

Cerys Versey - ACDA5018 - DIS assignment

by Cerys Versey

Submission date: 11-Nov-2024 12:24PM (UTC+0000)

Submission ID: 242296857

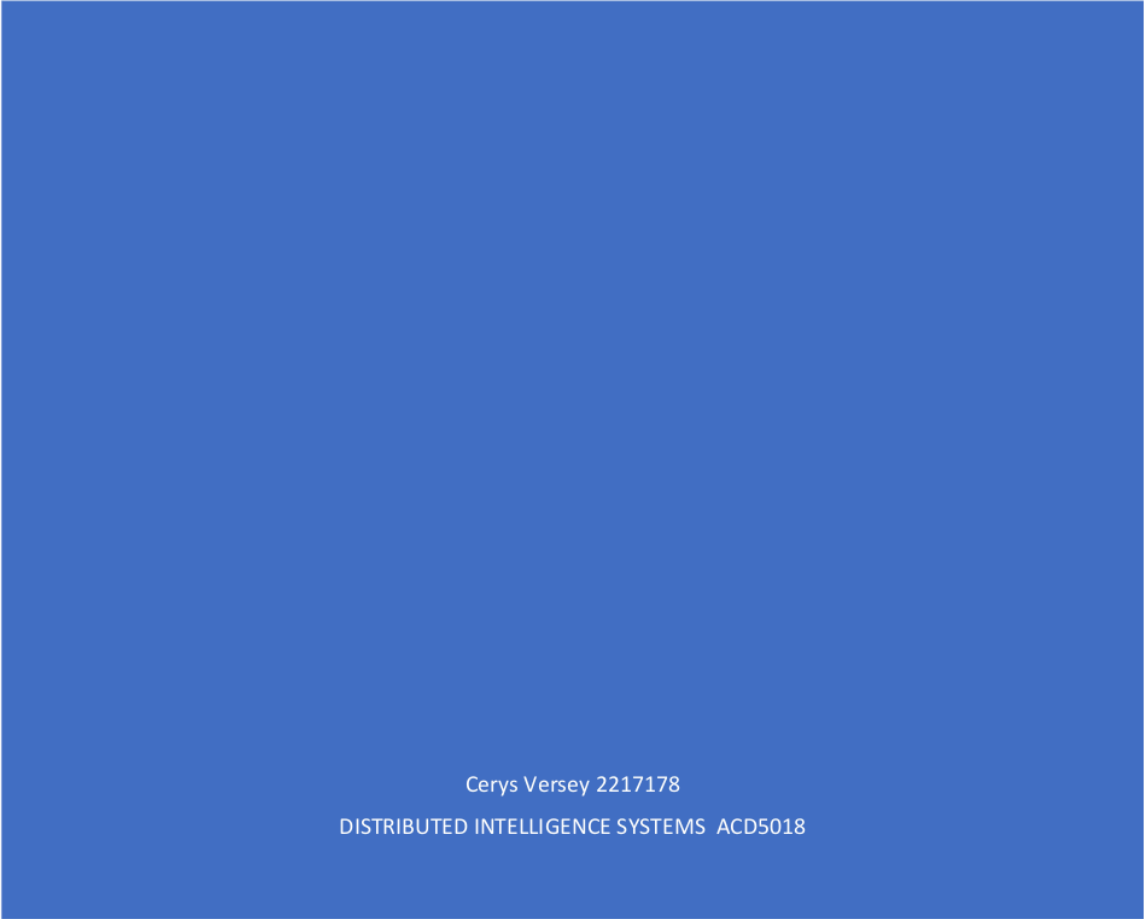
File name: 42325_Cerys_Versey_Cerys_Versey_-_ACDA5018_-_DIS_assignment_523660_1585641237.docx
(1.45M)

Word count: 3342

Character count: 16167



A SIMPLE TOKEN BASED DISTRIBUTED NETWORK



Cerys Versey 2217178
DISTRIBUTED INTELLIGENCE SYSTEMS ACD5018

Contents

- Abstract
- Introduction
- Research
- Aims
- Development
- Problems
- Outcome
- Conclusion

Abstract

The following report should give the reader a comprehensive understanding of single point, point to point and multi-master transmission systems. As well as a build guide and review of a simple token-based system that focuses on the front left of a car. The aim is to show how one chip can be used to instruct and another chip can be used to execute instructions. The whole system is done using I2C.

