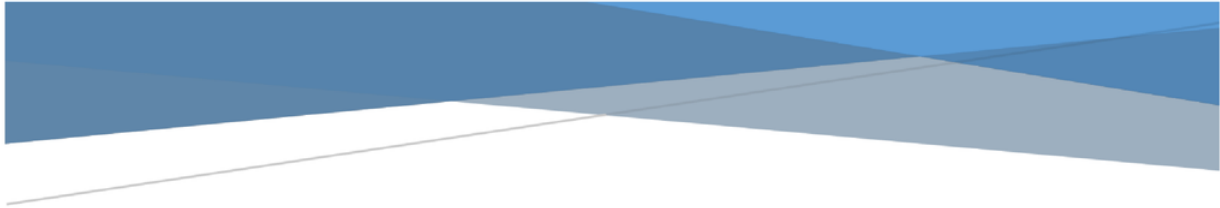


Final Report

by Nathan Smith

FILE	15460_NATHAN_SMITH_FINAL_REPORT_139109_2006558214.DOCX (8.2M)		
TIME SUBMITTED	01-FEB-2018 12:27 PM (UTC+0000)	WORD COUNT	5402
SUBMISSION ID	82660414	CHARACTER COUNT	29706



HND ELECTRICAL & ELECTRONIC ENGINEERING

PSDE 5013 CONFIGURATION & PROGRAMMING OF EMBEDDED SYSTEMS

ASSIGNMENT 1 CALANDER ALARM CLOCK

Nathan Smith Word Count = 5613

Contents

1. Introduction	3
2. Research	4
Timer Set Up and Registers	4
PIC 16F877 Microcontroller	8
3. System Design & Specification	10
Health and Safety and Ethics	12
4. Software Design	13
Code explanations	13
Function Definitions	25
5. Testing and Analysis	31
6. Conclusions and recommendations	40
7. References	41
Works Cited	41
8. Appendix	42

Figure 1: Block Diagram TIMER0.....	4
Figure 2: Option Register	5
Figure 3: INTCON Register	6
Figure 4: Pre scaler settings.....	6
Figure 5: General data sheet PIC 16F877	8
Figure 6: PIC 16F877	8
Figure 7: State Diagram For system.....	11
Figure 8: Head or Main file	13
Figure 9: Global and Interrupt	14
Figure 10: Main loop.....	15
Figure 11: Super Loop.....	16
Figure 12: Alarm set	17
Figure 13: Switch state.....	17
Figure 14: Case 1	18
Figure 15: Case 2	19
Figure 16: Case 3	20
Figure 17: Case 4 part 1.....	20
Figure 18: Case 4 part 2.....	21
Figure 19: Case 5.....	22
Figure 20: Case 6.....	22
Figure 21: Case 7 & 8	23
Figure 22: Case 9.....	24
Figure 23: Case 10.....	24
Figure 24: Display time function	25
Figure 25: Display Set Time	26
Figure 26: Display date 1	27
Figure 27: Display date 2	27
Figure 28: Display date 3	28
Figure 29: Display Alarm time	29
Figure 30: Display set alarm time	29
Figure 31: End of Code	30
Figure 32: Proteus Clock design	31
Figure 33: Code for 10ms.....	36
Figure 34: 5ms per division	36

Figure 35: Code for 1 second	37
Figure 36: 20ms division	37
Figure 37: Clock in Normal Mode	38
Figure 38: Alarm Active	38
Figure 39: Setting alarm	39
Figure 40: Alarm set to 4:00	39

1. Introduction

This assignment will look at designing an embedded system using a microcontroller and the use of the PIC 16F877 and HP488 development board. A suitable design will need to be designed built and tested.

The first part of the assignment will look at research into the features of the microcontroller and will include research on the timer set up and registers for the microcontroller and an investigation on the PIC 16F877 chip specifications and ports. The research will look at a written explanation and the use of images and data sheets to explain the features.

The design will be written in MP lab with the use of proteus design studio for the simulation. The assignment is asking for an alarm clock to be designed with the user being able to set the time and date from the design studio, set and enable an alarm, use the system with three buttons and output the design to a piece of hardware on the HP488 board.

The assignment asks for the main objectives to be covered which is to carry out the research as described above, create a state diagram to enable the operator to use the equipment simply and efficiently, design and test the code as its being built in stages and produce a test table, and download the program to the test board and practically test the system.

The assignment will be concluded and any recommendations will be included at the end of the assignment.

2. Research

Timer Set Up and Registers

The system requires correct timing in order to function correctly and operate the alarm clock accurately, this is done by implementing an ISR, this is an interrupt service routine and means the systems stops to perform the ISR if the qualifier is active. The time base used for the project is created with the application of an oscillator clock, this clock will determine the speed that the system will run in a scan time.

The timing is inserted internally to the ISR so that the program can update and run the time as soon as it receives a base time qualifier. The interrupt which will be utilised for the PIC 16F877 which is used in the assignment is the 'TIMER0' interrupt and below in figure 1 will show a timer block diagram.

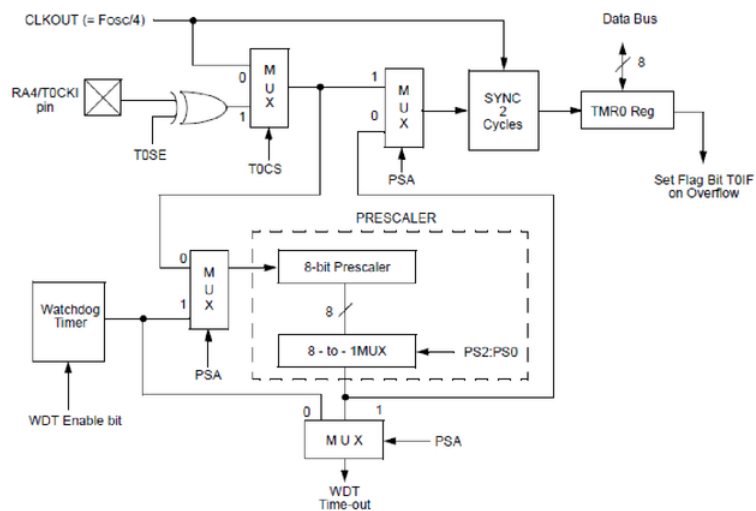


Figure 1: Block Diagram TIMER0

Two additional registers are to be added to the system to enable the TIMER0 to work which are the Option register and the Intcon register. These registers are explained below with the addition of diagrams explaining their operation.

The Options register is loaded with the binary inputs value of 0b10000111, figure 2 below shows the option register loaded with these values. The binary representation along the top of the data sheet shows the bit number with the value '1' and the middle text of the data sheet explains what each of the commands are doing, For example Bit 5 makes use of the clock TMR0 and transitions on RA4/T0CKI pin.

	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
	RBPUP	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7								bit 0
bit 7	RBPUP: 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: TMR0 Clock Source Select bit 1 = Transition on RA4/T0CKI pin 0 = Internal instruction cycle clock (CLKOUT)							
bit 4	T0SE: TMR0 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	TMR0 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 2: Option Register

The next register is the INTCON register, this register gives the capability to interrupt the system with TIMER0 so is critical to operate the system. The INTCON registers has the ability to be a writable and a readable register, and contains a variety of bits which can be enabled for the interrupt sources. The Name of the register 'INTCON' stands for **INT**errupt, **CON**trol. (Pictutorials, 2017)

The Register below in figure 3 has been set with the binary value of – 0b10100000, and as above with the options register the data sheet below shows what each command is allocated to what bit in the system.

