

Scott Dedoncker Ass 1 FINAL

by Scott Dedoncker

FILE	12968_SCOTT_DEDONCKER_SCOTT_DEDONCKER_ASS_1_FINAL_139109_568561630.PDF (1.26M)		
TIME SUBMITTED	01-FEB-2018 12:56PM (UTC+0000)	WORD COUNT	5096
SUBMISSION ID	82662379	CHARACTER COUNT	28691

HND Electrical & Electronic Engineering

PSDE 5013: Configuration and Programming of Embedded System



Assignment 1

Calendar Alarm Clock

Scott Dedoncker

Valero Energy Electrical Apprentice

Date of Submission: 1/02/2018

Word Count: 5084

Contents

Table of Figures.....	2
Introduction	3
1. Research.....	4
2. Design.....	8
2.1 Specification.....	8
2.2 Program Design.....	10
2.2.1 Main Program	10
2.2.2 Functions.....	21
3. Testing and Analysis of Results	31
3.1 Testing the Clock.....	32
3.2 Testing the alarm	34
3.3 Testing on the HP488 Development Board	37
4. Conclusions	39
5. Appendices.....	40
Bibliography	41

Table of Figures

Figure 1 - PIC 16F877 microcontroller Features (Microchip, n.d.).....	4
Figure 2 - PIC 16F877 Microcontroller	4
Figure 3 - OPTION_REG Register	5
Figure 4 - INTCON Register (The Bare Necessities, n.d.).....	6
Figure 5 - Prescaler Values	6
Figure 6 - System State Diagram	9
Figure 7 - Header files, Function Prototypes and Global Variables	10
Figure 8 - ISR Code	11
Figure 9 - Main Program Code 1	13
Figure 10 - Main Program Code 2	13
Figure 11 - Main program Code 3	14
Figure 12 - Main Program Code 4	15
Figure 13 - Main Program Code 5	16
Figure 14 - Main Program Code 6	16
Figure 15 - Main Program Code 7	17
Figure 16 - Main Program Code 8	18
Figure 17 - Main Program Code 9	18
Figure 18 - Main program Code 10	19
Figure 19 - Main Program Code 11	19
Figure 20 Main Program Code 12	19
Figure 21 - Main Program Code 13	20
Figure 22 - Main Program Code 14	20
Figure 23 - Display Time Function.....	21
Figure 24 - Display Set Time Function.....	22
Figure 25 - Display Date Function Days	23
Figure 26 - Display Date Function Months.....	24
Figure 27 - Display Date function Date string	25
Figure 28 - Set Date Function Days	26
Figure 29 - Set Date Function Months	27
Figure 30 - Set Date Function Strings.....	28
Figure 31 - Display Alarm Function	29
Figure 32 - Display Set Alarm Function	29
Figure 33 - Display State Function	30
Figure 34 - Delay Function	30
Figure 35 - MPLab Circuit.....	31
Figure 36 - Alarm test 1.....	34
Figure 37 - Alarm Test 2	34
Figure 38 - Alarm Test 3	35
Figure 39 - Alarm Test 4	35
Figure 40 - Alarm Test 5	36
Figure 41 - Alarm test 6.....	36

Introduction

The purpose of this assignment is to investigate and carry out a practical project based on the PIC 16F877 microcontroller. It will involve investigating the microcontroller itself, the various methods around how it is used, as well as a small investigation into its timing functions and how it manipulates C programming language. From that research then, an alarm clock will need to be designed using the PIC. The code will need to be written out in full on MPLab and tested in a Proteus simulation.

The alarm clock system must have the following specifications:

- Display the correct time and date on a 24 hour clock cycle
- Have the option to change and set the time and date
- Include an alarm feature
- Have the option to change, set and activate the alarm time
- Display the full date on an LCD screen
- Provide an LED from the system to signify the activation of the alarm

This assignment will have the following aims:

- Produce a good level of research on the topic
- Relate the research to the practical
- Produce a state diagram of the system process
- Provide a good explanation of the program code
- Provide a thorough testing plan and results

1. Research

In order to build the program for this assignment the correct research needed to be carried out into the specific component used. That component was a PIC 16F877 microcontroller.

Features	Parameter Name	Value
2 PWM 10-bit	Program Memory Type	Flash
258 Bytes EEPROM data memory	Program Memory (KB)	14
ICD	CPU Speed (MIPS)	5
25mA sink/source per I/O	RAM Bytes	368
Self Programming	Data EEPROM (bytes)	256
Parallel Slave Port	Digital Communication Peripherals	1-UART, 1-SPI, 1-I2C1-MSSP (SPI/I2C)
	Capture/Compare/PWM Peripherals	2 Input Capture, 2 CCP,
	Timers	2 x 8-bit, 1 x 16- bit
	ADC	8 ch, 10-bit
	Comparators	2
	Temperature Range (C)	-40 to 125
	Operating Voltage Range (V)	2 to 5.5
	Pin Count	40

Figure 1 - PIC 16F877 microcontroller Features (Microchip, n.d.)

The above figure shows the specifications of the PIC taken directly from the manufacturer's website. The left column details the PIC's specific features such as its ability to self-program, and the 25mA sinking and source current of each I/O, of which there are 4 named RA through to RD. The microcontroller also has 3 timers at its disposal, 2 of which are 8bit, and 1 which is 16bit. It operates at a volt range of 2v to 5.5v and has 14KB of flash memory. The 4 I/O ports will be the most valuable parts of this PIC for this assignment. These ports are known as RA1 to RA4 and will be used as inputs into the PIC controller for this assignment.

(Microchip, n.d.)

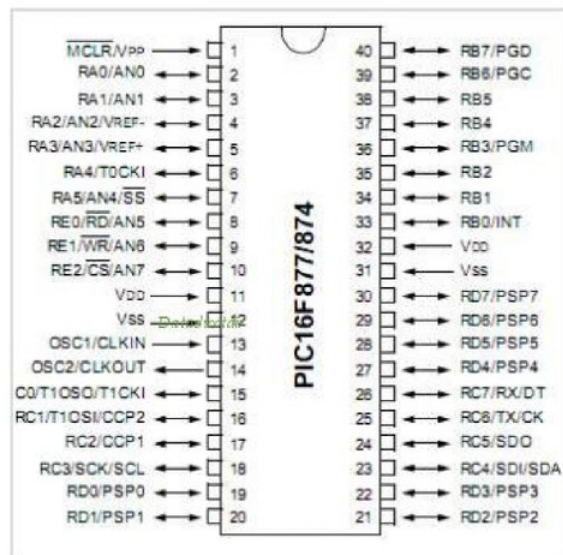


Figure 2 - PIC 16F877 Microcontroller

The manufacturer also states that in order for the timing functions to work, an interrupt service routine needs to be active. An interrupt service routine (ISR) is a software routine that is used to handle interrupts in a program. Microsoft explain that IDRs examine interrupts and assess them in order to determine how best to handle them. These interrupts are usually invoked by hardware on the system. ISRs must be extremely quick to deal with situations in front of them, otherwise this could have an adverse effect on the system and slow it down.

A good example of an ISR can be the even that takes place when a key on a keyboard is pressed. Each key of the keyboard determines a set action that needs to be completed by the CPU, outputting through the word processor or whatever program is being used as the current output. When a key is pressed the ISR will send a message to the CPU indicating which key has been depressed and will continue to do so until the key has been let go of. For this time the CPU will then output the correct value for whichever key is being pressed. Once the key has been let go, the ISR will interrupt the CPU state and tell it now that the key is no longer pressed, causing the CPU to stop processing the outputted value. This has been commonly referenced as a technological version of Newton's law of inertia, by which the program will continue to operate a function until it is interrupted. There are various interrupts available to the user of the PIC, and for this assignment the TIMER0 interrupt is the one that was used.

(Microsoft, 2006) (TechTerms, 2016)

The program will need to utilise data from two registers in order to function properly. This is due to all interrupts having two bit values, an input and an output, and each will be supplied by a universally used register. One of the two registers utilised is the OPTION_REG register and its data. Bit 5 is the piece of data that will be derived from this table and used in the assignment, as it is the one using TMRO and transitions on the RA4 pin, the one that will be connected to a button used to interrupt the cycle.

	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
	RBP0	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
	bit 7							bit 0
bit 7	RBP0: PORTB Pull-up Enable bit 1 = PORTB pull-ups are disabled 0 = PORTB pull-ups are enabled by individual port latch values							
bit 6	INTEDG: Interrupt Edge Select bit 1 = Interrupt on rising edge of RB0/INT pin 0 = Interrupt on falling edge of RB0/INT pin							
bit 5	T0CS: TMRO Clock Source Select bit 1 = Transition on RA4/T0CKI pin 0 = Internal instruction cycle clock (CLKOUT)							
bit 4	T0SE: TMRO Source Edge Select bit 1 = Increment on high-to-low transition on RA4/T0CKI pin 0 = Increment on low-to-high transition on RA4/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	PS2:PS0: Prescaler Rate Select bits							
	Bit Value	TMRO Rate	WDT Rate					
	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					

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 3 - OPTION_REG Register (The Bare Necessities, n.d.)

