

DIS Laboratory Report - Alistair Lounds - 2227990

by Alistair Lounds

Submission date: 12-Nov-2024 10:52AM (UTC+0000)

Submission ID: 242399900

File name: 43539_Alistair_Lounds_DIS_Laboratory_Report_-_Alistair_Lounds_-_2227990_523660_553891819.docx (1.36M)

Word count: 3535

Character count: 21225



Simple Token Based Distributed Network

By Alistair Lounds 2227990

Supervisor: Dr Gary Jones

Second Marker: Dr Stephen Mosey

Project submitted as part of the requirements for the award of
BEng (Hons) Embedded Electronic Systems Design

November 2024

Table of Contents

Research.....	3
Point to Point Transmission Systems.....	3
Point to Multipoint Transmission Systems	3
Multi-Master Transmission Systems	4
Quality Management Systems	4
Systems Design.....	5
Circuit Design	5
PIC Microcontroller Selection	5
Operation of Motorcycle Lights	5
Master to Slave Communication	6
Embedded C Program Listing	7
master.c Listing	7
master.h Listing	8
slave.c Listing	9
slave.h Listing.....	10
Operation of Program	11
master.c Program Flowchart	11
slave.c Program Flowchart	12
Testing & Verification	12
Rear Driving Lamp Test.....	13
Rear Brake Lamp Test	13
Flashing Rear Left Indication Lamp Test	14
Rear Fog Lamp Test	15
Error Mitigation	15
Conclusion	16
References.....	16

Table of Figures

Figure 1 - Point to Point Communication Diagram	3
Figure 2 - Point to Multipoint Communication Diagram	4
Figure 3 - Multi-Master Communications Diagram	4
Figure 4 - Simulated Distributed Network for a Motorcycle using Proteus Design Suite v8.16.....	5
Figure 5 - master.c Program Listing	8
Figure 6 - master.h Program Listing	8
Figure 7 - slave.c Program Listing	10
Figure 8 - slave.h Program Listing	10
Figure 9 - Flowchart for master.c Program	11
Figure 10 - Flowchart for slave.c Program	12
Figure 11 - VSM Studio Output - Compiling master.c and slave.c.....	13
Figure 12 - Simple Token based Distributed Network - Rear Driving Lamp	13
Figure 13 - Simple Token based Distributed Network - Rear Brake Lamp.....	14
Figure 14 - Simple Token based Distributed Network - Flashing Rear Left Indicator Lamp	14
Figure 15 - Simple Token based Distributed Network - Rear Fog Lamp.....	15
Figure 16 - I2C Serial EEPROM Memory Error	15

Abstract

This report details the programming of two PIC16F876 8-bit microcontrollers using the Embedded C programming language to produce a simple token based distributed network for an automotive application. This network replicates the rear left corner of a motorcycle; simulating the control of rear driving, brake, fog and left indicator lamps.

The networks design was programmed within the source code terminal of Proteus Design Suite Version 8.16. [1] This tool has been applied with the use of CCS C Compiler Version 5.103, allowing for the source code to be compiled and debugged. [2]

Research

In order to complete this assignment, a working knowledge and understanding was required of transmission network topologies.

Point to Point Transmission Systems

Point to Point transmission systems are communication networks where there is a dedicated common channel between two nodes. For instance, data is transmitted along a data bus from a master device to a receiving slave device. [3] Figure 1 illustrates this system as a simple box diagram. A benefit of this type of communication system is that it is considered more secure than point to multipoint and multi-master systems; as data is not shared across multiple channels/lines, denying disclosure of data to other devices. [3] However, this presents the drawback of the system being susceptible to poor resource allocation. As the entire networks bandwidth is reserved for only the two devices, this can introduce inefficiency to the systems design as the bandwidth may not be fully utilised. [3]

An example of this can be found in telecommunications and cellular architecture, where connections are made between two locations. [4] Point to point communication is used to connect 4th generation Long-term Evolution (LTE) telecommunications networks base stations 'Evolved Node B' (eNodeB) to servicing gateways (S-GW) in the core network. [5]

The use of two PIC16F876 microcontrollers for this simple token based distributed network is an example of a Point to Point Transmission system.

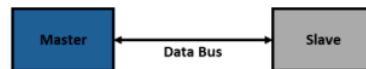


Figure 1 - Point to Point Communication Diagram

Point to Multipoint Transmission Systems

Point to Multi-point transmission systems are where there is a channel shared between multiple device nodes, with one master device controlling many slave devices, illustrated by Figure 2.

One benefit of this system is that network resources can be allocated more efficiently between multiple devices, enabling the capacity to be fully utilised. These systems allow for ease of installation as there is one master node, having room for expansion and connecting additional slave nodes. These networks also offer increased reliability as all slave nodes share one communication channel, not requiring complicated management processes to coordinate when nodes transmit data across the bus. [6] However, this carries the disadvantage that the networks may be considered less private as data is transmitted from one master node across multiple channels to all slave nodes. This can reduce network security as data is unnecessarily shared to wider network devices. [3]

An example of the application of Point to Multipoint transmission systems is that in factory automation, where one master Programmable Logic Controller (PLC) master node controls various slave sensors and mechanisms along a manufacturing production line. [7]

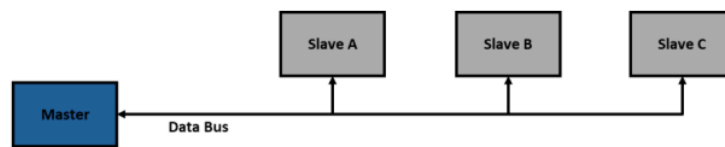


Figure 2 - Point to Multipoint Communication Diagram

Multi-Master Transmission Systems

Multi-Master transmission systems are where there are many nodes within the communication network that can initiate communication with other devices. Figure 3 illustrates this system as a simple box diagram.

One benefit of Multi-Master transmissions systems is that having various master nodes allows for the implementation of load balancing. This enables the distributed network to control traffic flow and improve system performance. [8] However, if this is mismanaged the network can experience drawbacks in performance such as network congestion and latency. As there are multiple nodes that can transceive in these distributed networks, loss of data packets and latency can occur due to high volumes of traffic. This can reduce system throughput, and therefore data flow control mechanisms are necessary to ensure the bus does not become congested. This is vital for maintaining good quality of service. [9]

Due to the complexity of these transmission systems, maintaining communication data links can become more complex and carry higher operating/maintenance costs; having reduced scalability in comparison to other distributed communications systems. [9] An example of where Multi-Master transmission is used is that of Controlled Area Network (CAN) systems found in vehicles for automotive and space flight applications. [10]



Figure 3 - Multi-Master Communications Diagram

Quality Management Systems

During the development of this simple token based distributed network, it was import to have awareness of Quality Management Systems and the need for continuous improvement.

