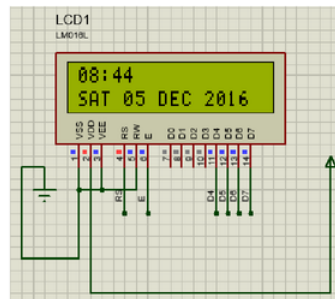


Figure 28 - Paused Values

The above figure shows values of the clock once the clock was arbitrarily paused, the values here are a placeholder to compare to incrementing values of the clock.

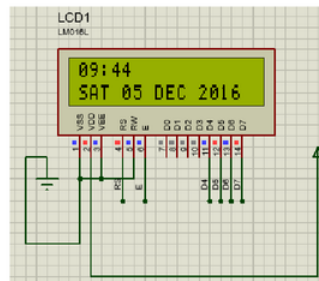


Figure 29 - Incrementing Hour

The above figure shows that the hours have been incremented by 1 compared to the original paused values.

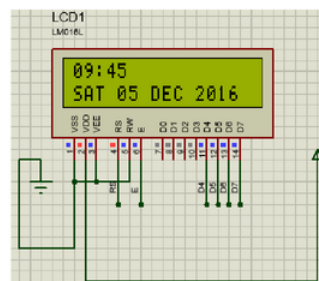


Figure 30 - Incrementing Minutes

The above figure shows that the minutes have been incremented by 1 compared to the previous figure.

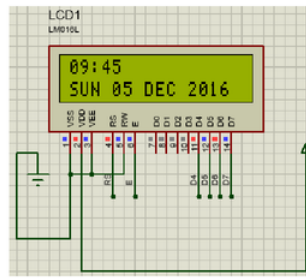


Figure 31 - Incrementing Day

The above figure shows that the day has been changed from Saturday to Sunday, which is the day being incremented by 1.

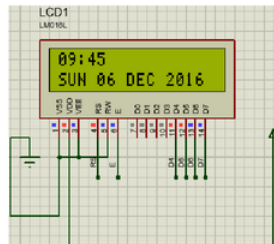


Figure 32 - Incrementing Date

The above figure shows that the date has been incremented from 05 to 06.

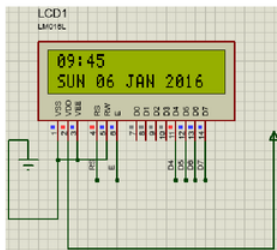


Figure 33 - Incrementing Month

The above figure shows that the month was changed from December to January, showing that the month has been incremented by 1.

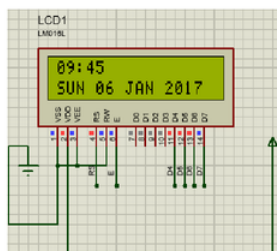


Figure 34 - Incrementing Year

The above figure shows evidence of the year being incremented by 1.

Earlier in the report (in the research section), it was explained that setting the TMR0 in the code to 63 would result in a 10ms cycle - in order to test this, an oscilloscope was connected to the circuit.

```
#include <pic.h>

int milli=0, secs=0; //Must be declared gl

void interrupt ISR (void)
{
    if(TOIF==1)
    {
        TOIF = 0;
        TMR0 = 63;
        // milli += 1;
        // if(milli==100)
        {
            // secs ++;
            // milli = 0;
            RB0 = ~RB0; // Toggle RB0 pin
        }
    }
}
```

Figure 35 - Showing that TMR0 = 63

The above code is what was used to achieve a timer cycle of 10ms.

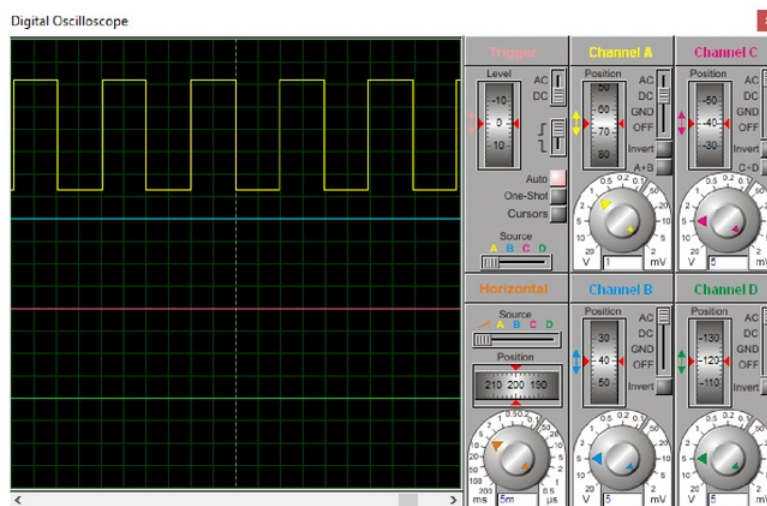


Figure 36 - Proof that TMR0 = 10ms

The oscilloscope above shows that the oscilloscope is down for two squares and up for two squares – each square is calibrated to show 5ms, and so the timer has a cycle of 10ms.

```
void interrupt ISR (void)
{
    if(TOIF==1)
    {
        TOIF = 0;
        TMRO = 63;
        milli += 1;
        if(milli==100)
        {
            secs ++;
            milli = 0;
            RB0 = ~RB0; // Toggle RB0 pin
        }
    }
}
```

Figure 37 - Changing the 10ms to 1s

The above code included an additional couple of lines within the if statement that changes the 10ms into 1s.