Introduction

Transmission systems come in many formats from the simplest point to point system to more complex multi-master transmission systems. They are used for everything from simply passing messages to complex network infrastructure like the once used for satellite communications. This report aims to use a practical example of a car's front left lights to explain to the reader the key parts of transmission systems as well as the underpinning protocols used. I2c or inter- integrated circuit protocol was chosen for this example but RS323 could also be used for even further simplicity if required.

I2C was designed in the 1980's to connect low speed devices such as microcontrollers over short distances [1][2]. It uses two wires the Serial Clock (SCL) and serial data line (SDA), to connect between devices. I2C can connect a number of peripheral (slave) devices to the controller (Master) devices[3]. These key attributes make it a good choice for chip to chip transmission of data. However, to avoid errors handshaking is required [4]. Handshaking is a way to control input and outputs of microcontrollers in this case to make sure they don't output at the same time [5].

Research (Task 1)

Another key area to understand is transmission systems and the different forms that they can take and the applications they have.

The simplest transmission system is a point to point system. This is simply where information is passed directly from one place to another and back again this can be thought of as a cup and string system. Information can only go to once place down the string the cup and is only connected to one place the other cup. There are many examples of where these types of transmission can be used including a walkie talkie set, two antennas across a room, a telephone between two people, or even an ethernet connecting two computers [6][7]. Point to point transmission can be used not only over a large area but is considered highly reliable due to its simplicity [8].

The next step up is a point to multipoint transmission system this is where a single point is connected to many other devices, this is usually thought of as a single master and multiple slave

system. An example of this is a laptop connecting to a number of peripheral devices like a printer, Bluetooth headset, monitor etc [9]. Another example of this is a star topology where multiple devices are connected to a central switch creating a look of a star [10]. There are many applications such as video surveillance where a base station is used to connect to multiple cameras for a central display, or used by a radio station to allow people to tune in or an ethernet hub (router) that can connect to multiple devices.

The most complex option is a multi-master system this is where there are multiple masters (controller) devices connect to multiple slave (peripheral) devices no one peripheral device has a designated controller. An example of this can be seen in microcontrollers where you have multiple devices controlling the same nodes for different uses.

Quality management

Quality management systems are defined by the Engineering Council as “activities that identify objectives, processes and resources required to achieve the desired results” [13]. As well as “providing the means to identify actions to address intended and unintended consequences in providing products and services” [13]. In practice this encompasses requirement capture processes, resource allocation, management and commercial expectations as well as intended and unintended consequences and their health and safety ramifications. Helping to underpin health and safety procedures as well as delivery.

QMS are crucial for enabling delivery that meets requirements not just on the surface but throughout the delivery process, it helps introduce SOP's (standard operating procedures) for safer delivery. As well helping to manage expectations of all participants throughout the management and delivery chains. This initial work load sets the back bone for reliable and consistent delivery showing how crucial they are not only in the design stage but the implementation and delivery stage.

QMS help to stimulate further development and continuous improvement by using a set of recurring activities that react to changes internally and externally to create new opportunities and maintain the projects impact [14].

Due to QMS inward and outward look, internal and external factors are allowed to weigh in on decisions allowing for updates in health and safety policy before an accident occurs; allowing for greater transparency. The tasks in QMS call for constant review and refinement of problems, allowing health and safety concerns that arise to be dealt with in a short period allowing for swift decisive action reducing the long-term impact. This helps not only reduce the risk of accidents but also gives confidence to the engineers involved.

Quality management systems are useful for both the customer and the engineer as it creates a culture of review and refinement not only improving the quality of the product but the way in which it is delivered.

Task 2

The following hopes to be a full guide to creation and evaluation of a simple token-based system. The system is used to replicate the functions of the front left of a car. The system includes a headlight, side lights, left indicator, Hazard, full beam, fog light and horn.

