

Assignment 1 Joshua Beer

by Joshua Beer

FILE	12940_JOSHUA_BEER_ASSIGNMENT_1_JOSHUA_BEER_139109_986936 294.DOCX (6.55M)		
TIME SUBMITTED	01-FEB-2018 10:53PM (UTC+0000)	WORD COUNT	5697
SUBMISSION ID	82687993	CHARACTER COUNT	33659



Calendar Alarm Clock

Configuration and Programming of Embedded System

Joshua Beer

PSDE5013

01/02/2018

Contents

.....	1
1. Introduction.....	5
2. Research	6
3. Design	10
4. Testing and Analysis of Results.....	41
5. Conclusions and Recommendations	54
6. Appendix and Bibliography	56
Bibliography	56
Figure 1 - TIMER0.....	6
Figure 2 - Option Register for PIC16F877	7
Figure 3 - PIC16F877 PIN Diagram	9
Figure 4 - Sequence of Design.....	10
Figure 5 - State Diagram of Design	11
Figure 6 - Code Function Prototype	12
Figure 7 - Code Global Variables	12
Figure 8 - Code Interrupt	13
Figure 9 - Code Main Function	14
Figure 10 - Code Starts Super Loop.....	15
Figure 11 - Code Display State (1)	16
Figure 12 - Code Case 0	17
Figure 13 - Code Case 1	18
Figure 14 - Code Case 2	20
Figure 15 - Code Case 3	20
Figure 16 - Code Case 4 (1).....	22
Figure 17 - Code Case 4 (2).....	23
Figure 18 - Code Case 5	24
Figure 19 - Code Case 6	25
Figure 20 - Code Case 7	25
Figure 21 - Code Case 8	26

Figure 22 - Code Case 9	26
Figure 23 - Code Case 10 and End of Super Loop	27
Figure 24 - Code Display Set Time	28
Figure 25 - Code Display Time.....	29
Figure 26 - Code Delay Function	30
Figure 27 - Code Display State Function.....	30
Figure 28 - Code Display Date Day.....	31
Figure 29 - Code Display Date Month (1)	33
Figure 30 - Code Display Date Month (2).....	34
Figure 31 - Code Display Date Placing On LCD	35
Figure 32 - Code Display Set Date (1)	36
Figure 33 - Code Display Set Date (2)	37
Figure 34 - Code Display Set Date (3)	38
Figure 35 - Code Display Set Date (4)	39
Figure 36 - Code Display Alarm Time	39
Figure 37 - Code Display Set Alarm	40
Figure 38 - Testing Table (1).....	42
Figure 39 - Normal Clock	43
Figure 40 - Setting Clock Hours	43
Figure 41 - Setting Clock Minutes	43
Figure 42 - Setting Clock Days.....	44
Figure 43 - Setting Clock Date	44
Figure 44 - Setting Clock Month.....	44
Figure 45 - Setting Clock Year	45
Figure 46 - Alarm Not Set.....	45
Figure 47 - Alarm Set	45
Figure 48 - Setting Alarm	46
Figure 49 - Setting Alarm Hours.....	46
Figure 50 - Setting Alarm Minutes.....	46
Figure 51 - Testing Table (2).....	47
Figure 52 - Alarm Gone Off.....	47
Figure 53 - Testing Leap Year.....	48
Figure 54 - Clock Pulse at 10ms	48
Figure 55 - Oscilloscope 10ms.....	49

Figure 56 - Clock Pulse at 1 Second	50
Figure 57 - Oscilloscope 1 Second	50
Figure 58 - Normal Clock Practical.....	51
Figure 59 - Alarm on Practical.....	52
Figure 60 - Setting Alarm Practical.....	52
Figure 61 - Setting Alarm Hours Practical	53

1. Introduction

The aims and objectives of this assignment are to be able to create and design an alarm clock using an embedded system based on a microcontroller using the configurable on-chip peripherals. The design will then be put onto a PIC 16F877 microcontroller and a HP488 development board. This will enable the design of a digital calendar alarm clock which has two lines of LCD which is shown on the device and 3 pushbuttons to alter and change the clock.

To be able to do this there will be a specification to follow, this is as follows:

Setting the time and date;

This must allow the user to press button 2 which allows them to alter the hour of the clock by toggling button 3. To toggle through the rest of the time changes (minutes, days, date, month and year) the user must press button 2 again and 3 to change whichever setting they are on. By pressing button 1 the changes made are saved and the clock will resume from the altered time.

Setting the alarm time;

This must allow the user to press button 1 which will display the alarm time that is already set. By pressing button 2 the user can select whether to change the alarm hours or minutes, this can then be toggled and changed by pressing button 3. By pressing button 1 the changes made are saved and the new alarm set and the clock will resume. It must also display ALARM on the LCD display when setting the alarm time.

Turning the alarm on/off;

When in normal mode i.e. the clock must be running and not in a setting time/alarm mode, by pressing button 3 will turn the alarm on/off. To make sure this is successful an exclamation mark (!) will appear on the LCD.

If all these criteria are met then alarm design will be considered a success.

2. Research

To be able to build a clock, research must be carried out into how a clock can accurately track time and how the microcontroller can aid this.

In this assignment the TIMER0 function on the microcontroller is being used to control the accurate timekeeping for the alarm clock, this utilises an interrupt service routine (ISR).

How this TIMER0 function utilises the other functions of the microcontroller is explained below using figure 1 as reference.

```
TRISA = 0x1E;           //Configure port A = 1111 1110
TRISB = 0x00;           //Configure port B = 0000 0000
TRISC = 0x00;           //Configure port C = 0000 0000
ADCON1 = 0x07;          //Configure ADCON1 = 0000 0111
OPTION_REG = 0x87;      //Configure OPTION_REG = 1000 0111
INTCON = 0xA0;          //Configure INTCON = 1010 0000
TMR0=255;               //Pre-loading TMR0
```

Figure 1 - TIMER0

The figure above shows the on-chip peripherals which are utilised.

TRISA; this is set to 0x1E (E = 14 for the right hand 4 binary digits) which sets all inputs for the switches on the board and output for the alarm indicator.

TRISB; this is set to 0x00 which sets all the LCD to outputs.

ADCON1; this is set to 0x07 which outputs RA0 and inputs for RA1, RA2 and RA3.

INTCON; this is set to 0xA0 which sets the global interrupts and the TRM0 as enabled (bit 7 and 5 respectively) while the peripheral interrupt is disabled by bit 6.

(Romux, 2010)

OPTION_REG; this is set to 0x87 which sets the prescaler rate and prescaler assignment bits on (0-3) which is 111 and 1:256 (seen in figure 2 below). It also sets the pull-up resistors to disabled.

REGISTER 5-1: OPTION_REG – OPTION REGISTER (ADDRESS: 81h OR 181h)

	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
	RAPU	INTEDG	T0CS	T0SE	PSA	PS2	PS0
bit 7							bit 0
bit 7	RAPU: PORTA Pull-up Enable bit 1 = PORTA pull-ups are disabled 0 = PORTA pull-ups are enabled by individual port latch values in WPUA register						
bit 6	INTEDG: Interrupt Edge Select bit 1 = Interrupt on rising edge of RA2/INT pin 0 = Interrupt on falling edge of RA2/INT pin						
bit 5	T0CS: TMR0 Clock Source Select bit 1 = Transition on RA2/T0CKI pin 0 = Internal instruction cycle clock (CLKOUT)						
bit 4	T0SE: TMR0 Source Edge Select bit 1 = Increment on high-to-low transition on RA2/T0CKI pin 0 = Increment on low-to-high transition on RA2/T0CKI pin						
bit 3	PSA: Prescaler Assignment bit 1 = Prescaler is assigned to the WDT 0 = Prescaler is assigned to the Timer0 module						
bit 2-0	PS<2:0>: Prescaler Rate Select bits						
	Bit Value	TMR0 Rate	WDT Rate ⁽¹⁾				
	000	1 : 2	1 : 1				
	001	1 : 4	1 : 2				
	010	1 : 8	1 : 4				
	011	1 : 16	1 : 8				
	100	1 : 32	1 : 16				
	101	1 : 64	1 : 32				
	110	1 : 128	1 : 64				
	111	1 : 256	1 : 128				

Note 1: A dedicated 16-bit WDT postscaler is available for the PIC16F688. See Section 11.7 “Watchdog Timer (WDT)” for more information.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Figure 2 - Option Register for PIC16F877

(Lab)

Once the peripherals are set on the microcontroller the scaling for a 1 second time count can now be added to the microcontroller.

The oscillating frequency being used in the microcontroller is 19.6608MHz. There are 4 pulses per complete cycle so therefore this number must be divided by 4 which $19,6608/4 = 4915200\text{Hz}$. As the system overflows at a value of 256 (this is why bit 5 is enabled in INTCON) a prescaler value is chosen, this is chosen to be the bit value of 111, which is the maximum prescaler value for the chip being used (1:256, shown in **Error! Reference source not found.**). The cycle time was previously calculated at 4915200Hz and therefore the increment time can be calculated.

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

For a 10ms count the increment time is divided into the count increment (with a 2 subtracted to account for error).

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

The preload count is then subtracted from the value of the prescaler to achieve the count that needs to be loaded into TMR0:

$$256 - 193 = 63$$

The testing and results of this will be shown below in task 3 and how the 10ms interrupt cycle displayed above will be turned into a 1 second count will be explained.

The assignment also uses the PIC16F887 as already mentioned above, this is the chip for the microcontroller which uses the peripherals stated and explained above.

The layout of a PIC16F877 is shown below in figure 3.