The second register used is the INTCON register as this would need to be used in conjunction with the first register. The INTCON register stands for interrupt control register, and is used to configure software interrupts, either starting or stopping processes in a system. The global interrupt enable/bit 7 is the most important part of this register as no interrupts will become active without this bit in effect. The 2 bits taken from this register will be bit 7 (the global interrupt enable) and bit 5 (timer 0 overflow enable). This will mean that when TIMER0 counts up to 256, it will intron make T01F turn to logic 1. This will then mean that as T01E is already logic 1 due to the TIMER0 being active, this will send a logic 1 signal through the system. This will cause the system to be interrupted.

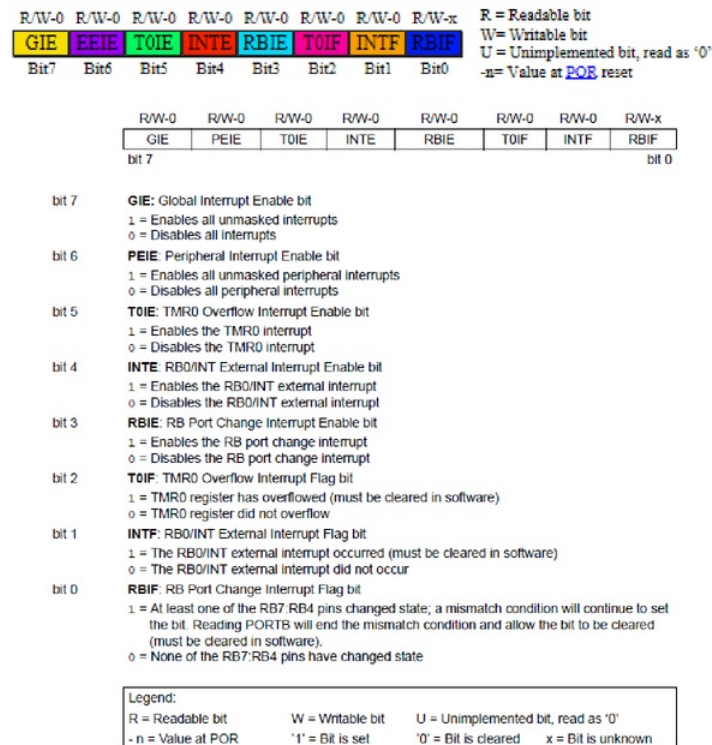


Figure 4 - INTCON Register (The Bare Necessities, n.d.)

So for this assignment the program will need a Oscillator clock, a 5MHz clock, two registers and a prescaler value. The prescaler value needs to be 256 or lower as the system peaks at 256. The prescaler values are calculated using the following table.

BIT Value	TMRO Rate	49152 / TMRO Rate	< 256 Yes/No
000	1 : 2	24576	No
001	1 : 4	12288	No
010	1 : 8	6144	No
011	1 : 16	3072	No
100	1 : 32	1536	No
101	1 : 64	768	No
110	1 : 128	384	No
111	1 : 256	193	Yes

Figure 5 - Prescaler Values

The calculations for this can be seen below.

Instruction cycle time: $1966800 / 4 = 4915200\text{HZ}$

$1 / 4915200 = 0.2\mu\text{s}$

Timer increment time = $0.2\mu\text{s} \times 256 \text{ prescale} = 51.2\mu\text{s}$

Counts for 10 ms = $10\text{ms} / 51.2\mu\text{s} = 195$ (take 2 off for error)

$195 - 2 = 193$

Preload count: $256 - 193 = 63$

This means that 63 needs to be preloaded into TMRO to get an interrupt delay of 10ms.

In the program the line:

```
→ milli += 10;  
→ if(milli == 1000);
```

Every time the Interrupt occurs 'milli' is increased by 10. This happens 100 times before 'seconds' increments by 1.

$100 \times 10\text{ms} = 1\text{sec}$

New Calculations

Instruction cycle time = $10,000,000 / 4 = 2,500,000$

$1 / 2.5 \text{ mil} = 4 \times 10^{-7} \text{ s}$

Timer increment time = $4 \times 10^{-7} \times 256 = 102.4\mu\text{s}$

Counts for 10ms = $10\text{ms} / 102.4\mu\text{s} = 98$

$98 - 2 = 96$ (take two off for error)

Preload count = $256 - 96 = 160$

*The calculation
is correct for
10MHZ clock.*

2. Design

2.1 Specification

Now that research had been completed on the PIC 16F877 Microcontroller, the next step to the assignment was designing the program. Using the microcontroller and a HP488 development board along with the two programs; MPLab and Proteus, a digital clock needed to be created with an alarm function. This clock needed to display a 24 hour clock, the day, the date, month and year in order to meet specification. The clock would also need to have the following functions shown in the state table below. It shows what each process is that the clock must achieve, and what steps need to be taken for that process to be achieved.

Process	State	RA1	RA2	RA3	RA4
Normal Clock Display	0	8	1	Toggle alarm	-
Set Hours	1	7	2	Increment +1	Decrement -1
Set Minutes	2	7	3	Increment +1	Decrement -1
Set Day	3	7	4	Increment +1	Decrement -1
Set Date	4	7	5	Increment +1	Decrement -1
Set Month	5	7	6	Increment +1	Decrement -1
Set Year	6	7	0	Increment +1	Decrement -1
Save Settings	7	-	-	-	-
Display Settings	8	0	9	-	-
Set Alarm Hours	9	7	10	Increment +1	Decrement -1
Set Alarm Minutes	10	7	8	Increment +1	Decrement -1

As the specification can be hard to visualise, a state diagram was produced in order to show the program moving from each state to the next.

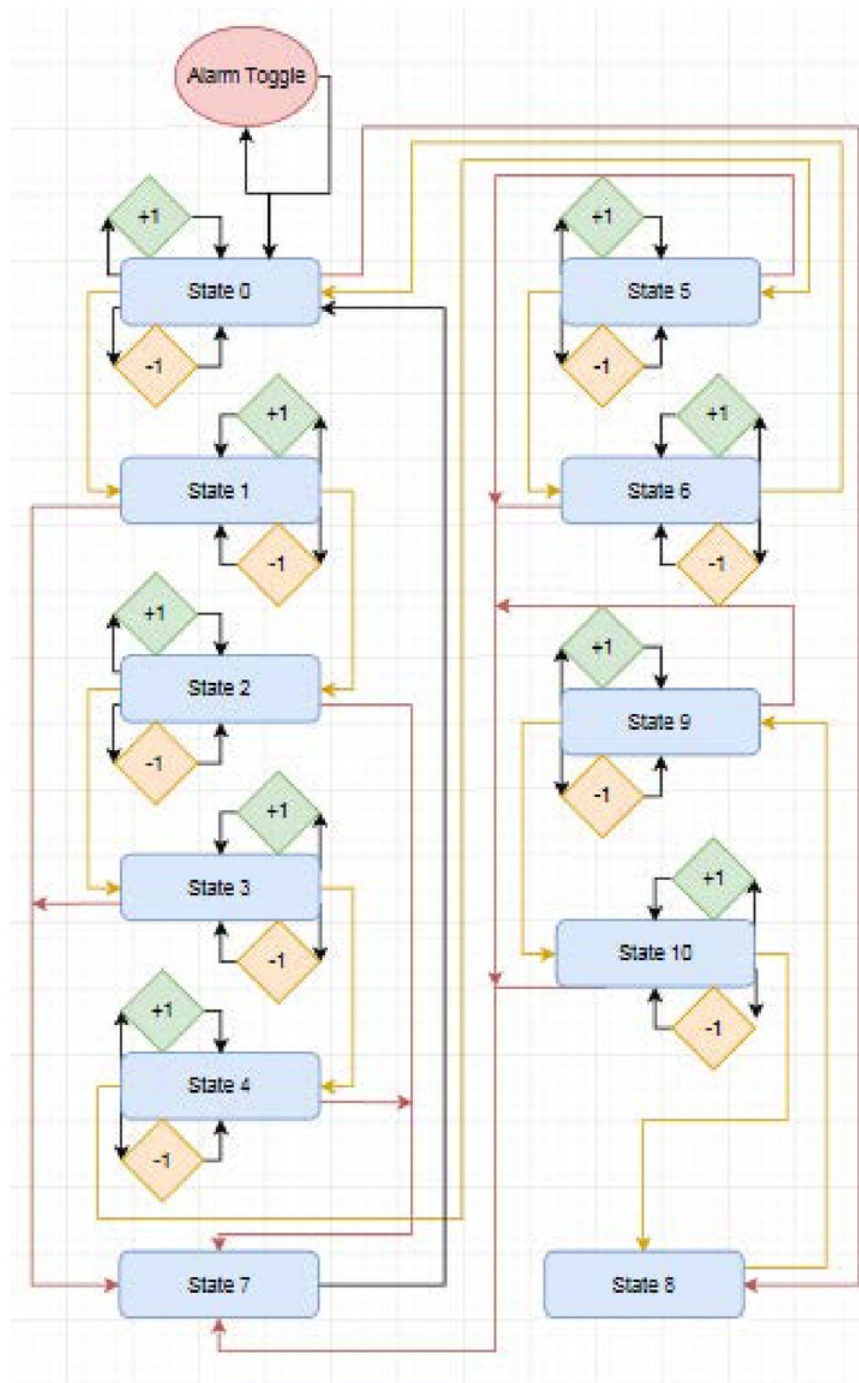


Figure 6 - System State Diagram

State
Diagram

2.2 Program Design

Now that each state and process had been derived the program needed to be built. This was done using MPLab and was written in the programming language of C. following now is the program code split into sections explaining how each part affected the final program.