Before any code could be created a schematic design had to be created. This was done using proteus an electronic CAD software. The schematic includes a set of buttons that were used to represent the

controls from inside a car, such as the hazard lights, horn, indicators etc. The schematic also includes a number of LED's as well as a buzzer to show the outputs from the system. As well as key components for enabling I2C to work within proteus, the most important component is the two PIC18F2525 chips. These are the components that allow the passing of bytes, as well as receiving the output from the buttons and sending out the relevant signal for the LED's. To make the system as accurate as possible each Led colour matches the colour that light would be in real life. Therefore, the Hazzard light are yellow along with the side lights and headlights. Whereas the full beam and fog lights are white lights.

The PIC18F2525 was chosen for its simplicity as well as its large number of ports allowing for future expansion and increased functionality. It's also readily available for purchase making it easy to turn the schematic into a real piece of hardware[11].

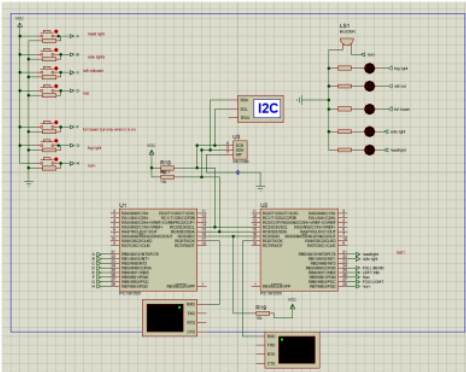


Figure 1 - Schematic of circuit

Once the schematic was made it was on to the flowchart to design the way in which the code would execute the various I/O required. The below flow chart shows the way in which button press' affect the system. If the button is pressed once it will turn the LED associated on and if it is twice will turn it off. It is also worth noting the use of variables to create if statements later on in other case conditions. This can be seen in the button F condition that until the side light (button C) have been pressed once the full beam won't come on, this is used to mimic real car function, i.e. the side lights have to be on to turn on the full beam. To help plan for the way the light functions would work a spread sheet was put together with all the conditions for lights being on. The key points to note are that both C and D turn on the left indicators, C is the left indicator controls and D is the hazard controls if this was a full light system then it would turn on both left and right indicators but due to it only being the front left you only see the left light.

	A	B	C	D	F	G	H
HEAD LI							
SIDE LI							
LEFT IND							
FULL BEAM							
FOG LI							
HORN							

Figure 2 - excel document

Once the excel document was finished it was much easier to see the conditions that were needed for lights being on and off. This was then turned into a flowchart.

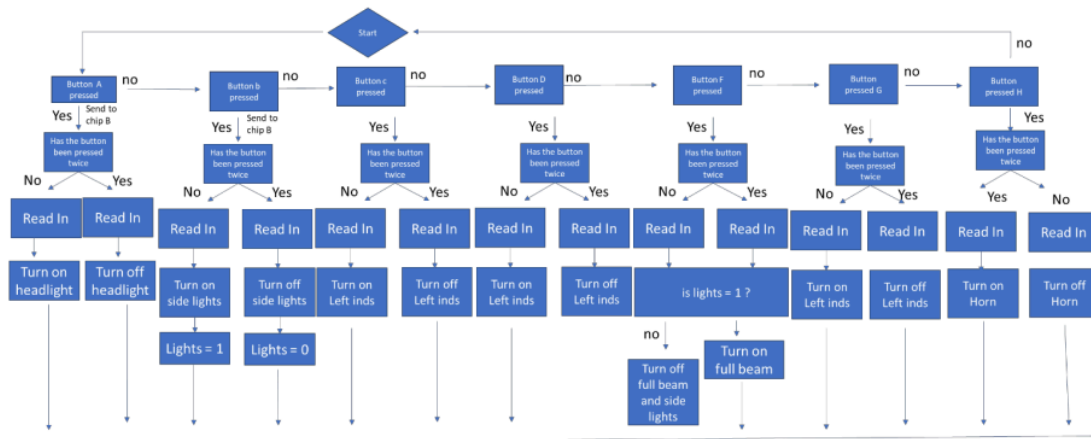


Figure 3- flowchart

Once the flowchart had been put together the code was created. Initially this was done by setting up the master and slave Main.C files to make it clear where the data would be sent too and from.

```
#include <main.h>
// slave
```

