

Assignment_1

by Emyr Dyer

FILE	7311_EMYR_DYER_ASSIGNMENT_1_139109_1026613299.DOCX (2.87M)		
TIME SUBMITTED	01-FEB-2018 11:42PM (UTC+0000)	WORD COUNT	3306
SUBMISSION ID	82690015	CHARACTER COUNT	24642

HND Electrical & Electronic Engineering

Calendar Alarm Clock

PSDE5013 – Configuration and
Programming of Embedded Systems

Author: Emyr Wyn Dyer
Submission Date: 01/02/2018
Word Count: 3306

CONTENTS

1	Introduction	1
2	Research.....	2
2.1	Timing.....	2
3	Design.....	3
3.1	Finding the States.....	3
3.2	Programming.....	5
3.2.1	Headers and Function Prototypes	5
3.2.2	Global Variables	5
3.2.3	Interrupt ISR	6
3.2.4	Main Function (Beginning).....	7
3.2.5	Start of Super-Loop	7
3.2.6	Switch Case 0	8
3.2.7	Switch Case 1	9
3.2.8	Switch Case 2	9
3.2.9	Switch Case 3	10
3.2.10	Switch Case 4	10
3.2.11	Switch Case 5	12
3.2.12	Switch Case 6	12
3.2.13	Switch Case 7	13
3.2.14	Switch Case 8	13
3.2.15	Switch Case 9	13
3.2.16	Switch Case 10 and Button While	14
3.2.17	Delay Function	14
3.2.18	DisplayTime Function	15
3.2.19	DisplaySetTime Function.....	16
3.2.20	DisplayDate and DisplaySetDateFunction.....	17
3.2.21	DisplaySetAlarmTime Function	18
3.2.22	DisplayState Function	19
3.2.23	Simulated Circuit	19
4	Testing and Analysis of Results	20
4.1	Simulated Test Plan.....	20
4.2	Simulated Test Screens	21
4.3	Simulated Test Summary	23

4.4	Uploaded Program to Board	24
5	Conclusions and Recommendations	28
6	Citation List	29

Figures

Figure 3.1: Headers and Function Prototypes	5
Figure 3.2: Global Variables	5
Figure 3.3: interrupt ISR	6
Figure 3.4: Main Function	7
Figure 3.5: Super-Loop	7
Figure 3.6: Switch Case 0	8
Figure 3.7: Switch Case 1	9
Figure 3.8: Switch Case 2	9
Figure 3.9: Switch Case 4	10
Figure 3.10: Switch Case 4	11
Figure 3.11: Switch Case 5	12
Figure 3.12: Switch Case 6	12
Figure 3.13: Switch Case 7	13
Figure 3.14: Switch Case 8	13
Figure 3.15: Switch Case 9	13
Figure 3.16: Switch Case 10	14
Figure 3.17: Delay Function	14
Figure 3.18: DisplayTime Function.....	15
Figure 3.19: DisplaySetTime Function.....	16
Figure 3.20: DisplayDateFunction (Day)	17
Figure 3.21: DisplayDateFunction (Month).....	17
Figure 3.22: DisplayDate Function (String Formatting)	18
Figure 3.23: DisplaySetAlarmTime Function	18
Figure 3.24: DisplayState Function	19
Figure 3.25: Complete Simulated Circuit	19
Figure 4.1: Case 0 Show D&T	21
Figure 4.2: Case 1 Increment Hour	21
Figure 4.3: Case 1 Decrement Hour	21
Figure 4.4: Case 2 Increment Minute.....	21
Figure 4.5: Case 2 Decrement Minute	22

Figure 4.6: Case 3 Change Day	22
Figure 4.7: Case 4 Change Date (Number)	22
Figure 4.8: Case 5 Change Month	22
Figure 4.9: Case 6 Change Year	22
Figure 4.10: Case 8 Show Alarm setting.....	23
Figure 4.11: Case 9 Change Hour setting	23
Figure 4.12: Case 10 Change Minute setting	23
Figure 4.13: Clock running on board.....	24
Figure 4.14: Alarm set	25
Figure 4.15: Edit Alarm	26
Figure 4.16: Changed Hour	27

1 INTRODUCTION

The aim of this investigation is to develop a working Digital Clock using the C Programming Language in conjunction with Proteus, to simulate the final product before uploading the final working code to the HP488 development board which used the PIC 16F877 microcontroller.

To develop an effective Digital Clock, there are a minimal number of design elements required to make the product viable, as the Digital Clock must:

- Display the date and time,
- Have the date and time set by the user using buttons,
- Have an alarm that can be set using buttons
- Be able to deactivate the alarm when ringing

There should be more to this.

2 RESEARCH

2.1 TIMING

A Digital Clock would be of no use if it could not track time without a solid degree of accuracy, so a method of ticking the seconds by must be coded using TMR0 (Timer 0), OPTION_REG (Option Register) and INTCON.

Both Registers used can be read and written to, and the functionality of both is that the OPTION_REG handles the scaling for the timer and the INTCON handles the pin interrupts for RB0 and the flag for the Timer.

To behave the way the registers must in order for the Digital Clock to function, the OPTION_REG must be configured using binary 0b10000111 or hex 0x87 so that the pre-scaling of the timer is 1:256 by setting bits 0 to 2 as 111, enabling the prescaler by setting bit 3 to 0, enabling an increment of low to high on pin RA4 by setting bit 4 to 0, enabling the internal clock with bit 5 by setting to 0, enabling the detection of the interrupt with a falling edge through RB0 by setting bit 6 to 0, and disabling the pull-up resistors by setting bit 7 to 1.

The INTCON register is configured with binary 0b10100000 or hex 0xA0, which enables the interrupts for the timer by setting bit 5 to 1, disabling all peripheral interrupts by setting bit 6 to 0, and enabling all unmasked interrupts by setting bit 7 to 1.

To produce accurate timing, the maximum prescaler value of 1:256 was used, which is fine considering the timer overflows once 256 is passed. As there are 4 square wave pulses for every instruction, the clock frequency on the board of 19.6608MHz would need to be divided by 4 to produce a 4.9152MHz frequency per instruction for timing. The time increments are calculated using the following equations.

$$1/4915200 \times 256 = 51.2 \times 10^{-6} \text{seconds}$$

This is not exactly 10ms as would be required, so the following calculation is carried out to find the value required to correct the scaling.

$$\left(10 \times 10^{-3} / 51.2 \times 10^{-6}\right) - 2 = 193$$

Taking the calculated value from the above equation and deducting it from 256,

$$256 - 193 = 63$$

By loading this compensating value into the timer, a viably round figure can now be used in conjunction with coding to provide 1 second timing.