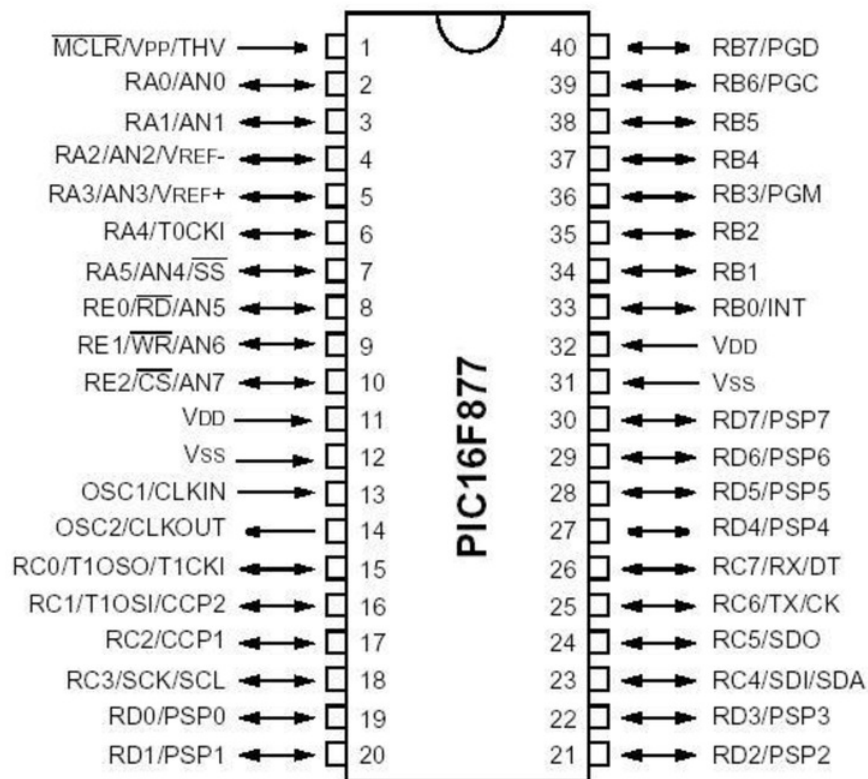


Figure 3 - PIC16F877 PIN Diagram

(Engineering)

Figure 3 shows that the PIC has 4 configurable input/output ports which are RA, RB, RC and RD. These can be used to program certain areas of the microcontroller and this will be shown in the assignment itself.

The main features of the PIC needed for the design are that it has enough I/O ports for all the buttons and clock function and it has enough memory space to be able to run the clock while storing new times for the clock itself and alarm times.

3. Design

The system design relies on pushing buttons to allow the clock time to be altered and the alarm time to be set and turned on/off. This design incorporates only 3 buttons which when pressed in a specific sequence will allow this to be achieved, the table below will show the outcome of each sequence of button pushes.

State	Action	Buttons			
		RA1	RA2	RA3	RA4
0	Normal function of clock which will display time and date, also if the alarm is turned on it will display a !	Sate 8	State 1	Turns alarm on/off	N/A
1	Function which allows the hours to be set.	State 7	State2	Increment hours	decrement hours
2	Function which allows the minutes to be set.	State 7	State 3	Increment minutes	decrement minutes
3	Function which allows the days to be set.	State 7	State 4	Increment days	decrement days
4	Function which allows the date to be set.	Sate 7	State 5	Increment date	decrement date
5	Function which allows the month to be set.	Sate 7	State 6	Increment month	decrement month
6	Function which allows the year to be set.	Sate 7	State 0	Increment years	decrement years
7	Save the new clock/alarm time and implements it.	N/A	N/A	N/A	N/A
8	Function which allows the alarm time to be altered and set.	State 0	State 9	N/A	N/A
9	Function which allows alarm hours to be altered.	State 7	State 10	Increment hours	N/A
10	Function which allows alarm minutes to be altered.	State 7	State 8	Increment minutes	N/A

Figure 4 - Sequence of Design

Figure 4 above shows the sequence of buttons pushed and the action which they cause. Once this has been produced a state diagram can follow which makes it easier to visualise what each button and setting will do.

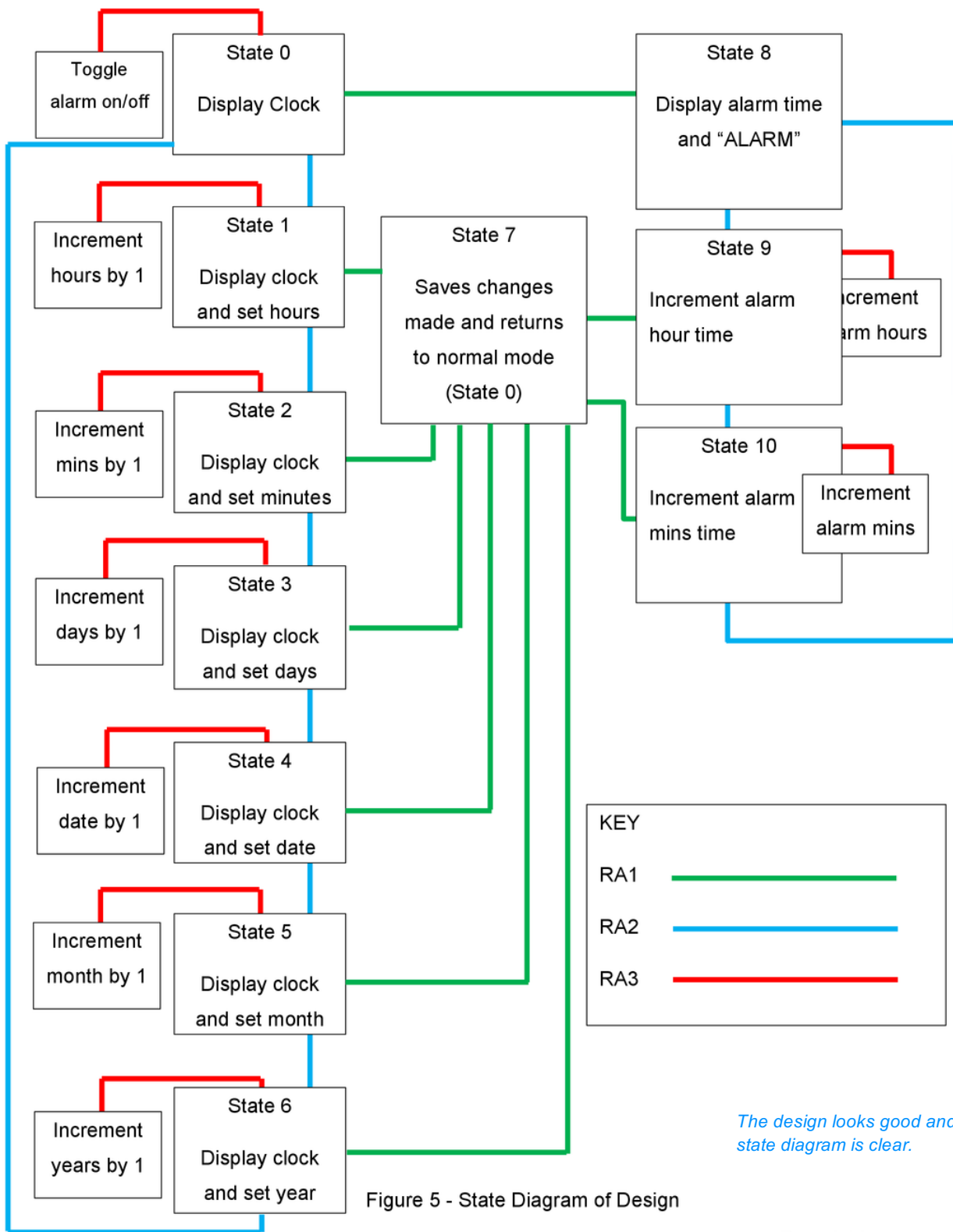


Figure 5 - State Diagram of Design

The design looks good and the state diagram is clear.

Figure 5 above shows the state diagram for the design which gives a better visualisation of how each state in the design will work and lead onto the next state.

This state diagram will also aid the design and writing of the code which is needed for the clock to work, this will be show and explained in the next part of the task below.

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

void DisplayTime(int line); //Function prototype
void DisplaySetTime(int line); //Function prototype
void DisplayState(int line); //Function prototype
void DisplayDate(int line); //Function prototype
void DisplaySetDate(int line); //Function prototype
void DisplayAlarmTime(int line); //Function prototype
void DisplaySetAlarmTime(int line); //Function prototype

void Delay(int t); //Function prototype
```

Figure 6 - Code Function Prototype

The code above shows the header files which are used from MPLAB and utilised in the design of the clock, these are pic.h and LCD.h.

The function prototypes are then defined to all be integers, this will allow the code to be modified further along and for the user/reader to know what each prototype is doing.

```
int milli=0, secs=0, mins=50, hours=23, day_code=2, days=1, month=12, //global variables
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;
```

Figure 7 - Code Global Variables

The code in figure 7 shows the global variables which will be initially used when the clock is started, these can be altered and changed when called upon and will be overwritten when needed. Note that all these global variables are integers as they all require whole numbers to function correctly.