	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
								bit 0
bit 7	GIE: Global Interrupt Enable bit 1 = Enables all unmasked interrupts 0 = Disables all interrupts							
bit 6	PEIE: Peripheral Interrupt Enable bit 1 = Enables all unmasked peripheral interrupts 0 = Disables all peripheral interrupts							
bit 5	TOIE: TMR0 Overflow Interrupt Enable bit 1 = Enables the TMR0 interrupt 0 = Disables the TMR0 interrupt							
bit 4	INTE: RB0/INT External Interrupt Enable bit 1 = Enables the RB0/INT external interrupt 0 = Disables the RB0/INT external interrupt							
bit 3	RBIE: RB Port Change Interrupt Enable bit 1 = Enables the RB port change interrupt 0 = Disables the RB port change interrupt							
bit 2	TOIF: TMR0 Overflow Interrupt Flag bit 1 = TMR0 register has overflowed (must be cleared in software) 0 = TMR0 register did not overflow							
bit 1	INTF: RB0/INT External Interrupt Flag bit 1 = The RB0/INT external interrupt occurred (must be cleared in software) 0 = The RB0/INT external interrupt did not occur							
bit 0	RBIF: RB Port Change Interrupt Flag bit 1 = At least one of the RB7:RB4 pins changed state; a mismatch condition will continue to set the bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared (must be cleared in software) 0 = None of the RB7:RB4 pins have changed state							

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: INTCON Register

With the two registers now operating and the time for TMR0 set, a one second time is now required which is done with the calculation below. There is an overflow in the system which is set at the value of 256, so the required value needs to be under 256. This is achieved using a pre scaler. Figure 4 below shows the setting for the pre scaler and the value that will be used for the calculation to achieve the 1 second is taken using the highest value on the settings, at 1:256 which makes all bits set to 111 as per the data sheet.

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 4: Pre scaler settings

The calculation is as follows;

$$\text{Cycle time (instruction)} = \frac{1966800}{4} = 4915200\text{HZ}$$

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

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

$$\text{Count } 10\text{ms} = \frac{10}{51.2} = 195 \quad (2 \text{ taken away for the error}) = 193$$

$$\text{The pre load count then} = 256 - 193 = 63$$

The pre load count is then pre-loaded in the timer TMR0 to achieve the delay for the interrupt which is 10ms.

The code in the system:

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

Each time the interrupt happens the value for the 'milli' will be increased by ten, 100 seconds will pass before the 'seconds' will then increase by the value of 1.

$$= 100 \times 10 = 1 \text{ second}$$

The config is not detailed enough. The research should include other MCU's with limitations and go further than the config explanation.

PIC 16F877 Microcontroller

The PIC 16F877 chip is a very widely used and popular microcontroller used in modern technologies. This Micro controller offers a wide range of advantages such as its competitive price, its strong and robust build quality to high standards, the many applications that it holds and the simplicity to use. The table below, figure 5, shows some of the general features found on the PIC16F877. (Circuits Today, 2015)

Single cycle instructions (Excluding program branches)
RISC CPU High performance
368 x 8 bit RAM Memory
Pin out to PIC 16C77, PIC 16C76, 16C74B.
Clock input 200MHz, Instruction cycle 200ns
Power on reset (POR)
Voltage range 2.0v – 5.56v
Interrupt capability 14 sources

Figure 5: General data sheet PIC 16F877

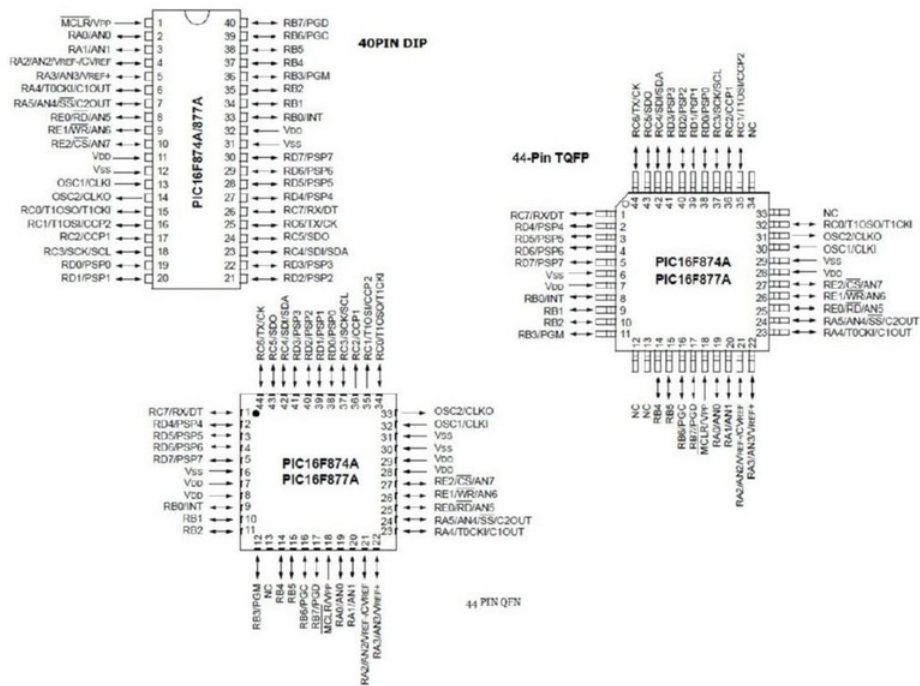


Figure 6: PIC 16F877

Above in figure 6 shows the pin diagram for the PIC16F877, this chip can be found in three different packages as shown above in the image. The chips key features are as follows, a frequency operating at 20MHz, the chip has three timers, four input ports (RA-RD), five output ports, two CCP modules, two serial comms ports, a PSP port, and a 10 bit analogue to digital module with 8 channels. The chip also has a memory of 368 bytes and a flash memory of 8KB or 14 bit words. (Circuits Today, 2015)

3. System Design & Specification

The system design for the assignment was use a PIC 16F877 as explained above in the research, along with the HP488 development board to design build and test a digital alarm clock with the use of a two line LCD and three pushbuttons RA1-RA4. The system will be spilt in to three main parts, setting the time and date, setting the alarm time and activating and de activating the alarm.

Setting the date and time;

To set the time and date this procedure must be designed and operated, press button 2 to enable the hours to be incremented with the use of button 3. Press button 2 again to toggle through to set the minutes, day, date and month with button 3 incrementing each of these variables. A press of button 1 will save the values set and return the system to normal operating mode.

Setting the Alarm

The use of button 1 when selected in normal operating mode will allow for the alarm display to be activated giving the user the option to set the alarm. The heading 'ALARM' will appear on the screen which mans the press of button 1 has successfully opened the alarm setting window. The use of button 2 can then change the alarm hours and minutes and button 3 can be used to increase the alarm hours and minutes. As above button 1 can then be pressed to save the alarm and return to the normal operating display.