This should extend beyond the configuration and

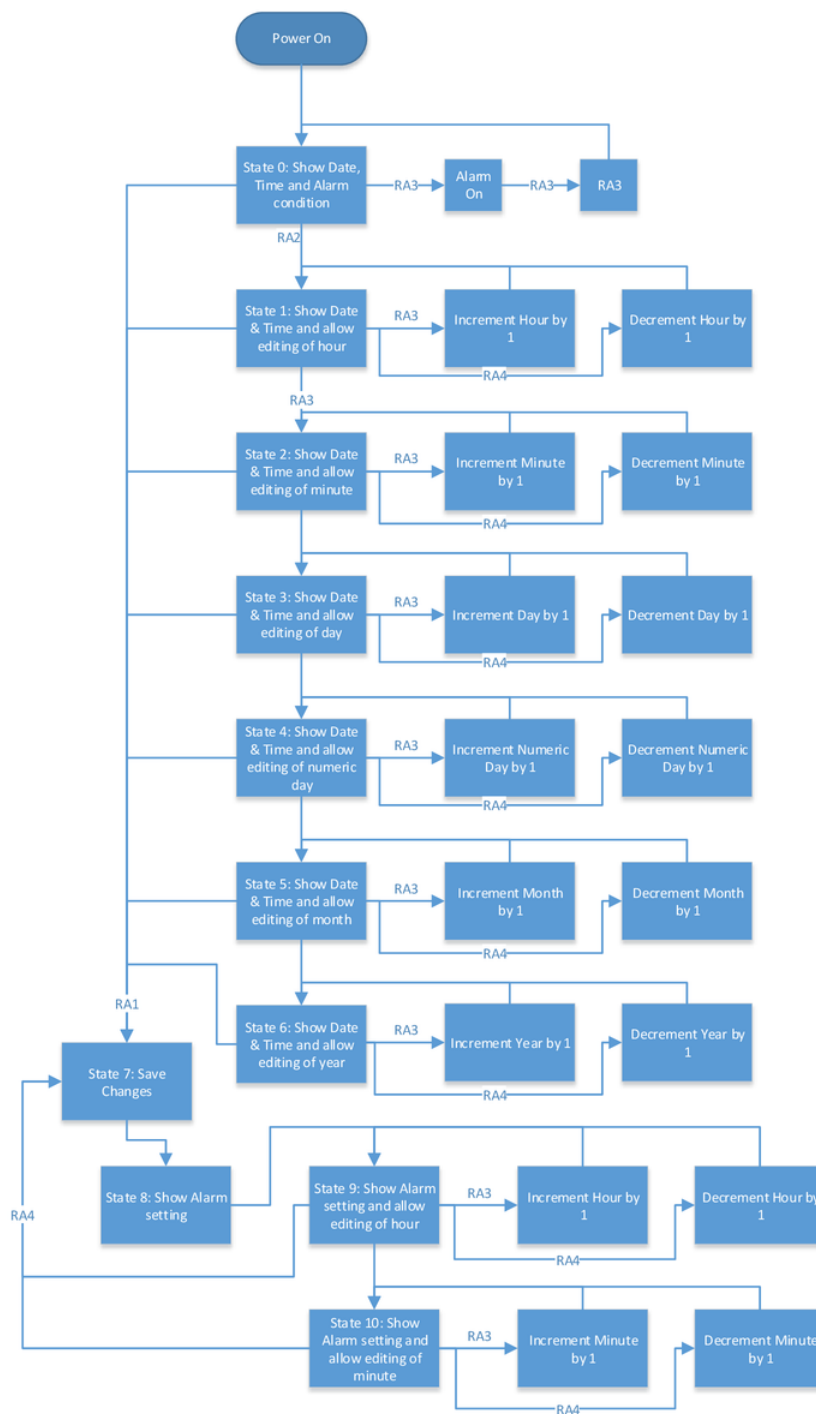
3 DESIGN

3.1 FINDING THE STATES

Before code can be written, an understanding of how the final product should work must be determined. The best way of achieving this is by creating a state table, and from that produce a state diagram, both of which help in writing code as well as producing a test plan which reflects how the product should behave.

State and Behaviour	Upon button press go to state		Action on button press	
	RA1	RA2	RA3	RA4
0: Show Date, Time and Alarm condition	7	1	Alarm On/Off	Do nothing
1: Show D&T and allow editing of hour	7	2	Increase set hour by 1	Decrease set hour by 1
2: Show D&T and allow editing of minute	7	3	Increase set minute by 1	Decrease set minute by 1
3: Show D&T and allow editing of day	7	4	Increase set day by 1	Decrease set day by 1
4: Show D&T and allow editing of numeric day	7	5	Increase set numeric day by 1	Decrease set numeric day by 1
5: Show D&T and allow editing of month	7	6	Increase set month by 1	Decrease set month by 1
6: Show D&T and allow editing of year	7	0	Increase set year by 1	Decrease set year by 1
7: Save changes	Do nothing	Do nothing	Do nothing	Do nothing
8: Show Alarm setting	0	9	Do nothing	Do nothing
9: Show Alarm setting and allow editing of hour	7	10	Increase set hour for alarm by 1	Decrease set hour for alarm by 1
10: Show Alarm setting and allow editing of minute	7	8	Increase set minute for alarm by 1	Decrease set minute for alarm by 1

From the above Truth Table, the State Diagram was produced and shown on the next page. From the Truth Table and the State Diagram, the code can be written with the idea of how information will need to flow through the Program.



This looks a little complex but it will work.

3.2 PROGRAMMING

3.2.1 Headers and Function Prototypes

Included at the header of the Main file, was a reference to the pic and LCD header files. The pic header file contains setting details for the PIC family of controllers in general like as the GPIO used by the PICs. The LCD header file refers to the LCD C file which handles the operation of the LCD screen used in simulation and implementation on the board.

Just below the Headers, the Function Prototypes are listed for the Functions coded at the foot of the Program. It should be noted that the Function Prototype for DisplayState has been commented out, for trying to reduce the effective file size of the program as the PIC has a limited amount of space for Program files.

```
/* Headers */
#include <pic.h>
#include "LCD.h"

/* Function Prototypes */
void Delay(int t);
void DisplayTime(int line);
void DisplaySetTime(int line);
//void DisplayState(int line); /* (uncomment for testing) */
void DisplayDate(int line);
void DisplaySetDate(int line);
void DisplayAlarmTime(int line);
void DisplaySetAlarmTime(int line);
```

Figure 3.1: Headers and Function Prototypes

3.2.2 Global Variables

Below are Global Variables used by the Digital Clock Program to function. It is necessary for these Variables to be made Global as the goal of this program is to make it possible to adjust the time through external controls (buttons). There are two distinct groups of Variables used with the top two rows being used to handle the data to be sent to the LCD, and temporary values identified by a leading 't' (bottom two rows) which are used to edit the current values of the top two rows of Variables.

```
/* Global Variables */
int milli=0, secs=0, mins=50, hours=23, dayCode=2, days=1, month=12,
    year=2016, state=0, alarmHours=0, alarmMins=0, timeOut=0, alarm=0,
    tHours=0, tMins=0, tDayCode=0, tDays=0, tMonth=0,
    tYear=0, tAlarmHours=0, tAlarmMins=0, tState=0;
```