2.2.1 Main Program

The first piece of code in 7 shows the 2 header files that were used for this program. These two were the pic.h file used for the C code, and the LCD.h file used to display on an LCD screen. This is how the clock will display to the user. Then all the function prototypes were listed out, all using an integer value of line except for the delay function, which would be using an integer value of t for time as this function will be delaying time. All the global variables were then listed. These would be variables that will be used throughout the entire program, and all come as integer values.

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

void DisplayTime(int line);                      //Function Prototypes, used 1
void DisplaySetTime(int line);
void DisplayState(int line);
void DisplayDate(int line);
void DisplaySetDate(int line);
void DisplayAlarmTime(int line);
void DisplaySetAlarmTime(int line);
void Delay(int t);

int milli=0, secs=0, mins=50, hours=23, day_code=2, days=1, month=12, year=2016,
    state=0, alarm_hours=0, alarm_mins=0, timeout=0, alarm=0,
    temp_hours=0, temp_mins=0, temp_day_code=0, temp_days=0, temp_month=0,
    temp_year=0, temp_al_hours=0, temp_al_mins=0, temp_state=0;
```

Figure 7 - Header files, Function Prototypes and Global Variables

The second section of code is for the interrupt service routine. This uses the TOIF and TMRO functions from the registers previously researched. This function works in that if milli reaches 100, the series of if statements will activate incrementing the seconds to their limit of 60 seconds, then the minutes to their limit of 60 minutes, then the hours to 24 hours, then the days to 31, then the months to 12, and finally the years indefinitely. This is the process of how the whole clock counts, by interrupting each stage once it reaches its limit. The last if statement also resets the day code to 0 once it tries to go beyond 7.

```
void interrupt ISR (void)
{
    if (TOIF==1)
    {
        TOIF = 0;           //Resets the flag interrupt to 0
        TMRO = 255;        //loads the preloaded time value
        milli += 10;        //increments in 10 millisecond steps
        if (milli==100)     //if milli is equal to 100, this if statement will activate
        {
            secs ++;        //increments seconds in steps of 1
            milli = 0;       //resets milli to 0
            if (secs==60)    //if secs equals 60, this if statement will activate
            {
                mins ++;    //increments mins in steps of 1
                secs = 0;    //Set secs to 0
                if (mins==60) //If mins equals 60, this if statement will activate
                {
                    hours ++; //Increment hours in steps of 1
                    mins = 0; //Set mins to 0
                    if (hours==24) //If hours equals 24, this if statement will activate
                    {
                        days ++; //Increment days in steps of 1
                        day_code ++; //Increment day_code in steps of 1
                        hours = 0; //sets hours to 0
                        if (days>31) //if days are greater than 31, this if statement will activate
                        {
                            month ++; //Increment month in steps of 1
                            days = 1; //Sets days to 1
                            if (month>12) //If month greater is greater than 12, this if statement will activate
                            {
                                year ++; //Increment year in steps of 1
                                month = 1; //Sets month to 1
                            }
                        }
                    }
                    if (day_code==7) day_code=0; //If day_code equals 7 then day_code is set to 0
                }
            }
        }
    }
}
```

This would be 100ms. the other divide by 10 will be included in another calculation.

Figure 8 - ISR Code

As it was not possible for the pic to run a clock of 1 second the code needed to be adjusted to accommodate milliseconds. This can be seen in the first if statement of the above figure. The following figures shows how that bit of code was derived. A block design was used on Proteus to test that section of code using an oscilloscope. The scope showed that the yellow plot rose for 5 sets of 0.2 microseconds. This meant that $2 \times 5 = 10$ which equates to 1 second. This is how the 1 second clock was installed and used on the PIC16F877.

*This should be in the testing section
as it is the prototype test for the 1
second timer.*

```
#include <pic.h>

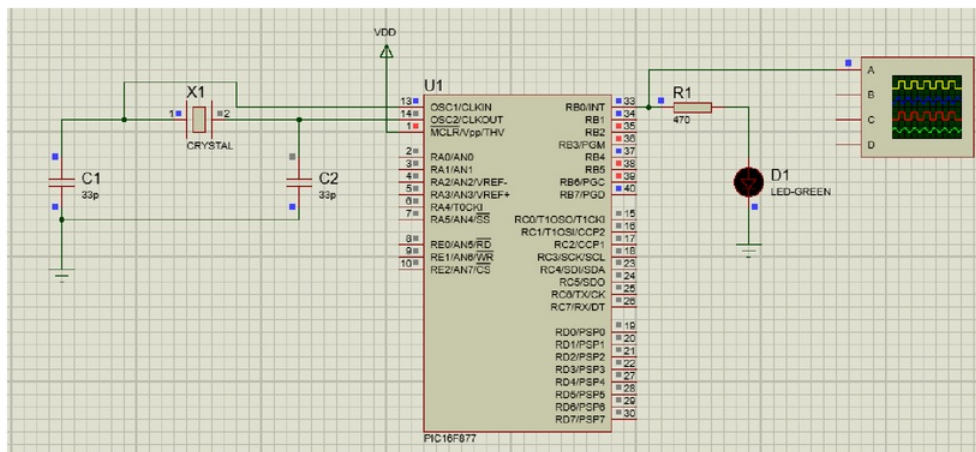
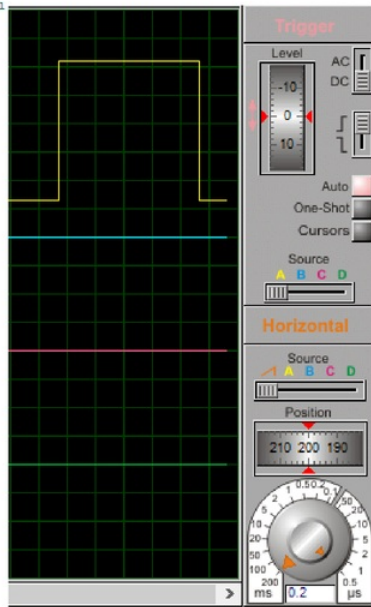
int milli=0, secs=0; //Must be declared global

void interrupt ISR (void)
{
    if(TOIF==1)
    {
        TOIF = 0;
        TH0 = 160;
        milli += 1;
        if(milli==100)
        {
            secs ++;
            milli = 0;
            RB0 = ~RB0; // Toggle RB0 pin
        }
    }
}

void main(void)
{
    TRISB = 0x00;

    OPTION_REG = 0x87;
    INCON = 0xA0;

    while(1)
    {
    }
}
```



The next function in the program is the main program. This is where all the functions are called and used. The device is configured in this section of the program using port configuration. This can be seen with ports TRISA/B/C being setup to equal their correct values. TRISA is set to 0001 1110 while the other 2 ports are set to 0000 0000. TMRO is loaded with the count limit of 255. Ports for the two registers of OPTION_Reg and INTCON are also set here. The LCD_Startup function is also activated here. This is a function taken from LCD.h and is not found within this programming code. It is used to activate the LCD screen.

```

////////////////////////////////////Main Program////////////////////////////////////
void main(void)                                //Main program, where all functions will be called
{
    TRISA = 0x1E;                               //Configure portA = 0001 1110
    TRISE = 0x00;                               //Configure portB = 0000 0000
    TRISC = 0x00;                               //Configure portC = 0000 0000
    ADCON1 = 0x07;                             //Configure portADCON1 = 0000 0111
    OPTION_REG = 0x87;                         //Configure OPTION_REG = 1000 0111
    INTCON = 0xA0;                             //Configure INTCON = 1010 0000
    TMRO=255;                                  //Preloads TMRO

    LCD_Startup();                             //Function call for LCD_startup
}

```

Needs more detail.

Figure 9 - Main Program Code 1

The first step in the main program is to activate a super loop. This was done using a while loop. This only activated once and while the LCD screen is switched on. The first if statement here determines whether a month has 30 or 31 days. This is so certain months will not reset back to date = 1 too early or too late. The second if statement is for a leap year, and will activate if it is the second month, 28 days, and the fourth year, it will then enter a 29th day for that month indicating a leap year. The third if statement is used for December. This activated if the month equals 12 (December) and the day equals 31. The super loop will then reset the clock to month 1 (January and day to equal 1. It will also increment the year value by 1. The fourth if statement resets the days from 7 to 0 when it tries to go beyond 7.

```

while(1)                                       //while starts the super-loop
{
    if((month==1)||month==3||month==5||month==7||month==8||month==10){days>31}
    //If month equals 1 or 3 or 5 or 7 or 8 or 10 and days > 31
    {
        month++;                             //Increments month in steps of 1
        days = 1;                             //Sets days to 1
    }

    if((month==2){days>28}{year%4 !=0})      //If month equals 2 and days is > 28 and year does not divide by 4, then this is not a leap year
    {
        month++;                             //Increments month in steps of 1
        days = 1;                             //Set days to 1
    }

    if((month==12){days>31})                 //If months equals 12 and days > 31, this iff statement activates
    {
        year++;                             //Increment year in steps of 1
        month = 1;                           //Set month to 1
        days = 1;                             //Set days to 1
    }

    if(day_code==7) day_code = 0;             //Sets day_code to 0 if day_code is equal to 7
}

```

The config settings are set here. This should be pointed out.

Figure 10 - Main Program Code 2

The next piece of code in this program was to check to see if the alarm has activated or not. The first if statement will activate if the alarm equals 1. This if statement will only work if the user has set the alarm to active on the clock. This if statement will then check the time until the current time on the clock matches up with the set alarm time in the clock settings. Once this happens it will then send an output signal to RA), this being the output pin that leads to the alarm LED. The second if statement will be active while alarm equals 0, and will therefore stop any output from RA0.

Good explanation of this logic function.

From here the switch function is activated. This is used to switch between states. From here each case will represent a state from the state diagram shown previously in the research. Case 0 shows that it will merely display the data on the clock LCD, showing the time if it equals logic 0, and date if it equals logic 1. The program then goes on to set each variable to its correct value.

