

Distributed Intelligence Systems

Assignment One

Jack Spillett - 2024

Introduction

This laboratory report aims to outline the steps (research, system design, programming and testing) that contributed to producing a simple token-based distributed network for a single corner of a vehicle, using a single master microcontroller. The rear left corner was chosen. Proteus needed to be used to design and implement the system, and adequate notes were required to be provided throughout the program for clarity, likewise, screenshots and flowcharts were integral to completing the task.

Proteus was initially used to design and test the circuit and program. CCS C Compiler was used to produce large elements of the program. Multiple steps needed to be carried out to complete the task. Firstly, research was required to understand transmission systems as well as the fundamentals of programming microcontrollers. The 'logic' of the programs then needed to be built into a flowchart to simplify programming. Next, the token-based distributed network could be built on Proteus, after which programming could begin. Having uploaded the final programs onto the microcontrollers on Proteus, simulations could be carried out.

Assigning this task to students offered an opportunity to learn the fundamentals of programming electronics. Specifically, microcontrollers, which are widely used in society.

Research

i. Description of a Point-to-Point Transmission System

A Point-to-Point Transmission System is a communication network where data is transmitted over a dedicated communication channel from one device to another. The primary difference between a Point-to-Point Transmission System and a Multipoint Transmission System or Multi-Master Transmission System is that a Point-to-Point Transmission System only transmits data between two specific endpoints rather than from one or more sources to multiple destinations. A simple example of a Point-to-Point Transmission System is a Telephone Line where only one person can talk and only one person can listen [2].

The primary components and characteristics of a Point-to-Point Transmission System are:

- Transmitter - The device which generates and encodes the signals into a format suitable for transmission.
- Transmission Medium - The physical connection between the two devices i.e. copper wires, fibre optic or even wireless links like Bluetooth.
- Receiver - The device which receives and decodes the signal into the originally intended format.
- Dedicated Communication Channel - Point-to-Point Transmission Systems normally use private, dedicated channels between the two devices. Though this reserves the entire communication channel of the transmission system for the two devices, there is a possibility of a waste of network bandwidth.
- Point-to-Point Protocols - The communication protocols which govern the transmission process i.e. how data is formatted, transmitted, received, and error-checked between the transmitter and receiver.
- Encoding - Often the data signals are encoded/modulated before transmission and demodulated at the receiver to retrieve the original data, to ensure efficient use of the transmission medium and to enable reliable long-distance communication.
- Reliability and Security - Point-to-Point Transmission Systems offer high reliability and security because of the direct connection between the transmitter and receiver which reduces the likelihood of interference or interception [3],[4].

ii. Description of a Multipoint Transmission System

A Multipoint Transmission System (MTS) is a communication network which allows for the simultaneous transmission of data from one transmitter to multiple receivers (nodes) within a network. A common example of an MTS is a local area network (LAN). An MTS consists of a transmitter which generates and encodes data and then transmits the signal onto a transmission medium connected to multiple receiving points or destinations. This can be achieved using various technologies, such as copper cables, wireless communication, satellite communication and optical fibre networks [5]. An MTS unlike a Point-to-Point Transmission System doesn't reserve the communication channel for the Transmitter and any special receiver so the maximum bandwidth of the system can always be utilised. The system typically uses a standardised communication protocol to manage the data and ensure it reaches the intended receivers efficiently and reliably [6].

A key advantage of a Multipoint Transmission System is that data can be transmitted to multiple devices simultaneously, improving network efficiency and scalability. Similarly, Multipoint Transmission Systems offer greater flexibility and resilience over Point-to-Point Transmission Systems because they easily adapt to changes in network topology and device connectivity. The primary disadvantage of a Multipoint Transmission System over a Point-to-Point Transmission System is the lack of security because the communication channel is shared [3], [4].

iii. Description of a Multi-Master Transmission System

A Multi-Master Transmission System is a type of communication network where multiple devices can transmit and receive communication, unlike traditional systems with a single master device (transmitter) that controls the communication process, thus a Multi-Master Transmission System allows for more flexibility and scalability. In a Multi-Master Transmission System, each device (node) within the network can both transmit and receive data independently. This means that any node can communicate with any other node without relying on a central authority i.e. a central transmitter [7], [3].

The main advantages of a Multi-Master Transmission System are that it allows for greater fault tolerance, as the failure of one node wouldn't necessarily disrupt the entire network and that Multi-Master Transmission System allows for better utilisation of network resources, as multiple nodes can communicate simultaneously without waiting for a single master to grant permission. However, the primary disadvantage of a Multi-Master Transmission System is the complexity in contrast to a single-master system i.e. Point-to-Point Transmission System or a Multipoint Transmission System. The complexity of the system occurs when multiple devices attempt to transmit data simultaneously. Though protocols and algorithms are used to coordinate communication and resolve conflicts in an orderly manner, like with a Point-to-Point Transmission System or a Multipoint Transmission System, there is added complexity as a result of multiple transmitters [7].

Multi-Master Transmission Systems should be used where flexibility, scalability, and specific fault tolerances are required, some common uses include distributed computing, industrial automation and networking applications [5].

System Design

The assignment brief stated the need to produce a simple token-based distributed network for a single corner of a vehicle, to justify the requirements, the rear left corner was used to complete the task. The rear lights of a modern car include Daytime Running Lights, Brake Lights, Turn Signals (Indicators), Reversing Lights and Rear Fog Lights [1].