```

void interrupt ISR (void)                                //Setting the clock to work
{
    if(TOIF==1)
    {
        TOIF = 0;                                        //Reset interrupt flag, set to zero
        TMRO = 255;                                     //Preload timing value
        milli += 10;                                    //Increment in milliseconds
        if(milli==100)                                  //If milliseconds equals 100
        {
            secs ++;                                    //Increment seconds
            milli = 0;                                  //Reset milliseconds to 0
            if(secs==60)                                 //If seconds equal 60
            {
                mins ++;                                //Increment mins
                secs = 0;                                //Set seconds to 0
                if(mins==60)                             //If mins equal 60
                {
                    hours ++;                            //Increment hours
                    mins = 0;                            //Set mins to 0
                    if(hours==24)                        //If hours equal
                    {
                        days ++;                          //Increment days
                        day_code ++;                     //Increment day_code

                        hours = 0;                        //Hours equal 0

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

```

Figure 8 - Code Interrupt

The code in figure 8 is showing how the clock is being set up. Therefore if TOIF is equal to 1 then the loop will start again as TOIF is then set to 0 thus resetting the interrupt flag. As shown in the research the pic itself can only have a maximum of 10ms as its cycle therefore this part of the code will make this a 1 second pulse.

It's done by using the value calculated for TMR0 (255) and setting it as the timing value and then incrementing milliseconds by 1 (figure shows 10 because it is sped up for testing but Figure 56 shows 1) until milliseconds equal 100 (1 second).

After this loop has finished the seconds are incremented by one and milliseconds reset to 0 to allow the seconds to be incremented again and therefore this is the fundamentals of the clock.

If seconds equals 60 then the minutes are incremented by 1 and seconds reset to 0.

This is repeated in the code for minutes and therefore hours increment by 1 and minutes set to 0.

If hours equals 24 then days are incremented by 1 and hours set to 0.

If days are greater than 31 then the month is incremented by 1 and the days set to 1.

If months are greater than 12 the year is incremented by one and month set to 1.

Every time hours equals 24 then day code will increment by one and if day code equals 7 then day code is set back to 0.

This is the basic clock function and therefore will keep running and incrementing the time until told otherwise. There is of course exceptions to these rules and these are shown in the code below.

This is explained well.

```
////////////////////////////////////  
void main(void)                                //Main Function  
{  
  
    TRISA = 0x1E;                               //Configure port A = 1111 1110 Input switches and output alarm indicator (E = 14)  
    TRISB = 0x00;                               //Configure port B = 0000 0000 all outputs LCD  
    TRISC = 0x00;                               //Configure port C = 0000 0000 port C not used  
    ADCON1 = 0x07;                             //Configure ADCON1 = 0000 0111 output for RA0 and inputs for RA1, 2 and 3  
    OPTION_REG = 0x87;                         //Configure OPTION_REG = 1000 0111  
    INTCON = 0xA0;                             //Configure INTCON = 1010 0000 (A = 10)  
    TMR0 = 255;                                //Pre-loading TMR0  
  
    LCD_Startup();                             //Function header, runs function in LCD.c
```

Figure 9 - Code Main Function

The code in figure 9 has been explained in the research section of the assignment apart from the LCD_Startup which will run the function head in the LCD.c file.

*Should point out here that
the main function is where
the configuration is included.*

```

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 is greater than 31
        {
            month++; //Increment month
            days = 1; //Set days to 1
        }

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

    if((month==12) && (days>31))
        //If month equals 12 and days is greater than 31
        {
            year++; //Increment year
            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 10 - Code Starts Super Loop

Figure 10 shows the start of the super loop for the main function.

This is setting out the exceptions to the rules for days in the month, this means that months with less than 31 days (months 4, 6, 9, 11) will increment the month on the 30th day. Unless it's month 2 and not a leap year then the month will increment after the 28th day. After these have finished the day will then set back to 1.

Month 12 will increment on the 31st day but this will then set the months and days back to 1 and increment the year by 1.

```

////////////////////// DisplayState(1); //////////////// displays the state on line 1 of LCD
if(alarm==1) //If alarm equals 1 step into this part of the code
{
    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 to next part of code
        {
            RA0 = 1; //Set output for RA0 to 1 (this turns the alarm on if the time is set is met)
        }
}
if(alarm==0) //If alarm equals zero step in to next bit of code
{
    RA0 = 0; //Set output RA0 to 0
}

```

Again if the day code equals 7 then the day code is set back to 0.

Figure 11 - Code Display State (1)

Figure 11 above shows the code for turning the alarm on. If the alarm is equal to 1 (alarm is turned on) and the alarm hours and alarm minutes match the time on the clock then RA0 is set to output 1 (turning on RA0 output) and thus turning the alarm on.

This is correct.

If the alarm equals 0 then RA0 is set to 0 and the alarm is then off.

```
case 0:                                     //Case 0 will run if the 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 (sets the alarm)
    if(RA2) state = 1;                     //If RA2 is equal to 1 set state to equal 1 (sets the clock)

    if(RA3==1)                             //If RA3 is equal to 1 step in (this is turning the alarm on and off)
    {
        if(alarm==0)                       //If alarm equals 0
        {
            alarm = 1;                     //Sets alarm to equal 1
        }
        else                               //Else command, if alarm is not equal to zero step in to next part of code
        {
            alarm = 0;                     //Sets alarm to 0
        }
    }

    break;                                 //This will break out from the case it's in
```

Figure 12 - Code Case 0

Figure 12 shows the code for case 0, this will run if the state of the clock is also 0.

The function header will display the time on line 0 of the LCD and will display the date on line 1 of the LCD.

The next part of the code is setting the clock time to equal any temporary time the user has set by making the temp_hours equal to hours etc.

If RA1 is pressed then the state will equal 8 and therefore enter case 8 in the code to set the alarm.

If RA2 is pressed then the state will equal 1 and therefore enter case 1 in the code to set the clock.

If RA3 is pressed then the alarm will be turned on/off dependant on what state it was already in.

```
case 1:                                     //Case 1, run if state equals 1 (this case is setting the hours by pressing RA3 or RA4)
    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
        if(temp_hours>23) temp_hours=0; //If temp_hours is greater than 23 set to 0
    }
    if(RA4)                                //If RA4 = 1
    {
        while(RA4);
        Delay(100);

        temp_hours--;                     //deccrement temp_hours
        if(temp_hours<0) temp_hours=23; //If temp_hours is greater than 23 set to 0
    }

    break;                                //This will break out from the case it's in
```

Break will then break out from the case it's in.

Figure 13 - Code Case 1

Figure 13 shows the code for case 1, this will be entered if the state equals 1 and allows the user to set the hours of the clock.

The time will be shown on the LCD while in this state, running the displaytime prototype.

If RA1 is pressed then the state will equal 7 and case 7 will be entered which will save any changes to the time.

If RA2 is pressed then the state will equal 2 and case 2 will be entered which will allow the user to change the minutes for the clock.

If RA3 is pressed it will increment the hours for the clock by 1 and if the hours are greater than 23 then temp_hours will equal 0.

The next bit of code was added as an extra for the assignment which will allow the hours to be decremented by 1 by pressing RA4, there is a delay of 100 for this.

Again break will then break out from the case it's in.


```

case 2:                                     //Case 2, run if state equals 2 (this case is setting the minutes by pressing RA3 or RA4)
    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
        if(temp_mins>59) temp_mins=0;      //If temp_mins is greater than 59 set to 0
    }

    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 0
    }
    break;                                 //This will break out from the case it's in

```

Figure 14 - Code Case 2

Figure 14 shows the code for case 2, this will run if the state equals 2 and allows the user to set the minutes for the clock.

This has exactly the same principle of code as explained above for figure 13 but now pressing RA2 will make the state equal 3 and enter case 3.

```

case 3:                                     //Case 3, run if state equals 3 (this case is setting the days by pressing RA3 or RA4)
    DisplaySetDate(1);                     //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
    {
        temp_day_code++;                   //Increment temp_day_code
        if(temp_day_code>6) temp_day_code=0; //If temp_day_code is greater than 6 set to 0
    }

    if(RA4)                                //If RA4 = 1
    {
        while(RA4);
        Delay(100);

        temp_day_code--;                   //decrement temp_day_code
        if(temp_day_code<0) temp_day_code=6; //If temp_day_code is less than 0 set to 6
    }

    break;                                 //This will break out from the case it's in

```

By pressing RA3 the minutes are incremented by 1 and RA4 to decrement by 1.

Figure 15 - Code Case 3

Figure 15 shows the code for case 3, this will run if the state equals 3 and allows the user to set the days for the clock.

This has exactly the same principle of code as explained above for figure 14 but now will display the date on LCD line 1 and by pressing RA2 will make the state equal 4 and enter case 4.

By pressing RA3 the days will increment by 1 and pressing RA4 will decrement by 1.

```
case 4:                                     //Case 4, run if state equals 4 (this case is setting the date (days numerical) by pressing RA3 or RA4)
  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
  {
    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))
    {
      temp_days = 1;                       //If temp_month equals 1 or 3 or 5 or 7 or 8 or 10 and temp_days greater than 31
      //Set days to 1
    }

    if(((temp_month==2)&&(temp_days>28)&&(temp_year% 4 != 0))
    {
      temp_days = 1;                       //If temp_month equals 2 and temp_days greater than 28 and 4 does not divide into year then set days to equal 1
      //Set days to 1
    }

    if(((temp_month==2)&&(temp_days>29)&&(temp_year% 4 == 0))
    {
      temp_days = 1;                       //If temp_month equals 2 and temp_days greater than 29 and 4 divides into year step in because it's a leap year
      //Set days to 1
    }

    if(((temp_month==4)||(temp_month==6)||(temp_month==9)||(temp_month==11))&&(temp_days>30))
    {
      temp_days = 1;                       //If temp_month equals 4 or 6 or 9 or 11 and temp_days greater than 30
      //Set days to 1
    }

    if(RA4)                                //If RA4 is equal to 1
    {
      while(RA4);
      Delay(100);
      temp_days--;                         //decrement days
    }
  }
}
```

If the temp_day_code is greater than 6 it will be set to 0.