The Engineering Council defines Quality Management Systems (QMS) to be the formalisation of systems of work and procedures that are required to ensure the achievement of key organisational aims. [11] QMS aims to provide consistency in standard operating procedures, meaning that that tailored processes are in place so that value is provided to stakeholders. [11] QMS encourages management to utilise resources effectively, promoting organisationally efficiency and consideration to be made regarding the consequences of decisions, both short and long term. Risk-based thinking can be applied to mitigate/reduce unintended consequences. [12]

QMS emphasises the importance for engineering practices to be sustainable and geared towards long-term success. This involves there being adequate processes in place to ensure organisations are in compliance with international and industry standards; for instance following good engineering practices detailed in 'BS EN ISO 9001'. [14] This helps organisations fulfil essential regulatory and legal requirements.

For continuous improvement to be achieved, organisations must be reactive, taking onboard feedback and making data driven decisions. To ensure there is continuous improvement, QMS must foster a workplace culture that encourages employee involvement across all levels of the organisation.

A focus on continuous improvement may result in enhanced health and safety, through the development of safer working practices. By conducting regular auditing and reviews, organisations can gain insightful data and feedback, aiding fulfilment of QMS aims. An example of this is, the Occupational Health and Safety Management standard ISO 45001, that incorporates risk assessments and auditing into the quality management process. [13]

Systems Design

Circuit Design

Figure 4 shows the Simple Token Based Distributed Network for a Motorcycle designed and simulated within Proteus Design Suite.

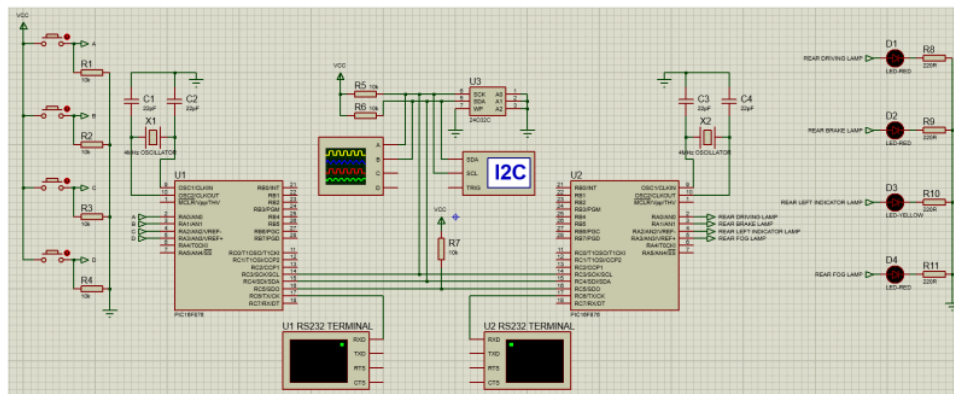


Figure 4 - Simulated Distributed Network for a Motorcycle using Proteus Design Suite v8.16

PIC Microcontroller Selection

Figure 4 shows that two PIC16F876 microcontrollers were chosen to act as the Master and Slave chips in this network. A single Peripheral Interface Controller (PIC) master component 'U1' provides signals to the single slave component 'U2' to operate the vehicles lamps.

The PIC16F876 microcontroller was selected for three main reasons. Firstly, the chip has a three Input/Output (I/O) ports. This allows for PortA to be allocated for button/token inputs and PortC for communication between 'U1' and 'U2' PICs and the I2C memory chip 'U3'. With both having PortB remaining unutilised for both the master and slave PICs, available for future system expansion. The PIC16F876 microcontroller also has very low power consumption, being less than 0.6mA at 3 Volts when operated at a clock speed of 4MHz. [15] The PIC16F876 is also cheaper relative to many PIC18F alternatives, providing a cost saving to the manufacturer in this large scale automotive application.

However, due to the PIC16F876 not having an internal oscillator. An external 4MHz quartz crystal oscillator has been added to both the microcontroller chips to provide a system clock source for program timing. Due to this, I had to write the header files master.h and slave.h such that they instruct the program that an external crystal oscillator is being used. Therefore, modifying line 12 of the header files from '#use delay(internal=4MHz)' to become '#use delay(crystal=4MHz)'. [16]

Operation of Motorcycle Lights

The vehicle chosen for this assessment is that of a motorcycle, simulating a simple tokenised distributed network for the rear left corner of the lighting system. Therefore the program requires the implementation of four lamps:

Rear Driving Light

Consisting of the 'D1' red Light Emitting Diode (LED) connected to a 220 Ω resistor; fed by the output pin RA0 of the slave PIC. When the Single Pole Single Throw (SPST) button A is pressed, the switch is momentarily on; this pulse is detected triggering switch case 'A'. A second press of this button triggers switch case 'a'.

Rear Brake Light

Consisting of the 'D2' red LED connected to a 220 Ω resistor; fed by the output pin RA1 of the slave PIC. When the SPST button B is pressed, the switch is momentarily on; this pulse is detected triggering switch case 'B'. A second press of this button triggers switch case 'b'.

Rear Left Indicator Light

Consisting of the 'D3' yellow LED connected to a 220 Ω resistor; fed by the output pin RA2 of the slave PIC. When the SPST button C is pressed, the switch is momentarily on; this pulse is detected triggering switch case 'C'. A second press of this button triggers switch case 'c'.

Rear Fog Light

Consisting of the 'D4' red LED connected to a 220 Ω resistor; fed by the output pin RA3 of the slave PIC. This LED can only be turned by pressing button D, when the rear driving light is switched on. This is done by setting the fog variable to equal 1, when the driving light output pin RA0 is set to a High state; and 0 when set to a Low state. However, the rear fog light can be switched off by pressing SPST button D, regardless of state of the RA0 pin, the rear driving light output.

Master to Slave Communication

Inter-Integrated Communication Protocol (I2C) has been selected to enable dataflow between the master and slave PIC16F876 microcontrollers for the distributed network. This protocol is used to pass data between the PIC16F876 microcontrollers and component U3, the '24C02C' Serial Electrically Erasable Programmable Read-Only Memory (EEPROM) integrated circuit. [17]

I2C is a synchronous protocol, therefore requiring a clock signal taken from the PIC16F876 microcontroller. This is provisioned by the Serial Clock Line (SCL) connected to the RC3 pins, and connected to the Serial Clock (SCK) port of the I2C serial EEPROM chip. The Serial Data Line (SDA) enables data transmission between the master and slave PIC16F876's. [17]