```

if(alarm==1)                                //If alarm equals 1, this if statement activates
{
    if((hours==alarm_hours)&&(mins==alarm_mins))
    //If hours equals alarm_hours and mins equals alarm_mins or retained, alarm equals 1 and this if statement activates
    {
        RA0 = 1;                            //Set output RA0 to 1
    }
}
if(alarm==0)                                //If alarm equals 0 this if statement activates
{
    RA0 = 0;                                //Set output RA0 to 0
}

switch(state)
{
    case 0:
        DisplayTime(0);
        DisplayDate(1);

        temp_hours = hours;
        temp_mins = mins;
        temp_day_code = day_code;
        temp_days = days;
        temp_month = month;
        temp_year = year;

```

Figure 11 - Main program Code 3

The next step in the program code is the continuation of state switching. From here the program looks for either RA1 or RA2 to be activated. If RA1 is pressed, state 8 is the chosen path, if RA2 then state 1. If neither is pressed then the program will continue to display the incrementing time on the LCD display. The third if statement is for when RA3 is pressed in while the clock is in normal display mode. This if function and its sub if functions will (if off) now set the alarm to active, or (if on) deactivate. From here there is a break in the program which breaks out from case 0. The next case is case 1. This case first runs the display set time function to show the correct time on the clock. It then has its if statement options for RA1 to be pressed and go to case 7, RA2 to go to case 2, or RA3 to be pressed and increment the hours in steps of +1. If the hours then reach 23, the sub if statement will activate, setting the hours back to 0 to signal midnight and the start of a new day cycle. There fourth if statement is if RA4 is pressed. This will activate a while loop for the time of RA4 being pressed, and will call the delay function, giving it an integer variable of 100. The hours will then decrement in steps of -1, once again resetting but this time from 0 to 23 rather than the opposite way. There is then a break in the case.

```

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

if(RA3==1)                   //If RA3 is equal to 1 this if statement activates
{
    if(alarm==0)              //If alarm equals 0
    {
        alarm = 1;            //Sets alarm to equal 1
    }
    else                      //Else command, if alarm is not equal to 0
    {
        alarm = 0;            //Sets alarm to 0
    }
}

break;

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

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

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

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

    break;                    //This breaks out from each case

```

Figure 12 - Main Program Code 4

Case 2 is now shown. This once again displays the set time. Options for the if statements are RA1 = case 7, RA2 = case 3, and RA4 is used to increment the minutes in steps of +1, resetting to 0 after 59 signalling the passing of 60 minutes. RA4 then calls for the delay function again, giving it the 100 integer and the minutes are then decremented by steps of -1, resetting at 0 to 59. There is then a break in the code.

```

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

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

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

```

Figure 13 - Main Program Code 5

Case 3 once again displays the set date. It has options for RA1 = case 7, RA2 = case 4, RA3 to increment the days, resetting at day 7 to 0, and RA4 calling the delay function, decrementing the days and resetting at 0 to 7. There is then the case break.

```

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

    if(RA1) state = 7;                     //If RA1 is equal to 1 set state to equal 7
    if(RA2) state = 4;                     //If RA2 is equal to 1 set state to equal 4
    if(RA3)                                //If RA3 is equal to 1 step in
    {
        temp_day_code++;                   //Increment temp day_code by 1
        if(temp_day_code>6) temp_day_code=0; //If temp_day_code is greater than 6 set to zero
    }

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

        temp_day_code--;                   //decrement temp_day_code by 1
        if(temp_day_code<0) temp_day_code=6; //If temp_day_code is less than 0 set to 6
    }
    break;

```

Figure 14 - Main Program Code 6

Case 4 follows and also displays the clock date on the LCD display. The options for the if statements are RA1 = case 7, RA2 = case 5, RA3 for day incrementing and RA4 for day decrementing. The program then uses a series of if statements depending on whether RA3 or RA4 have been pressed, to determine whether the day minimum/maximum cap for that month has been reached. There is then a case break.

```

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

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

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

        Delay(100);

        temp_days--;                       //deccrement temp_days by 1

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

    break;

```

Figure 15 - Main Program Code 7

Case 5 starts off by displaying the set date again, having RA1 = case 7, RA2 = case 6, RA3 to increment the months in steps of +1 and have them reset at 12 back to 1, and RA4 used to decrement the months in steps of -1, resetting at 0 to 12. There is then a break at the end.

```

case 5:                                     //Case 5, run if state equals 5
    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 step in
    {
        temp_month++;                     //Increment temp_month by 1
        if(temp_month>12) temp_month=1;    //If temp_month is greater than 12 set to 1
    }

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

        Delay(100);

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

    break;

```

Figure 16 - Main Program Code 8

Case 6 starts off by displaying the set date again, having RA1 = case 7, RA2 = case 0, RA3 to increment the year in steps of +1, and RA4 used to decrement the years in steps of -1. There is then a break at the end.

```

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

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

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

        Delay(100);

        temp_year--;                     //decrement temp_year by 1
    }

    break;

```

Figure 17 - Main Program Code 9

Case 7 shows that the time will be displayed if a logic 0 is detected, and the date will be displayed if a logic 1 is detected. The variables are then set to equal the temporary alarm times. There is also a function call for the delay function and then state 0 is called, returning to the first state.

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

    hours = temp_hours;
    mins = temp_mins;
    day_code = temp_day_code;
    days = temp_days;
    month = temp_month;
    year = temp_year;
    alarm_hours = temp_al_hours;
    alarm_mins = temp_al_mins;

    Delay(100);
    state = 0;
    break;
```

Figure 18 - Main program Code 10

Case 8 runs the display for the alarm date and alarm time. These are called in function headers. The temporary variables for alarm hours and mins are then set to the actual alarm hours and mins variables. RA1 takes the program to case 0 while RA2 takes the program to state 9.

```
case 8:
    DisplayDate(1);
    DisplayAlarmTime(0);
    temp_al_hours = alarm_hours;
    temp_al_mins = alarm_mins;

    if(RA1) state = 0;
    if(RA2) state = 9;

    break;
```

Figure 19 - Main Program Code 11

Case 9 is used to increment the set alarm hours. RA1 takes the program to case 7, while RA2 takes the program to case 10. RA3 then increments the hours in steps if +1.

```
case 9:
    DisplaySetAlarmTime(0);
    if(RA1) state = 7;
    if(RA2) state = 10;
    if(RA3)
    {
        temp_al_hours++;
        if(temp_al_hours>23) temp_al_hours=0;
    }

    break;
```

//Case 9, run if state equals 9

//If RA1 is equal to 1 set state to equal 7

//If RA2 is equal to 1 set state to equal 10

//If RA3 is equal to 1 step in

//Increment temp_al_hours by 1

//If temp_al_hours is greater than 23 set to 0

Figure 20 Main Program Code 12

Case 10 increments the alarm minutes. RA1 goes to case 7, RA2 to case 8 and RA3 increments the minutes in steps of +1.

```

case 10:                                     //Case 10, run if state equals 10
    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;

```

Figure 21 - Main Program Code 13

The final statement in this super loop is a while statement. This statement stops the program if any buttons are pressed after the case statement.

```

    break;
}
while(RA1 || RA2 || RA3 || RA4)            //While RA1 or RA2 or RA3 are active do nothing
{
}

```

Figure 22 - Main Program Code 14

The
state
function
ns are
explained
reason
ably
well.