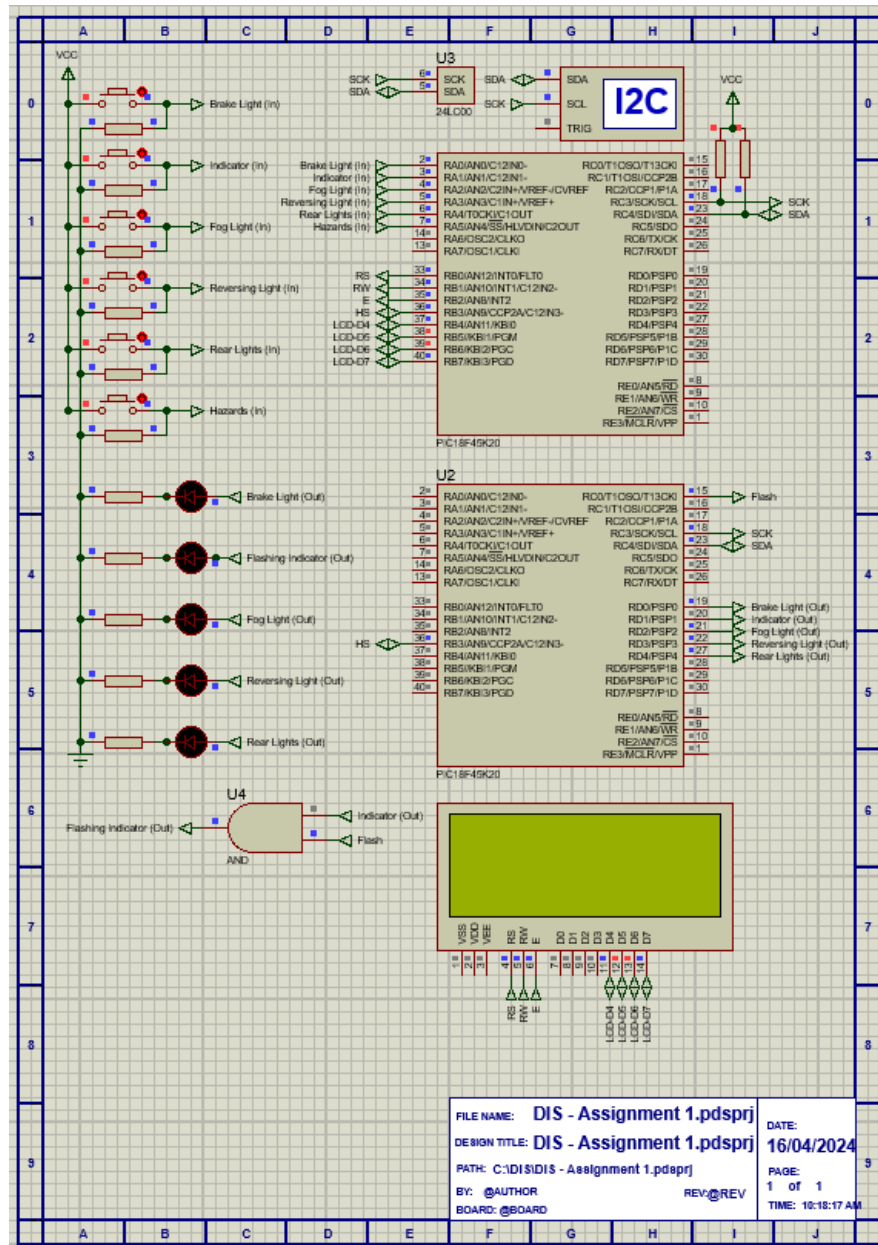


Figure 1: Final System Design on Proteus.

The system consisted of two 18F45K20 PIC Microcontrollers, labelled U1 and U2, with U1 being the master and U2 being the slave, for a Multipoint System. I2C was used as the communication method between the Microcontrollers, along the SCK and SDA terminals. In layman's terms: PIC U1 receives the input, writes to the I2C memory, PIC U2 reads the memory, turns the relevant output on or off and then writes to memory. After that PIC U1 reads the memory and displays the relevant data onto the LCD.

A VCC supply was inserted in parallel with the SCK and SDA terminals through resistors to supply power for communication. A 24LC00 was installed and labelled as U3 to act as EEPROM. Handshaking between the Microcontrollers was configured for when both microcontrollers are writing to the memory, using terminals B4 of U1 and U2. An I2C Debugger and an LCD were installed and programmed into the circuit to aid in debugging, with the LCD terminals being connected to Terminals B of U1.

Buttons were used to temporarily switch the states of the inputs of U1 along Terminals A, a variety of LEDs were used to mimic the outputs of U2 along Terminals D, and resistors were placed in series to alter the brightness of the LED. Terminal C0 was attached to an AND Gate (U4), through the programming it was made to flash. Combining the solid output of terminal D1 of U2 onto the AND Gates with the switching or 'flashing' output of C0, the Indicator LED is made to 'flash' when working as an Indicator or a Hazard. To reduce the complexity of the circuit, Input, Output and BIDIR Terminals were used where possible, improving the appearance of the circuit and making fault finding easier.

Name	Purpose
Brake Light (In)	Button Input to PIC U1
Indicator (In)	Button Input to PIC U1
Fog Light (In)	Button Input to PIC U1
Reversing Light (In)	Button Input to PIC U1
Rear Lights (In)	Button Input to PIC U1
Hazards (In)	Button Input to PIC U1
Brake Lights (Out)	PIC U2 Output to LED
Indicator (Out)	PIC U2 Output to AND Gate
Flashing Indicator (Out)	AND Gate Output to LED
Fog Light (Out)	PIC U2 Output to LED
Reversing Light (In)	PIC U2 Output to LED
Rear Lights (Out)	PIC U2 Output to LED

Name	Purpose
Flash	PIC U2 Output to AND Gate
SCK	I2C Communication
SDA	I2C Communication
RS	LCD Register Select Pin
RW	LCD Read and Write Pin
E	Enable PIN
HS	Hand Shaking for Memory
LCD - D4	LCD Data Pin
LCD - D5	LCD Data Pin
LCD - D6	LCD Data Pin
LCD - D7	LCD Data Pin

Table 1: Input, Output and BIDIR Terminals used in Proteus.

Programs

U1 - Main C	
	See Appendix 1
U1 - Main H	
	See Appendix 2
U2 - Main C	
	See Appendix 3
U1 - Main H	
	See Appendix 4

Table 2: List of Programs and the relevant Appendices.

Operation of the Programs

U1 - Main C

The Main C Program for U1 (the Master PIC) consists of six sections:

- The first section (lines 1 to 13) defines the LCD and the Handshaking Pins and references the LCD Source File (line 13). This section was drawn from CCS C Compiler after using the Project Wizard function to set the relevant parameters, those selected can be seen in Appendix 5.
- The next section (lines 15 to 18) names functions for the inputs, reading from the memory, writing to the memory and clearing the memory.
- Lines 21 to 65 are the third section of the program, in this section, the discrete value of the inputs (on or off) are used to switch a specific memory location.
- In the next section (lines 67 to 73) hexadecimal values from the memory are displayed on the LCD based on the last input that was pressed.
- The fifth section (from lines 78 to 119) sets the I2C functions for the functions named in section two, i.e. how to read from the memory, how to write to the memory and clearing the memory.
- The final section (lines 121 onwards) sets the working parameters for the inputs.

U1 - Main H

The Main H Program for U1 is fairly simple, it defines the parameters of the PIC and references the 18F45K20 Source File. The Fuses for the PIC are also set within this program.

U2 - Main C

The Main C Program for the U1 PIC can be split into 7 sections:

1. The first section (lines 1 to 9) defines the Read and Write to Memory Functions, without this U2 would not be able to use I2C communication. The section also defines an integer for a counter, states that Main H must be included for U2, defines the Handshaking Pin and references back to a Timer 2 Function (line 9).
2. Lines 11 to 19 cause PIN C0 to switch on and off (flash) which is then combined with an AND Gate to cause the Indicator and Hazard Light to Flash. Switching On and Off is done using a counter and a timer.
3. Between lines 24 and 26, the parameters for the timer are defined, this section was written using CCS Compiler.
4. Section four is found between lines 28 to 40, the function for distinguishing between new data and old data within the memory is established. Port B is set up for Handshaking. The Data for the program is drawn from the Memory. An If statement is used so that when the new Data from the Memory is different to the Old Data, the relevant states of the Port D outputs change. Their states are then written to the memory.
5. Section 5 of U2 - Main C is between lines 42 and 61 and creates the function for the Indicator to Flash when the Hazard button is pressed, regardless of whether the Indicator is on or off to begin with.
6. Lines 63 to 81 describe how the memory should be read using handshaking and I2C functions.
7. Finally lines 83 to 95 describe how the memory should be written to, again using handshaking and I2C functions.