Figure 16 - Code Case 4 (1)

```

if(((temp_month==1)||((temp_month==3)||((temp_month==5)||((temp_month==7)||((temp_month==9)
||((temp_month==10)||((temp_month==12)||((temp_days<1)))
//If temp_month equals 1 or 3 or 5 or 7 or 9 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 because it's not a leap year
{
    temp_days = 28;          //Set days to 28
}
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 because it's a leap year
{
    temp_days = 29;          //Set days to 29
}
if(((temp_month==4)||((temp_month==6)||((temp_month==8)||((temp_month==10)||((temp_month==12)||((temp_days<1)))
//If temp_month equals 4 or 6 or 8 or 10 or 12 and temp_days greater than 30 step in
{
    temp_days = 30;          //Set days to 30
}

}

//end if RA4

break;                      //This breaks out from each case

```

Figure 17 - Code Case 4 (2)

Figures 16 and 17 show the code for case 4, this is entered if the state equals 4. This allows the user to set the date (numerical days) for the clock.

This code is slightly different as it includes the exception for the month of February (month 2) as every 4 years is a leap year. This is explained below.

If RA1 is pressed the state equals 7 and case 7 is entered and the time is saved.

If RA2 is pressed the state equals 5 and case 5 is entered which allows the months to be altered for the clock.

If RA3 is pressed then the date is incremented by 1. The exceptions for the months with less than 31 days have been explained above except for the leap year exception, therefore;

If the month equals 2 and the year is divisible by 4 then month 2 must have a date greater than 29 for the month to increment by 1. This allows there to be 29 days in February every 4 years.

Again the date can be decremented by pressing RA4 and the case will break out from the case it's in if the state equals something else.

```

case 5:                                     //Case 5, run if state equals 5 (this case is setting the month by pressing RA3 or RA4)
  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 = 6;                       //If RA2 is equal to 1 set state to equal 6
  if(RA3)                                  //If RA3 is equal to 1
  {
    temp_month++;                          //Increment temp_month
    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_month_code
    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 18 - Code Case 5

Figure 18 shows the code for case 5 which is entered if the state equals 5, this allows the user to set the month for the clock.

This has exactly the same principle of code as explained above for figure 15 but now pressing RA2 will make the state equal 6 and enter case 6 which will allow the user to set the year for the clock.

By pressing RA3 the month will increment by 1 and pressing RA4 will decrement by

```

case 6:                                     //Case 6, run if state equals 6 (this case is setting the year by pressing RA3 or RA4)
  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
  {
    temp_year++;                          //Increment temp_year
  }
  if(RA4)                                  //If RA4 = 1
  {
    while(RA4);
    Delay(100);

    temp_year--;                          //decrement temp_year
  }

  break;                                  //This breaks out from each case

```

1. If the temp_month is greater than 12 it will be set to 1.

Figure 19 - Code Case 6

Figure 19 above shows the code for case 6 which is entered if the state equals 6, this allows the user to set the year for the clock.

This has exactly the same principle of code as explained above for figure 18 but now pressing RA2 will make the state equal 0 and enter case 0 which will start the clock again but will now show the time and date set in cases 1-6.

By pressing RA3 the year is incremented by 1 and RA4 decremented by 1.

Again break will then break out from the case it's in.

This will now set the clock to the desired time by the user.

```
case 7:                                     //Case 7, run if state equals 7 (this case saves the new clock and alarm times)
    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 0 (once the new times are saved it starts the clock again)
    break;                                  //This breaks out from each case
```

Figure 20 - Code Case 7

Figure 20 shows the code for case 7, if the state equals 7 then case 7 is entered and this will then save the temporary time set in cases 1-6 and make this the actual time in the global variables.

While in case 7 the time and date are displayed on the LCD on their respective lines and once the global variables have been changed the state will equal 0 and case 0 will be entered after a delay of 100.

This will start the clock again and display the new time.

```

case 8:                                     //Case 8, run if state equals 8 (this case allow the alarm time to be set)

    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

```

Figure 21 - Code Case 8

Figure 21 shows the code for case 8, if the state equals 8 then case 8 is entered and this will then allow the user to set the alarm time.

This will display the date on LCD line 1 and display the alarm time on LCD line 0.

If RA1 is pressed the state equals 0 and the clock will resume.

If RA2 is pressed the state equals 9 and case 9 will be entered which allows the user to change the alarm hours and set it.

```

case 9:                                     //Case 9, run if state equals 9 (this case allows the hours for the alarm to be incremented by 1)

    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 22 - Code Case 9

Figure 22 shows the code for case 9, if the state equals 9 then case 9 is entered and this will then allow the user to set the alarm time hours.

This will display the alarm time on LCD line 0.

If RA1 is pressed the state equals 7 which allows the new alarm time to be saved.

If RA2 is pressed the state equals 10 and case 10 will be entered which allows the user to change the alarm minutes and set it.

If RA3 is pressed the alarm hours are incremented by 1 and if temp_al_hours is greater than 23 then temp_al_hours will equal 0.

```

case 10:                                     //Case 10, run if state equals 10 (this case allows the minutes for the alarm to be incremented by 1)
    DisplaySetAlarmTime(0);                 //Function header, runs DisplaySetAlarmTime function prototype

    if(RA1) state = 7;                       //If RA1 is equal to 1 set state to equal 7
    if(RA2) state = 8;                       //If RA2 is equal to 1 set state to equal 8
    if(RA3)                                  //If RA3 is equal to 1
    {
        temp_al_mins++;                     //Increment temp_al_mins by 1
        if(temp_al_mins>59) temp_al_mins=0; //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
}

```

The state part is explained, but it should be noted that the while loops for inc and dec and at the end is to counter switch bounce.

Figure 23 - Code Case 10 and End of Super Loop

Figure 23 shows the code for case 10, if the state equals 10 then case 10 is entered and this will then allow the user to set the alarm time minutes.

This will display the alarm time on LCD line 0.

If RA1 is pressed the state equals 7 which allows the new alarm time to be saved.

If RA2 is pressed the state equals 8 and case 8 will be entered which allows the user to change the alarm time again or save the alarm time and start the clock.

If RA3 is pressed the alarm minutes are incremented by 1 and if temp_al_mins is greater than 59 then temp_al_mins will equal 0.

Again break will then break out from the case it's in.

This will then carry on the super loop and keep incrementing the time of the clock indefinitely.

This ends the main program, the code below is describing the function definitions which is called upon by the main program when needed.

```

////////////////////////////////////display set time function////////////////////////////////////
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
}

```

Figure 24 - Code Display Set Time

The code in figure 24 shows the displaysettime function prototype and the code behind it. The display set time code is used to display the time set on the LDC line of choice depending on what case the user is in.

This will display the time on the LCD on line 0 (because displaystring is set as (line, time_string) as follows; 00:00. With the hours being the first two digits and the minutes the second two digits separated by a : this will be utilising the first 5 digits in the time_string array.

If the alarm is set to on and equals 1 the exclamation mark (!) will be shown on character 14 of the string. The string is ended by placing a 0 in the last string.

```

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
}

```

Figure 25 - Code Display Time

Figure 25 shows the code for the displaytime function, this is exactly the same code as in figure 24 except it uses the global varial value for hours and mins and not temp_hour and temp_mins, and will display the time which has been set by the user after they have saved the changes.

```

////////////////////////////////////delay function,
void Delay(int t) // Start of function
{
    int i;
    for(i=0;i<t;i++);

}

```

Figure 26 - Code Delay Function

Figure 26 shows the delay function which is used in some cases to delay switching back to case 0 or when decrementing. It works by setting i to equal 0 and while i is less than t, i will increment.

Therefore setting t to 100 then delay will count up to 100 and then move on.

```

////////////////////////////////////display state function,
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 27 - Code Display State Function

Figure 27 shows the code for the displaystate, this uses the current state of the clock and puts it into the state_string array in character 6.

```

////////////////////////////////////display date function////////////////////////////////////
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)
    {
        day_code_string[0] = (char) 83; //char S
        day_code_string[1] = (char) 85; //char U
        day_code_string[2] = (char) 78; //char 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)
    {
        day_code_string[0] = (char) 77; //char M
        day_code_string[1] = (char) 79; //char O
        day_code_string[2] = (char) 78; //char 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==2)
    {
        day_code_string[0] = (char) 84; //char T
        day_code_string[1] = (char) 85; //char U
        day_code_string[2] = (char) 69; //char E
        day_code_string[3] = ' '; //creates space
        day_code_string[4] = 0; //Ends the string by placing 0 in the last string
    }
    if (day_code==3)
    {
        day_code_string[0] = (char) 87; //char W
        day_code_string[1] = (char) 69; //char E
        day_code_string[2] = (char) 68; //char D
        day_code_string[3] = ' '; //creates space
        day_code_string[4] = 0; //Ends the string by placing 0 in the last string
    }
    if (day_code==4)
    {
        day_code_string[0] = (char) 84; //char T
        day_code_string[1] = (char) 72; //char H
        day_code_string[2] = (char) 85; //char U
        day_code_string[3] = ' '; //creates space
        day_code_string[4] = 0; //Ends the string by placing 0 in the last string
    }
    if (day_code==5)
    {
        day_code_string[0] = (char) 70; //char F
        day_code_string[1] = (char) 82; //char R
        day_code_string[2] = (char) 73; //char I
        day_code_string[3] = ' '; //creates space
        day_code_string[4] = 0; //Ends the string by placing 0 in the last string
    }

    if (day_code==6)
    {
        day_code_string[0] = (char) 83; //char S
        day_code_string[1] = (char) 65; //char A
        day_code_string[2] = (char) 84; //char T
        day_code_string[3] = ' '; //creates space
        day_code_string[4] = 0; //Ends the string by placing 0 in the last string
    }
}

```

Figure 28 - Code Display Date Day

Figure 28 shows the code for the displaydate function, specifically in figure 27 it shows the day_code_string in more detail. This is a string with 5 elements or memory spaces. This means that when the day_code_string function is called in the main program the day will be shown on the LCD.

This works by using the ASCII table (see appendix) and displaying the correct letter for the value set in the table. There is a space after the day and then the array is ended by a 0 which is a null character in the table.

This also utilises the day_code function which is incremented when the days change in the main program, this is then linked to the correct characters to display the day to be read by the user.