2.2.2 Functions

Function definitions make up the bulk of a program. To make a program more fluid they are derived into function calls that are prototyped at the top of a program and called from outside the main program. The first function created in this program was the display time function. This function uses a 16 value string array with 16 declared variables of 0 due to the LCD screen being 16 columns long. The array is set to char as this is what is used in C programming to store characters. Each 0 represents a character of the ASCII table. All 16 0's as it stands represent nothing. Each character of the string will then be declared a purpose value, and that value is depending on whether the alarm is activated or not. Each line of the time strings represent one vertical column on the LCD Display. So time strings 0 and 1 represent the two digits that will be declared for the hours, and strings 3 and 4 will represent the two digits that will be declared for the minutes. Each char value is divided by 10 and then added to 48 to give it the ASCII value equivalent that can be displayed by the LCD display. The ASCII table can be found in the appendices. Each pair of apostrophes represent 1 space. Time string 2 is used to display the clock ':'. the only difference between the 2 if statements here is that if the alarm is active, time string 14 will display a char value of '!' on the LCD display.

```
void DisplayTime(int line)
{
    char time_string[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    if (alarm==1)
    {
        time_string[0] = (char) hours / 10 + 48;
        time_string[1] = (char) hours % 10 + 48;
        time_string[2] = ':';
        time_string[3] = (char) mins / 10 + 48;
        time_string[4] = (char) mins % 10 + 48;
        time_string[5] = ' ';
        time_string[6] = ' ';
        time_string[7] = ' ';
        time_string[8] = ' ';
        time_string[9] = ' ';
        time_string[10] = ' ';
        time_string[11] = ' ';
        time_string[12] = ' ';
        time_string[13] = ' ';
        time_string[14] = (char) 33;
        time_string[15] = 0;
    }

    if (alarm==0)
    {
        time_string[0] = (char) hours / 10 + 48;
        time_string[1] = (char) hours % 10 + 48;
        time_string[2] = ':';
        time_string[3] = (char) mins / 10 + 48;
        time_string[4] = (char) mins % 10 + 48;
        time_string[5] = ' ';
        time_string[6] = ' ';
        time_string[7] = ' ';
        time_string[8] = ' ';
        time_string[9] = ' ';
        time_string[10] = ' ';
        time_string[11] = ' ';
        time_string[12] = ' ';
        time_string[13] = ' ';
        time_string[14] = ' ';
        time_string[15] = 0;
    }
    DisplayString(line, time_string);
}
```

This explanation could be clearer.

The function here is the LCD function with the create

Figure 23 - Display Time Function

The second function used in this program is very similar to the first. This function works in the exact same way to the display time function, however this one merely uses values that are used for editing the clocks time. This is shown by the use of temp hours and temp mins variables when converting to ASCII for the LED display.

```

////////////////////////////////////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};

    if(alarm==1)
    {
        time_string[0] = (char) temp_hours / 10 + 48;
        time_string[1] = (char) temp_hours % 10 + 48;
        time_string[2] = ':';
        time_string[3] = (char) temp_mins / 10 + 48;
        time_string[4] = (char) temp_mins % 10 + 48;
        time_string[5] = ' ';
        time_string[6] = ' ';
        time_string[7] = ' ';
        time_string[8] = ' ';
        time_string[9] = ' ';
        time_string[10] = ' ';
        time_string[11] = ' ';
        time_string[12] = ' ';
        time_string[13] = ' ';
        time_string[14] = (char) 33;
        time_string[15] = 0;
    }

    if(alarm==0)
    {
        time_string[0] = (char) temp_hours / 10 + 48;
        time_string[1] = (char) temp_hours % 10 + 48;
        time_string[2] = ':';
        time_string[3] = (char) temp_mins / 10 + 48;
        time_string[4] = (char) temp_mins % 10 + 48;
        time_string[5] = ' ';
        time_string[6] = ' ';
        time_string[7] = ' ';
        time_string[8] = ' ';
        time_string[9] = ' ';
        time_string[10] = ' ';
        time_string[11] = ' ';
        time_string[12] = ' ';
        time_string[13] = ' ';
        time_string[14] = ' ';
        time_string[15] = 0;
    }

    DisplayString(line, time_string);
}

```

Figure 24 - Display Set Time Function

The display date function is the largest function as it has to convert all the variables for the date into ASCII values to be displayed by the LCD screen. The first part of the code shows char arrays being declared for the days, months and date strings. This works in the same way as previously explained. Each if statement for the day codes then display 3 characters for each day of the week depending on what the day code is. So for example a day code = 0 = Mon, day code = 4 = Fri. The number given to each char value represents the ASCII equivalent of the letter.

```

//////////////////////////////////////Display Date Function//////////////////////////////////////
void DisplayDate(int line)
{
    char day_code_string[5] = {0, 0, 0, 0, 0};
    char month_string[5] = {0, 0, 0, 0, 0};
    char date_string[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    if(day_code==0)
    {
        day_code_string[0] = (char) 83;
        day_code_string[1] = (char) 85;
        day_code_string[2] = (char) 78;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if(day_code==1)
    {
        day_code_string[0] = (char) 77;
        day_code_string[1] = (char) 79;
        day_code_string[2] = (char) 78;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if(day_code==2)
    {
        day_code_string[0] = (char) 84;
        day_code_string[1] = (char) 85;
        day_code_string[2] = (char) 69;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if(day_code==3)
    {
        day_code_string[0] = (char) 87;
        day_code_string[1] = (char) 69;
        day_code_string[2] = (char) 68;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if(day_code==4)
    {
        day_code_string[0] = (char) 84;
        day_code_string[1] = (char) 72;
        day_code_string[2] = (char) 86;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if(day_code==5)
    {
        day_code_string[0] = (char) 70;
        day_code_string[1] = (char) 82;
        day_code_string[2] = (char) 73;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if(day_code==6)
    {
        day_code_string[0] = (char) 83;
        day_code_string[1] = (char) 68;
        day_code_string[2] = (char) 84;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
}

```

Figure 25 - Display Date Function Days

Below is the code for the month strings in the display date function. This works in the exact same way as the day code strings, just merely using the variables for the months instead.

```
if (month==1)
{
    month_string[0] = (char) 74;
    month_string[1] = (char) 65;
    month_string[2] = (char) 78;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==2)
{
    month_string[0] = (char) 70;
    month_string[1] = (char) 69;
    month_string[2] = (char) 66;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==3)
{
    month_string[0] = (char) 77;
    month_string[1] = (char) 65;
    month_string[2] = (char) 82;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==4)
{
    month_string[0] = (char) 65;
    month_string[1] = (char) 80;
    month_string[2] = (char) 82;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==5)
{
    month_string[0] = (char) 77;
    month_string[1] = (char) 65;
    month_string[2] = (char) 89;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==6)
{
    month_string[0] = (char) 74;
    month_string[1] = (char) 85;
    month_string[2] = (char) 78;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==7)
{
    month_string[0] = (char) 74;
    month_string[1] = (char) 85;
    month_string[2] = (char) 76;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==8)
{
    month_string[0] = (char) 65;
    month_string[1] = (char) 85;
    month_string[2] = (char) 71;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==9)
{
    month_string[0] = (char) 83;
    month_string[1] = (char) 69;
    month_string[2] = (char) 80;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==10)
{
    month_string[0] = (char) 79;
    month_string[1] = (char) 67;
    month_string[2] = (char) 84;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==11)
{
    month_string[0] = (char) 78;
    month_string[1] = (char) 79;
    month_string[2] = (char) 86;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month==12)
{
    month_string[0] = (char) 68;
    month_string[1] = (char) 69;
    month_string[2] = (char) 67;
    month_string[3] = ' ';
    month_string[4] = 0;
}
```

Figure 26 - Display Date Function Months

The last section of this function is used to set the bits for the date string array to be displayed. This compiles all the converted ASCII codes together and arranges them into the correct bit columns of the array to be displayed on the LCD screen. The first 4 bits of the array are used for the day code, then bits 4 and 5 are used for the actual days, converting them there into their ASCII equivalent. Bit 6 uses a spacer before bits 7 to 10 are used to display the month string. Bits 11 to 14 are then used to display the year, converting each value into its ASCII equivalent. The function then ends with a function call for the Display String function, setting line, and date string as its variables.

```

    date_string[0] = day_code_string[0];    /
    date_string[1] = day_code_string[1];
    date_string[2] = day_code_string[2];
    date_string[3] = day_code_string[3];
    date_string[4] = days / 10 + 48;
    date_string[5] = days % 10 + 48;
    date_string[6] = ' ';
    date_string[7] = month_string[0];
    date_string[8] = month_string[1];
    date_string[9] = month_string[2];
    date_string[10] = month_string[3];
    date_string[11] = year / 1000 + 48;
    date_string[12] = year % 1000 / 100 + 48;
    date_string[13] = year % 100 / 10 + 48;
    date_string[14] = year % 10 + 48;
    date_string[15] = 0;

    DisplayString(line, date_string);
}

```

More is needed here.

Figure 27 - Display Date function Date string

This is function outline

A carbon copy of the previous function, display set date is exactly the same as the display date function, however it is used when editing the parameters of the clock specifically. This uses temporary values for the date, month and years.

```

//////////////////////////////////////Display Set Date Function//////////////////////////////////////
void DisplaySetDate(int line)
{
    char day_code_string[5] = {0, 0, 0, 0, 0};
    char month_string[5] = {0, 0, 0, 0, 0};
    char date_string[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    if(temp_day_code==0)
    {
        day_code_string[0] = (char) 83;
        day_code_string[1] = (char) 85;
        day_code_string[2] = (char) 78;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if(temp_day_code==1)
    {
        day_code_string[0] = (char) 77;
        day_code_string[1] = (char) 79;
        day_code_string[2] = (char) 78;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if(temp_day_code==2)
    {
        day_code_string[0] = (char) 84;
        day_code_string[1] = (char) 85;
        day_code_string[2] = (char) 69;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if(temp_day_code==3)
    {
        day_code_string[0] = (char) 87;
        day_code_string[1] = (char) 69;
        day_code_string[2] = (char) 68;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if(temp_day_code==4)
    {
        day_code_string[0] = (char) 84;
        day_code_string[1] = (char) 72;
        day_code_string[2] = (char) 85;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if(temp_day_code==5)
    {
        day_code_string[0] = (char) 70;
        day_code_string[1] = (char) 82;
        day_code_string[2] = (char) 73;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if(temp_day_code==6)
    {
        day_code_string[0] = (char) 83;
        day_code_string[1] = (char) 65;
        day_code_string[2] = (char) 84;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
}