Activating the alarm

In normal operating mode button 3 can be pressed to activate and deactivate the alarm, when this is done the symbol of an exclamation mark (!) is displayed and this shows the user the alarm is set or not set.

The system will work on a state system design, a state system has a list of every state the program can operate in or be set to. A state diagram has been produced below in figure 7 to show every state of the system built and what pressing each button will do or what state will be set. The state diagram is basically a flow chart giving a clear representation of what happens to the system when each button is pressed and how to navigate through the system. This state diagram is essential for a user as it acts as an instruction chart and without it the program may be misused or not operated to its full capacity.

State diagram for Clock

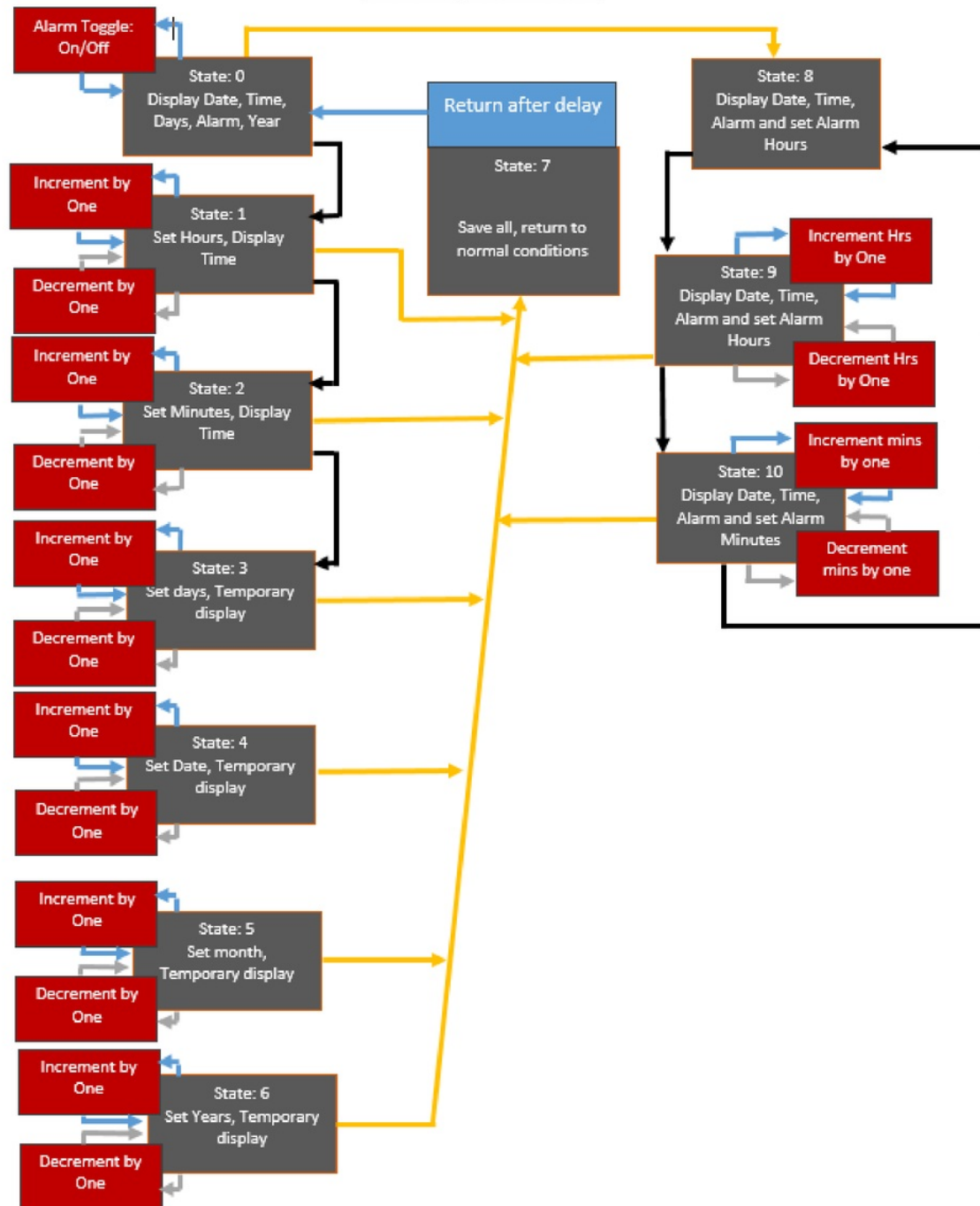


Figure 7: State Diagram For system

Health and Safety and Ethics

Health and safety is a massive part of the modern engineering world and plays a key role for In this case the lecturer and students. The college has its own health and safety policies which apply from a wide range of variables from to the equipment used in rooms, the condition of the buildings and the rules adhered to on campus.

The main health and safety points followed for this application were as follows,

1. Ensure the college was open, with lecturers present in the teaching rooms and persimmon was granted to enter the rooms.
2. Exits were free from obstructions and clearly indicated
3. Safety PPE was acquired and used when carrying out practical activities
4. Seats and tables were in good working order and the correct amount of personal space and postures were adopted
5. All electrical equipment was PAT tested with a suitable sticker present before using the equipment.

The code created was either created by the author of this assignment or taken from the lecturers notes in an open source such as Moodle or the public drive, the information used was taken from reliable credited sources and referenced correctly using Harvard style were applicable.

4. Software Design

The code was built over a number of sessions in the college lab and was built in a number of different programs which were modified and added to over time until they were added together to the main file to complete the system. This main file is what will be explained with a number of screenshots taken of the program to shown the individual parts of the program and explain what is happening in each part.

The programs crated were done in a sequence so that the program could be built on and added to achieve the next step, the first programme built was the code to display and adjust the time. This was then added to in another program to display and adjust the date, these two programs were then added together and tested to ensure there operation before the programme was taken further. With the test done and positive results the alarm was then programmed with the alarm set feature added in another programme which was tested individually then added to the main programme and tested to the proteus project and all the stages and states tested as per part 5 pf this assignment.

The explanation of the code is as follows from the top of the programme down to the bottom with key areas screen grabbed and explained.

Code explanations

Part 1 – Head or Main file setting

```
#include <pic.h>                                //Head or main file
#include "LCD.h"                                //Head or main file

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

Figure 8: Head or Main file

This part of the code is located at the very top of the program, its function is to defined the names of the functions prototypes for what they will be called through the program, the functions are also in order of what they will be programmed through the code. As explained above the programme was built in an order and added to so when they were all brought together in this main code file they were in order as built so that they operate correctly.

Part 2 – Global variables and Interrupt

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

int milli=0, secs=0, mins=10, hours=23, state=0, days=29, month=12, //global variables used for final values for time and date
day_code=4, year=2015, alarm_hours=0, alarm_mins=0, alarm=0,
temp_hours=0, temp_mins=0, temp_state=0, temp_day_code=0, temp_days=0,
temp_month=0, temp_year=0, temp_al_hours=0, temp_al_mins=0; //global variables for temp

void interrupt ISR (void)
{
    if(TOIF==1)
    {
        TOIF = 0; //Reset the interrupt flag, set to zero
        TMR0 = 255; //Reload preload value for time
        milli += 10; //Increase the milliseconds
        if(milli==10) //If milli equals 10
        {
            secs ++; //Increase the seconds by 1
            milli = 0; //Reset milli to zero
            if(secs==60) //If secs equals 60
            {
                mins ++; //Increase the mins by 1
                secs = 0; //Set secs to zero
                if(mins==60) //If mins equals 60
                {
                    hours ++; //Increase hours by 1
                    mins = 0; //Set mins to zero value
                    if(hours==24) //If hours equals 24
                    {
                        days ++; //Increase days by 1
                        day_code ++; //Increase day_code by 1
                        hours = 0;
                        if(days>31)
                        {
                            month ++; //Increase month by 1
                            days = 1; //Set days to one
                            if(month>12) //If month greater than 12
                            {
                                year ++; //Increase year by 1
                                month = 1; //Sets the month to 1
                            }
                        }
                    }
                }
            }
        }
        if(day_code==7) day_code=0; //If day_code equals 7 then day_code=zero
    }
}
```