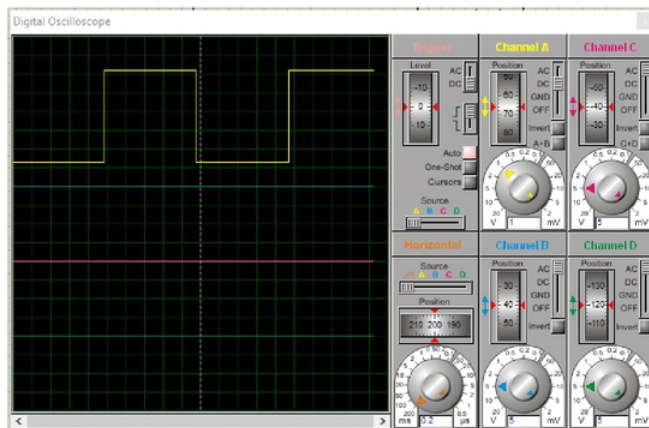


Figure 38 - 1s Cycle

The squares in the above oscilloscope screenshot are set to space at 200ms, the cycle takes five squares, and so $200\text{ms} \times 5 = 1\text{s}$, proving that it is indeed 1s.

Practical

The practical for this assignment involved uploading the code to the board with the PIC and the LCD connections. The following figure shows the calendar alarm clock actually function correctly – in the figure, the design is in the alarm-setting mode, as indicated by the text “ALARM” on the top right of the LCD.



Figure 39 - Functioning Practical Board

6. Conclusions and Recommendations

The completed system functioned as a digital calendar alarm clock and had multiple essential features:

- An LCD which displayed the time, date, day, month and year, as well as details such as whether the alarm was on or off
- The time and date could be set on the system without the need for additional hardware
- The alarm could be set to any given time to the hours and minute
- The alarm functionality could be turned off and on
- A functioning alarm

These functions all play into a very robust and effective calendar alarm clock, thus meeting the aims and objectives of the assignment.

One recommendation to improve the system would be to add additional functionality to the alarm, as the current design only allows the user to set the alarm to specific hours and minutes; it could be upgraded to allow the alarm to be set to specific dates, days or months. Another recommendation that could be made, which would improve the functionality and make the design process easier would be to use a microcontroller such as an Arduino instead of the PIC, as Arduinos have much more online support due to their popularity and in my opinion are more intuitive, with more libraries and functionalities at the ready, allowing for a faster development process.

7. References

4. *Timers*. (2018). Retrieved from MikroElektronika: <https://www.mikroe.com/ebooks/pic-microcontrollers-programming-in-assembly/timers>
- Datasheet. (2018). *PIC16F877 Data Sheet*. Retrieved from Microchip: <http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf>

Assignment 1

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

54/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

Text Comment. reasonable intro. A diagram would be useful here.

PAGE 5

PAGE 6

PAGE 7

Text Comment. Research should extend further with more relevant sources.

PAGE 8

PAGE 9

Text Comment. State diagram is clear and well presented.

PAGE 10

PAGE 11

PAGE 12

PAGE 13

Text Comment. The configuration is explained and the hardware components are illustrated. This section could be structured in a better way because this will cause confusion with the test section.

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

Text Comment. What about state 10.

PAGE 20

Text Comment. State code is explained fairly well. What about the while statement at the end. This is to counter switch bounce.

Text Comment. Must include the 'DisplayString' function call at the end here because it is the function used in the LCD display.

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

JOSEFF JONES

0

RESEARCH-INFORMED LITERATURE



Needs more research and relevant sources. The research should extend further than the config registers. Alternative MCU's and limitations could be included.

9/25

KNOWLEDGE AND UNDERSTANDING



Intro is a little short but there is an intro. 2/5 The system design and explanation is reasonable. This could have been improved with more in depth explanations. 6/10 The software is explained but more detail is needed. 6/10

14/25

ANALYSIS



Testing is good 7/10 Conclusion has got some valid points. 5/10

12/20

PRACTICAL APPLICATION AND DEP.



Software explanation is good. however there are sections that could be clearer. 6/10 Testing is good. 7/10

13/20

SKILLS FOR PROFESSIONAL PRAC.



Report is ok.3/5 Team work is good 3/5

6/10