Embedded C Program Listing

master.c Listing

Figure 5 shows the Embedded C programs source code file 'master.c' for the master PIC16F876 component U1 and comments explaining its structure and operation.

```

1  /* Student Number: 2227990
2  Name: Alistair Lounds
3  Last Modified: 07/11/2024
4  Description: This program is a simple token based distributed network, consisting of two PIC16F876 8-bit microcontrollers,
5              one acting as the master PIC and the other as the slave. This circuit simulates the operation of Lights for
6              the rear left corner of a motorcycle.
7
8  */
9  /***master***/
10 #include <master.h> //include master.h header file
11
12 /***global variables***/
13 char buttons=0; //initialise buttons variable as character data type and set to equal 0
14 char output; //initialise output variable as a character data type
15 char txbyte; //initialise txbyte variable as a character data type
16
17 unsigned int value=0x00; //initialise value variable as a unsigned integer data type
18
19 char read_buttons(void); //define subroutine read_buttons
20 void ep_write(char, char); //define subroutine ep_write
21
22 /***main***/
23 void main(void)
24 {
25     while(TRUE) //Loop forever
26     {
27         buttons=read_buttons(); //set buttons variable to be value returned by read_buttons subprocess
28         switch(buttons) //switch statement for buttons
29         {
30             case 1: if(output & 0x01) //switches rear driving Light on and off
31             {
32                 output-=1; //sets bit position 0 as off
33                 txbyte='a'; //sets the txbyte token to be string 'a'
34             }
35             else
36             {
37                 output+=1; //sets bit position 0 as on
38                 txbyte='A'; //sets the txbyte token to be string 'A'
39             }
40             break; //break enables the program to break out of switch block and stop execution - exiting Loop
41
42             case 2: if(output & 0x02) //switches rear brake Light on and off
43             {
44                 output-=2; //sets bit position 1 as off
45                 txbyte='b'; //sets the txbyte token to be string 'b'
46             }
47             else
48             {
49                 output+=2; //sets bit position 1 as on
50                 txbyte='B'; //sets the txbyte token to be string 'B'
51             }
52             break; //break enables the program to break out of switch block and stop execution - exiting Loop
53
54             case 4: if(output & 0x04) //switches rear Left indicator Light on and off
55             {
56                 output-=4; //sets bit position 2 as off
57                 txbyte='c'; //sets the txbyte token to be string 'c'
58             }
59             else
60             {
61                 output+=4; //sets bit position 2 as on
62                 txbyte='C'; //sets the txbyte token to be string 'C'
63             }
64             break; //break enables the program to break out of switch block and stop execution - exiting Loop
65
66             case 8: if(output & 0x08) //switches rear fog Light on and off
67             {
68                 output-=8; //sets bit position 3 as off
69                 txbyte='d'; //sets the txbyte token to be string 'd'
70             }
71             else
72             {
73                 output+=8; //sets bit position 3 as on
74                 txbyte='D'; //sets the txbyte token to be string 'D'
75             }
76             break; //break enables the program to break out of switch block and stop execution - exiting Loop
77             default: break; //default is an optional statement which specifies to break out of switch statement Loop if no case is matched
78         }
79
80         printf("%c", txbyte); //print the token character to the rs232 virtual terminal
81         ep_write(0, txbyte); //writes the data in memory at address location 0
82     }
83 }
84
85 /***read_buttons subroutine***/
86 char read_buttons(void)
87 {
88     char value=0; //initialises a value variable for the buttons and sets it to equal 0
89     while(!input_a()); //while there is not an input to PORTA wait for an input - button press event
90     value=input_a(); //set value variable to equal input from PortA
91     delay_ms(500); //500ms delay time provided for button release
92     return(value); //when read_buttons subroutine is called and executed this returns the value as a character
93 }
94
95 /***ep_write subroutine***/
96 void ep_write(char address, char data)
97 {
98     while(!input(pin_c5)); //waits for memory of chip to be available
99     output_low(pin_c5); //set Serial Data Out(SDO) pin Low (memory busy)
100    i2c_start(); //sends i2c start condition pulse

```

```

101     i2c_write(0xA0);           //sets device address to 0xA0
102     i2c_write(address);       //sends the device address to be write to
103     i2c_write(data);          //sends the data to write to the device address
104     delay_ms(delay);          //wait for 20ms
105     i2c_stop();               //sends i2c stop condition pulse
106     delay_ms(delay);          //wait for 20ms - gives time to allow i2c chip to store data to 0xA0 memory address
107     input(pin_c5);            //releases Serial Data Out(SDO) pin RC5
108 }

```

Figure 5 - master.c Program Listing

master.h Listing

Figure 6 shows the Embedded C programs header file 'master.h' for the master PIC16F876 component U1 and comments explaining its structure and operation.

```

1  /**master header file**
2  #include <16F876.h>           //header file for PIC used in circuit "PIC16F876"
3  #device ADC=10               //sets read_adc return size to 10 bits
4
5  #FUSES NOPUT                 //no power up timer
6  #FUSES BROWNOUT              //reset when brownout detected
7  #FUSES LVP                   //low voltage programming on B3(PIC16)
8  #FUSES NOCPD                 //no EE protection
9  #FUSES NOWRT                 //program memory not write protected
10 #FUSES NOPROTECT             //code not protected from reading
11
12 #use delay(crystal=4MHz)      //no internal oscillator - external clock speed set to 4MHz
13 #use rs232(baud=9600,parity=N,xmit=pin_c6,rcv=pin_c7,bits=8,stream=PORT1,errors) //rs232 communication pins - pin RC6 is transmit(xmit) - pin RC7 is receive
14 #use i2c(Master,Fast,sda=pin_c4,scl=pin_c3,force_hw) //i2c communication pins - pin RC3 is serial clock line (SCL) - pin RC4 is serial data line (SDA)
15
16 #define delay 20              //delay variable set to a value of 20ms - therefore "delay_ms(delay);" would be 20ms