U2 - Main H

The Main H Program for U2 is identical to the Main H Program in U1, it again defines the parameters of the PIC, references the 18F45K20 Source File and sets the Fuses for the PIC.

Miscellaneous Programs

LCD and Specific PIC 18F45K20 Files were included within the Proteus PIC Source Files for ease of use, without these files, the basic functions of the LCD and PICs would have to be written manually, which would be fairly complex and very time-consuming.

The purpose of the U1 LCD program is to provide compatibility between the PIC and the connected, LCD. The program was copied from the CCS Compiler as a source code. Without this, the PIC would not communicate with the LCD unless the routines were written manually. The program contains routines which are called back in the U1 Main C Program. The program is not needed by U2 however was copied from U1 if it was required for fault-finding purposes or future expansion.

The purpose of the 18F45K20 within the U1 Folder was again to remove the need to manually write the basic functions, chip identities and port parameters. The purpose of the 18F45K20 within the U2 Folder is identical to U1, the program was again copied from the CCS Compiler as a source code.

Flowchart

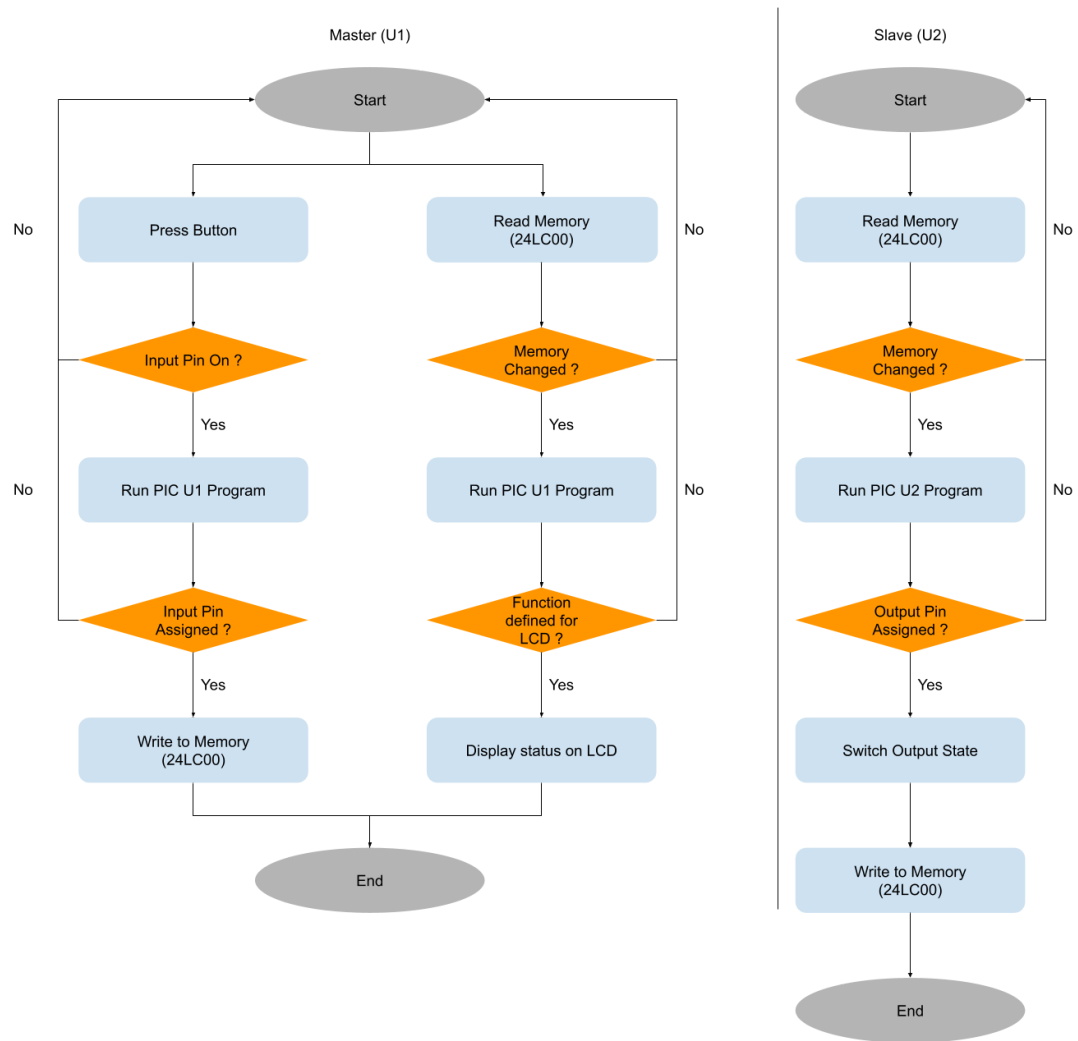


Figure 2: Flow Chart of Program Logic.