Figure 3.2: Global Variables

3.2.3 Interrupt ISR

The interrupt ISR works directly with the PIC's timer and registers (identified in the research). The actual configuration of the registers is handled in the next code segment, but in the ISR code segment, the 10ms can be scaled up or down here, and through the use of nested if statements, the number of seconds to a minute, the number of minutes to an hour, the number of hours to a day, the number of days to a month, and the number of months to a year is established, an onward progression of the timing is controlled.

```
void interrupt ISR (void)
{
    if(TOIF==1)
    {
        TOIF = 0; //Set the interrupt flag to zero
        TMRO = 255; //Pre-load the initial time value
        milli += 1; //Increment milli(seconds) by 1
        if(milli==100) //If milli equals 10
        {
            secs ++; //Increment seconds by 1
            milli = 0; //Reset Milli to zero
            if(secs==60) //If secs equals 60
            {
                mins ++; //Increment mins by 1
                secs = 0; //Set secs to zero
                if(mins==60) //If mins equals 60
                {
                    hours ++; //Increment hours by 1
                    mins = 0; //Set mins to zero
                    if(hours==24) //If hours equals 24
                    {
                        days ++; //Increment days by 1
                        dayCode ++; //Increment dayCode by 1

                        hours = 0; //Set hours to 0

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

Figure 3.3: interrupt ISR

3.2.4 Main Function (Beginning)

At the head of the Main Function are settings used to configure the PIC. TRISA was the only I/O to be used, so this was then only element to be configured of the TRISEs. ADCON1 sets the pins used as Digital I/O. The OPTION_REG and INTCON are configured as previously discussed in the research. The LCD_Startup function

```
/* Main Function */
void main(void)
{
    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 with 255

    LCD_Startup();           //Calls function in LCD.c
}
```

Figure 3.4: Main Function

3.2.5 Start of Super-Loop

To make the Digital Clock work according to the Gregorian calendar, the number of days to month are fixed here, and once the daily and monthly limits are exceeded, the year is incremented forward.

```
while(1) //Begin super-loop
{
    /* For months that have 31 days to a month */
    if(((month==1)||(month==3)||(month==5)||(month==7)||(month==8)||(month==10))&&(days>31))
    {
        month++; //Increment month by 1
        days = 1; //Set days to 1
    }
    /* For February that has 28 days that month */
    if((month==2)&&(days>28)&&(year%4!=0))
    {
        month++; //Increment month by 1
        days = 1; //Set days to 1
    }
    /* For February that has 29 days on a leap year */
    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
    }
    /* For December */
    if(dayCode==7) dayCode = 0; //Sets dayCode to 0 if dayCode is equal to 7

    /* Would display the current state, uncomment below when testing. */
    // DisplayState(1);

    if(alarm==1) //If alarm set to 1, intervene
    {
        /* If the hours is equal to the alarmHours and the mins is equal to the alarmMins... */
        if((hours==alarmHours)&&(mins==alarmMins))
        {
            RA0 = 1; //Set output RA0 to 1 (Beep Beep, Beep Beep)
        }
    }
    if(alarm==0) //If alarm is set to 0
    {
        RA0 = 0; //Set output RA0 to zero (Knock the Alarm off)
    }
}
```

Figure 3.5: Super-Loop

3.2.6 Switch Case 0

In case 0, the current date and time values are displayed using the called Functions DisplayTime and DisplayDate, and the temporary values are set using the current running Date and Time values. In each case statement as shown in the state diagram, RA1 and RA2 are used to navigate between states, with RA3 and RA4 being used to adjust values. The current state of the alarm is also shown.

```
switch (state) /* The following cases are run provided the
               corresponding state value is met, i.e.
               State 0 - Case 0 */
{
    case 0:
        DisplayTime(0); //Call DisplayTime function
        DisplayDate(1); //Call DisplayDate function

        tHours = hours; //Sets tHours to equal hours
        tMins = mins;   //Sets tMins to equal mins
        tDayCode = dayCode; //Sets tDayCode to equal dayCode
        tDays = days;    //Sets tDays to equal days
        tMonth = month;  //Sets tMonth to equal month
        tYear = year;    //Sets tYear to equal year

        if(RA1) state = 8; //If RA1 is set to 1 set state to 8
        if(RA2) state = 1; //If RA2 is set to 1 set state to 1
        if(RA3==1) //If RA3 is set to 1, intervene
        {
            if(alarm==0) //If alarm is set to zero (not primed)
            {
                alarm = 1; //Sets alarm to 1 (primed)
            }
            else //Otherwise if RA3 is set to 0...
            {
                alarm = 0; //Set alarm to zero (not primed)
            }
        }
        break; //Break out
}
```

Figure 3.6: Switch Case 0

3.2.7 Switch Case 1

In case 1 by using inputs RA3 and RA4, the user can change the temporary value for hour, and when done can save by pressing RA1. If the selection exceeds 23 hours, then a setting of 0 is made, also if a selection goes the other way from 0, then a setting of 23 hours is made

```
case 1:
    DisplaySetTime(0);

    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
    {
        tHours++;                //Increment tHours by 1
        if(tHours>23) tHours=0;   //If tHours is greater than 23 set to zero
    }
    if(RA4)                       //If RA4 = 1
    {
        while(RA4);
        Delay(100);
        tHours--;                //decrement tHours by 1
        if(tHours<0) tHours=23;   //If tHours is greater than 23 set to zero
    }
    break;                       //Break out
```

Figure 3.7: Switch Case 1

3.2.8 Switch Case 2

Functioning similarly to case 1, the temporary value for minutes can be set here. A delay is present in both case 1 and here to slow down value changes.

```
case 2:
    DisplaySetTime(0);

    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, intervene
    {
        tMins++;                //Increment tMins by 1
        if(tMins>59) tMins=0;    //If tMins is greater than 59 set to zero
    }

    if(RA4)                       //If RA4 is = 1
    {
        while(RA4);
        Delay(100);
        tMins--;                //decrement tMins by 1
        if(tMins<0) tMins=59;    //If tMins is greater than 59 set to zero
    }
    break;                       //Break out
```

Figure 3.8: Switch Case 2

3.2.9 Switch Case 3

Switch Case 3 is used to change the text string for the day of the week e.g. MON for Monday. The sequence starts with a Sunday and progresses to Saturday whereby it then loops back around.