```

Figure 6 - master.h Program Listing

slave.c Listing

Figure 7 shows the Embedded C programs source code file 'slave.c' for the slave PIC16F876 component U2 and comments explaining its structure and operation.

```

1  /* Student Number: 2227990
2  Name: Alistair Lounds
3  Last Modified: 07/11/2024
4  Description: This program is a simple token based distributed network, consisting of two PIC16F876 8-bit microcontrollers,
5  one acting as the master PIC and the other as the slave. This circuit simulates the operation of lights for
6  the rear left corner of a motorcycle.
7  */
8
9  /***slave***/
10 #include <slave.h> //include slave.h header file
11
12 /***Global variables***/
13 char buttons=0; //Initialise buttons variable as character data type and set to equal 0
14 char old_buttons=0; //Initialise old_buttons variable as character data type and set to equal 0
15 char output=0; //Initialise output variable as character data type and set to equal 0
16 char q=0; //Initialise q variable as character data type and set to equal 0
17
18 unsigned int counter=0; //Initialise counter variable as unsigned integer data type and set to equal 0
19 unsigned int flash=0; //Initialise flash variable as unsigned integer data type and set to equal 0
20 unsigned int left=0; //Initialise left variable as unsigned integer data type and set to equal 0
21 unsigned int fog=0; //Initialise fog variable as unsigned integer data type and set to equal 0
22
23 void ep_write(char address,char data); //define subroutine ep_write
24 char ep_read(char); //define subroutine ep_read
25 void indicator_flash(void); //define subroutine indicator_flash
26 void motorcycleLights(void); //define subroutine motorcycleLights
27
28 /***timer subroutine***/
29 #INT_TIMER2
30 void TIMER2_isr(void)
31 {
32     counter++; //Increment counter variable - increase counter by a value of 1
33     if(counter==40) //conditional statement, when counter is equal to 40 the statement is True - executes following code
34     {
35         counter=0; //resets counter value to 0
36         if(flash) flash=0; //sets flash variable to equal 0
37         else flash=1; //else set flash variable to equal 1
38         indicator_flash(); //call and execute indicator_flash subroutine
39     }
40 }
41
42 /*******
43 void main(void)
44 {
45     setup_timer_2(T2_DIV_BY_16,255,4); //timer - 255 is where the clock value resets - 4 timer overflows before interrupt occurs
46     enable_interrupts(INT_TIMER2); //Initialises the INT_TIMER2 interrupt
47     enable_interrupts(GLOBAL); //Initialises the global interrupts
48     while(true) //Loop forever
49     {
50         buttons=ep_read(0x00); //read data byte from keyboard
51         if(old_buttons!=buttons) //conditional statement - if old_buttons does not equal buttons execute the following code within curly brackets
52         {
53             printf("%c",buttons); //print the button token character to rs232 virtual terminal
54             ep_write(0x10,buttons); //write the received value into memory at address location 0x10
55             delay_ms(delay); //wait for 20ms
56             motorcycleLights(); //call and execute motorcycleLights subroutine
57         }
58         old_buttons=buttons; //sets old_buttons variable to equal previous value - buttons
59     }
60 }
61
62 /***motorcycle subroutine***/
63 void motorcycleLights(void)
64 {
65     switch(buttons) //switch statement for buttons
66     {
67         //rear driving lamp
68         case 'A': output+=0x01; //adds 0x01 to output to turn on bit 0 (rear driving light)
69                 fog=1; //sets fog variable to equal 1
70                 break; //break enables the program to break out of switch block and stop execution - exiting loop
71
72         case 'B': output=output & 0xfe; //bitwise AND the output byte with 0xfe (11111110) to clear bit position 0
73                 fog=0; //sets fog variable to equal 0
74                 break; //break enables the program to break out of switch block and stop execution - exiting loop
75
76         //rear brake lamp
77         case 'B': output+=0x02; //adds 0x02 operand to output to turn on bit 1 (rear brake lamp)
78                 break; //break enables the program to break out of switch block and stop execution - exiting loop
79
80         case 'b': output=output & 0xfd; //bitwise AND the output byte with 0xfd (11111101) to clear bit position 1
81                 break; //break enables the program to break out of switch block and stop execution - exiting loop
82
83         //rear left indicator lamp
84         case 'C': output+=0x04; //adds 0x04 operand to output to turn on bit 2 (left indicator lamp)
85                 left=1; //set left variable to equal 1
86                 break; //break enables the program to break out of switch block and stop execution - exiting loop
87
88         case 'c': output=output & 0xfb; //bitwise AND the local output byte with 0xfb (11111011) to clear bit position 2
89                 left=0; //set left variable to equal 0
90                 break; //break enables the program to break out of switch block and stop execution - exiting loop
91
92         //rear fog lamp
93         case 'D': if(fog==1) //conditional statement - only when fog equals 1 (rear driving light on) - fog light can be turned on
94                 {
95                     output+=0x08; //adds 0x08 operand to output to turn on bit 3 (fog lamp) if rear driving lights are on
96                     break; //break enables the program to break out of switch block and stop execution - exiting loop
97                 }
98
99         case 'd': output=output & 0xf7; //bitwise AND the output byte with 0xf7 (11110111) to clear bit position 3
100                 break; //break enables the program to break out of switch block and stop execution - exiting loop
101
102         default: break; //default is an optional statement which specifies to break out of switch statement loop if no case is matched
103     }
104 }

```

```

101     output_a(output);           //sends output variable value to PORTA of slave PIC16F876 microcontroller
102 }
103
104 /***ep_write subroutine*****
105 void ep_write(char address,char data)
106 {
107     while(!input(pin_c5)); //waits for memory of chip to be available
108     output_low(pin_c5); //set Serial Data Out(SDO) pin low (memory busy)
109     i2c_start(); //sends i2c start condition pulse
110     i2c_write(0xA0); //sets device address to 0xA0
111     i2c_write(address); //send the device address to write to
112     i2c_write(data); //send the data to write to device address
113     delay_ms(delay); //wait for 20ms
114     i2c_stop(); //sends i2c stop condition pulse
115     delay_ms(delay); //wait for 20ms - gives time to allow i2c chip to store data to 0xA0 memory address
116     input(pin_c5); //releases Serial Data Out(SDO) pin RCS
117 }
118
119 /***ep_read subroutine*****
120 char ep_read(char address)
121 {
122     char data=0; //sets data to equal 0 and be a character data type
123     while(!input(pin_c5)); //waits for memory of chip to be available
124     output_low(pin_c5); //set Serial Data Out(SDO) pin low (memory busy)
125     i2c_start(); //sends i2c start condition pulse
126     i2c_write(0xA0); //sets device address to 0xA0
127     i2c_write(address); //sends the device address to be read from
128     delay_ms(delay); //wait for 20ms - provides time for chip to set its for reading memory
129     i2c_start(); //sends second i2c start condition pulse
130     delay_ms(delay); //wait for 20ms - provides time for chip to be ready
131     i2c_write(0xA1); //sets device address to 0xA1
132     data=i2c_read(0); //reads data from the chip - value '0' means do not acknowledge
133     delay_ms(delay); //wait for 20ms
134     i2c_stop(); //sends i2c stop condition pulse
135     delay_ms(delay); //wait for 20ms
136     input(pin_c5); //releases Serial Data Out(SDO) pin RCS
137     return(data); //when ep_read subroutine is called and executed this returns the data as a character
138 }
139
140 /***indicator_flash subroutine*****
141 void indicator_flash(void)
142 {
143     if(left & flash)output_high(pin_a2); //sets pin RA2 high - turning Rear Left Indicator Lamp on
144     else output_low(pin_a2); //else sets pin RA2 low - turning Rear Left Indicator Lamp off
145 }

```

Figure 7 - slave.c Program Listing

slave.h Listing

Figure 8 shows the Embedded C programs header file 'slave.h' for the slave PIC16F876 component U2 and comments explaining its structure and operation.

```

1  /***slave header file*****
2  #include <16F876.h> //header file for PIC used in circuit 'PIC16F876'
3  #define ADC=10 //sets read_adc return size to 10 bits
4
5  #FUSES NOPUT //no power up timer
6  #FUSES BROWNOUT //reset when brownout detected
7  #FUSES LVP //Low voltage programming on B3(PIC16)
8  #FUSES NOCPD //no EE protection
9  #FUSES NOWRT //program memory not write protected
10 #FUSES NOPROTECT //code not protected from reading
11
12 #use delay(crystal=4MHz) //no internal oscillator - external clock speed set to 4MHz
13 #use rs232(baud=9600,parity=N,mit=pin_c6,rcv=pin_c7,bits=8,stream=PORT1,errors) //rs232 communication pins - pin RC6 is transmit(xmit) - pin RC7 is receive
14 #use i2c(Master,Fast,sda=pin_c4,scl=pin_c3,force_hw) //i2c communication pins - pin RC3 is serial clock line (SCL) - pin RC4 is serial data line (SDA)
15
16 #define delay 20 //delay variable set to a value of 20ms - therefore "delay_ms(delay);" would be 20ms