```

if (month==1)
{
    month_string[0] = (char) 74;    //char J
    month_string[1] = (char) 65;    //char A
    month_string[2] = (char) 78;    //char N
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==2)
{
    month_string[0] = (char) 70;    //char F
    month_string[1] = (char) 69;    //char E
    month_string[2] = (char) 66;    //char B
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==3)
{
    month_string[0] = (char) 77;    //char M
    month_string[1] = (char) 65;    //char A
    month_string[2] = (char) 82;    //char R
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==4)
{
    month_string[0] = (char) 65;    //char A
    month_string[1] = (char) 80;    //char P
    month_string[2] = (char) 82;    //char R
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==5)
{
    month_string[0] = (char) 77;    //char M
    month_string[1] = (char) 65;    //char A
    month_string[2] = (char) 89;    //char Y
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==6)
{
    month_string[0] = (char) 74;    //char J
    month_string[1] = (char) 85;    //char U
    month_string[2] = (char) 78;    //char N
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==7)
{
    month_string[0] = (char) 74;    //char J
    month_string[1] = (char) 85;    //char U
    month_string[2] = (char) 76;    //char L
    month_string[3] = ' ';
    month_string[4] = 0;
}

```

Figure 29 – Code Display Date Month (1)

```

if (month==8)
{
    month_string[0] = (char) 65;    //char A
    month_string[1] = (char) 85;    //char U
    month_string[2] = (char) 71;    //char G
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==9)
{
    month_string[0] = (char) 83;    //char S
    month_string[1] = (char) 69;    //char E
    month_string[2] = (char) 80;    //char P
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==10)
{
    month_string[0] = (char) 79;    //char O
    month_string[1] = (char) 67;    //char C
    month_string[2] = (char) 84;    //char T
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==11)
{
    month_string[0] = (char) 78;    //char N
    month_string[1] = (char) 79;    //char O
    month_string[2] = (char) 86;    //char V
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==12)
{
    month_string[0] = (char) 68;    //Ascii code 68 = D
    month_string[1] = (char) 69;    //Ascii code 69 = E
    month_string[2] = (char) 67;    //Ascii code 67 = C
    month_string[3] = ' ';        //Creates space
    month_string[4] = 0;          //Ends the string by placing 0 in the last string
}

/*Places day_code, days, month, and year into the date_string*/

```

Figure 30 - Code Display Date Month (2)

Figures 29 and 30 show the code for the displaydate function, specifically in figure the month_string in more detail.

It's the same code as the day_code_string explained above except it has 12 months and utilises the month function when it is incremented.

```

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;           // day 10's
date_string[5] = days % 10 + 48;         // day units
date_string[6] = ' ';                     //Creates space
date_string[7] = month_string[0];
date_string[8] = month_string[1];         //get month chars
date_string[9] = month_string[2];
date_string[10] = month_string[3];
date_string[11] = year / 1000 + 48;       // year 1000's
date_string[12] = year % 1000 / 100 + 48; // year 100's
date_string[13] = year % 100 / 10 + 48;  // year 10's
date_string[14] = year % 10 + 48;        // year units
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
}

```

Figure 31 - Code Display Date Placing On LCD

Figure 31 shows the code for all the functions above which are now being utilised in the date_string array mentioned above in figure 28.

The date string take the day code string and places the values in the array into the first 4 characters of it's array, it then uses the value for the date and places it in the next 2 characters of the array and then uses a space as there is not one there as the date does not have an array with one in like the other strings do.

The month string is then placed in the next 4 places in the date string array followed by the values for the years and this is then ended by the 0 (null character).

The display string then uses the date string and sends it to LCD line 1 to complete the clock. If correct the date should read something like; MON 12 JAN 2016.

The end function call takes the string created as an argument for the LCD function call. This is how the data gets displayed on the LCD.

```

////////////////////////////////////Display set date function////////////////////////////////////

void 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
    {
        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
    {
        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
    {
        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
    {
        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
    {
        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
    }
    if(temp_day_code==5) //If temp_day_code is equal to 5
    {
        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
    {
        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
    }
}

```

Figure 32 - Code Display Set Date (1)

```

if(temp_month==1)                //If temp_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(temp_month==2)                //If temp_month is equal to 2
{
    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
{
    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
{
    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
}
if(temp_month==5)                //If temp_month is equal to 5
{
    month_string[0] = (char) 77;  //Ascii code 77 = M
    month_string[1] = (char) 65;  //Ascii code 65 = A
    month_string[2] = (char) 89;  //Ascii code 89 = Y
    month_string[3] = ' ';        //Creates space
    month_string[4] = 0;          //Ends the string by placing 0 in the last string
}
if(month==6)
{
    month_string[0] = (char) 74;  //char J
    month_string[1] = (char) 85;  //char U
    month_string[2] = (char) 78;  //char N
    month_string[3] = ' ';
    month_string[4] = 0;
}
if(month==7)
{
    month_string[0] = (char) 74;  //char J
    month_string[1] = (char) 85;  //char U
    month_string[2] = (char) 76;  //char L
    month_string[3] = ' ';
    month_string[4] = 0;
}
}

```

Figure 33 - Code Display Set Date (2)

```

if (month==8)
{
    month_string[0] = (char) 65;    //char A
    month_string[1] = (char) 85;    //char U
    month_string[2] = (char) 71;    //char G
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==9)
{
    month_string[0] = (char) 83;    //char S
    month_string[1] = (char) 69;    //char E
    month_string[2] = (char) 80;    //char P
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==10)
{
    month_string[0] = (char) 79;    //char O
    month_string[1] = (char) 67;    //char C
    month_string[2] = (char) 84;    //char T
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==11)
{
    month_string[0] = (char) 78;    //char N
    month_string[1] = (char) 79;    //char O
    month_string[2] = (char) 86;    //char V
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==12)
{
    month_string[0] = (char) 68;    //Ascii code 68 = D
    month_string[1] = (char) 69;    //Ascii code 69 = E
    month_string[2] = (char) 67;    //Ascii code 67 = C
    month_string[3] = ' ';        //Creates space
    month_string[4] = 0;          //Ends the string by placing 0 in the last string
}

```

Figure 34 - Code Display Set Date (3)

```

        /*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] = temp_days / 10 + 48;
        date_string[5] = temp_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] = temp_year / 1000 + 48;
        date_string[12] = temp_year % 1000 / 100 + 48;
        date_string[13] = temp_year % 100 / 10 + 48;
        date_string[14] = temp_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
}

```

Figure 35 - Code Display Set Date (4)

Figures 32-35 show the code for displaysetdate function, this is exactly the same code as the displaydate but allows the user to see what changes they are making while altering the date.

```

////////////////////////////////////Display alarm time function////////////////////////////////////
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
}

```

Figure 36 - Code Display Alarm Time

Figure 36 shows the code for displayingalarmtime function, this is utilising the alarm_string array and putting the alarm_hours value into the first two spaces of the array followed by a :, this then places the alarm_minutes value into the next two spaces.

The next 5 spaces are used as spaces with the last next 5 spaces displaying the word ALARM on the LCD line 0 using the ASCII values in the array spaces to spell ALARM. This is then ended by a 0 (null value) in the last array space.

If successful it should look something like; 07:15 ALARM