```
case 3:
    DisplaySetDate(1);

    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, intervene
    {
        tDayCode++;              //Increment temp dayCode by 1
        if(tDayCode>6) tDayCode=0; //If tDayCode is greater than 6 set to zero
    }
    if(RA4)                       //If RA4 = 1
    {
        while(RA4);
        Delay(100);
        tDayCode--;              //decrement tDayCode by 1
        if(tDayCode<0) tDayCode=6; //If tDayCode is less than 0 set to 6
    }
    break;                       //Break out
```

Figure 3.9: Switch Case 4

3.2.10 Switch Case 4

Case 4 handles the oddities of the Gregorian calendar such as the varying maximum days in a particular month, as well as whether or not that particular year is a leap year or not, which is achieved through if statements that identify the current day, month and year stored in the running variables.

```

case 4:
    DisplaySetDate(1);

    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, intervene
    {
        tDays++;                //Increment tDays by 1
        if(((tMonth==1)|| (tMonth==3)|| (tMonth==5)|| (tMonth==7)|| (tMonth==8)
        || (tMonth==10)|| (tMonth==12)) && (tMonth>31))
        /* If tDays equals 1 or 3 or 5 or 7 or 9 or 10
        and tDays greater than 31, intervene */
        {
            tDays = 1;           //Set days to 1
        }
        if((tMonth==2)&&(tDays>28)&&(tYear% 4 != 0))
        /* If tDays equals 2 and tDays greater than 29 and
        4 does not divide into year, intervene (not leap year) */
        {
            tDays = 1;           //Set days to 1
        }
        if((tMonth==2)&&(tDays>29)&&(tYear% 4 == 0))
        /* If tDays equals 2 and tDays greater than 29 and
        4 divides into year, intervene (leap year) */
        {
            tDays = 1;           //Set days to 1
        }
        if(((tMonth==4)|| (tMonth==6)|| (tMonth==9)|| (tMonth==11)) && (tMonth>30))
        /* If tMonth equals 4 or 6 or 9 or 11 and tMonth
        greater than 30, intervene */
        {
            tDays = 1;           //Set days to 1
        }
    }
    if(RA4)                       //If RA4 = 1
    {
        while(RA4);
        Delay(100);
        tDays--;                //decrement tDays by 1
        if(((tMonth==1)|| (tMonth==3)|| (tMonth==5)|| (tMonth==7)|| (tMonth==8)
        || (tMonth==10)|| (tMonth==12)) && (tMonth<1))
        /* If tMonth equals 1 or 3 or 5 or 7 or 8 or 10 and tDays greater than 31, intervene */
        {
            tDays = 31;          //Set days to 31
        }
        if((tMonth==2)&&(tMonth<1)&&(tYear% 4 != 0))
        /* If tMonth equals 2 and tDays greater than 29 and
        4 does not divide into year, intervene (not leap year) */
        {
            tDays = 28;          //Set days to 28
        }
        if((tMonth==2)&&(tMonth<1)&&(tYear% 4 == 0))
        /* If tMonth equals 2 and tDays greater than 29 and
        4 divides into year, intervene (leap year) */
        {
            tDays = 29;          //Set days to 29
        }
        if(((tMonth==4)|| (tMonth==6)|| (tMonth==9)|| (tMonth==11)) && (tMonth<1))
        /* If tMonth equals 4 or 6 or 9 or 11 and tDays greater than 30, intervene */
        {
            tDays = 30;          //Set days to 30
        }
    }
    break;                       //Break out

```

Figure 3.10: Switch Case 4

3.2.11 Switch Case 5

Case 5 is used to increment or decrement the chosen value for the month. The temporary value here is then used later to set the displayed variable for month.

```
case 5:
    DisplaySetDate(1);
    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, intervene
    {
        tMonth++;                //Increment tMonth by 1
        if(tMonth>12) tMonth=1;  //If tMonth is greater than 12 set to 1
    }
    if(RA4)                       //If RA4 = 1
    {
        while(RA4);
        Delay(100);
        tMonth--;                //decrement tDayCode by 1
        if(tMonth<1) tMonth=12;  //If tMonth is less than 0 set to 12
    }
    break;                       //Break out
```

Figure 3.11: Switch Case 5

3.2.12 Switch Case 6

This case allows the user to increment or decrement the chosen value for year. For obvious and practical reasons, the progression of years into millennia is not considered.

```
case 6:
    DisplaySetDate(1);
    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, intervene
    {
        tYear++;                //Increment tYear by 1
    }
    if(RA4)                       //If RA4 = 1
    {
        while(RA4);
        Delay(100);
        tYear--;                //decrement tYear by 1
    }
    break;                       //Break out
```

Figure 3.12: Switch Case 6

3.2.13 Switch Case 7

Case 7 is used to save all the changes that the user has selected for Date and Time as well as any settings chosen for when the Alarm should ring. After this step the program returns to Case 0.

```
case 7:
    DisplayTime(0);
    DisplayDate(1);

    hours = tHours;           //Sets hours to equal tHours
    mins = tMins;             //Sets mins to equal tMins
    dayCode = tDayCode;       //Sets dayCode to equal tDayCode
    days = tDays;             //Sets days to equal tDays
    month = tMonth;           //Sets month to equal tMonth
    year = tYear;             //Sets year to equal tYear
    alarmHours = tAlarmHours; //Sets alarmHours to equal tAlarmHours
    alarmMins = tAlarmMins;   //Sets alarmMins to equal tAlarmMins
    Delay(100);
    state = 0;                //Sets state to equal zero
    break;                    //Break out
```

Figure 3.13: Switch Case 7

3.2.14 Switch Case 8

This is the initial state for the user before setting the Alarm, where the default values for the Alarm are loaded into the temporary values.

```
case 8:
    DisplayDate(1);
    DisplayAlarmTime(0);
    tAlarmHours = alarmHours; //Sets tAlarmHours to equal alarmHours
    tAlarmMins = alarmMins;   //Sets tAlarmMins to equal alarmMins
    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;                    //Break out
```

Figure 3.14: Switch Case 8

3.2.15 Switch Case 9

Functioning similarly to when the user selected an hour value for the time, the user selects the temporary value for the Alarm hour before it is saved in state 7.

```
case 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, intervene
    {
        tAlarmHours++;        //Increment tAlarmHours by 1
        if(tAlarmHours>23) tAlarmHours=0; //If tAlarmHours is greater than 23 set to 0
    }
    break;                    //Break out
```

Figure 3.15: Switch Case 9

3.2.16 Switch Case 10 and Button While

This case allows the user to select a temporary value for minutes for the alarm before saving the setting in case 7. At the end outside of the Switch Case statement, there is a while running so that if any of the first three buttons are used, the other buttons should not trigger unless the pressed button is released.