```

Figure 8 - slave.h Program Listing

Operation of Program

master.c Program Flowchart

Figure 9 shows a flowchart describing the operation and logic for the C program ‘master.c’. [18]

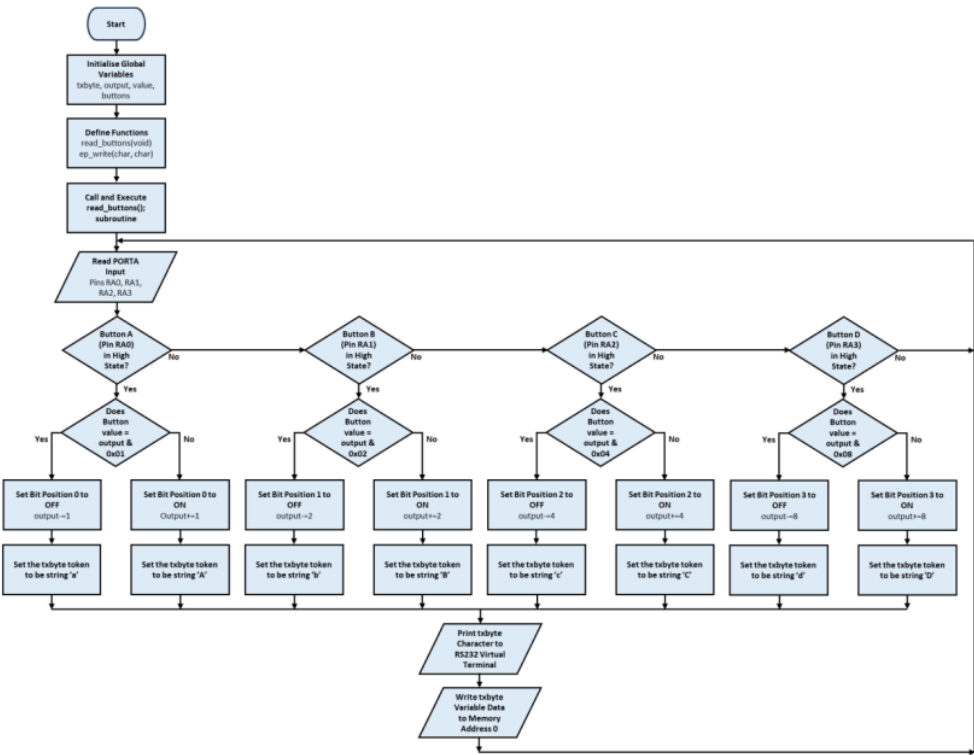


Figure 9 - Flowchart for master.c Program

slave.c Program Flowchart

Figure 10 shows a flowchart describing the operation and logic for the C program 'slave.c'. [18]