Figure 9: Global and Interrupt

Figure 9 above shows the global variables are set by taking them out of the main loop and defining them outside, these variables are used throughout the code as variable values. The Interrupt ISR section comes next, this section will control the timing used in the system and project. The 'if' statement comes first and is used for the program looking for a value or variable to be selected or met then 'if' this is met the program will continue.

The first 'if' is for the TOIF, when or 'if' this interrupt flag is met the system will reset and move on to the next stage of preloading the value of 255 into TMR0. The next part is the timing, one cycle is 10ms so 100 cycles will be equal to the 1 second needed for the program. Once the value for 10 is reached the program will continue. The seconds then moves on to the minutes were once 60 seconds is reached it will move on to the minutes then after 60 minutes is reached it will move on the hours as above. The hours then looked for a cycle of 24 hours before its moved onto days and

Part 3 – Main loops

Figure 10: Main loop

The value calculated for the pre load is then entered into the TMR0 which as above is 255 for this program. The LCD_Startup function is then defined and this is taken from the library which is built by an external source and inputted into this system. The file is not part of the assignment so it can be taken from the library along with other files in this system.

Part 4 – Super loop

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

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

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

Figure 11: Super Loop

This super loop starts by looking at the days in the months, because there is a different number of days in some months this is addressed in the first two parts by looking at the months with 31 days, the if statements then activates each month with the first statement being for the months with 31 days, then second statement for the months with 28 days and after 4 years becomes a leap year. The third statement looks at the months reaching 12 months which will increase the year and the final statement looks at a full week cycle of 7 days.

Part 5 – Alarm

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

Figure 12: Alarm set

This section of code is looking to see if the alarm is activated, this will then check to see if the actual time is the same at the set alarm time to activate the alarm through RA0. If the alarm actual time is not the same as the alarm set time then it will be set at 0 at RA0, until the time is reached. Then second part says if the alarm has not been set then RA0 is always set to 0 as it has no output.

Part 6 – Switch state

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

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

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

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

    break; //This will break out from each case
}
```

Figure 13: Switch state

This part looks at the switch statement, this refers to the state diagram and is set out in the same fashion of following each state in order. The first case (0) is the standard display of the clock, this is show with the display time and date statements which tell the clock to show the time and date. The temp variables come next and these are for when the operator decides to change a value for time or the date but as it's a temp statement the normal time will continue. RA1 Ra2 are the buttons for selecting a different function as per the diagram and Ra1 can then be pressed again to save the values as above. Ra3 is for the alarm and will see if the alarm is off or on.

Part 7 – Case 1

```

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++;                     //Increase temp_hours by 1
        if(temp_hours>23) temp_hours=0;    //If temp_hours is greater than 23 set to zero
    }
    if(RA4)                                //If RA4 = 1
    {
        while(RA4);                       // This will cause the value to decrement by a value of 1
        Delay(100);
        temp_hours--;                     //decrement the temp_hours value by 1
        if(temp_hours<0) temp_hours=23;    //If temp_hours is greater than 23 set to zero
    }

    break;                                 //This will break out from each case

```

Figure 14: Case 1

Case 1 is for the function of display set time, Ra1 and Ra2 are used to select the variables to set the time as per the state diagram and they will set the system to another state. Ra3 is used to increase the hours and Ra4 is used to decrease the hours both by the value of 1.

Part 8 – Case 2

```
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++;                       //Increase 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);                        // This will cause the value to decrease by 1
        Delay(100);
        temp_mins--;                       //decrease temp_mins value by 1
        if(temp_mins<0) temp_mins=59;      //If temp_mins is greater than 59 set to zero
    }
    break;                                 //This breaks out from each case
```

Figure 15: Case 2

Case 2 is similar to case 1 but it now looks at the minutes instead of the hours in the same order you would set a clock or read the time. Ra1 and Ra2 again are used to select the different states and Ra3 will increase the minutes and Ra4 will decrease the minutes.

Part 9 – Case 3 and 4

```

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

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

    }

    if(RA4)                                //If RA4 is equal to 1
    {
        while(RA4);
        Delay(100);
        temp_day_code--;
    }                                     //end if RA4

    break;                                //This will break out from each case

```

Figure 16: Case 3

Case 3 is the same as above but setting the day codes now in the same fashion as case 1 and 2.

```

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)
    {
        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 */
        {
            temp_days = 1;                 //Set days to 1
        }                                 //end if

        if(((temp_month==2)||((temp_days>28)||((temp_year% 4 != 0)))
        /*If temp_month equals 2 and temp_days greater than 28 and
        4 does not divide into year - do this (not leap year)*/
        {
            temp_days = 1;                 //Set days to 1
        }                                 //end if

        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
        }                                 //end if

        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 */
        {
            temp_days = 1;                 //Set days to 1
        }                                 //end if
    }                                     //end if RA3

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

        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 = 31;                //Set days to 31
        }
        if(((temp_month==2)||((temp_days>1)||((temp_year% 4 != 0)))
        /*If temp_month equals 2 and temp_days greater than 29 and
        4 does not divide into year step in (not leap year)*/
        {
            temp_days = 28;                //Set days to 28
        }
        if((temp_month==2)||((temp_days>1)||((temp_year% 4 == 0)))
        /*If temp_month equals 2 and temp_days greater than 29 and
        4 divides into year step in (leap year)*/
        {

```

Figure 17: Case 4 part 1

```

temp_days--;

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 28
}
if((temp_month==2) && (temp_days<1) && (temp_year% 4 == 0))
/*If temp_month equals 2 and temp_days greater than 29 and
4 divides into year step in (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
}

}

```

Figure 18: Case 4 part 2

Case 4 is used to display the date of the system, this part also allows the date to be set and looks at using ra3 and 4 to increase and decrease the days. The program will then process through and check the number of days in the selected month in order to decide which month it is in by the known number of days in each month. This will also check then for a leap year.

Part 10 – Case 5 & 6

```
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
{
temp_month++; //Increment temp_month by 1
if(temp_month>12) temp_month=1; //If temp_month is greater than 12 set to 1
} //end if
if(RA4) //If RA4 is equal to 1
{
while(RA4);
Delay(100);
temp_month--;
if(temp_month<1) temp_month=12; //If temp_month is less than 0 set to 12
} //end if RA4

break; //This breaks out from each case
```

Figure 19: Case 5

This part of the code looks at case 5 which is setting the date, ra1 and 2 are the same as above were they follow the states and can select the date part of the clock. Ra3 and 4 are then used again the increase and decrease the month values through the 12 months.

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

if(RA1) state = 7; //If RA1 is equal to 1 set state to equal 7
if(RA2) state = 0; //If RA2 is equal to 1 set state to equal 0
if(RA3) //If RA3 is equal to 1
{
temp_year++; //Increment temp_year by 1
}
if(RA4) //If RA4 is equal to 1
{
while(RA4);
Delay(100);
temp_year--;
} //end if RA4
break; //This breaks out from each case
```

Figure 20: Case 6

This part is similar to the section above but it is looking to set the year, ra1 and 2 is as above were they are used to select the state and ra3 and 4 will increase and decrease the year. The year value should really only be increased but in some cases the date may need to be changed to a past year for testing purposes.