```
case 10:
    DisplaySetAlarmTime(0);

    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
    {
        tAlarmMins++;           //Increment tAlarmMins by 1
        if(tAlarmMins>59) tAlarmMins=0; //If tAlarmMins is greater than 59 set to 0
    }
    break;                       //Break out
while(RA1 || RA2 || RA3);       //While RA1 or RA2 or RA3 are active do nothing
```

Figure 3.16: Switch Case 10

3.2.17 Delay Function

The Delay Function is located at the end of the Main Function, and is used whenever a Delay is required, usually to act as a means of de-bouncing the inputs.

```
/* Functions */
void Delay(int t)
{
    int i;                       // Counting variable is stored here
    for(i=0;i<t;i++);           /* for loop will run so long as the value of 'i' does
                                not exceed the value of 't' that is imported into this function,
                                the effect of this will create a loop that is variable
                                depending on the desired duration of the delay*/
}
```

Figure 3.17: Delay Function

3.2.18 DisplayTime Function

This function through the use of ASCII values and strings is used to format the Time shown for both the normal running of the clock and displaying the alarm setting.

```
void DisplayTime(int line)
{
    char timeString[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, intervene
    {
        timeString[0] = (char) hours / 10 + 48; //Displays the first hour digit.
        timeString[1] = (char) hours % 10 + 48; //Displays the second hour digit.
        timeString[2] = ':'; //Displays the : digit.
        timeString[3] = (char) mins / 10 + 48; //Displays the first minute digit.
        timeString[4] = (char) mins % 10 + 48; //Displays the second minute digit.
        timeString[5] = ' '; //Spaces
        timeString[6] = ' ';
        timeString[7] = ' ';
        timeString[8] = ' ';
        timeString[9] = ' ';
        timeString[10] = ' ';
        timeString[11] = ' ';
        timeString[12] = ' ';
        timeString[13] = ' ';
        timeString[14] = (char) 33; //ASCII 33 = !
        timeString[15] = 0; //Ends the string by placing 0 in the last string
    }

    if (alarm==0) //If alarm is equal to 0, intervene
    {
        timeString[0] = (char) hours / 10 + 48; //Displays the first hour digit.
        timeString[1] = (char) hours % 10 + 48; //Displays the second hour digit.
        timeString[2] = ':'; //Displays the : digit.
        timeString[3] = (char) mins / 10 + 48; //Displays the first minute digit.
        timeString[4] = (char) mins % 10 + 48; //Displays the second minute digit.
        timeString[5] = ' '; //Spaces
        timeString[6] = ' ';
        timeString[7] = ' ';
        timeString[8] = ' ';
        timeString[9] = ' ';
        timeString[10] = ' ';
        timeString[11] = ' ';
        timeString[12] = ' ';
        timeString[13] = ' ';
        timeString[14] = ' ';
        timeString[15] = 0; //Ends the string by placing 0 in the last string
    }

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

Figure 3.18: DisplayTime Function

The 'DisplayString' function should be included and pointed out that this is the function that takes the created string and displays the data content on LCD.

3.2.19 DisplaySetTime Function

Functioning in the same manner as the previous Function, this code handles time formatting.

```
void DisplaySetTime(int line)
{
    char timeString[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, intervene
    {
        timeString[0] = (char) tHours / 10 + 48; //Displays the first hour digit.
        timeString[1] = (char) tHours % 10 + 48; //Displays the second hour digit.
        timeString[2] = ':'; //Displays the : digit.
        timeString[3] = (char) tMins / 10 + 48; //Displays the first minute digit.
        timeString[4] = (char) tMins % 10 + 48; //Displays the second minute digit.
        timeString[5] = ' '; //Spaces
        timeString[6] = ' ';
        timeString[7] = ' ';
        timeString[8] = ' ';
        timeString[9] = ' ';
        timeString[10] = ' ';
        timeString[11] = ' ';
        timeString[12] = ' ';
        timeString[13] = ' ';
        timeString[14] = (char) 33; //ASCII 33 = !
        timeString[15] = 0; //Ends the string by placing 0 in the last string
    }

    if(alarm==0) //If alarm is equal to 0, intervene
    {
        timeString[0] = (char) tHours / 10 + 48; //Displays the first hour digit.
        timeString[1] = (char) tHours % 10 + 48; //Displays the second hour digit.
        timeString[2] = ':'; //Displays the : digit.
        timeString[3] = (char) tMins / 10 + 48; //Displays the first minute digit.
        timeString[4] = (char) tMins % 10 + 48; //Displays the second minute digit.
        timeString[5] = ' '; //Spaces
        timeString[6] = ' ';
        timeString[7] = ' ';
        timeString[8] = ' ';
        timeString[9] = ' ';
        timeString[10] = ' ';
        timeString[11] = ' ';
        timeString[12] = ' ';
        timeString[13] = ' ';
        timeString[14] = ' ';
        timeString[15] = 0; //Ends the string by placing 0 in the last string
    }
    DisplayString(line, timeString); //Sends the string to the LCD on line 0
}
```

Figure 3.19: DisplaySetTime Function

3.2.20 DisplayDate and DisplaySetDateFunction

This function's purpose is to return a character string back to the calling element depending on the integer sent to DisplayDate. ASCII values are used to represent letters on the LCD. An integer of 0 would show SUN on the LCD, and 2 would show TUE.

```
void DisplayDate(int line)
{
    char dayCodeString[5] = {0, 0, 0, 0, 0}; //Local string declared with 5 elements
    char monthString[5] = {0, 0, 0, 0, 0}; //Local string declared with 5 elements
    char dateString[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //Local string declared with 16 elements

    if(dayCode==0) //If dayCode is equal to zero
    {
        dayCodeString[0] = (char) 83; //ASCII 83 = S
        dayCodeString[1] = (char) 85; //ASCII 85 = U
        dayCodeString[2] = (char) 78; //ASCII 78 = N
        dayCodeString[3] = ' ';
        dayCodeString[4] = 0; //Ends the string by placing 0 in the last string
    }
    if(dayCode==1) //If dayCode is equal to 1
    {
        dayCodeString[0] = (char) 77; //ASCII 77 = M
        dayCodeString[1] = (char) 79; //ASCII 79 = O
        dayCodeString[2] = (char) 78; //ASCII 78 = N
        dayCodeString[3] = ' ';
        dayCodeString[4] = 0;
    }
}
```

Figure 3.20: DisplayDateFunction (Day)

Also, in this function is the formatting for the Month where an integer of 5 would show MAY on the LCD.

```
if(month==1) //If month is equal to 1
{
    monthString[0] = (char) 74; //ASCII 74 = J
    monthString[1] = (char) 65; //ASCII 65 = A
    monthString[2] = (char) 78; //ASCII 78 = N
    monthString[3] = ' ';
    monthString[4] = 0; //Ends the string by placing 0 in the last string
}
if(month==2) //If month is equal to 2
{
    monthString[0] = (char) 70; //ASCII 70 = F
    monthString[1] = (char) 69; //ASCII 69 = E
    monthString[2] = (char) 66; //ASCII 66 = B
    monthString[3] = ' ';
    monthString[4] = 0;
}
```