Testing

All Off

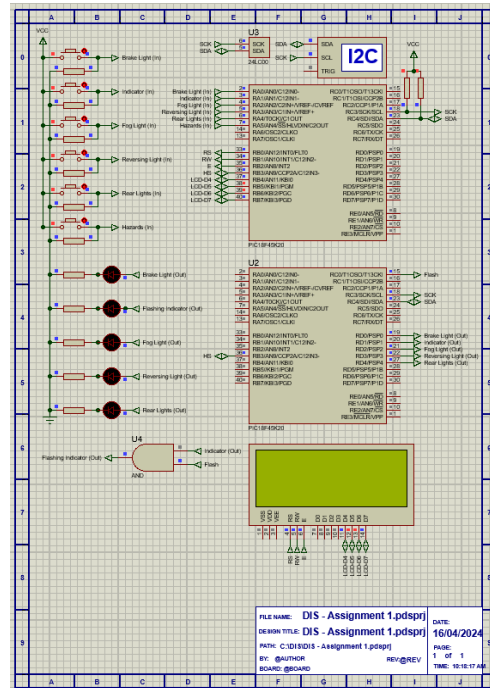


Figure 3: Testing the System Design on Proteus - All Off

All On

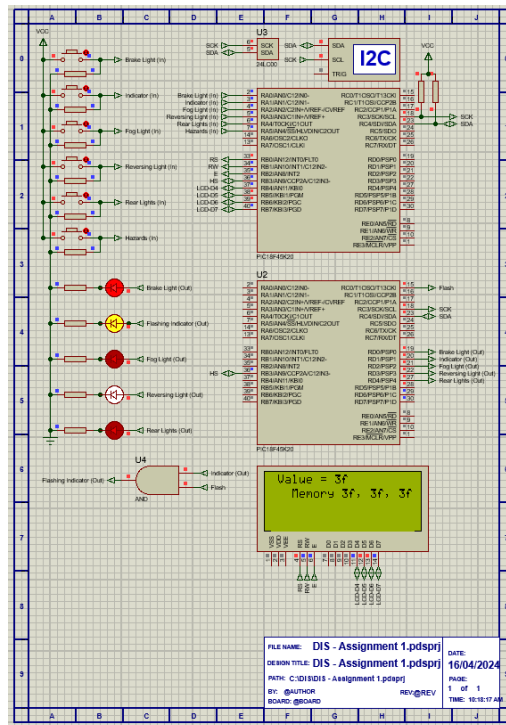


Figure 4: Testing the System Design on Proteus - All On

One On

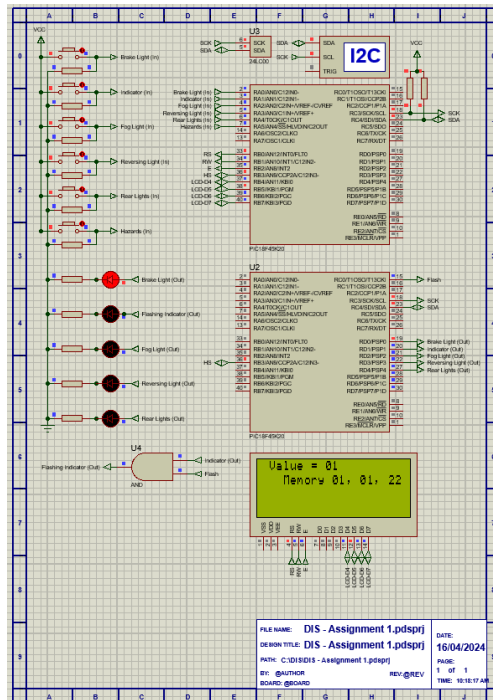


Figure 5: Testing the System Design on Proteus - One-On

Indicator Flashing

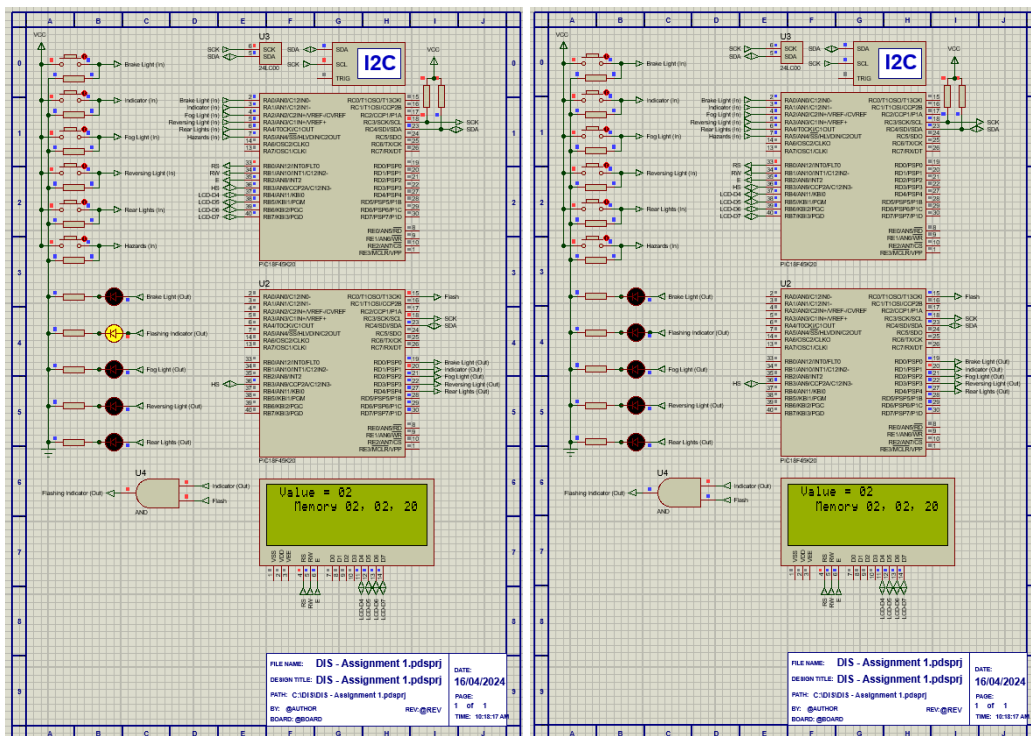


Figure 6: Testing the System Design on Proteus - Indicator Flashing

Hazards Flashing

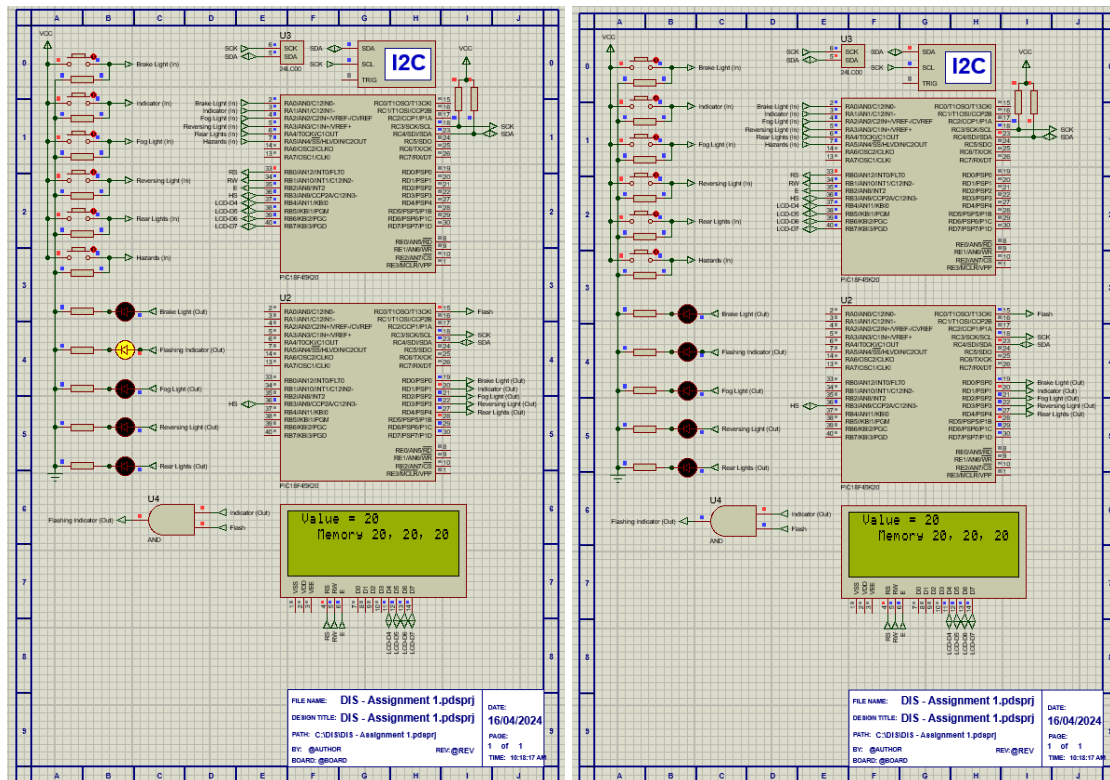


Figure 7: Testing the System Design on Proteus - Indicator Flashing as Hazards

Recording of Simulation

https://drive.google.com/file/d/19jNcbnnZE8p_4FTkQREV3-PGENvVY46q/view?usp=sharing

Review of Simulation

Using Proteus to simulate the system was beneficial, as it showed an issue within the program that was not addressed. The first draft of the system contained an Indicator and a Hazard Light, both of which would flash when the Hazard button was pressed. The Hazard Light was removed as it would not be true to life, where the indicator alone would flash.