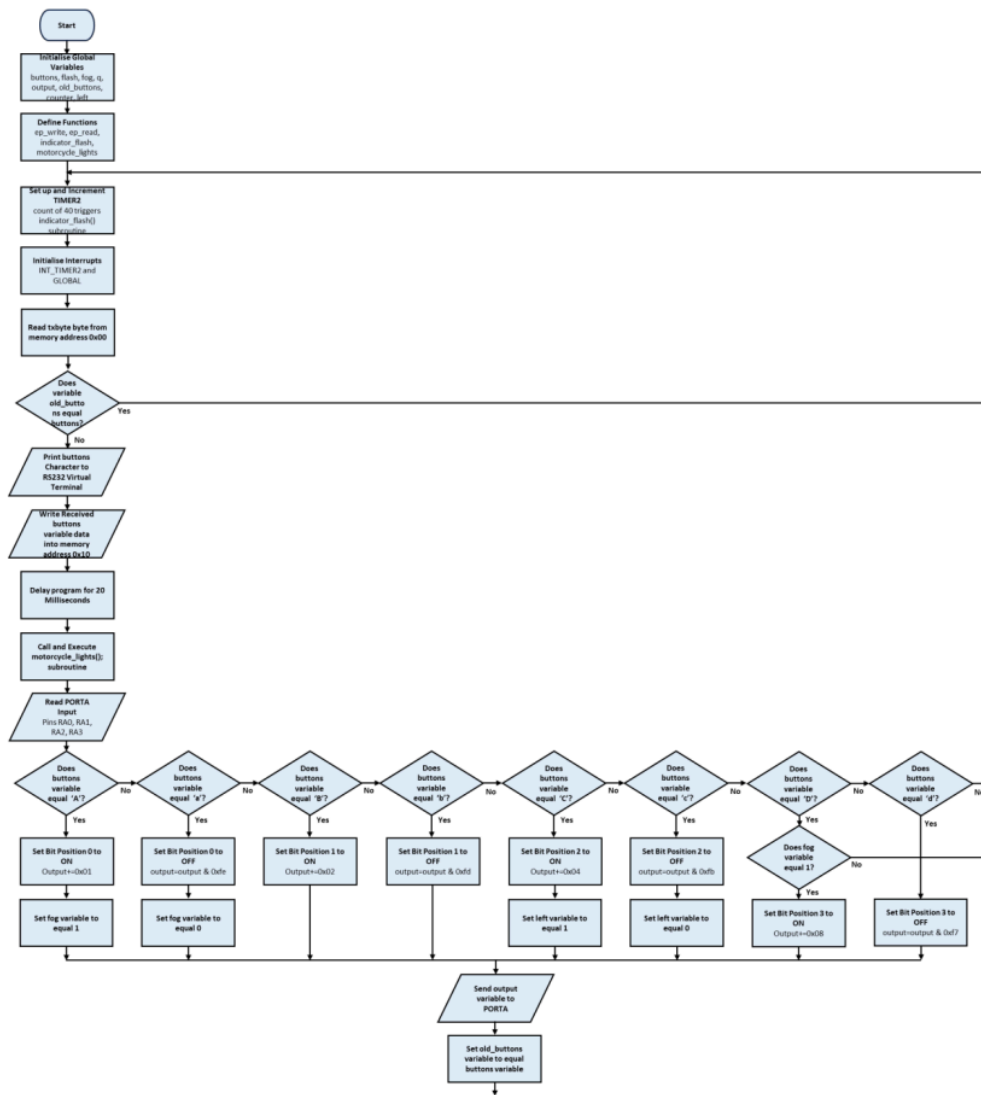


Figure 10 - Flowchart for slave.c Program

Testing & Verification

The process of debugging the program used the VSM Studio terminal to observe the errors output by the CCS C compiler. Figure 11 shows that after several iterations of software development, source code files 'master.c' and 'slave.c' built successfully returning zero error messages.

```

VSM Studio Output
ccsc.exe -J +DF +ES -P -Z +O8 +CC +EXPORT +FM "master.c"
ccsc.exe -J +DF +ES -P -Z +O8 +CC +FM LINK="Debug.cof=master.o"

0 Errors, 0 Warnings.
Build Successful.

C:\Users\alist\AppData\Local\Temp\VSM Studio\26648b51b42c45d19f3e691f8319b2b2\PIC16F876_1\master.o ==> 0 Errors, 0 Warnings
Build Successful.
Compiled successfully.

VSM Studio Output
ccsc.exe -J +DF +ES -P -Z +O8 +CC +EXPORT +FM "slave.c"
ccsc.exe -J +DF +ES -P -Z +O8 +CC +FM LINK="Debug.cof=slave.o"

0 Errors, 0 Warnings.
Build Successful.

C:\Users\alist\AppData\Local\Temp\VSM Studio\26648b51b42c45d19f3e691f8319b2b2\PIC16F876_2\slave.o ==> 0 Errors, 0 Warnings.
Build Successful.
Compiled successfully.

```

Figure 11 - VSM Studio Output - Compiling master.c and slave.c

After this development, I observed the response of the simple token based distributed network during simulation, to verify whether the program functions had been implemented correctly:

Rear Driving Lamp Test

Figure 12 shows that red LED 'D1' illuminates when button A is pressed, displaying an upper case 'A' on the both RS232 virtual terminals. When button A is pressed a second time, we can observe a lower case 'a' character on the RS232 terminals and the LED turning off. This demonstrates that the rear driving lamp is functioning as intended.

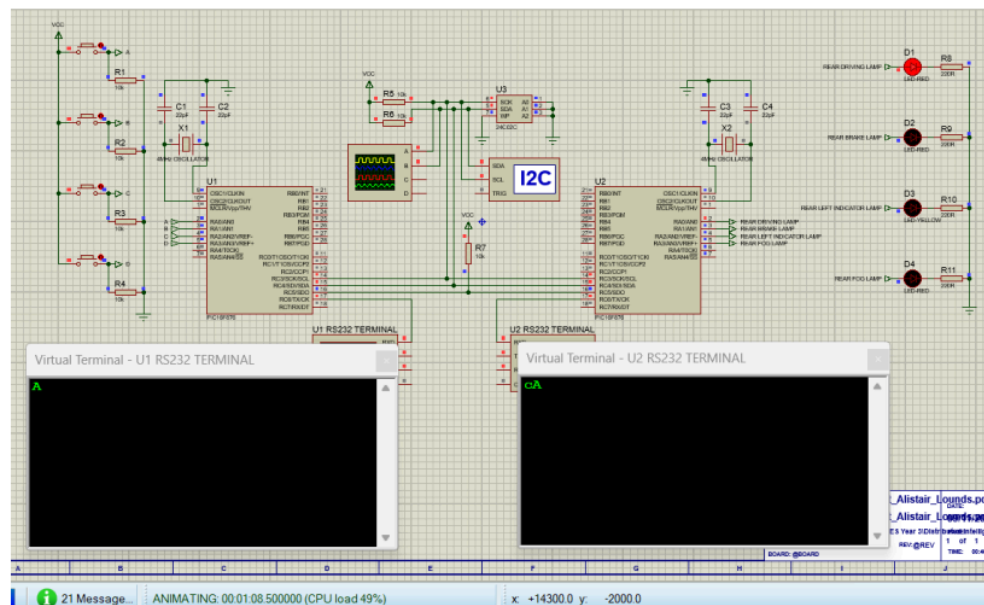


Figure 12 - Simple Token based Distributed Network - Rear Driving Lamp

Rear Brake Lamp Test

Figure 13 shows that red LED 'D2' illuminates when button B is pressed; displaying an upper case 'B' on the both RS232 virtual terminals. When button B is pressed a second time, we can observe a lower case 'b' character on the terminals and the LED turning off. This demonstrates that the rear brake lamp is functioning as intended.

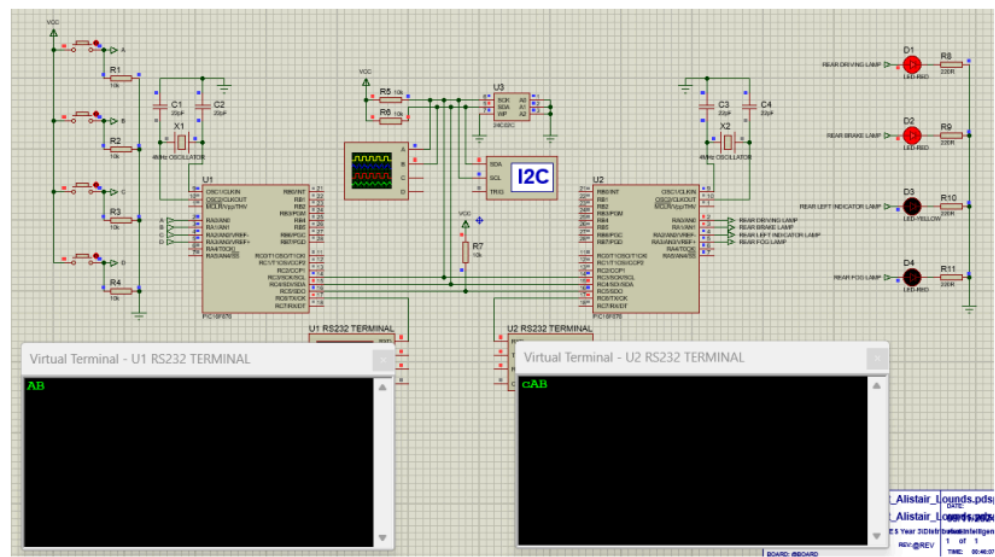


Figure 13 - Simple Token based Distributed Network - Rear Brake Lamp

Flashing Rear Left Indication Lamp Test

Figure 14 shows that yellow LED 'D3' illuminates when button C is pressed; displaying a upper case 'C' on the both RS232 virtual terminals. This lamp flashes on and off periodically, as determined by the TIMER2 subroutine in the slave.c program. When button C is pressed a second time, we can observe a lower case 'c' character on the RS232 terminals and the LED turning off. This demonstrates that the rear left indicator lamp is functioning as intended.