Figure 3.21: DisplayDateFunction (Month)

This code section concatenates all the date elements required into a string for the LCD, which is then sent along to the DisplayString Function.

```
/* Places dayCode, days, month, and year into the dateString
before forwarding to the LCD */
dateString[0] = dayCodeString[0];
dateString[1] = dayCodeString[1];
dateString[2] = dayCodeString[2];
dateString[3] = dayCodeString[3];
dateString[4] = days / 10 + 48;
dateString[5] = days % 10 + 48;
dateString[6] = ' ';
dateString[7] = monthString[0];
dateString[8] = monthString[1];
dateString[9] = monthString[2];
dateString[10] = monthString[3];
dateString[11] = year / 1000 + 48;
dateString[12] = year % 1000 / 100 + 48;
dateString[13] = year % 100 / 10 + 48;
dateString[14] = year % 10 + 48;
dateString[15] = 0; //Ends the string by placing 0 in the last string

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

Figure 3.22: DisplayDate Function (String Formatting)

It should be stated that the DisplaySetDate Function is coded in exactly the same manner with the only difference being the name of the function itself and is used for temporary values of date.

3.2.21 DisplaySetAlarmTime Function

This Function is used to display the user's selected time for the Alarm, along with an ASCII message to show that the Alarm time is shown (ALARM).

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

    alarmString[0] = (char) tAlarmHours / 10 + 48; //Displays the first hour digit.
    alarmString[1] = (char) tAlarmHours % 10 + 48; //Displays the second hour digit.
    alarmString[2] = ':'; //Displays the : digit.
    alarmString[3] = (char) tAlarmMins / 10 + 48; //Displays the first minute digit.
    alarmString[4] = (char) tAlarmMins % 10 + 48; //Displays the second minute digit.
    alarmString[5] = ' ';
    alarmString[6] = ' ';
    alarmString[7] = ' ';
    alarmString[8] = ' ';
    alarmString[9] = ' ';
    alarmString[10] = (char) 65; //ASCII 65 = A
    alarmString[11] = (char) 76; //ASCII 76 = L
    alarmString[12] = (char) 65; //ASCII 65 = A
    alarmString[13] = (char) 82; //ASCII 82 = R
    alarmString[14] = (char) 77; //ASCII 77 = M
    alarmString[15] = 0; //Ends the string by placing 0 in the last string

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

Figure 3.23: DisplaySetAlarmTime Function

3.2.22 DisplayState Function

The only purpose of this code element is to display the current state of the Switch Case statement for the purposes of testing. In actual application, it is possible to comment out this code segment for the purposes of reducing the effective file size.

```
void DisplayState(int line)
{
    char stateString[8] = {0, 0, 0, 0, 0, 0, 0, 0};

    stateString[0] = ' ';
    stateString[1] = ' ';
    stateString[2] = ' ';
    stateString[3] = ' ';
    stateString[4] = ' ';
    stateString[5] = ' ';
    stateString[6] = (char) state + 48;
    stateString[7] = 0;