Though the Hazard Light was removed from the system, the program itself was not changed, as a result when the Hazard Button was pressed, the Indicator Output Pin, D1, and the previous Hazard Light Output Pin, D5, would switch on. Though initially an attempt was made to remove the Hazard Light Output Pin from switching, it was eventually kept as part of the program. This is because when attempting to remove it, it was difficult to distinguish between when the Indicator Light was working as the Hazard Light or as the Indicator. Thus when fault finding, Pin D5, being On, distinguishes the indicator flashing as a Hazard rather than an Indicator.

Summary

This laboratory report has detailed the steps that have contributed to producing a simple token-based distributed network for the single corner of a vehicle on Proteus. A program was produced with regular notes, screenshots of the system were provided and flowcharts were used to show the operating principle of the system i.e. Write to Memory, Read from Memory.

Proteus was used to design and test the circuit and program. CCS Compiler was also used to produce large fundamental elements of the program. Background research was presented to show an understanding of Transmission Systems and some of the fundamentals of Microcontrollers. The 'logic' of the system has been organised using a flowchart to make programming the system simpler. With the program completed and installed onto the Microcontroller on Proteus, simulations were then completed.

Overall the research and lab work required to complete the laboratory report has created the opportunity to learn the fundamentals of programming PIC microcontrollers, which are widely used in society. Very few recommendations can be offered on improvements to the completion of the task besides removing PIN D5 flashing when the Hazzard Button is pressed, although it was found beneficial to keep it within the program.

References

- [1] - MyCarDictionary, "Rear light | Chassis | My car dictionary," *www.my-cardictionary.com*, Feb. 12, 2016.
<https://www.my-cardictionary.com/lighting/rear-light.html#:~:text=Rear%20lights%20provide%20greater%20safety>
- [2] - ScienceDirect, "Point-to-Point Communication - an overview | ScienceDirect Topics," *www.sciencedirect.com*, 2011.
<https://www.sciencedirect.com/topics/computer-science/point-to-point-communication>
- [3] - D. Serpanos and T. Wolf, *Architecture of Network Systems*. Elsevier, 2011.
- [4] - M. Plonus, *Electronics and Communications for Scientists and Engineers*, 2nd ed. Oxford: Butterworth-Heinemann, 2020.
- [5] - C. Diot, W. Dabbous, and J. Crowcroft, "Multipoint communication: a survey of protocols, functions, and mechanisms," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 3, pp. 277–290, Apr. 1997, doi: <https://doi.org/10.1109/49.564128>.
- [6] - H. Minn and D. Munoz, "Channel Knowledge Acquisition in Relay and Multipoint-to-Multipoint Transmission Systems," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 4, pp. 1416–1434, Apr. 2015, doi: <https://doi.org/10.1109/tvt.2014.2330992>.
- [7] - Y. Cheng-ying and C. Yong, "Research on Multi-master Communication System Based on RS485 Bus," *Journal of Physics: Conference Series*, vol. 1237, no. 4, p. 042078, Jun. 2019, doi: <https://doi.org/10.1088/1742-6596/1237/4/042078>.
- [8] - P. Horowitz, W. Hill, and Oxford University Press, *The art of electronics*, 3rd ed. New York: Cambridge University Press, 2018.
- [9] - G. Luecke, *Analog and digital circuits for electronic control system applications: using the TI MSP430 microcontroller*. Amsterdam; Boston: Elsevier/Newnes, 2005.
- [10] - J. Mankar, C. Darode, K. Trivedi, M. Kanoje, P. Shahare, and A. Professor, "REVIEW OF I2C PROTOCOL," *International Journal of Research in Advent Technology*, vol. 2, no. 1, 2014. Available:
<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=314537daa1f601f83044b25b68e2af6c8f331f3f>
- [11] - J. Sanchez and M. P. Canton, *Microcontroller Programming*. CRC Press, 2018.