Part 11 case 7 & 8

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

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

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

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

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

    if(RA1) state = 0;                     //If RA1 is equal to 1 set state to equal 0
    if(RA2) state = 9;                     //If RA2 is equal to 1 set state to equal 9
    break;                                 //This breaks out from each case
```

Figure 21: Case 7 & 8

Case 7 is the section where the temp values which can be set and changed without changing the actual time value, are saved and set to be displayed on the screen of the clock as the new value. The alarm which is edited is also saved and is now an active alarm.

Case 8 is where the alarm value set by the users is then displayed to the screen and ra1 will return to standard display and ra2 will set the alarm time.

Part 11 – case 9 & 10

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

    break; //This breaks out from each case
```

Figure 22: Case 9

This part of the code looks at case 9 which is for setting the alarm values for the hours. The time will come up on the screen and the operator can then adjust the alarm times with ra1 and 2 to select hours and ra3 and 4 to increase and decrease the values.

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

Figure 23: Case 10

This part of the code looks at case 10 which is for setting the alarm values for the minutes. The time will come up on the screen and the operator can then adjust the alarm times with ra1 and 2 to select minutes and ra3 and 4 to increase and decrease the values.

The final part of the program is the line starting with the 'while' statement which ends the super loop.

Function Definitions

The function definitions are the next main part of the system, this piece of code in the system is the area where the commands called for in the main program are carried out. When a piece of the function definitions are defined into the main program as shown above, the code will carry out the command and then once completed return to where it was in the program.

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

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

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

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

.....//display set time function//.....

```

Figure 24: Display time function

The code for the display time starts off with a void key word, this key word instructs the program to not look for a return when the code is completed. This is followed by the key word int line which means the value for line has been taken from the main program into this function definition program or piece of code.

The next part is the declaration of the arrays, these arrays declare a time string in the value of (0) with 16 elements. The char at the front of the array means the (0) is represented by a value in an ASCII sheet. The zero in the sheet is equivalent to a

null character which gives a blank screen before anything is set. The value for the alarm section will decide what is displayed onto the LED screen.

The code inside the alarm section starts with the first 2 lines being for the clock hours with the first line displaying the first hour digit and the second line displaying the second hour digit. The ASCII value for 48 is 0 as shown in the comments. The third line will then display the value for nothing on the clock. The fourth and fifth lines are the same but for the first and second minute values. Lines 6 down create a blank space on the clock with the apostrophes and blank space displaying null on the clock display. The displaystring function along the end of this piece of code is used to display the values to the LCD.

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

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

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

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

Figure 25: Display Set Time

This part of code for the display set time is very similar to the code explained above for the display time, and is used to change the time values. Temp hours and mins are used to work out the ASCII values.

The next part is the display date section. This is a large section of code so the screen shots will be shown and an explanation after.

```

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

    if (day_code == 0) //If day_code is equal to zero
    {
        day_code_string[0] = (char) 83; //Ascii code 83 = S
        day_code_string[1] = (char) 85; //Ascii code 85 = U
        day_code_string[2] = (char) 78; //Ascii code 78 = F
        day_code_string[3] = ' '; //Creates space
        day_code_string[4] = 0; //Ends the string by placing 0 in the last string
    }
    if (day_code == 1) //If day_code is equal to 1
    {
        day_code_string[0] = (char) 77;
        day_code_string[1] = (char) 78;
        day_code_string[2] = (char) 78;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if (day_code == 2) //If day_code is equal to 2
    {
        day_code_string[0] = (char) 84;
        day_code_string[1] = (char) 85;
        day_code_string[2] = (char) 83;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if (day_code == 3) //If day_code is equal to 3 step in
    {
        day_code_string[0] = (char) 87;
        day_code_string[1] = (char) 83;
        day_code_string[2] = (char) 83;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if (day_code == 4) //If day_code is equal to 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 (day_code == 5) //If day_code is equal to 5 step in
    {
        day_code_string[0] = (char) 70;
        day_code_string[1] = (char) 82;
        day_code_string[2] = (char) 78;
        day_code_string[3] = ' ';
        day_code_string[4] = 0;
    }
    if (day_code == 6) //If day_code is equal to 6
    {
        day_code_string[0] = (char) 83;
    }
}

```

Figure 26: Display date 1

```

}
if (day_code == 6) //If day_code is equal to 6
{
    day_code_string[0] = (char) 83;
    day_code_string[1] = (char) 85;
    day_code_string[2] = (char) 84;
    day_code_string[3] = ' ';
    day_code_string[4] = 0;
}
if (month == 1) //If month is equal to 1
{
    month_string[0] = (char) 74; //Ascii code 74 = J
    month_string[1] = (char) 69; //Ascii code 69 = A
    month_string[2] = (char) 78; //Ascii code 78 = F
    month_string[3] = ' '; //Creates space
    month_string[4] = 0; //Ends the string by placing 0 in the last string
}
if (month == 2) //If month is equal to 2
{
    month_string[0] = (char) 70;
    month_string[1] = (char) 69;
    month_string[2] = (char) 66;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month == 3) //If month is equal to 3
{
    month_string[0] = (char) 77;
    month_string[1] = (char) 65;
    month_string[2] = (char) 82;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month == 4) //If month is equal to 4
{
    month_string[0] = (char) 65;
    month_string[1] = (char) 80;
    month_string[2] = (char) 82;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month == 5) //If month is equal to 5
{
    month_string[0] = (char) 77;
    month_string[1] = (char) 65;
    month_string[2] = (char) 89;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month == 6) //If month is equal to 6
{
    month_string[0] = (char) 74;
    month_string[1] = (char) 85;
    month_string[2] = (char) 78;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if (month == 7) //If month is equal to 7
{
    month_string[0] = (char) 74;
}
}

```

Figure 27: Display date 2