```

////////////////////////////////////Display set alarm time function////////////////////////////////////
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
}

```

Figure 37 - Code Display Set Alarm

Figure 37 shows the code for displaysetalarmtime, this code is exactly the same as in figure 36 except it uses the global variable value for temp_al_hours and temp_al_mins instead of alarm_hours and alarm_mins. This function is to allow the user to view the changes to the alarm they are making when toggling the hours and minute values.

The LCD function call is used in the same way for all the created arrays in the functions. It passes the array strings as arguments for the LCD function call and displays on the LCD.

4. Testing and Analysis of Results

Testing was carried out to check that all the design functions worked correctly, this is aided by a testing table which is shown below. The clock was sped up to aid testing and left to run for “months” and the clock performed perfectly as it should have done. This part of the testing is not documented as it is hard to gain evidence for this but it was successful.

Other parts of testing which is not shown or discussed in the evidence is the decrementing part of the clock, this is because it was an added extra to the assignment criteria and was not necessary, regardless; this was put into the code of the clock and when tested did work correctly.

Current State	Test Procedure	Evidence (Figure No.)	Expected Result	Actual Result	Success?
0	Display clock time and date on LCD	38, 45 and 46	Display the set time and date on screen with clock running and turns on alarm when RA3 is pressed	Display the set time and date on screen with clock running and turns on alarm when RA3 is pressed	Yes
1	Increment hours by 1	39	Increment hours by 1 when RA3 is pressed	Increment hours by 1 when RA3 is pressed	Yes
2	Increment minutes by 1	40	Increment minutes by 1 when RA3 is pressed	Increment minutes by 1 when RA3 is pressed	Yes
3	Increment days by 1	41	Increment days by 1 when RA3 is pressed	Increment days by 1 when RA3 is pressed	Yes
4	Increment date by 1	42	Increment date by 1 when RA3 is pressed	Increment date by 1 when RA3 is pressed	Yes
5	Increment month by 1	43	Increment month by 1 when RA3 is pressed	Increment month by 1 when RA3 is pressed	Yes
6	Increment year by 1	44	Increment year by 1 when RA3 is pressed	Increment year by 1 when RA3 is pressed	Yes
7	Saves the altered time and date and alarm time and returns the clock to normal	49 and 45	Saves the altered times and alarms and clock continues	Saves the altered times and alarms and clock continues	Yes
8	Displays alarm time and date along with ALARM	47	Alarm time is shown when RA1 is pressed from state 0	Alarm time is shown when RA1 is pressed from state 1	Yes
9	Increment alarm hours by 1	48	Increment alarm hours by 1 when pressing RA3	Increment alarm hours by 1 when pressing RA4	Yes
10	Increment alarm minutes by 1	49	Increment alarm minutes by 1 when pressing RA3	Increment alarm minutes by 1 when pressing RA4	Yes

Figure 38 - Testing Table (1)

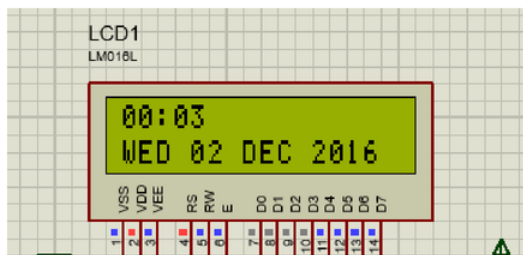


Figure 39 - Normal Clock

Press RA2 to start setting clock time.

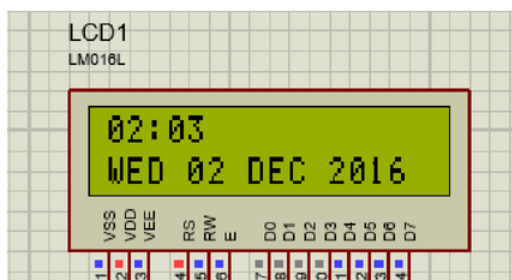


Figure 40 - Setting Clock Hours

RA3 to toggle hours (above) RA2 to move to minutes.

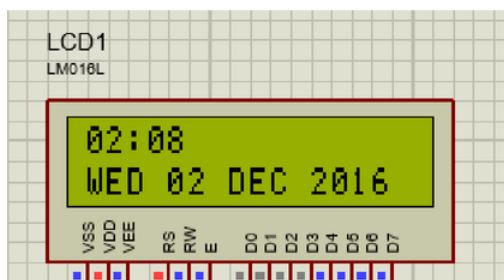


Figure 41 - Setting Clock Minutes

RA3 to toggle minutes (above) RA2 to move to days.

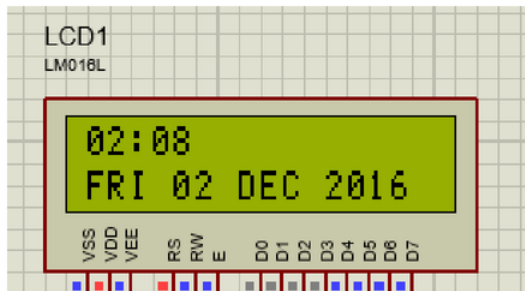


Figure 42 - Setting Clock Days

RA3 to toggle days (above) RA2 to move to date.

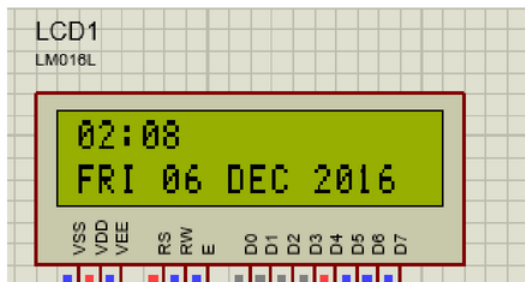


Figure 43 - Setting Clock Date

RA3 to toggle date (above) RA2 to move to month.

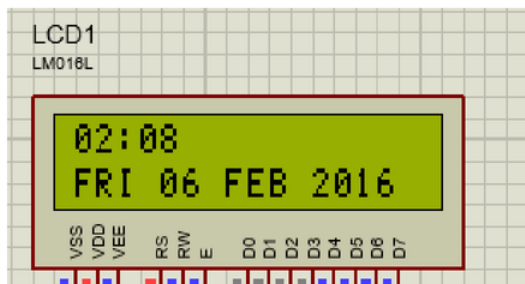


Figure 44 - Setting Clock Month

RA3 to toggle month (above) RA2 to move to year.

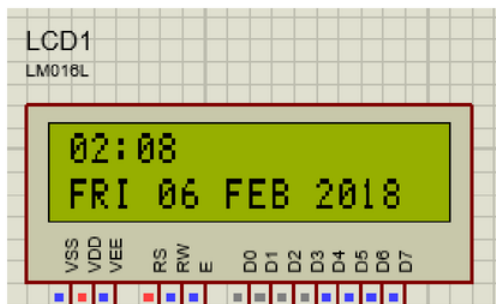


Figure 45 - Setting Clock Year

RA3 to toggle month (above) and pressing RA1 to return to normal and run clock (same figure as evidence (44)).

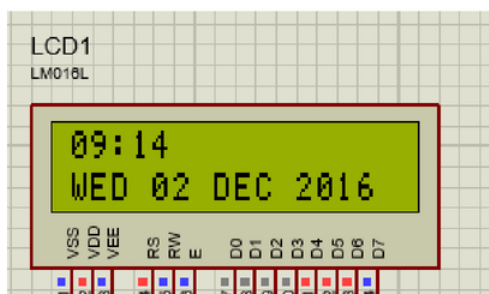


Figure 46 - Alarm Not Set

Press RA3 in state 0 to set alarm (below)

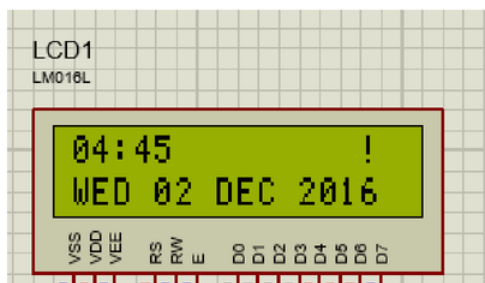


Figure 47 - Alarm Set

Alarm is now set and displaying exclamation mark (!)

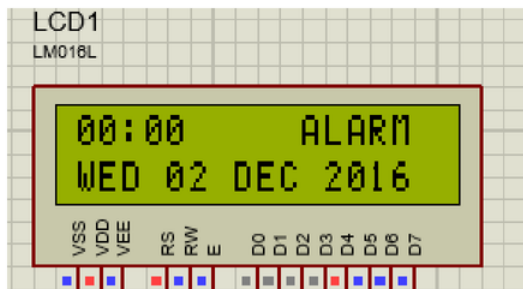


Figure 48 - Setting Alarm

Pressing RA1 in state 0 allows user to set alarm and displays ALARM.

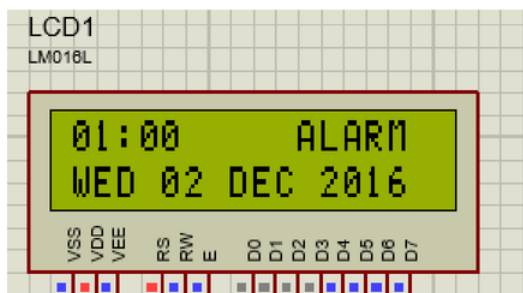


Figure 49 - Setting Alarm Hours

Pressing RA2 allows hours to be set and RA3 to change hours

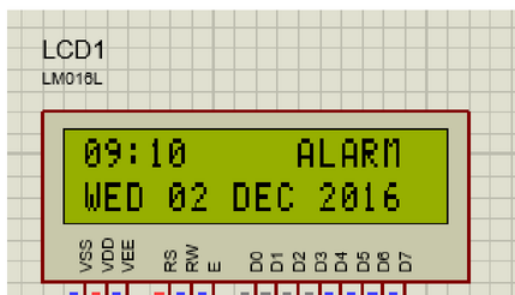


Figure 50 - Setting Alarm Minutes

Pressing RA2 again allows minutes to be set and RA3 to change minutes

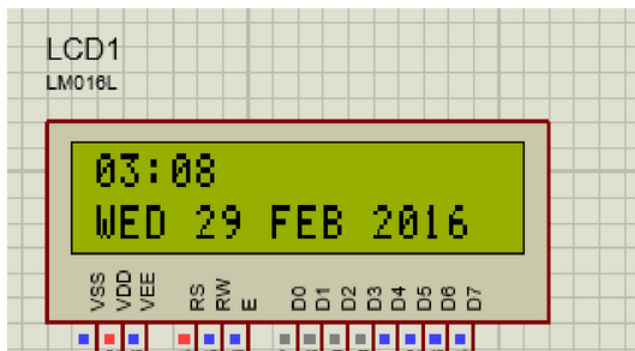


Figure 53 - Testing Leap Year

Figure 53 shows the clock date as being 29th of February 2016 which is a leap year because it is divisible by 4 and thus allowing the date in February to be 29.

Supporting evidence of testing is shown below of the 1 second clock pulse for the PIC, as explained above the PIC is limited to 10ms by its processing power and therefore uses the code to make 1 second, this is shown below.