Appendices

Appendix 1 - Program Listing for U1 Main C

```
1  #include
   <main.h>
                                     //CCS
2
3  #define LCD_ENABLE_PIN
   PIN_B2
                                     //CCS
4  #define LCD_RS_PIN
   PIN_B0
                                     //CCS
5  #define LCD_RW_PIN
   PIN_B1
                                     //CCS
6  #define LCD_DATA4
   PIN_B4
                                     //CCS
7  #define LCD_DATA5
   PIN_B5
                                     //CCS
8  #define LCD_DATA6
   PIN_B6
                                     //CCS
9  #define LCD_DATA7
   PIN_B7
                                     //CCS
10
11 #define HS
   PIN_B3
                                     //Handshaking between PICs
12
13 #include
   <lcd.c>
                                     //CCS
14
15 unsigned int Input_Buttons(void);
                                     //Function for Inputs
16 unsigned int Read_Memory(unsigned int);
                                     //Function for Reading Memory
17 void Write_Memory(unsigned int,unsigned int);
                                     //Function for Writing to Memory
18 void Clear_Memory(unsigned int);
                                     //Function for Clearing Memory
19
20
21 void main()
                                     //Main Program Below
22 {
23     unsigned int Data,Value,Local=0;;
                                     //Function for Data, Value and Local
24     port_b_pullups(0x08);
                                     //Set Handshaking on Port B with Pull Up
25
26     lcd_init();
                                     //LCD Programming Below
27
28     while(TRUE)
                                     //Start of LCD Loop
29     {
30         Value=0;
```

```

31         //Set Value as 0
        Value=Input_Buttons();

32        //Inputs are the Value in Loop
        switch(value)

33            //Start of LCD Program
        {
34                case 0x01:         if(!(Local & 0x01))Local+=1;
                                    //If Off turn On
35                                    else Local=Local & 0xfe;
                                    //If On turn Off
36                                    Write_Memory(0x00,Local);
                                    //Write to Memory
37                                    break;
                                    //End of Case

38                case 0x02:         if(!(Local & 0x02))Local+=2;
                                    //If Off turn On
39                                    else Local=Local & 0xfd;
                                    //If On turn Off
40                                    Write_Memory(0x00,Local);
                                    //Write to Memory
41                                    break;
                                    //End of Case

42                case 0x04:         if(!(Local & 0x04))Local+=4;
                                    //If Off turn On
43                                    else Local=Local & 0xfb;
                                    //If On turn Off
44                                    Write_Memory(0x00,Local);
                                    //Write to Memory
45                                    break;
                                    //End of Case

46                case 0x08:         if(!(Local & 0x08))Local+=8;
                                    //If Off turn On
47                                    else Local=Local & 0xf7;
                                    //If On turn Off
48                                    Write_Memory(0x00,Local);
                                    //Write to Memory
49                                    break;
                                    //End of Case

50                case 0x10:         if(!(Local & 0x10))Local+=0x10;
                                    //If Off turn On
51                                    else Local=Local & 0xef;
                                    //If On turn Off
52                                    Write_Memory(0x00,Local);
                                    //Write to Memory
53                                    break;
                                    //End of Case

54                case 0x20:         if(!(Local & 0x20))Local+=0x20;
                                    //If Off turn On
55                                    else Local=Local & 0xdf;
                                    //If On turn Off
56                                    Write_Memory(0x00,Local);
                                    //Write to Memory
57                                    break;
                                    //End of Case

58                case 0x40:         if(!(Local & 0x40))Local+=0x40;
                                    //If Off turn On
59                                    else Local=Local & 0xbf;
                                    //If On turn Off
60                                    Write_Memory(0x00,Local);
                                    //Write to Memory
61                                    break;
                                    //End of Case

62                case 0x80:         if(!(Local & 0x80))Local+=0x80;
                                    //If Off turn On
63                                    else Local=Local & 0x7f;
                                    //If On turn Off
64                                    Write_Memory(0x00,Local);
                                    //Write to Memory
65                                    break;
                                    //End of Case

66                case 0x00:         if(!(Local & 0x00))Local+=0x00;
                                    //If Off turn On
67                                    else Local=Local & 0xff;
                                    //If On turn Off
68                                    Write_Memory(0x00,Local);
                                    //Write to Memory
69                                    break;
                                    //End of Case

        }

        if(Value)

        //If Value
        {
            Data=Read_Memory(0x00);

```

```

//Data
in Memory is Data for Loop
70     printf(lcd_putc, "\f Value = %x", Data);
//Print on LCD Value
followed by the Stored Data
71     lcd_gotoxy(3,3);

//Positioning of Digits onto LCD
72     printf(lcd_putc, " Memory %x, %x, %x", Read_Memory(0x00), Read_Memory(0x05),
Read_Memory(0x09)); //Print Data from Specific Memory Locations onto LCD
73     }
74     }
75     }
76     }
77
78     unsigned int Read_Memory(unsigned int Address)
//Function
for Reading Memeory
79     {
80         unsigned int val;

//Start
81         while(!input(HS));

//If Handshaking Line is High
82         output_low(HS);

//Set Handshaking Line Low
83         i2c_start();

//I2C Start Pulse
84         i2c_write(0xA0);

//Set PIC Type and Number
85         i2c_write(Address);

//Set Address
86         delay_ms(10);

//Delay for PIC
87         i2c_start();

//I2C Start Pulse
88         delay_ms(10);

//Delay for PIC
89         i2c_write(0xA1);

//Set PIC Type and Number
90         val=i2c_read();

//Read from PIC
91         delay_ms(10);

//Delay for PIC
92         i2c_stop();

//I2C Stop Pulse
93         delay_ms(10);

//Delay for PIC
94         input(HS);

//Set Handshaking Line High
95         return(val);

//End
96     }
97
98     void Write_Memory(unsigned int Address, unsigned int Data)

```

```

//Function for
Writing to Memeory
99 {
100     while(!input(HS));
//If Handshaking Line is High
101     output_low(HS);
//Set Handshaking Line Low
102     i2c_start();
//I2C Start Pulse
103     i2c_write(0xA0);
//Set PIC Type and Number
104     i2c_write(Address);
//Set Address
105     i2c_write(Data);
//Write Data to Address
106     delay_ms(10);
//Delay for PIC
107     i2c_stop();
//I2C Stop Pulse
108     delay_ms(10);
//Delay for PIC
109     input(HS);
//Set Handshaking Line High
110 }
111
112 void Clear_Memory(unsigned int Data, unsigned int Clear=0)
//Function for
Clearing Memeory
113 {
114     i2c_start();
//I2C Start
115     i2c_write(0xA0);
//Set PIC Type and Number
116     i2c_write(Clear);
//Set Clear
117     i2c_write(Data);
//Write Data to Address
118     i2c_stop();
//I2C Stop Pulse
119 }
120
121 unsigned int Input_Buttons(void)
//Function for
Inputs
122 {
123     unsigned int value=0;
//Read Values
124     value=input_a();
//Values are derived from Terminals A
125     return(value);

```

126

```
//End of Function  
}
```

Appendix 2 - Program Listing for U1 Main H

```
1  #include
   <18F45K20.h>
2                                     //CCS
3
4  #device
   ADC=10
5                                     //CCS
6
7  #FUSES
   NOWDT
8                                     //No Watch Dog Timer
9
10 #FUSES
   FCMEN
11                                     //Fail-safe clock monitor enabled
12
13 #FUSES
   IESO
14                                     //Internal External Switch Over mode enabled
15
16 #FUSES
   PUT
17                                     //Power Up Timer
18
19 #FUSES
   BROWNOUT
20                                     //Reset when brownout detected
21
22 #FUSES
   BORV27
23                                     //Brownout reset at 2.7V
24
25 #FUSES
   LPT1OSC
26                                     //Timer1 configured for Low-power operation
27
28 #FUSES
   HFOFST
29                                     //High Frequency INTRC starts clocking CPU immediately
30
31 #FUSES
   NOPROTECT
32                                     //Code not protected from reading
33
34 #FUSES
   NOWRT
35                                     //Program memory not write protected
36
37 #FUSES
   NOEBTR
38                                     //Memory not protected from table reads
39
40 #FUSES
   NOMCLR
41                                     //Memory not protected from memory clear
42
43 #use
   delay(internal=16MHz)
44                                     //CCS
45
46 #use
   i2c(Master,Fast,sda=PIN_C4,scl=PIN_C3,force_hw)
47                                     //CCS
48
49 #define DELAY
   500
50                                     //CCS
```