```

Figure 28 - Set Date Function Days

```

if (temp_month==1)
{
    month_string[0] = (char) 74;
    month_string[1] = (char) 65;
    month_string[2] = (char) 78;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (temp_month==2)
{
    month_string[0] = (char) 70;
    month_string[1] = (char) 69;
    month_string[2] = (char) 66;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (temp_month==3)
{
    month_string[0] = (char) 77;
    month_string[1] = (char) 65;
    month_string[2] = (char) 82;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (temp_month==4)
{
    month_string[0] = (char) 65;
    month_string[1] = (char) 80;
    month_string[2] = (char) 82;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (temp_month==5)
{
    month_string[0] = (char) 77;
    month_string[1] = (char) 65;
    month_string[2] = (char) 89;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (temp_month==6)
{
    month_string[0] = (char) 74;
    month_string[1] = (char) 85;
    month_string[2] = (char) 78;
    month_string[3] = ' ';
    month_string[4] = 0;
}

if (temp_month==7)
{
    month_string[0] = (char) 74;
    month_string[1] = (char) 85;
    month_string[2] = (char) 76;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (temp_month==8)
{
    month_string[0] = (char) 65;
    month_string[1] = (char) 85;
    month_string[2] = (char) 71;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (temp_month==9)
{
    month_string[0] = (char) 83;
    month_string[1] = (char) 69;
    month_string[2] = (char) 80;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (temp_month==10)
{
    month_string[0] = (char) 79;
    month_string[1] = (char) 67;
    month_string[2] = (char) 84;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (temp_month==11)
{
    month_string[0] = (char) 78;
    month_string[1] = (char) 79;
    month_string[2] = (char) 86;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (temp_month==12)
{
    month_string[0] = (char) 68;
    month_string[1] = (char) 69;
    month_string[2] = (char) 67;
    month_string[3] = ' ';
    month_string[4] = 0;
}

```

Figure 29 - Set Date Function Months

```

    date_string[0] = day_code_string[0];    //Plac
    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] = ' ';
    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;

    DisplayString(line, date_string);
}

```

Figure 30 - Set Date Function Strings

The display alarm time and display set alarm time were the next 2 functions used in the program. These were carbon copies of the display date and set date functions explained previously and would work in the same way. The only difference being that this would be used for when the alarm settings were being displayed on the LCD and the when her alarm was being set.

```

////////////////////////////////////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};

    alarm_string[0] = (char) alarm_hours / 10 + 48;
    alarm_string[1] = (char) alarm_hours % 10 + 48;
    alarm_string[2] = ':';
    alarm_string[3] = (char) alarm_mins / 10 + 48;
    alarm_string[4] = (char) alarm_mins % 10 + 48;
    alarm_string[5] = ' ';
    alarm_string[6] = ' ';
    alarm_string[7] = ' ';
    alarm_string[8] = ' ';
    alarm_string[9] = ' ';
    alarm_string[10] = (char) 65;
    alarm_string[11] = (char) 76;
    alarm_string[12] = (char) 65;
    alarm_string[13] = (char) 82;
    alarm_string[14] = (char) 77;
    alarm_string[15] = 0;

    DisplayString(line, alarm_string);
}

```

Figure 31 - Display Alarm Function

```

////////////////////////////////////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};

    alarm_string[0] = (char) temp_al_hours / 10 + 48;
    alarm_string[1] = (char) temp_al_hours % 10 + 48;
    alarm_string[2] = ':';
    alarm_string[3] = (char) temp_al_mins / 10 + 48;
    alarm_string[4] = (char) temp_al_mins % 10 + 48;
    alarm_string[5] = ' ';
    alarm_string[6] = ' ';
    alarm_string[7] = ' ';
    alarm_string[8] = ' ';
    alarm_string[9] = ' ';
    alarm_string[10] = (char) 65;
    alarm_string[11] = (char) 76;
    alarm_string[12] = (char) 65;
    alarm_string[13] = (char) 82;
    alarm_string[14] = (char) 77;
    alarm_string[15] = 0;

    DisplayString(line, alarm_string);
}

```

Figure 32 - Display Set Alarm Function

The display state function is important as it tells the LCD display what state the program is currently in. This is a useful tool for fault-finding. It uses an array and ASCII conversion similar to the previous arrays used in the program.

```
//////////////////////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 33 - Display State Function

The last function used in the program is the delay function. This uses a set amount of time cycles to delay the program. The value of t typically used is 100 as this is a solid value for the function to count to. Once the function reaches 100 it breaks out of the delay and the program cycle continues.