```

}
if(month==8) //If month is equal to 8
{
    month_string[0] = (char) 65;
    month_string[1] = (char) 86;
    month_string[2] = (char) 71;
    month_string[3] = ' ';
    month_string[4] = 0;
}
if(month==9) //If month is equal to 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) //If month is equal to 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) //If month is equal to 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) //If month is equal to 12
{
    month_string[0] = (char) 68;
    month_string[1] = (char) 69;
    month_string[2] = (char) 67;
    month_string[3] = ' ';
    month_string[4] = 0;
}
/*Places day_code, days, month, and year into the date_string*/
date_string[0] = day_code_string[0];
date_string[1] = day_code_string[1];
date_string[2] = day_code_string[2];
date_string[3] = day_code_string[3];
date_string[4] = days / 10 + 48;
date_string[5] = days % 10 + 48;
date_string[6] = ' '; //Creates space
date_string[7] = month_string[0];
date_string[8] = month_string[1];
date_string[9] = month_string[2];
date_string[10] = month_string[3];
date_string[11] = year / 1000 + 48;
date_string[12] = year % 1000 / 100 + 48;
date_string[13] = year % 100 / 10 + 48;
date_string[14] = year % 10 + 48;
date_string[15] = 0; //Ends the string by placing 0 in the last string
DisplayString(line, date_string); //Ends the string to the LCD on line i
}

```

Figure 28: Display date 3

This part of the code is for the display date section, figure 26 starts the part of code with the three arrays being declared. The local strings have 5 elements assignment and are similar to explained previously in the assignment. The day code is then next, this part will output three values which will indicate the day, a variable for the day code will increase each day and display the actual day with the if statement for all seven days as shown in figures 26 and 27.

The month if statements are next and have one for each month to give a total of 12 statements. Like above the system will output three characters in each if statement to indicate what day of the month it is and uses the ASCII table to work out the letter for the number calculation.

The last section as shown in figure 28 puts all the information gathered for the days and months and outputs them to the LCD display. The first three lines are for the day display in the ASCII value and are followed by a space. Then date is then displayed followed by the month represented in ASCII values. There is then another space followed by the year displayed. This is ended with the displaystring keyword which calls from the lcd.h file and outputs to the LCD screen. The display set date piece of code which follows this code is very similar but it's used for editing the outputs with temp functions.

The next part is the display alarm time section as shown below in figure 29.

```

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

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

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

```

Figure 29: Display Alarm time

This section looks at the display alarm time, this section is used to display to the operator what the alarm is set too, and it uses the same arrays and set out as the coding above.

```

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

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

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

```

Figure 30: Display set alarm time

Figure 30 shows the display alarm time piece of code which is used them as above but to change the values of the alarm time.

```

/

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

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

    DisplayString(line, state_string);
}

```

Figure 31: End of Code

Figure 31 above is the end of the code for the system, the delay key word is there to delay the program before it is broken out and returned to the main program. The last part of the program which has the display state is used as a fault finding part and refers back to the state diagram above in the assignment.

5. Testing and Analysis

Following on from the construction of the system and program comes the testing stage. The test table below shows each individual state and test that can be carried out. Testing is very important as the program may build successful but could be containing bugs in the system and not operate correctly. Below in figure 32 shows the clock system built in the proteus design suite which the code is written to in order to test the system.

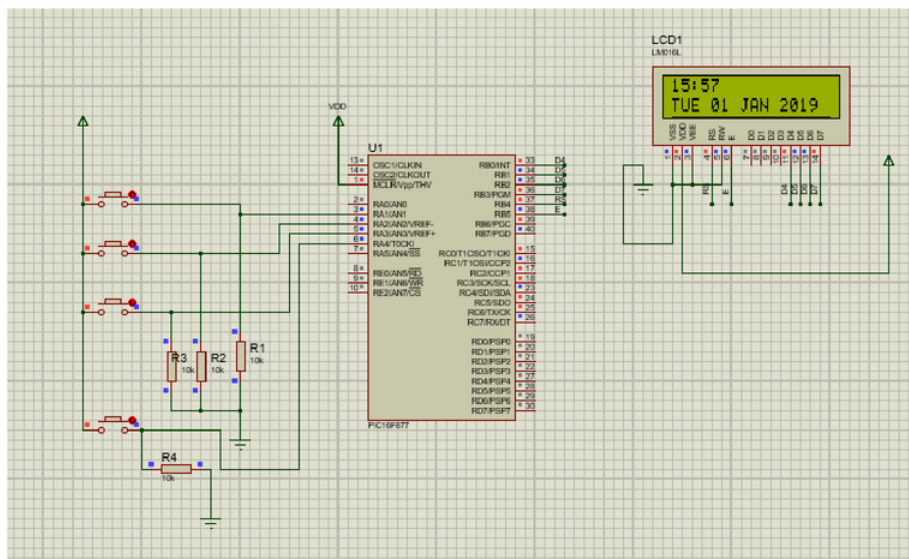
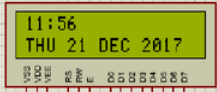
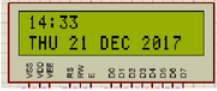
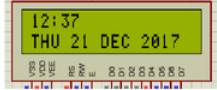
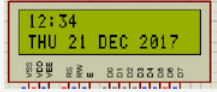
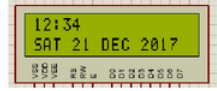
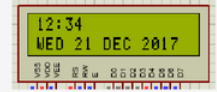
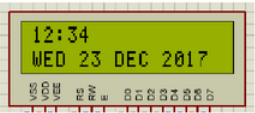
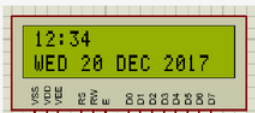
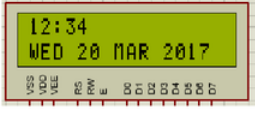
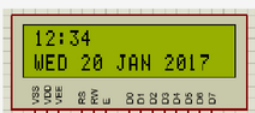
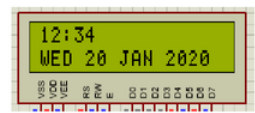
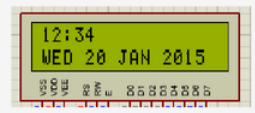
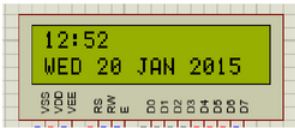
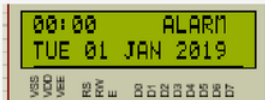
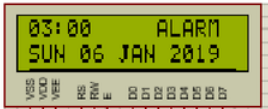
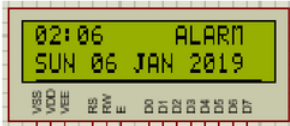


Figure 32: Proteus Clock design

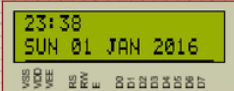
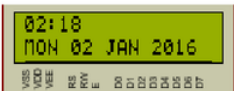
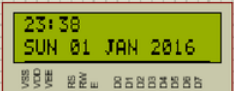
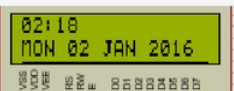
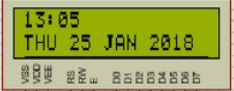
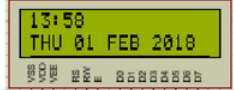
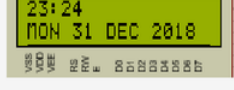
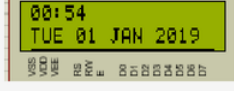
The test table is inserted below and contains a test for each and every state in order to efficiently test the program for its function.

State of Machine	Procedure	Results expected	Proved results	Pass/Fail
0	Normal Display (Date and Time)	Normal Display		Pass
1	Hours Increase by 1	1 hour increments with RA3		Pass
1	Hours decrease by 1	1 hour decrements with RA4		Pass
2	Minutes Increase by 1	1 Minute Increments with RA3		Pass
2	Minutes Decrease by 1	1 Minute Decrements with RA4		Pass
3	Day Increase by 1	1 Day Increment with RA3		Pass
3	Day Decrease by 1	1 Day Decrement with RA4		Pass

4	Date Increase by 1	1 Date Increment with RA3		Pass
4	Date Decrease by 1	1 Date Decrement with RA4		Pass
5	Month Increase by 1	1 Month Increment with RA3		Pass
5	Month Decrease by 1	1 Month Decrement with RA4		Pass
6	Year Increase by 1	1 Year Increment with RA3		Pass
6	Year Decrease by 1	1 Year Decrement with RA4		Pass

7	Values set saved by pressing RA1	Returns to Normal Screen with New Set Values Entered. (Dates are not accurate to Real Time)		Pass
8	Display alarm time and date	Press Ra1 and alarm time is displayed		Pass
9	Increase alarm hours	Press Ra3 to increase alarm hours		Pass
10	Decrease alarm hours	Press Ra3 to decrease alarm hours by doing full hours cycle		Pass
11	Increase alarm Minutes	Press Ra3 is when minutes selected to increase		Pass
12	Decrease alarm minutes	Press Ra43 when minutes selected to full cycle to		Pass

		decrease minutes		
--	--	---------------------	--	--

Procedure	Result Expected	Proved Results	Test Step	Pass/Fail
Correct displayed time	Time on Clock should match real time	N/A	1	Pass
Day increases when time passes full 24hrs	Day changes from Sunday to Monday when clock cycles 24 hrs	 	2	Pass
Date Increases when time passes full 24 hrs	Date changes from 1 Jan to 2 Jan after 24 hrs	 	3	Pass
Month Increases with the date increasing	Month changes from Jan to Feb with increasing dates	 	4	Pass
Year Increases with increasing dates of the full 12 months	Year to change from Dec 2018 to Jan 2019	 	5	Pass

Shows when alarm is on	The display should indicate a (!) when alarm is activated		6	Pass
------------------------	---	---	---	------

The testing stage proved successful and the program operated as expected with no faults, the final conclusions and recommendations will evaluate this in more detail.