Appendix 3 - Program Listing for U2 Main C

```
1  #include
   <main.h>
2                                     //CCS
3  #define HS
   PIN_B3
4                                     //CCS
5  unsigned int Read_Memory(unsigned int);
6  //Function for Reading Memeory
   unsigned int counter=0;
7  //Function for Counter
   void Write_Memory(unsigned int,unsigned int);
8  //Function for Writing to Memory
9  #INT_TIMER2
                                     //CCS
10
11 void TIMER2_isr(void)
12     //Function for Timer
13     {
14         counter++;
15         //Interrupt
16         if(counter==40)
17             //When Counter
18             {
19                 output_toggle(pin_c0);
20             }
21         //Switch Ouput of C0
22         counter=0;
23         //Set Counter to Zero
24     }
25
26 void main()
27     //Main Program Below
28     {
29         setup_timer_2(T2_DIV_BY_16,249,10);
30         //Timer Parameters
31         enable_interrupts(INT_TIMER2);
32         //Enables Timer 2 Interrupts
33         enable_interrupts(GLOBAL);
34         //Enable ALL Interrupts
35         unsigned int Data,Old_Data;
36         //Function for Data and Old Data
37         port_b_pullups(0x08);
38         //Set Handshaking on Port B with Pull Up
39         while(TRUE)
40             //Repeat the Code Below
41             {
42                 delay_ms(10);
43                 //Delay for PIC
```

```

34         Data=Read_Memory(0x00);
35         //Read from Memory
36         if(Data!=Old_Data)
37             //When Data is Different
38             {
39                 output_d(Data);
40                 //Write onto Output D
41                 Write_Memory(0x05,Data);
42                 //Write to Memory Location
43             }
44             if(Data & 0x20)
45                 //Function for
46                 Hazards
47                 //If
48                 {
49                     if(!input(PIN_A0))
50                     //Check Status
51                     {
52                         output_high(PIN_D1);
53                         //Set Output
54                     }
55                     On
56                     if(Data!=Old_Data)
57                         //When
58                         the Data is Different
59                         Write_Memory(0x09,Data);
60                         //Write Memmory
61                     }
62                     else
63                     //If Not
64                     {
65                         output_low(PIN_D1);
66                         //Set
67                     }
68                     Output Off
69                     if(Data!=Old_Data)
70                         //When
71                         the Data is Diffrent
72                         Write_Memory(0x09,0xff);
73                         //Write
74                     }
75                     Memmory
76                 }
77             }
78             Old_Data=Data;
79             //End of Function
80         }
81     }
82     unsigned int Read_Memory(unsigned int address)
83         //Function
84         for Reading Memeory
85         {
86             unsigned int val;
87             //Start
88             while(!input(HS));
89             //Is Handshaking Line High ?
90             output_low(HS);

```

```

68      //Set Handshaking Line Low
        i2c_start();

69      //I2C Start Pulse
        i2c_write(0xA0);

70      //Set PIC Type and Number
        i2c_write(address);

71      //Set Address
        delay_ms(10);

72      //Delay for PIC
        i2c_start();

73      //I2C Start Pulse
        delay_ms(10);

74      //Delay for PIC
        i2c_write(0xA1);

75      //Set PIC Type and Number
        val=i2c_read();

76      //Read from PIC
        delay_ms(10);

77      //Delay for PIC
        i2c_stop();

78      //I2C Stop Pulse
        delay_ms(10);

79      //Delay for PIC
        input(HS);

80      //Set Handshaking Line High
        return(val);

81      //End
    }
82
83 void Write_Memory(unsigned int address,unsigned int data) //Function for
    Writing to Memeory
84     {
85         while(!input(HS));

86         //Is Handshaking Line High ?
        output_low(HS);

87         //Set Handshaking Line Low
        i2c_start();

88         //I2C Start Pulse
        i2c_write(0xA0);

89         //Set PIC Type and Number
        i2c_write(address);

90         //Set Address
        i2c_write(data);

91         //Write Data to Address
        delay_ms(10);

92         //Delay for PIC
        i2c_stop();

        //I2C Stop Pulse

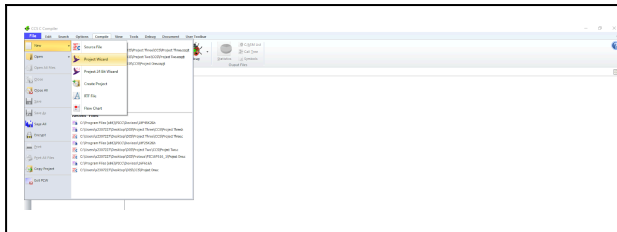
```

```
93         delay_ms(10);  
94         //Delay for PIC  
          input(HS);  
95         //Set Handshaking Line High  
96     }
```

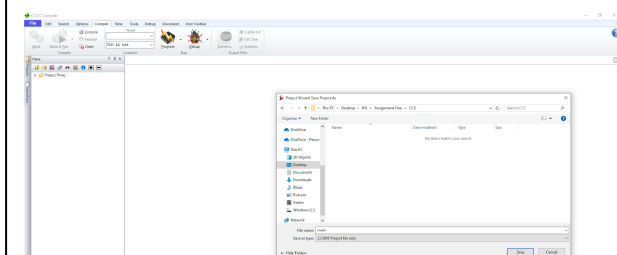
Appendix 4 - Program Listing for U2 Main H

```
1  #include
   <18F45K20.h>
2                                     //CCS
3
4  #device
   ADC=10
5                                     //CCS
6
7  #FUSES
   NOWDT
8                                     //No Watch Dog Timer
9
10 #FUSES
   FCMEN
11                                     //Fail-safe clock monitor enabled
12
13 #FUSES
   IESO
14                                     //Internal External Switch Over mode enabled
15
16 #FUSES
   PUT
17                                     //Power Up Timer
18
19 #FUSES
   BROWNOUT
20                                     //Reset when brownout detected
21
22 #FUSES
   BORV27
23                                     //Brownout reset at 2.7V
24
25 #FUSES
   LPT1OSC
26                                     //Timer1 configured for Low-power operation
27
28 #FUSES
   HFOFST
29                                     //High Frequency INTRC starts clocking CPU immediately
30
31 #FUSES
   NOPROTECT
32                                     //Code not protected from reading
33
34 #FUSES
   NOWRT
35                                     //Program memory not write protected
36
37 #FUSES
   NOEBTR
38                                     //Memory not protected from table reads
39
40 #FUSES
   NOMCLR
41                                     //Memory not protected from memory clear
42
43 #use
   delay(internal=16MHz)
44                                     //CCS
45
46 #use
   i2c(Master,Fast,sda=PIN_C4,scl=PIN_C3,force_hw)
47                                     //CCS
48
49 #define DELAY
   500
50                                     //CCS
```

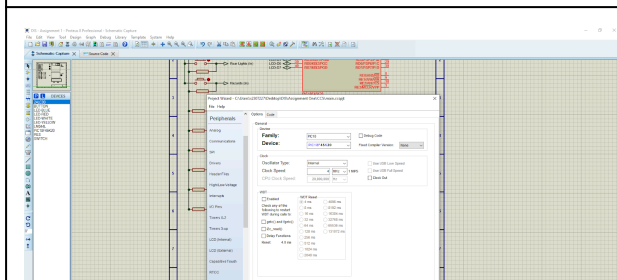
Appendix 5 - CCS Project Wizard Step-by-Step