```
//////////////////////Delay Function//
void Delay(int t) // Start of function
{
    int i;
    for(i=0;i<t;i++);
}
```

Figure 34 - Delay Function

3. Testing and Analysis of Results

Once the program had been built it needed to be thoroughly tested. Each possible state and the process of getting from one state to the next needed to be tested. This was done by building a circuit in Proteus using the PIC 16F877. The built circuit can be seen below.

The LCD screen is shown on the right, with the PIC 16F877 Microcontroller in the middle. An led is taken out of port RA0 as the output to show the alarm activating.

Switch RA1 would be used to open up the alarm settings and used to confirm the inputted alarm time.

Switch RA2 is used to select hours, minutes, day, date, month, year for increment and decrement.

Switch RA3 is used to increment each value. When not in the alarm settings it is also used to activate the alarm. This is done by displaying an '!' mark on the clock.

RA4 switch is used to decrement each value.

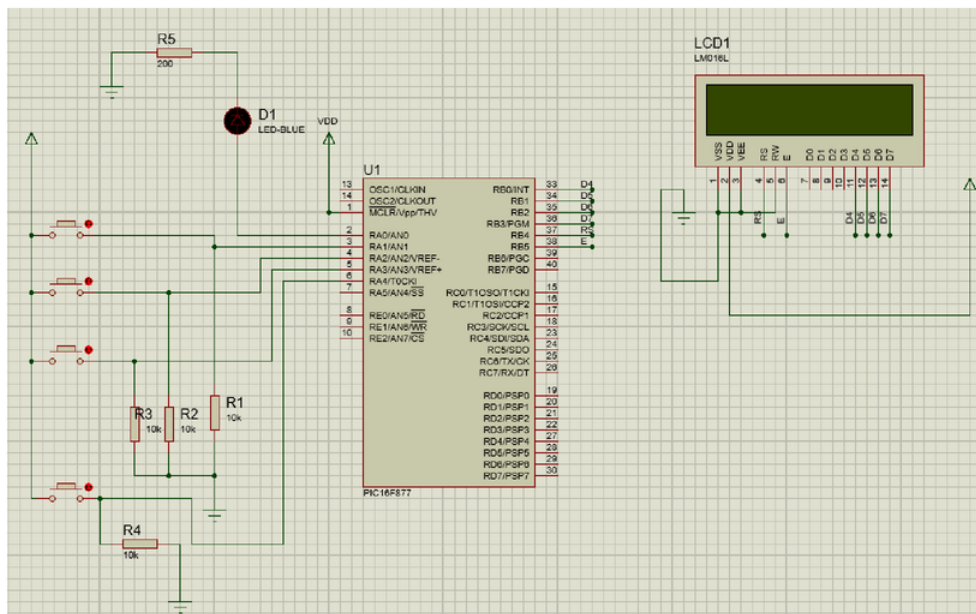


Figure 35 - MPLab Circuit

3.1 Testing the Clock

The first tests were done to make sure the clock would increment and decrement correctly, as well as display the correct values on the LCD display.

Process	State	Expected Result	Actual Result
Normal Clock Display	0	Clock would display normal time	15:21 SAT 08 DEC 2018
Increment Hours	1	Clock would increment hours	16:21 SAT 08 DEC 2018
Decrement Hours	1	Clock would decrement hours	14:21 SAT 08 DEC 2018
Increment Minutes	2	Clock would increment minutes	14:22 SAT 08 DEC 2018
Decrement Minutes	2	Clock would decrement minutes	14:20 SAT 08 DEC 2018
Increment Days	3	Clock would increment days	14:20 SUN 08 DEC 2018
Decrement Days	3	Clock would decrement days	14:20 FRI 08 DEC 2018
Increment Date	4	Clock would increment date	14:20 FRI 09 DEC 2018
Decrement Date	4	Clock would decrement date	14:20 FRI 07 DEC 2018
Increment Month	5	Clock would increment month	14:20 FRI 07 JAN 2019
Decrement Month	5	Clock would decrement month	14:20 FRI 07 NOV 2018
Increment Year	6	Clock would increment year	14:20 FRI 07 NOV 2019

Decrement Year	6	Clock would decrement year	14:20 FRI 07 NOV 2017
Save Settings	7	Clock would save settings	14:33 FRI 07 NOV 2017
Display Settings	8	Clock would display settings	14:20 FRI 07 NOV 2017

3.2 Testing the alarm

The last tests were to make sure the alarm functioned correctly. The figure below shows the clock running in its normal state. The RA1 button will be pressed now to activate the alarm settings. The alarm will then be set to 2:15 on Wednesday 2nd December 2016. When the alarm activates RA0 will switch red and the blue LED will light up.

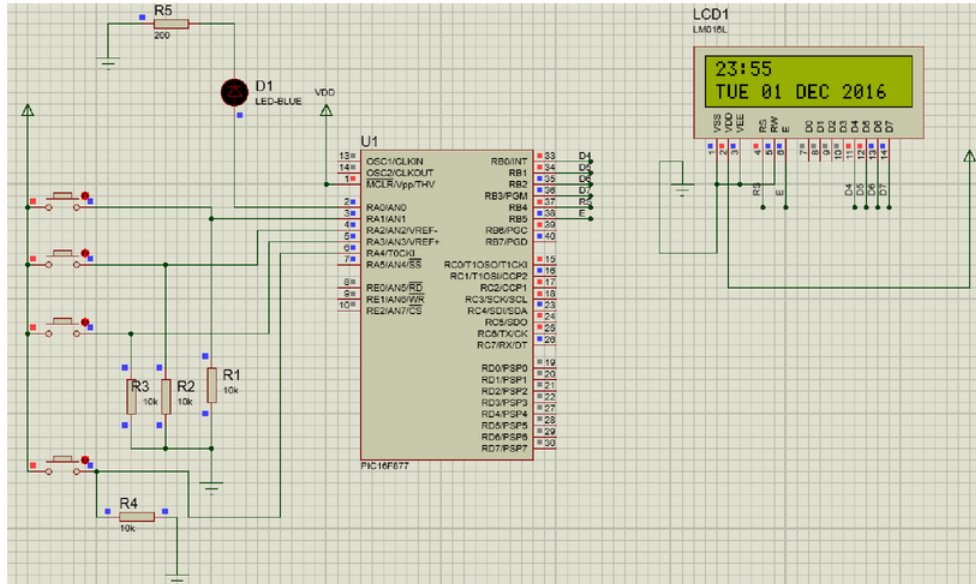


Figure 36 - Alarm test 1

The following figure shows the circuit once RA1 has been pressed.

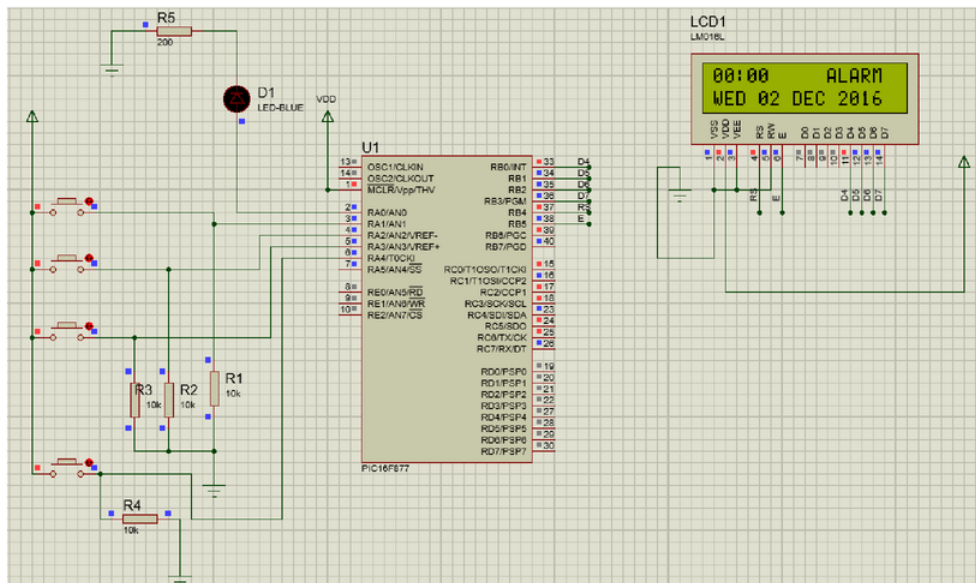


Figure 37 - Alarm Test 2

RA2 was pressed to select hours, and RA3 was used to increment the hours to 2.

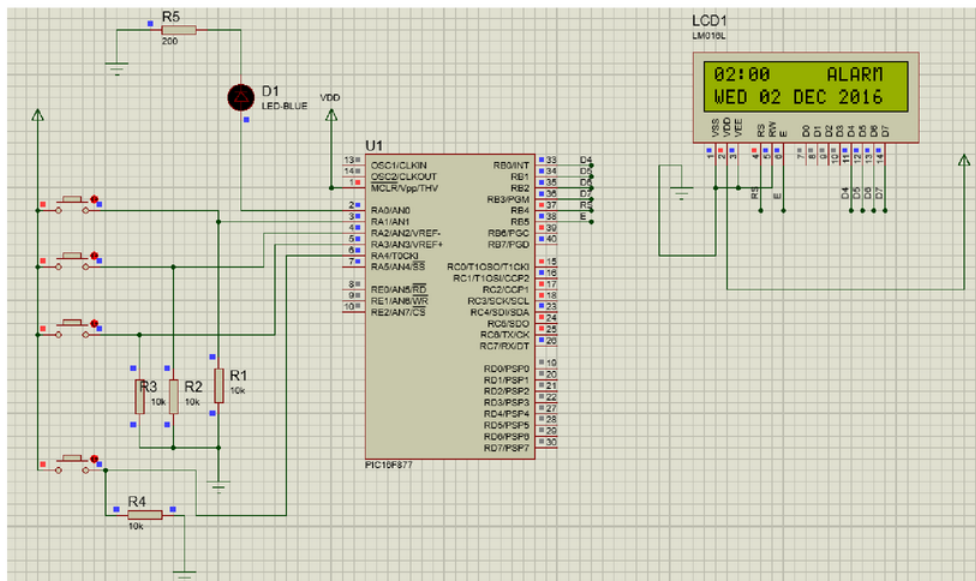


Figure 38 - Alarm Test 3

RA2 was then pressed again to select minutes and RA3 was used to increment the minutes to 15. The rest of the date had already been set.

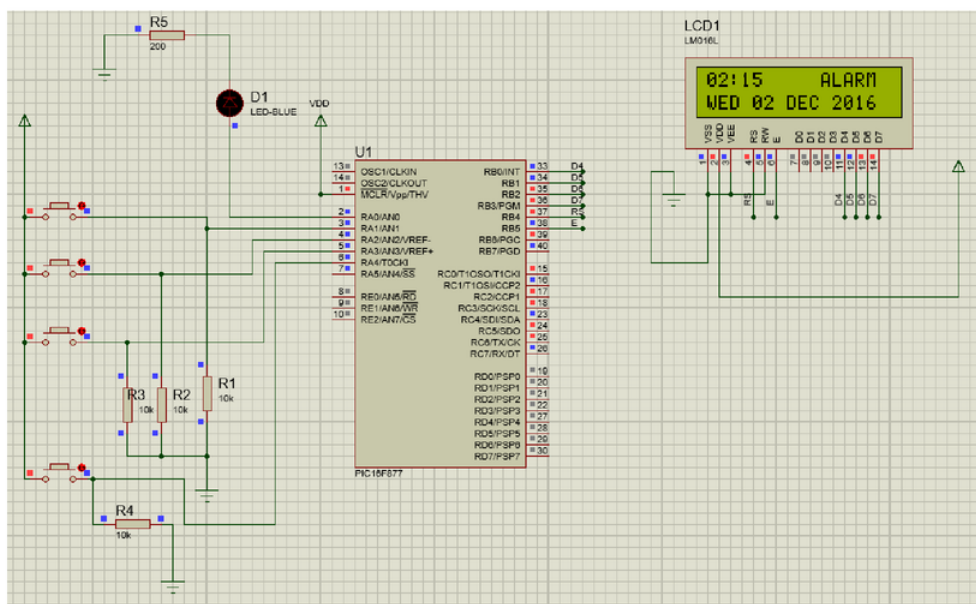


Figure 39 - Alarm Test 4

RA1 was then pressed to save the alarm settings. RA3 was pressed to activate the alarm so that it would go off when set to. This can be seen from the "!" mark on the clock face.

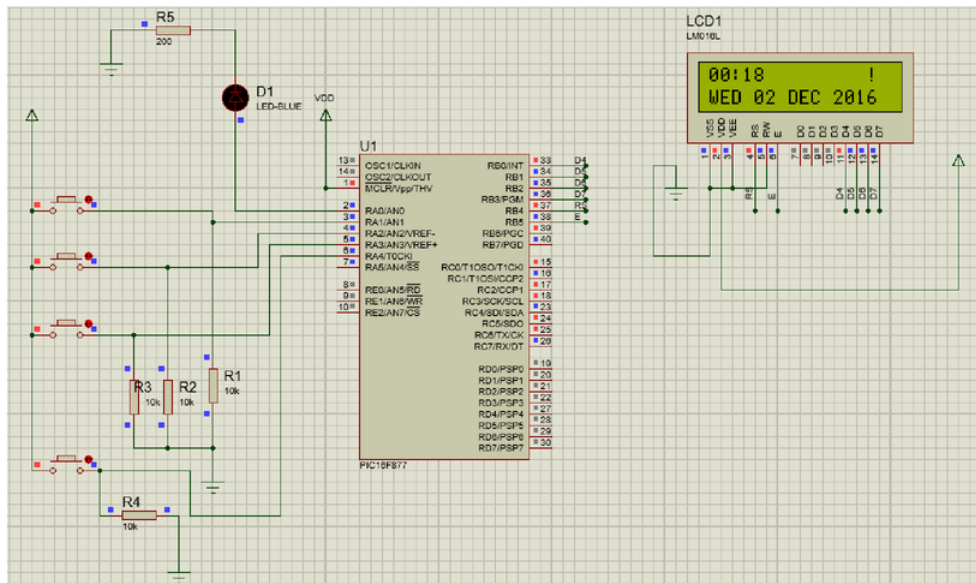


Figure 40 - Alarm Test 5

Once 2:15 was reached on the clock the alarm activated as expected. This is shown by the blue LED lighting up.

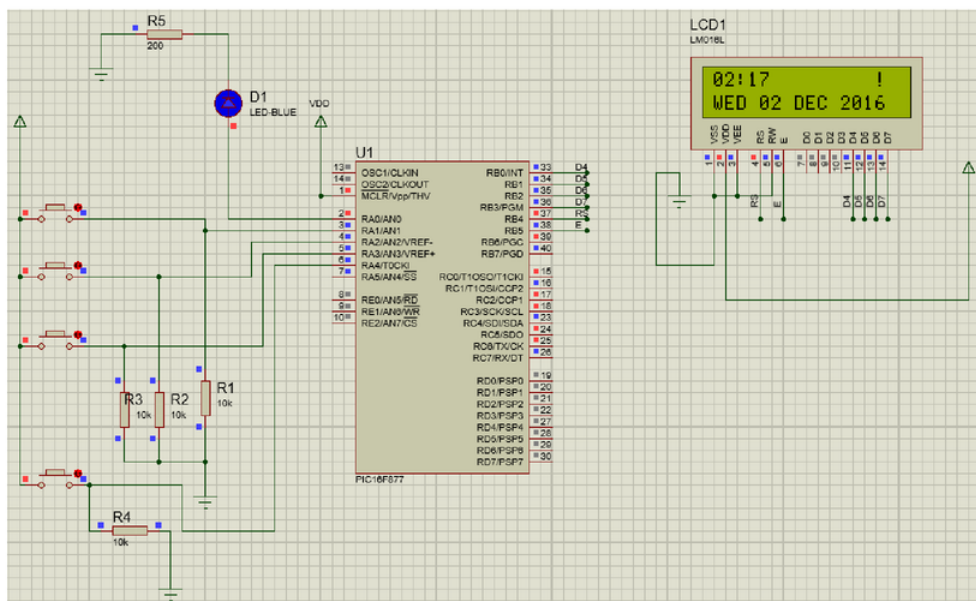
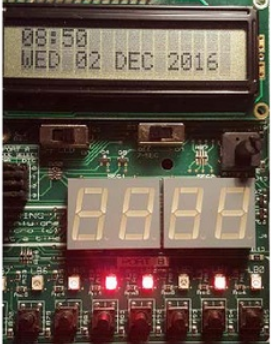
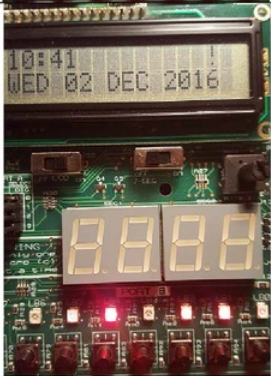
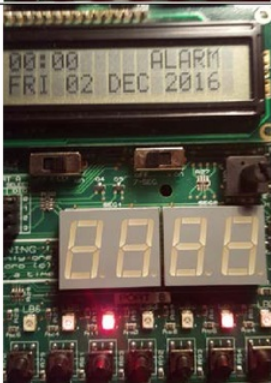
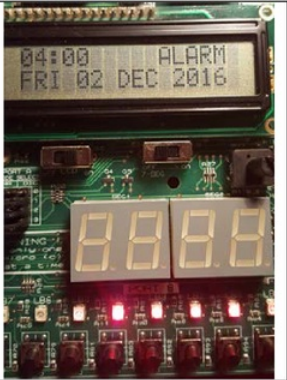


Figure 41 - Alarm test 6

3.3 Testing on the HP488 Development Board

Now that the program had been fully simulated using Proteus and MPLab, it needed to be downloaded to the HP488 development board and physically tested. The testing plan can be seen below.

Process	State	Expected Outcome	Result
Normal Display	0	Normal clock would display on LCD screen	
Alarm Set	0 (with alarm set)	Exclamation mark would appear to show the alarm has been set to on	
Alarm Settings	8	Alarm settings would be displayed	

Alarm Time Set	0 (with alarm activating)	Alarm settings would be saved	
----------------	---------------------------	-------------------------------	--

4. Conclusions

The overall purpose of this assignment was to investigate and carry out a practical project based on the PIC 16F877 microcontroller. Research was carried out on this microcontroller and was used to derive calculations that would then be used in designing an alarm clock. The alarm clock had a set list of specifications that needed to be met using. All of this centred on the design of the PIC 16F877 microcontroller.

The research delved into what a PIC 16F877 microcontroller is, how it works, its features, and various methods of how it can be applied in practise. Research was also done on various registers that are used when programming microcontrollers such as the PIC 16F877. These registers were the INTCON and OPTION_REG registers.

The alarm clock system upon completion had all of the following features:

- Displays the correct time and date on a 24 hour clock cycle
- Has the option to change and set the time and date
- Includes an alarm feature