    DisplayString(line, stateString);
}
```

Figure 3.24: DisplayState Function

3.2.23 Simulated Circuit

Below is the simulated circuit for which the code is written, using a suitable LCD and buttons interacting with a matching PIC microcontroller.

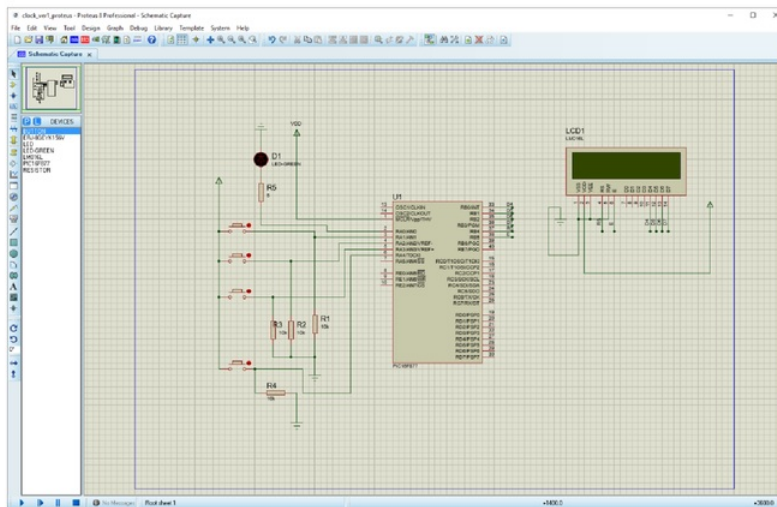


Figure 3.25: Complete Simulated Circuit

The code is explained well. There are some parts missing but the structure makes this

4 TESTING AND ANALYSIS OF RESULTS

4.1 SIMULATED TEST PLAN

To test the performance of the Digital Clock, there are two tests that should be run. The first test is to verify the performance of the Simulated circuit in Proteus, re-enabling the DisplayState Function and effectively comparing to the State Table and Diagram.

State under test	Method	Expected Result	Actual Result	Pass?/Fail?	Figure Reference
0	Observe for current Date and Time	Date and Time to be formatted as expected	Matching Expected	Pass	Figure 4.1
1	Increase set hour by 1	Hour to increase	Matching Expected	Pass	Figure 4.2
1	Decrease set hour by 1	Hour to decrease	Matching Expected	Pass	Figure 4.3
2	Increase set minute by 1	Minute to increase	Matching Expected	Pass	Figure 4.4
2	Decrease set minute by 1	Minute to decrease	Matching Expected	Pass	Figure 4.5
3	Increase set day by 1	Day to increase	Matching Expected	Pass	Figure 4.6
3	Decrease set day by 1	Day to decrease	Matching Expected	Pass	Figure 4.6
4	Increase set numeric day by 1	Numeric day to increase	Matching Expected	Pass	Figure 4.7
4	Decrease set numeric day by 1	Numeric day to decrease	Matching Expected	Pass	Figure 4.7
5	Increase set month by 1	Month to increase	Matching Expected	Pass	Figure 4.8
5	Decrease set month by 1	Month to decrease	Matching Expected	Pass	Figure 4.8
6	Increase set year by 1	Year to increase	Matching Expected	Pass	Figure 4.9
6	Decrease set year by 1	Year to decrease	Matching Expected	Pass	Figure 4.9
7	Save changes to settings	Changes set	Matching Expected	Pass	
8	Show Alarm setting	Current alarm settings shown	Matching Expected	Pass	Figure 4.10
9	Increase set hour for alarm by 1	Alarm hour to increase	Matching Expected	Pass	Figure 4.11

9	Decrease set hour for alarm by 1	Alarm hour to decrease	Matching Expected	Pass	Figure 4.11
10	Increase set minute for alarm by 1	Alarm minute to increase	Matching Expected	Pass	Figure 4.12
10	Decrease set minute for alarm by 1	Alarm minute to decrease	Matching Expected	Pass	Figure 4.12

4.2 SIMULATED TEST SCREENS

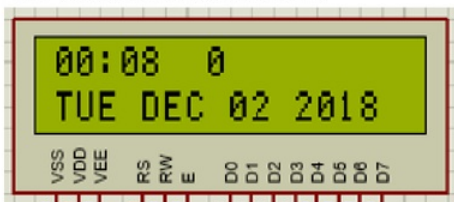


Figure 4.1: Case 0 Show D&T

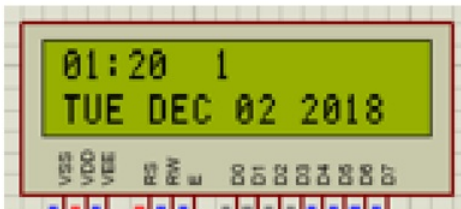


Figure 4.2: Case 1 Increment Hour

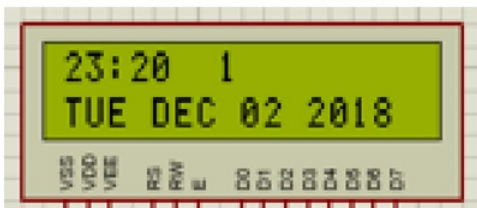


Figure 4.3: Case 1 Decrement Hour



Figure 4.4: Case 2 Increment Minute

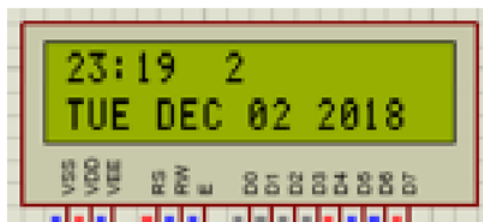


Figure 4.5: Case 2 Decrement Minute

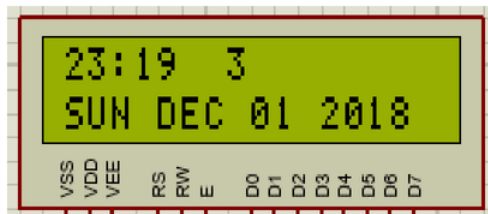


Figure 4.6: Case 3 Change Day

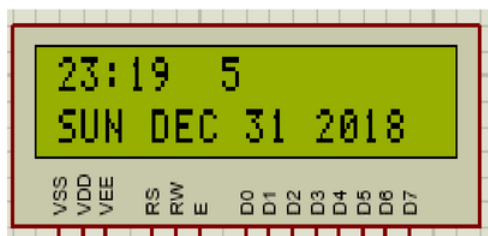


Figure 4.7: Case 4 Change Date (Number)

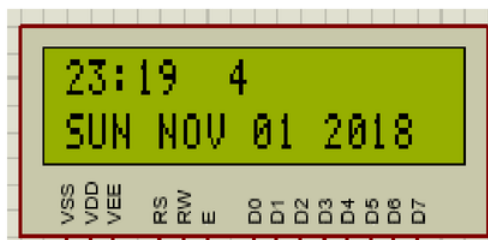


Figure 4.8: Case 5 Change Month

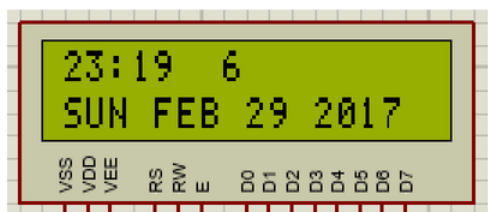


Figure 4.9: Case 6 Change Year

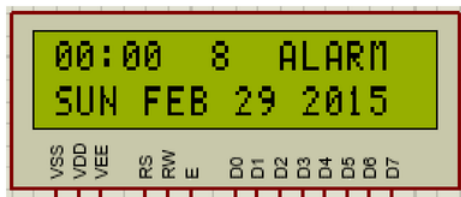


Figure 4.10: Case 8 Show Alarm setting

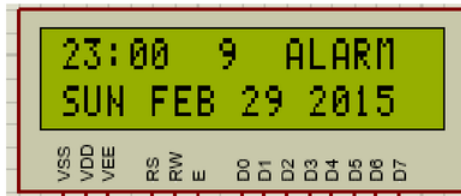


Figure 4.11: Case 9 Change Hour setting

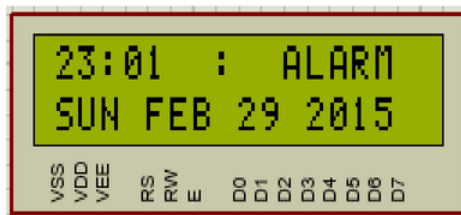


Figure 4.12: Case 10 Change Minute setting

4.3 SIMULATED TEST SUMMARY