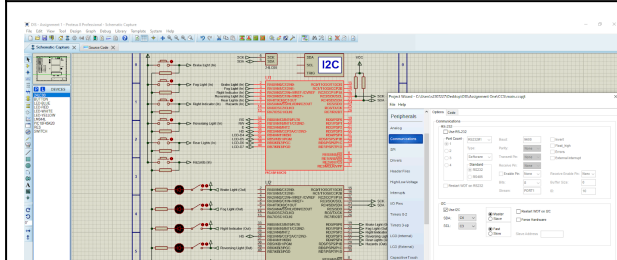
Open CCS,
Then go to File, New, Project Wizard



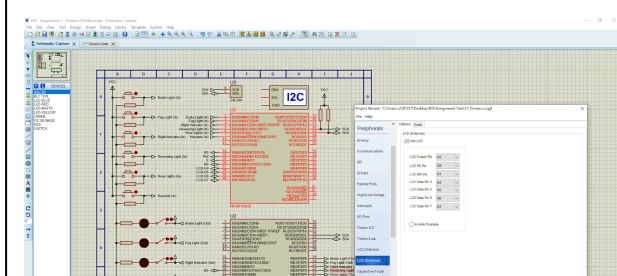
Save to a new dedicated folder on the computer.



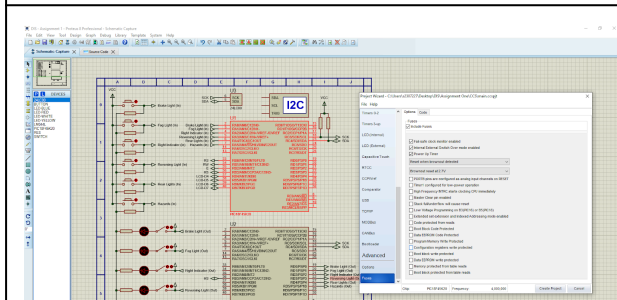
Set the Family, Device, Oscillator Type and Clock Speed. For example, in this report a PIC18; PIC1845K20; Internal Oscillator with a 4MHz Clock Speed.



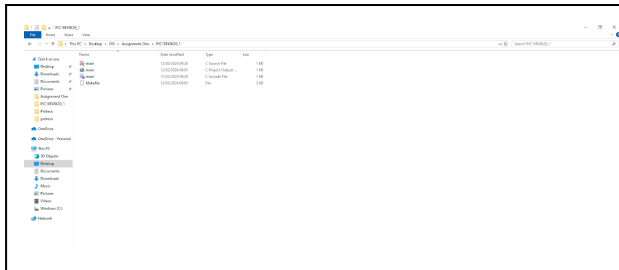
In Communications, set the Master and Slave settings. For this report, I2C was used, with the SDA being set to C4 and the SCL being set to C3.



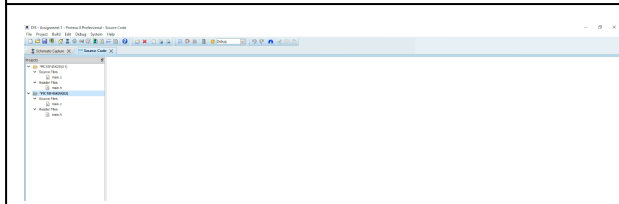
Set up the LCD in the LCD (External Tab). In this report, the LCD Enable Pin has been set to B2, the RS Pin is B0, the RW pin is B1, The Data Pin 4 is B4, the Data Pin 5 is B5, the Data Pin 6 is B6 and the Data Pin 7 is B7.



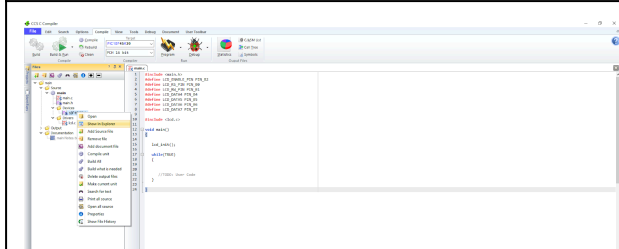
In Fuses, set enable fuses and then only select Fail-Safe Clock Monitor Enabled, Internal External Switch Over Mode Enabled and Power Up Timer. Then set the first dropdown menu to Reset when Brownout Detected. Then in the second dropdown select Brownout reset at 2.7V.



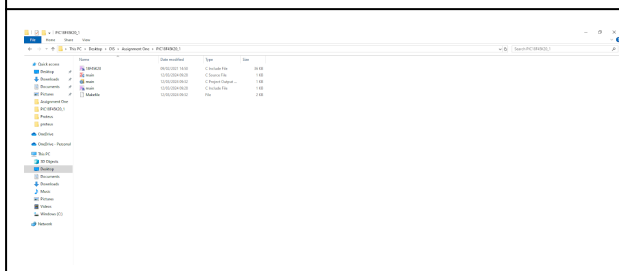
Within Proteus, under Project, Project Settings, de-select Embed Files, this creates a new folder where the CCS Files can be added. Save the Main C and Main H, CCS files into the P18F45K20_1 Folder.



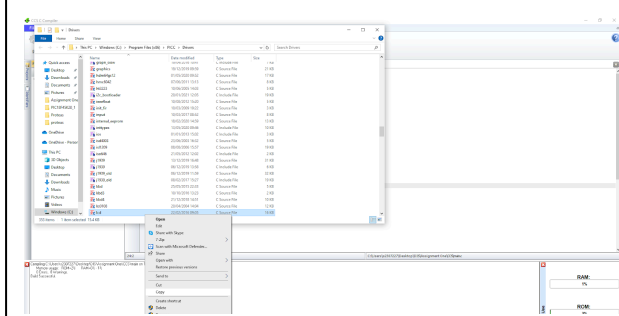
Within Proteus, these files can then be loaded by right-clicking and selecting New File.



In CCS, right-click on 18F45K20 under the Device Files Folder. Then select Show in Explorer and copy the highlighted file.

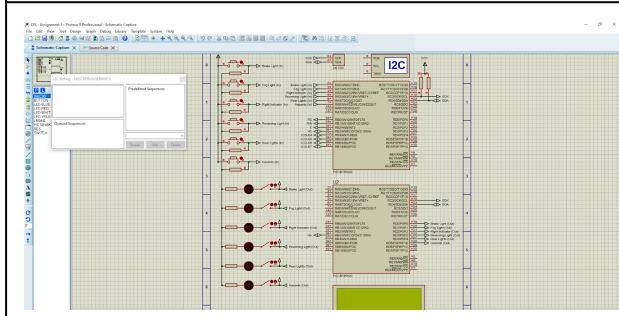


Paste the Highlighted File into the P18F45K20_1 Folder.



In CCS, right-click on LCD under the Drivers Folder. Then select Show in Explorer and copy the highlighted file.

Paste the Highlighted File into the P18F45K20_1 Folder.



Proteus should now be able to Run although there is no 'main' program written yet.