- Has the option to change, set and activate the alarm time
- Displays the full date on an LCD screen
- Provides an LED from the system to signify the activation of the alarm

There also included a well thought out and thorough testing plan that detailed every possible state the clock could enter, and whether or not it did in practise. The function testing of the clock, both in normal and alarm state was a success.

5. Appendices

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	ˆ
1	1	1	1	(START OF HEADING)	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	(START OF TEXT)	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	(END OF TEXT)	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	(END OF TRANSMISSION)	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	(ENQUIRY)	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	(ACKNOWLEDGE)	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	(BELL)	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	(BACKSPACE)	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	(HORIZONTAL TAB)	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	(LINE FEED)	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	(VERTICAL TAB)	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	(FORM FEED)	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	(CARRIAGE RETURN)	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	(SHIFT OUT)	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	(SHIFT IN)	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	(DATA LINK ESCAPE)	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	(DEVICE CONTROL 1)	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	(DEVICE CONTROL 2)	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	(DEVICE CONTROL 3)	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	(DEVICE CONTROL 4)	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	(NEGATIVE ACKNOWLEDGE)	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	(SYNCHRONOUS IDLE)	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	(END OF TRANS. BLOCK)	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	(CANCEL)	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	(END OF MEDIUM)	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	(SUBSTITUTE)	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	(ESCAPE)	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	(FILE SEPARATOR)	76	4C	1001100	114	L	124	7C	1111100	174	
29	1D	11101	35	(GROUP SEPARATOR)	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	(RECORD SEPARATOR)	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	(UNIT SEPARATOR)	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	(SPACE)	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	\$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(	88	58	1011000	130	X					
41	29	101001	51	)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

(ZZT32, 2007)

Bibliography

Microchip, n.d. *PIC16F877*. [Online]

Available at: <http://www.microchip.com/wwwproducts/en/en010241>

Microsoft, 2006. *Interrupt Service Routines*. [Online]

Available at: <https://msdn.microsoft.com/en-us/library/ms892408.aspx>

TechTerms, 2016. *ISR*. [Online]

Available at: <https://techterms.com/definition/isr>

The Bare Necessities, n.d. *The INTCON Register*. [Online]

Available at: https://pictutorials.com/The_Intcon_Register.htm

ZZT32, 2007. *ASCII TABLE*. [Online]

Available at: <https://commons.wikimedia.org/wiki/File:ASCII-Table.svg>

[Accessed 31 1 2018].

Scott Dedoncker Ass 1 FINAL

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

55/100

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

Text Comment. The calculation is correct for 10MHZ clock.

PAGE 9

PAGE 10

Text Comment. State diagram is clear and the system can be designed from this.

PAGE 11

PAGE 12

Text Comment. This would be 100ms. the other divide by 10 will be included in another calculation.

PAGE 13

Text Comment. This should be in the testing section as it is the prototype test for the 1 second timer.

PAGE 14

Text Comment. Needs more detail.

Text Comment. The config settings are set here. This should be pointed out.

PAGE 15

Text Comment. Good explanation of this logic function.

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

Text Comment. The state functions are explained reasonably well.

PAGE 22

Text Comment. This explanation could be clearer.

Text Comment. The function here is the LCD function with the created array as the argument with line number. This is the function call that displays the data on the LCD.

PAGE 23

PAGE 24

PAGE 25

PAGE 26

Text Comment. More is needed here.

Text Comment. This function should be explained.

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

GRADING FORM: CSG-RUBIC

SCOTT DEDONCKER

0

RESEARCH-INFORMED LITERATURE



Limited research. This should investigate other MCU's and discuss the limitations with reference to this task.

10/25

KNOWLEDGE AND UNDERSTANDING



Intro is weak. 1/5 System design is good in parts but the config needs more in depth explanations. 6/10 Software explanation could have been more detailed and better structured. 6/10

13/25

ANALYSIS



Testing is good. 8/10 Conclusion is limited. 5/10

13/20

PRACTICAL APPLICATION AND DEP.



Some parts of the code needed more in depth explanation. 6/10 Testing of the practical is good. 7/10

13/20

SKILLS FOR PROFESSIONAL PRAC.



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

6/10