With the Test Plan carried out on the Simulated version of the Digital Clock, and showing a positive result for functionality, it was determined that it was time to upload the program to the Development board.

4.4 UPLOADED PROGRAM TO BOARD

The next five images show the Program functioning on the Development Board, showing the Alarm being primed and set

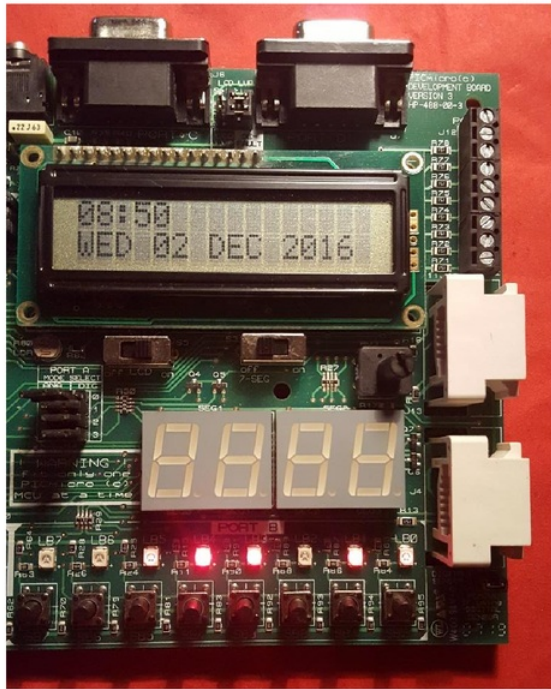


Figure 4.13: Clock running on board

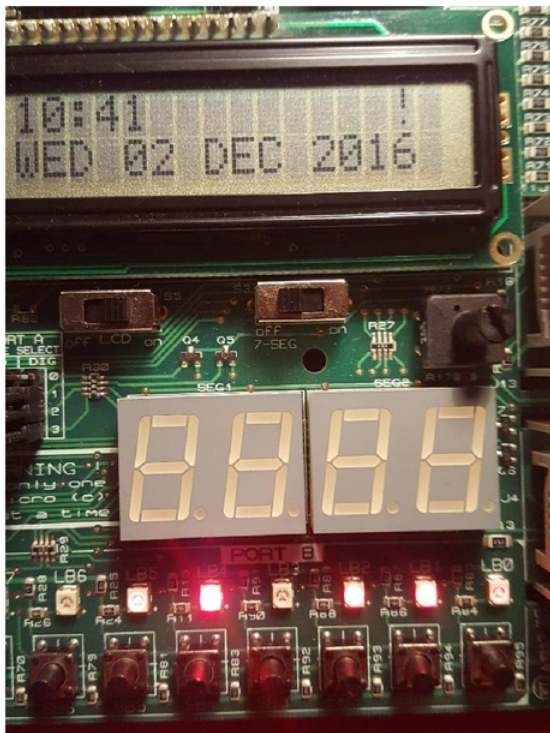


Figure 4.14: Alarm set

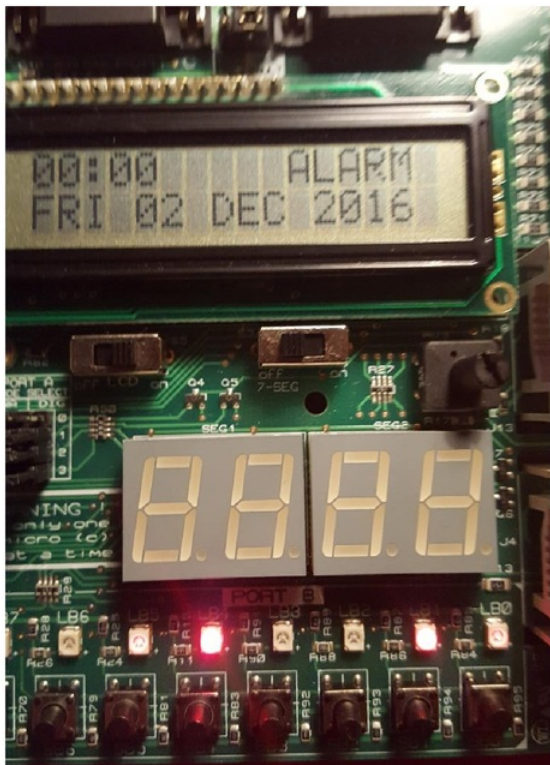


Figure 4.15: Edit Alarm

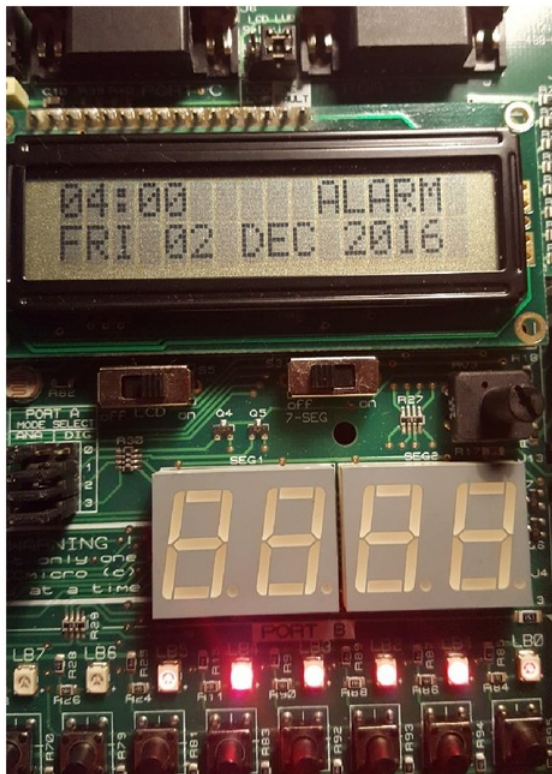


Figure 4.16: Changed Hour

5 CONCLUSIONS AND RECOMMENDATIONS

Using a State Table and Diagram, a viable code was programmed that addressed the needs of the Digital Clock. The testing of the Simulated Digital Clock showed that the Code functioned before uploading to the PIC on the Development Board.

There are a significant number of improvements that can be made to the Digital Clock

- Re-write the code in such a way as to reduce file size,
- When in set mode, the element being changed (hour, minute) could flash, indicating that changes can be made,
- The text Day could be changed according to date, month and year, though this would require more complex coding,
- Compartmentalise the code further by moving repeating code segments into other C files and referencing appropriately in the header (ASCII character string library function),
- An adjustable sleep function simplified to 5, 10 or 15 minute selections,
- Choice of alarm tunes stored in suitable format to be played by a buzzer, akin to old mobile phone ringtones.

Or should be noted that most of the recommendations stated above may not be possible to implement on hardware used in this investigation, so other hardware choices could be made.

*T
hi
s
m
a
k
e
s
s
e
n
s
e

.
C
o
ul
d
b
e
m
o
r
e
o
n
c
o
n
cl
u
si*

6 CITATION LIST

Assignment_1

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

52/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

Text Comment. There should be more to this.

PAGE 6

Text Comment. This should extend beyond the config and discuss other MCU alternatives with limitation to this task.

PAGE 7

PAGE 8

Text Comment. This looks a little complex but it will work.

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

Text Comment. The 'DisplayString' function should be include and pointed out that this is the function that takes the created string and displays the data content on LCD.

PAGE 20

PAGE 21

PAGE 22

PAGE 23

Text Comment. The code is explained well. There are some parts missing but the structure makes this part clear.

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

Text Comment. This makes sense. Could be more on conclusion.

PAGE 33

GRADING FORM: CSG-RUBIC

EMYR DYER

0

RESEARCH-INFORMED LITERATURE

 limited research and no refs used.

6/25

KNOWLEDGE AND UNDERSTANDING

 Intro is short but to the point. 2.5/5 System design is reasonable. Could do with more explanation. There is an explanation for the config. 5.5/10 Software and explanation is good and easy to follow. 7/10

15/25

ANALYSIS

 Some testing is missing e.g. alarm test. 6.5/10 Conclusion could have more content but good points on recommendations. 5.5/10

12/20

PRACTICAL APPLICATION AND DEP.

 The software design and application is good and demonstrates understanding. 7/10 Testing has parts missing i.e. alarm activation. 6/10

13/20

SKILLS FOR PROFESSIONAL PRAC.

 The report is good and easy to follow. 3/5 Team work is good. 3/5

6/10