```
#include <pic.h>

int milli=0, secs=0; //Must be declared global for function prototype variables

void interrupt ISR (void)
{
    if(TOIF==1)
    {
        TOIF = 0; //Reset interrupt flag, set to zero
        TMRO = 63; //Reload preload time value. TMRO IS SET FOR 100ms this is the largest time we can have from the TMRO. Clk = 10MHZ
        // milli += 1; //Increment milliseconds by 10 to increase the value of the timer in steps of 10
        // if(milli==100) //If milli equals 100 increment sec by 1, clear milli = 0 and toggle RBO
        {
            secs ++; //Increment seconds by 1 goes here
            milli = 0; //Reset milli to zero
            RBO = ~RBO; // Toggle RBO pin
        }
    }
}

void main(void) //Main Function
{
    TRISE = 0x00; //Configure port B = 0000 0000
    OPTION_REG = 0x87;
    INTCON = 0x00;

    while(1) //Start super-loop
    {
        //
    }
}
```

Figure 54 - Clock Pulse at 10ms

Figure 54 above shows the preloaded value of 63 which is calculated in the research, this give a 10ms clock pulse which is proven below by placing a oscilloscope on TMR0.

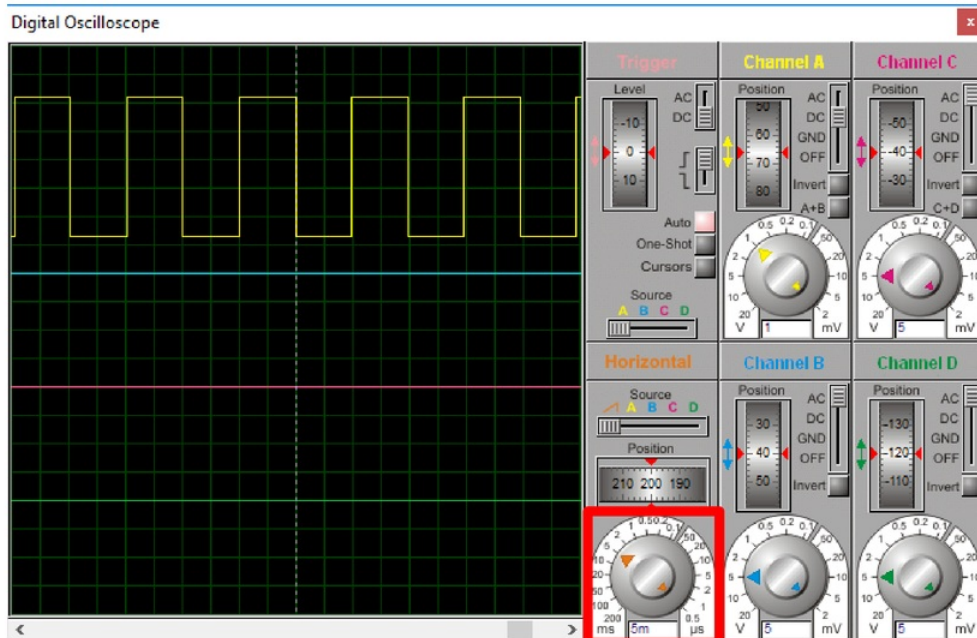


Figure 55 - Oscilloscope 10ms

Figure 55 shows that the source is set to 5ms per division (highlighted) and there are 2 divisions per pulse so therefore one pulse is 2 x 5ms which equals 10ms.

```

#include <pic.h>

int milli=0, secs=0; //Must be declared global for function prototype variables

void interrupt ISR (void)
{
    if(TOIF==1)
    {
        TOIF = 0; //Reset interrupt flag, set to zero
        TMR0 = 63; //Reload preload time value. TMR0 IS SET FOR 100ms this is the largest time we can have from the TMR0. Clk = 10MSE
        milli += 1; //Increment milliseconds by 10 to increase the value of the timer in steps of 10
        if(milli==100) //If milli equals 100 increment sec by 1, clear milli = 0 and toggle RB0
        {
            secs++; //Increment seconds by 1 goes here
            milli = 0; //Reset milli to zero
            RB0 = ~RB0; // Toggle RB0 pin
        }
    }
}

void main(void) //Main Function
{
    TRISB = 0x00; //Configure port B = 0000 0000

    OPTION_REG = 0x87;
    INTCON = 0x00;