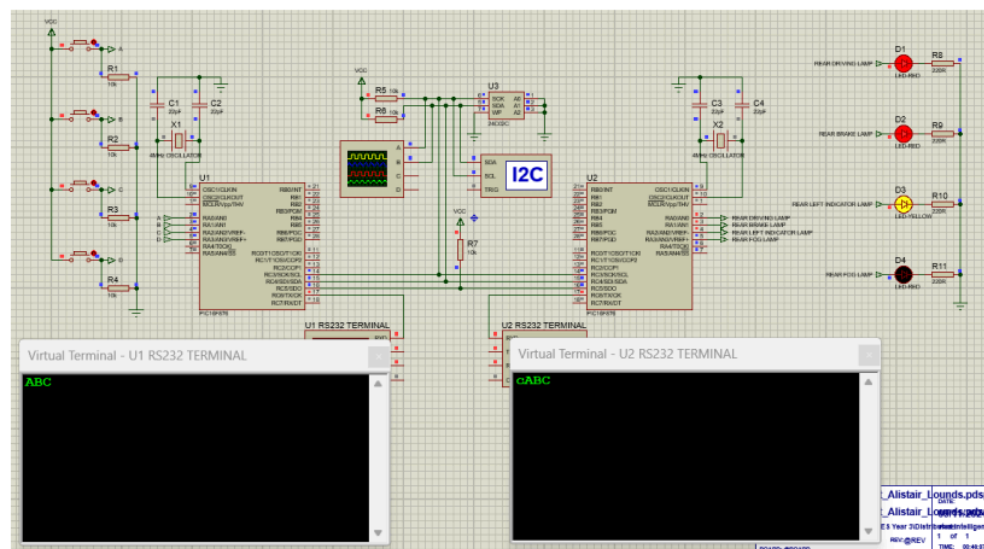


Figure 14 - Simple Token based Distributed Network - Flashing Rear Left Indicator Lamp

Rear Fog Lamp Test

Figure 15 shows that red LED 'D4' illuminates when button D is pressed; displaying a upper case 'D' on the both RS232 virtual terminals. When button D is pressed a second time, we can observe a lower case 'd' character on the terminals and the LED turning off. The rear fog light does not turn on unless the rear driving lamp is also in an 'On' state. This demonstrates that the rear fog lamp is functioning as intended.

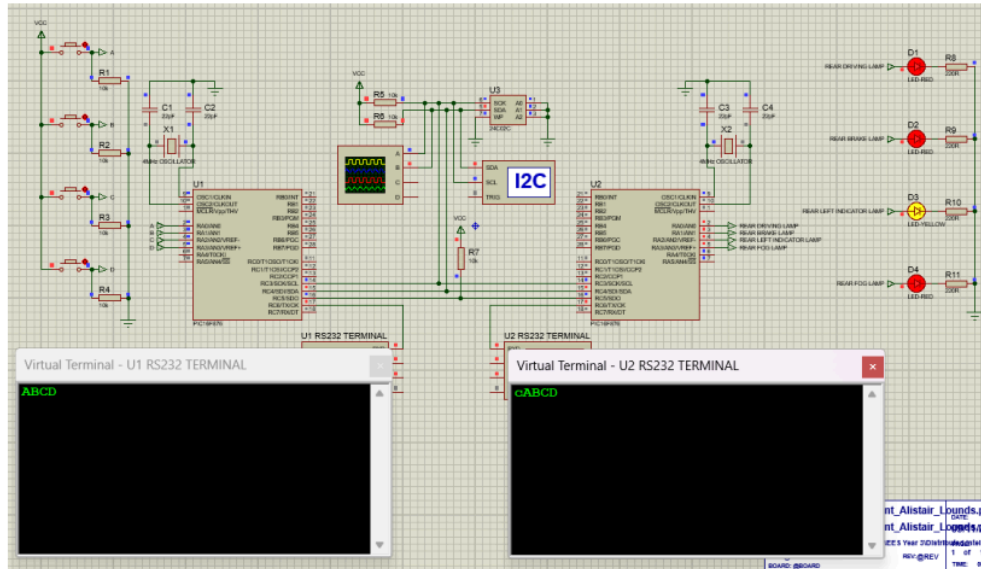


Figure 15 - Simple Token based Distributed Network - Rear Fog Lamp

Error Mitigation

Figure 16 shows the "[I2CMEM] Stop condition whilst memory is transmitting data is unreliable" error message found during the development of this program. This error message relates to component U3, being the '24C02C' I2C Serial Electrically Erasable Programmable Read-Only Memory (EEPROM) integrated circuit.

This was caused due to the lack of an optional acknowledge (ack) parameter within the parenthesis of the 'data = i2c_read();' function. This instruction reads a byte of data from the integrated circuit using the I2C interface and stores the byte as the 'data' variable. When using this syntax, if the parenthesis are left blank the default value is set to 1. A value of '1' indicates to send an acknowledgement message when a byte of data has been read. [16]

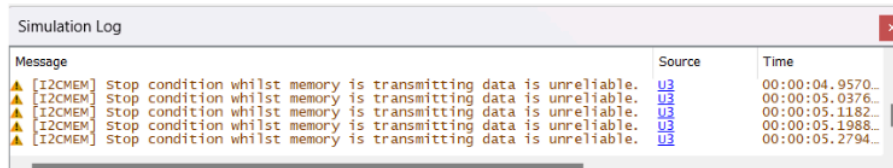


Figure 16 - I2C Serial EEPROM Memory Error

By inserting a value of '0' within the parenthesis on line 132 of the slave.c program, we are instructing the program to 'not acknowledge' when a byte of data has been read. This stops the program from looking for a stop condition response for the I2C memory, therefore preventing the serial EEPROM memory error message. [16]

Conclusion

In conclusion, it has been demonstrated that PIC16F876 microcontrollers can be programmed using the C programming language to produce a simple token based distributed network to simulating the operation of motorcycle lamps in an automotive application.

In hindsight the use of a PIC16F876 may not be the optimal solution for this automotive application as this PIC does not have an internal oscillator. Due to this I had to add two 22pF capacitors and an external quartz crystal oscillators to both the master and slave chips. This introduces increased cost of manufacture, as components need to be incorporated into the circuit design, pick and placed during board assembly, and further quality assurance to verify whether appropriate standards of manufacture have been met. This added complexity also present more potential failure points for system design.