```

#include <pic.h>
int milli=0, sec=0; //Must be declared global for function prototype variables

void interrupt ISR (void)
{
    if(TOIF==1)
    {
        TOIF = 0; //Reset interrupt flag, set to zero
        TMR0 = 60; //Reload preload time value. TMR0 IS SET FOR 100ms this is the largest time we can have from the TMR0. Clk = 10MHz
        //milli += 1; //Increment milliseconds by 10 to increase the value of the timer in steps of 10
        //if(milli==100) //If milli equals 100 increment sec by 1, Clear milli = 0 and toggle RBO
        {
            // sec += 1; //Increment seconds by 1 goes here
            // milli = 0; //Reset milli to zero
            RBO = ~RBO; // Toggle RBO pin
        }
    }
}

void main(void) //Main Function
{
    TRISB = 0x00; //Configure port B = 0000 0000
    OPTION_REG = 0x07;
    INTCON = 0x00;

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

```

Figure 33: Code for 10ms

The above code in figure 33 is set up for the value of 10ms limited by the PIC 16F877 internals. The scope value is shown below and needed to be changed to become 1 second as per the calculation in the research. Below in figure 34 is 5ms per division. So $2 \times 5 = 10\text{ms}$.

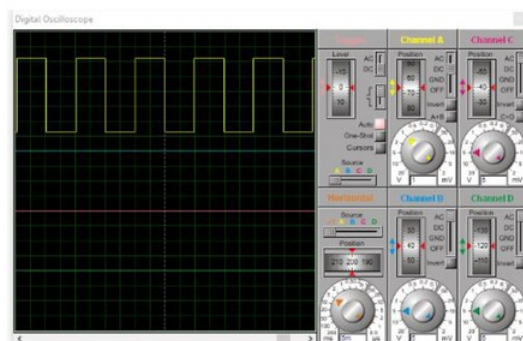


Figure 34: 5ms per division

The Below code in figure 35 has been changed and set up for the value of 1 second. This is done because the system limited by the PIC 16F877 internals. The scope value is shown below in figure 36 and has been changed to become 1 second. Below in figure 36 is 20ms per division. So $20 \times 5 = 1$ second.

```
#include <pic.h>

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

void interrupt ISR (void)
{
    if(TOIF==1)
    {
        TOIF = 0; //Reset interrupt flag, set to zero
        TMR0 = 65; //Reload preload time value. TMR0 IS SET FOR 100ms this is the largest time we can have from the TMR0. Clk = 10MHz
        milli += 1; //Increment milliseconds by 10 to increase the value of the timer in steps of 10
        if(milli==100) //If milli equals 100 increment sec by 1, clear milli = 0 and toggle RB0
        {
            secs++; //Increment seconds by 1 goes here
            milli = 0; //Reset milli to zero
            RB0 = ~RB0; // Toggle RB0 pin
        }
    }
}

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

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

Figure 35: Code for 1 second

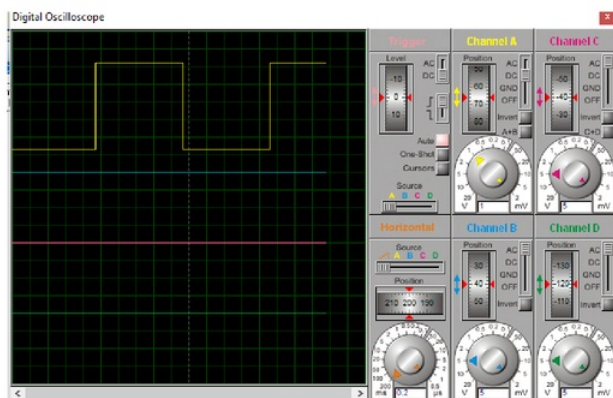


Figure 36: 20ms division

The HP488 board was connected to the computer and the system was downloaded onto the board and tested as below. The clock on the LCD screen is the same as the one designed in proteus to give a clear understanding of testing.

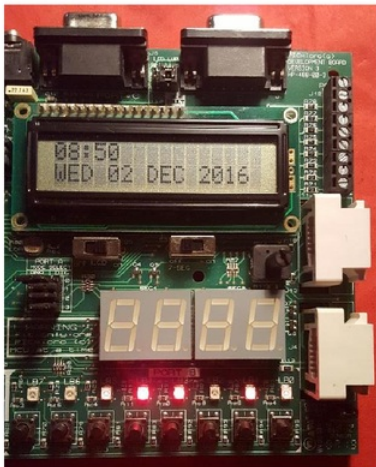


Figure 37: Clock in Normal Mode

Figure 37 above shows the board powered up and operating in its normal state, the clock is showing a date which is not true to the time it was tested but it shows the clock is operating.