    while(1) //Start super-loop
    {
    }
}

```

Figure 56 - Clock Pulse at 1 Second

Figure 56 shows the code has now been modified to count 100 times for the milliseconds to equal 1 second.

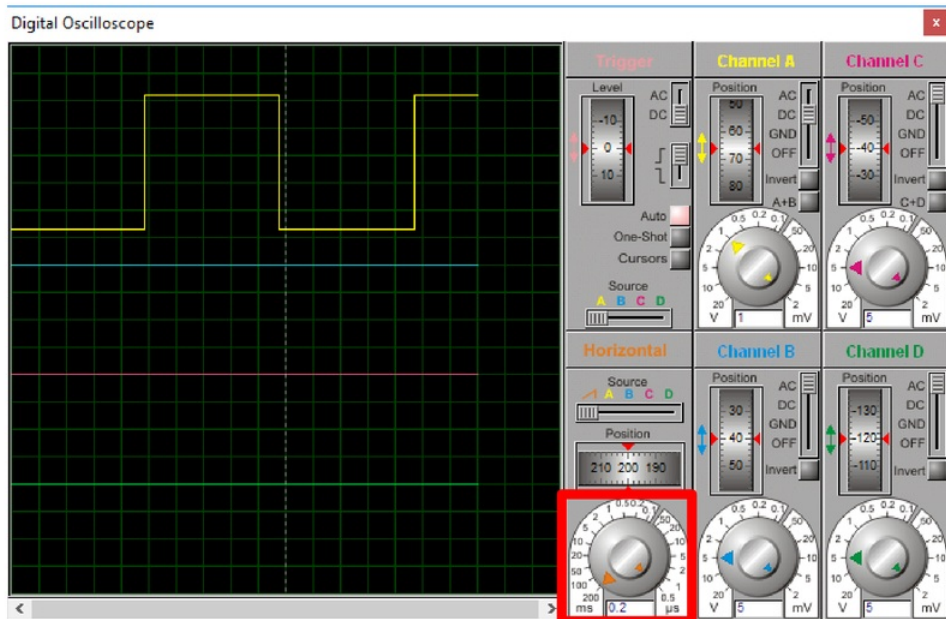


Figure 57 - Oscilloscope 1 Second

Figure 57 above shows that the source is set to 20ms per division (highlighted) and there are 5 divisions per pulse so therefore one pulse is 20ms x 5 which equals 1 second.

Practically testing the HP488 development board

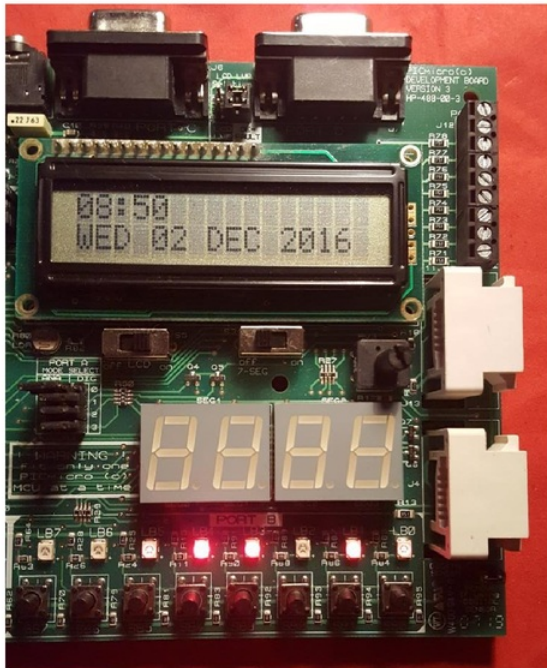


Figure 58 - Normal Clock Practical

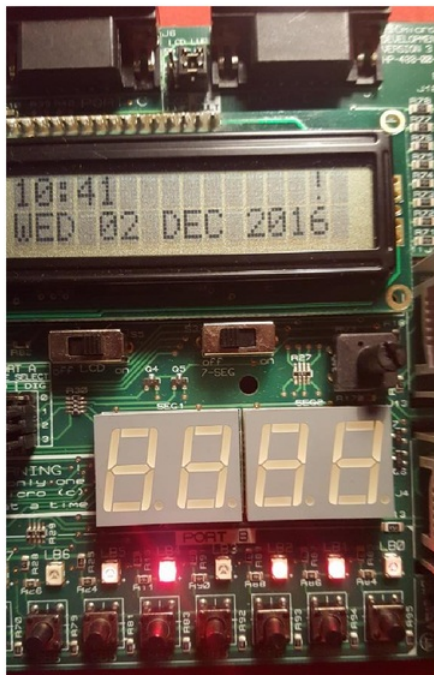


Figure 59 - Alarm on Practical



Figure 60 - Setting Alarm Practical

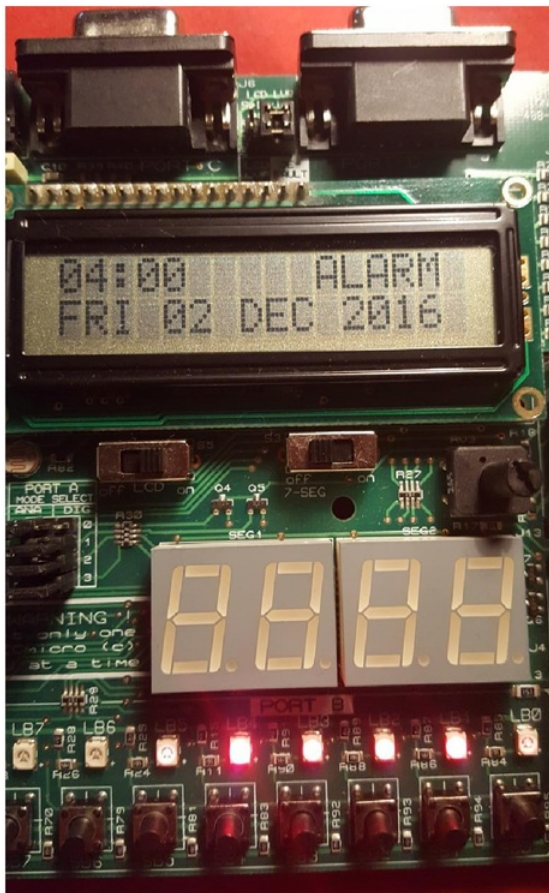


Figure 61 - Setting Alarm Hours Practical

This is the end of testing; testing has been successful as all expected outcomes have happened and the code has worked correctly for the alarm clock.

5. Conclusions and Recommendations

To be able to conclude the aims and objectives of the assignment must be evaluated to see if they have been met, this is shown below.

The aims and objectives of this assignment were to be able to create and design an alarm clock using an embedded system based on a microcontroller using the configurable on-chip peripherals. The design was then be put onto a PIC 16F877 microcontroller and a HP488 development board. This enabled the design of the digital calendar alarm clock which has two lines of LCD and 3 pushbuttons to alter and change the clock.

These aims above were met as an alarm clock was designed and implemented on the PIC 16F877 microcontroller and downloaded onto the HP488 development board, evidence of this working is provided and works successfully so this part of the assignment is deemed a success.

The specification for the alarm clock was as follows:

Setting the time and date;

This must allow the user to press button 2 which allows them to alter the hour of the clock by toggling button 3. To toggle through the rest of the time changes (minutes, days, date, month and year) the user much press button 2 again and 3 to change whichever setting they are on. By pressing button 1 the changes made are saved and the clock will resume from the altered time.

Setting the alarm time;

This must allow the user to press button 1 which will display the alarm time that is already set. By pressing button 2 the user can select whether to change the alarm hours or minutes, this can then be toggled and changed by pressing button 3. By pressing button 1 the changes made are saved and the new alarm set and the clock will resume. It must also display ALARM on the LCD display when setting the alarm time.

Turning the alarm on/off;

When in normal mode i.e. the clock must be running and not in a setting time/alarm mode, by pressing button 3 will turn the alarm on/off. To make sure this is successful an exclamation mark (!) will appear on the LCD.

These criteria were all met and testing and evidence is proof of this, this therefore proves that the alarm clock works fully and that the assigned as a whole can be deemed a success as all the aims and objectives have been met.

Recommendations:

The code for the clock could be worked upon as there is room to make the code more efficient, for example; the days of the month should be linked to the date and therefore when altering the date the days automatically change too.

This was discovered as a recommendation when the date for the leap year in February was changed as the days did not and therefore the day of the 29th of February 2016 might not have been accurate.

Adding noises to the PIC microcontroller and code might be advantageous as the alarm could make a sound and different tones could be used when altering the time and date and a separate tone used for altering the alarm time.

In conclusion the code works well and the assignment is a success.

6. Appendix and Bibliography

Bibliography

_site_media/ascii/ascii-chars-table-landscape.jpg. (n.d.). Retrieved from <http://www.asciitbl.com/>: http://www.asciitbl.com/_site_media/ascii/ascii-chars-table-landscape.jpg

Engineering, P. (n.d.). *2009/06/microcontroller-pic-16f877-features-pin.html*. Retrieved January 2018, from <http://ugpro143.blogspot.co.uk>: <http://ugpro143.blogspot.co.uk/2009/06/microcontroller-pic-16f877-features-pin.html>

Lab, E. (n.d.). *blog/wp-content/uploads/2010/11/OPTION_REG.png*. Retrieved 1 2018, from <http://embedded-lab.com>: http://embedded-lab.com/blog/wp-content/uploads/2010/11/OPTION_REG.png

Romux. (2010). *Interrupts*. Retrieved 1 31, 2017, from <http://www.romux.com/tutorials/pic-tutorial/interrupts>

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0 0	000		NULL	32 20	040	 	Space	64 40	100	@	@	96 60	140	`	,			
1 1	001		Start of Header	33 21	041	!	i	65 41	101	A	A	97 61	141	a	a			
2 2	002		Start of Text	34 22	042	"	"	66 42	102	B	B	98 62	142	b	b			
3 3	003		End of Text	35 23	043	#	#	67 43	103	C	C	99 63	143	c	c			
4 4	004		End of Transmission	36 24	044	$	\$	68 44	104	D	D	100 64	144	d	d			
5 5	005		Enquiry	37 25	045	%	%	69 45	105	E	E	101 65	145	e	e			
6 6	006		Acknowledgment	38 26	046	&	&	70 46	106	F	F	102 66	146	f	f			
7 7	007		Bell	39 27	047	'	'	71 47	107	G	G	103 67	147	g	g			
8 8	010		Backspace	40 28	050	(	(	72 48	110	H	H	104 68	150	h	h			
9 9	011		Horizontal Tab	41 29	051	)	)	73 49	111	I	I	105 69	151	i	i			
10 A	012		Line feed	42 2A	052	*	*	74 4A	112	J	J	106 6A	152	j	j			
11 B	013		Vertical Tab	43 2B	053	+	+	75 4B	113	K	K	107 6B	153	k	k			
12 C	014		Form feed	44 2C	054	,	,	76 4C	114	L	L	108 6C	154	l	l			
13 D	015		Carriage return	45 2D	055	-	-	77 4D	115	M	M	109 6D	155	m	m			
14 E	016		Shift Out	46 2E	056	.	.	78 4E	116	N	N	110 6E	156	n	n			
15 F	017		Shift In	47 2F	057	/	/	79 4F	117	O	O	111 6F	157	o	o			
16 10	020		Data Link Escape	48 30	060	0	0	80 50	120	P	P	112 70	160	p	p			
17 11	021		Device Control 1	49 31	061	1	1	81 51	121	Q	Q	113 71	161	q	q			
18 12	022		Device Control 2	50 32	062	2	2	82 52	122	R	R	114 72	162	r	r			
19 13	023		Device Control 3	51 33	063	3	3	83 53	123	S	S	115 73	163	s	s			
20 14	024		Device Control 4	52 34	064	4	4	84 54	124	T	T	116 74	164	t	t			
21 15	025		Negative Ack.	53 35	065	5	5	85 55	125	U	U	117 75	165	u	u			
22 16	026		Synchronous idle	54 36	066	6	6	86 56	126	V	V	118 76	166	v	v			
23 17	027		End of Trans. Block	55 37	067	7	7	87 57	127	W	W	119 77	167	w	w			
24 18	030		Cancel	56 38	070	8	8	88 58	130	X	X	120 78	170	x	x			
25 19	031		End of Medium	57 39	071	9	9	89 59	131	Y	Y	121 79	171	y	y			
26 1A	032		Substitute	58 3A	072	:	:	90 5A	132	Z	Z	122 7A	172	z	z			
27 1B	033		Escape	59 3B	073	;	;	91 5B	133	[	[	123 7B	173	{	{			
28 1C	034		File Separator	60 3C	074	<	<	92 5C	134	\	\	124 7C	174	|				
29 1D	035		Group Separator	61 3D	075	=	=	93 5D	135	]	^	125 7D	175	}	}			
30 1E	036		Record Separator	62 3E	076	>	>	94 5E	136	^	_	126 7E	176	~	~			
31 1F	037		Unit Separator	63 3F	077	?	?	95 5F	137	_	-	127 7F	177		Del			

Assignment 1 Joshua Beer

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

62/100

Instructor

test comment

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

Text Comment. Research is limited and should extend to more than just the configuration information.

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

Text Comment. The design looks good and the state diagram is clear.

PAGE 12

PAGE 13

PAGE 14

Text Comment. This is explained well.

Text Comment. Should point out here that the main function is where the configuration is included.

Text Comment. Explained well.

Text Comment. This is correct.

Text Comment. The state part is explained, but it should be noted that the while loops for inc and dec and at the end is to counter switch bounce.

Text Comment. The end function call takes the string created as an argument for the LCD function call. This is how the data gets displayed on the LCD.

PAGE 37

PAGE 38

PAGE 39

PAGE 40

Text Comment. The LCD function call is used in the same way for all the created arrays in the functions. It passes the array strings as arguments for the LCD function call and displays on the LCD.

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50

PAGE 51

PAGE 52

PAGE 53

PAGE 54

PAGE 55

PAGE 56

PAGE 57

GRADING FORM: CSG-RUBIC

RESEARCH-INFORMED LITERATURE



This could be much better and should extend to more than the registers. .6/15 The research is weak and more academic sources could have been used. 4/10

10/25

KNOWLEDGE AND UNDERSTANDING



Introduction gives a good overview. 3/5 The configuration is there but needs a better explanation. The system design is good. 6.5/10 Some points were not covered with code explanation.6.5/10

16/25

ANALYSIS



The testing is good and covers the CAD and dev board which shows the compatibility of the Proteus CAD architecture.8.5/10 Conclusion is good. This could have included recommendations and the compatibility factor with CAD and dev board. 6.5/10

15/20

PRACTICAL APPLICATION AND DEP.



Practical application is good and the prototypes were built and combined together for the final system. A little more involving the prototype stages could have been included in the explanation. 6.5/10

15/20

The testing is substantial. 8.5/10

SKILLS FOR PROFESSIONAL PRAC.



The report is good. 3/5 Team working is good. 3/5

6/10