To further improve the distributed network, I could reduce complexity in system design by opting for a PIC16/PIC18 microcontroller chip with an internal oscillator. For instance, a more suitable alternative may be to use the 'Microchip Technologies PIC16F887-I/P'. Unlike the microcontroller selected for my simple token based distributed network, this chip is delivered as a 'commercial off the shelf' (COTS) component containing two internal oscillators, one being a high frequency oscillator with a default 8MHz clock speed. [19]

This PIC16F887-I/P also may be more cost efficient in this automotive industry application; with lower unit cost having significant cost savings should the vehicle be manufactured in large volumes. Comparatively the PIC16F887-I/P costs £2.97 per unit, in comparison to PIC16F876-I/SP costing £5.81. [20][21]

References

- [1] Labcenter Electronics, "Proteus 8.16", North Yorkshire: England, Mar. 2023, [Online] Available: <https://www.labcenter.com/whatsnew/8.16/> [accessed Nov 01. 2024]
- [2] Custom Computer Services Inc, "CCS Downloads", Wisconsin: United States, Feb. 2024, [Online] Available: <https://www.ccsinfo.com/downloads.php#manuals> [accessed Nov 01. 2024]
- [3] GeeksforGeeks, "Differences between Point-to-Point and Multi-point Communication", Uttar Pradesh: India, Feb. 2022, [Online] Available: <https://www.geeksforgeeks.org/differences-between-point-to-point-and-multi-point-communication/> [accessed Nov 08. 2024]
- [4] Dr. B Feng, Sanny Telecom Equipment Co. Ltd, Shenzhen University, "Point to Point Communication Comprehensive Guide", Guangdong Province: China, Apr. 2024, [Online] Available: <https://www.sannytelecom.com/point-to-point-communication-comprehensive-guide/> [accessed Nov 08. 2024]
- [5] Huawei Technologies Co. Ltd, "LTE Network Architecture" in "NetEngine AR V300R021 CLI-based Configuration Guide - Interface Management", Shenzhen Guangdong Province: China, Jul. 2023, [Online] Available: <https://support.huawei.com/enterprise/en/doc/EDOC1100203257/f110c2e7/lte-network-architecture> [accessed Nov 08. 2024]
- [6] Chicago Communications LLC, "Benefits of Point-to-Multipoint Connectivity", Illinois: United States, Aug. 2024, [Online] Available: <https://www.chicomm.com/blog/the-benefits-of-point-to-multipoint-connectivity-in-communications> [accessed Nov 08. 2024]
- [7] RS Components Ltd, "What's the Role of PLCs in Automation?", Corby: England, Apr. 2024, [Online] Available: https://uk.rs-online.com/web/content/discovery/ideas-and-advice/plc-automation-guide?srsId=AfmBOOpMGd_x2AH5tar7mFdlemZzTOSMZVj7iTJcHtrBHR-bskDz6_hq [accessed Nov 08. 2024]
- [8] Stormatics PTE. Ltd, "What is Multi-Master architecture?" in "Understanding Multi-Master architecture in PostgreSQL: Benefits and Trade-offs", Singapore, May. 2024, [Online] Available: <https://stormatics.tech/blogs/understanding-multi-master-architecture-in-postgresql-benefits-and-trade-offs> [accessed Nov 08. 2024]

- [9] GeeksforGeeks, "Point-to-Point Communication in Distributed", Uttar Pradesh: India, Jun. 2024, [Online] Available: <https://www.geeksforgeeks.org/point-to-point-communication-in-distributed-systems/> [accessed Nov 08. 2024]
- [10] Renesas Electronics America Inc, "Using CAN Bus Serial Communications in Space Flight Applications", California: United States, 2018, [Online] Available: <https://www.renesas.com/en/document/whp/using-can-bus-serial-communications-space-flight-applications> [accessed Nov 08. 2024]
- [11] Engineering Council, "Quality Policy", London: England, 2023, [Online] Available: <https://www.engc.org.uk/quality/quality-policy/> [accessed Nov 11. 2024]
- [12] American Society for Quality, "What Is a Quality Management System (QMS)?", 2024, Wisconsin: United States, [Online] Available: https://asq.org/quality-resources/quality-management-system?srsId=AfmBOooB_pRz0jqSdzW-WWUq5bQj5_fg9EaExqQKh204bGb1IU4bl82 [accessed Nov 11. 2024]
- [13] International Standards Organisation, "ISO 45001:2018 - Occupational health and safety management systems — Requirements with guidance for use", Geneva: Switzerland, Mar. 2018, [Online] Available: <https://www.iso.org/standard/63787.html> [accessed Nov 11. 2024]
- [14] British Standards Institutes, "BS EN ISO 9001:2015 – TC - Quality management systems", Milton Keynes: England, Sep. 2015, [Online] Available: <https://knowledge.bsigroup.com/products/quality-management-systems-requirements-1?version=tracked> [accessed Nov 11. 2024]
- [15] Microchip Technology Inc, "PIC16F87X - 28/40-Pin 8-Bit CMOS FLASH Microcontrollers", Arizona: United States, Nov. 2012, [Online] Available: <https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/30292D.pdf> [accessed Nov 09. 2024]
- [16] Custom Computer Services Inc, "CCS C Compiler Manual", Wisconsin: United States, Nov. 2021, [Online] Available: https://www.ccsinfo.com/downloads/ccs_c_manual.pdf [accessed Nov 06. 2024] pp. 183, pp.323-324
- [17] Microchip Technology Inc, "I2C Master Mode - Overview and Use of the PICmicro MSSP I2C Interface with a 24xx01x EEPROM", Arizona: United States, 2001, [Online] Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/i2c.pdf> [accessed Nov 09. 2024]
- [18] Ir. Sofie Beerens, European Commission, "5.1 Flowchart" in "C for Embedded Systems", Brussels: Belgium, [Online] Available: https://ec.europa.eu/programmes/erasmus-plus/project-result-content/693a717c-12aa-4849-8ed1-2ef31999b4b8/Desire_C_for_embedded_Sofie_Beerens.pdf [accessed Nov 07. 2024] pp.55
- [19] Microchip Technology Inc, "4.5 Internal Clock Modes" in "PIC16F882/883/884/886/887 - 28/40/44-Pin Flash-Based, 8-Bit CMOS Microcontrollers", Arizona: United States, Jan. 2015, [Online] Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/40001291H.pdf> [accessed Nov 09. 2024] pp.67
- [20] Digikey Electronics, "PIC16F876A-I/SP", Minnesota: United States, Nov. 2024, [Online] Available: <https://www.digikey.co.uk/en/products/detail/microchip-technology/PIC16F876A-I-SP/446140> [accessed Nov 09. 2024]
- [21] Digikey Electronics, "PIC16F876A-I/SP-ND", Minnesota: United States, Nov. 2024, [Online] Available: <https://www.digikey.co.uk/en/products/detail/microchip-technology/PIC16F887-I-P/1015608> [accessed Nov 09. 2024]

DIS Laboratory Report - Alistair Lounds - 2227990

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

72/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

GRADING FORM: SMU GENERIC_GJ

ALISTAIR LOUNDS**71.90****RESEARCH-INFORMED LITERATURE**

You have explained what point to point and point to multipoint and multimaster are correctly. 68%

13.6**KNOWLEDGE AND UNDERSTANDING**

The design of your circuit is suitable for the task 75%

18.75**ANALYSIS**

You have completed the programs for both devices and the circuit works well. 70%

17.5**PRACTICAL APPLICATION AND DEP.**

Your explanation is good and acknowledges that both pics speak to memory IC 72%

10.8

SKILLS FOR PROFESSIONAL PRAC.



The report is well presented with sections to explain the code and include a reasonable number of references. 75%

11.25