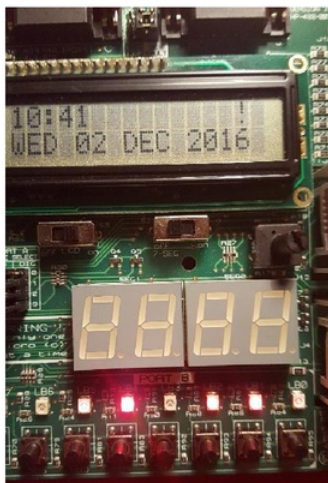


Figure 38: Alarm Active

Figure 38 above shows the clock with the alarm active, as stated above in the assignment the exclamation mark shows the alarm is active and set.



Figure 39: Setting alarm

Figure 39 above shows the clock in the alarm setting where the alarm can be set by pressing Ra1 and then Ra3 & 4 to increase and decrease the values for hours and minutes.



Figure 40: Alarm set to 4:00

Figure 40 above shows the alarm being set to the time of 4:00am and is incremented and decremented with Ra3 and Ra4, this proves the project works when testing down onto the HP488 board.

6. Conclusions and recommendations

The assignment aimed to make use of the PIC 16F877 microcontroller and the HP488 development board along with an embedded system C program to design an alarm clock as per the assignment brief. The embedded system was designed and the system was tested as per the assignment brief.

The research was designed to investigate the timer set up along with the PIC 16877 specifications and ports, the research helped to gain valuable information and knowledge on the microcontroller which was then used to help to design and explain how the system worked. The timer set up also gained good knowledge and helped towards designing the system. The research covered all areas asked and was explained with the use of block text and images and data sheets, this gives the reader a broader understanding and shows the use of theory and application.

The system was designed to meet the required specifications, the user was able to set the time and date without effecting the real time and then save to display the changed time, the user can set an alarm along with activating and deactivating the alarm with the click of a button. The LCD screen was implemented and displayed the correct values with the correct spacing and locations, the output could then have an LED attached to indicate the alarm if required.

Overall the design testing and simulation was a success, the system work as required and no problems were encountered, a newer microcontroller would provide better use and a recommendations and updated computers and monitors working at a faster speed could prove useful in the future.

7. References

Works Cited

AsciiTable, 2010. *ASCII Table and Description*. [Online]

Available at: <http://www.asciitable.com/>

[Accessed 30 01 2018].

Circuits Today, 2015. *Introduction to PIC 16F877*. [Online]

Available at: <http://www.circuitstoday.com/introduction-to-pic-16f877>

[Accessed 30 01 2018].

Microchip, 2017. *PIC16F877*. [Online]

Available at: <http://www.microchip.com/wwwproducts/en/PIC16F877>

[Accessed 30 01 2018].

Pictutorials, 2017. *PIC 16X84 Basics*. [Online]

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

[Accessed 30 01 2018].

wiki.osdev.org, 2013. *Interrupt_Service_Routines*. [Online]

Available at: http://wiki.osdev.org/Interrupt_Service_Routines

[Accessed 19 December 2013].

8. Appendix

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	{null}	32	20	040	Space	64	40	100	64;	@	96	60	140	96;	`
1	1	001	SOH	{start of heading}	33	21	041	!	65	41	101	65;	A	97	61	141	97;	a
2	2	002	STX	{start of text}	34	22	042	"	66	42	102	66;	B	98	62	142	98;	b
3	3	003	ETX	{end of text}	35	23	043	#	67	43	103	67;	C	99	63	143	99;	c
4	4	004	EOT	{end of transmission}	36	24	044	\$	68	44	104	68;	D	100	64	144	100;	d
5	5	005	ENQ	{enquiry}	37	25	045	%	69	45	105	69;	E	101	65	145	101;	e
6	6	006	ACK	{acknowledge}	38	26	046	&	70	46	106	70;	F	102	66	146	102;	f
7	7	007	BEL	{bell}	39	27	047	'	71	47	107	71;	G	103	67	147	103;	g
8	8	010	BS	{backspace}	40	28	050	(	72	48	110	72;	H	104	68	150	104;	h
9	9	011	TAB	{horizontal tab}	41	29	051	)	73	49	111	73;	I	105	69	151	105;	i
10	A	012	LF	{NL line feed, new line}	42	2A	052	*	74	4A	112	74;	J	106	6A	152	106;	j
11	B	013	VT	{vertical tab}	43	2B	053	+	75	4B	113	75;	K	107	6B	153	107;	k
12	C	014	FF	{NP form feed, new page}	44	2C	054	,	76	4C	114	76;	L	108	6C	154	108;	l
13	D	015	CR	{carriage return}	45	2D	055	-	77	4D	115	77;	M	109	6D	155	109;	m
14	E	016	SO	{shift out}	46	2E	056	.	78	4E	116	78;	N	110	6E	156	110;	n
15	F	017	SI	{shift in}	47	2F	057	/	79	4F	117	79;	O	111	6F	157	111;	o
16	10	020	DLE	{data link escape}	48	30	060	0	80	50	120	80;	P	112	70	160	112;	p
17	11	021	DC1	{device control 1}	49	31	061	1	81	51	121	81;	Q	113	71	161	113;	q
18	12	022	DC2	{device control 2}	50	32	062	2	82	52	122	82;	R	114	72	162	114;	r
19	13	023	DC3	{device control 3}	51	33	063	3	83	53	123	83;	S	115	73	163	115;	s
20	14	024	DC4	{device control 4}	52	34	064	4	84	54	124	84;	T	116	74	164	116;	t
21	15	025	NAK	{negative acknowledge}	53	35	065	5	85	55	125	85;	U	117	75	165	117;	u
22	16	026	SYN	{synchronous idle}	54	36	066	6	86	56	126	86;	V	118	76	166	118;	v
23	17	027	ETB	{end of trans. block}	55	37	067	7	87	57	127	87;	W	119	77	167	119;	w
24	18	030	CAN	{cancel}	56	38	070	8	88	58	130	88;	X	120	78	170	120;	x
25	19	031	EM	{end of medium}	57	39	071	9	89	59	131	89;	Y	121	79	171	121;	y
26	1A	032	SUB	{substitute}	58	3A	072	:	90	5A	132	90;	Z	122	7A	172	122;	z
27	1B	033	ESC	{escape}	59	3B	073	;	91	5B	133	91;	[	123	7B	173	123;	{
28	1C	034	FS	{file separator}	60	3C	074	<	92	5C	134	92;	\	124	7C	174	124;	
29	1D	035	GS	{group separator}	61	3D	075	=	93	5D	135	93;	]	125	7D	175	125;	}
30	1E	036	RS	{record separator}	62	3E	076	>	94	5E	136	94;	^	126	7E	176	126;	~
31	1F	037	US	{unit separator}	63	3F	077	?	95	5F	137	95;	_	127	7F	177	127;	DEL

(AsciiTable, 2010)

Final Report

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

44/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

Text Comment. The config is not detailed enough. The research should include other MCU's with limitations and go further than the config explanation.

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

GRADING FORM: CSG-RUBIC

NATHAN SMITH

0

RESEARCH-INFORMED LITERATURE



Very limited.

5/25

KNOWLEDGE AND UNDERSTANDING



Should be better explained.

11/25

ANALYSIS



Explanations could be better.

11/20

PRACTICAL APPLICATION AND DEP.



Good in parts, but not fully explained.

12/20

SKILLS FOR PROFESSIONAL PRAC.



Reasonable

5/10