Figure 4 - main . c files

```
#include <main.h>
// master
```

Figure 5- main .c files

Once the pages were organised both pages needed an area to create global subroutines and variables. A global variable is a variable that can be used anywhere within that script rather than confined to a loop or a subroutine. Within this block of code there is a variety of different variable types such as char which is short for character this can include any letters, numbers or even symbols. Whereas an unassigned integer can only be an integer value. Once the variables had been set up it was on to the creation of the cases.

```
// variable and subroutine creation and assignment ////////////////////////////////////////////
void ep_w(char,char);           // creating a subroutine - which outputs two characters in an array (x,y) (where,what)
char read_buttons(void);        // creating a variable to store values later
char ep_r(char );              // creating a variable to store values for later catagorised as a character

char local_output;              // creating a variable to store values later
char txbyte;                    // creating a variable to store values later
char q=0;                       // creating a variable and assigning it an initial start value
unsigned int value=0x00;        // creating a variable and assigning it a start value
```

Figure 6- Master variables

```

//variable and subroutine creation and assignment //////////////////////////////////
char local_output=0; // assigning a character variable an initial value
char control_byte=0; // setting up an initial value to use later for if statements i.e. if lights is on do blank
unsigned int lights=0; // setting up an initial value to use later for if statements i.e. if counter is on do blank
unsigned int counter=0; // setting up an initial value to use later for a while loop for the flash statement
unsigned int left=0; // setting up an initial value to use later for a if statement for the left flash condition
// if you wanted to implement a right indicator you would need to set us a variable for that too here
CHAR buts=0; // creating the variable buts (buttons) to use later for switch conditions
char old_but=0; // creating a value to store the old button chosen for use inn switch conditions

char ep_r(char); // ep_r or eeprom read is the variable that will later used to read data in and out from a specific location
void ep_w(char where,char what); // ep_w or eeprom write is the variable that is used to write characters into specifcll memory locations on the chip (where it
void flash_control(void); // creating a subroutine for the flashing instructions for the indicators and hazard lights to be stored
void sc(void); // sc or switch conditions is the subroutine that holds the switch condions and there operations

```

Figure 7 - slave variables

Cases

Cases are used to create instruction sets with conditions attached much like attaching an if statement to a subroutine. As can be seen the case has an if statement attached to it; this is the condition on which the following lines of code will be executed. In this case the slave sets up the local output values as well as sends the relevant character to show that that case has been selected.

To make sure that the button doesn't accidentally get turned on again if it is already on the off condition comes first and if that condition isn't met it then turns on the LED making sure it only goes on, once the button has been pressed.

```

case 1: if(local_output & 0x01)
{
    local_output-=1;
    txbyte='@';
}
else
{
    local_output+=1;
    txbyte='A';
}
break;

```

Figure 8 - case example

Once the master case conditions were set up it was important to setup the slave (peripheral) cases.

```

////////////////////////////////////
case 'A': control_byte+=0x01; // head light - outbut 0x01 which will turn my healight l
break; // breaks out of that case

case 'a': control_byte=control_byte & 0xfe; // AND the control byte with 0xfe to clear
break; // breaks out of that case

```

Figure 9- Case examples

As you see now the cases are no longer 1,2,4,8, etc to batch the binary values of the buttons being turned on but are no A, a, b, B, C, c etc to map to the character that the master sends telling the slave chip which case to act upon. As can be seen from case A the cases are rather simplistic simply turning on or off a light this is done by imaging port B of the slave as a binary list. To then turn it of you simply need to do some Hex maths as can be seen below.



Figure 10 - apply binary values to the port for light control

Hex value: ff – 01 = FE

Figure 11- example of hex maths

H and G run a similar simple process to A by simply using control byte for turning lights on and off. However B,C,D,F have more complex code to reflect the increase of complexity and functionality required.

B (side lights) has a lights variable attached to be able to be used as an if condition later on in case F. This means that the main beam will only come on if the side lights are on.

```
case 'B':  
    control_byte=control_byte+=0x06;  
    lights=1;  
    break;  
  
case 'b':  
    control_byte=control_byte & 0xf1;  
    lights=0;  
    break;  
  
case 'F':  
    if(lights)  
    {  
        control_byte=control_byte+=0x08;  
    }  
    break;  
  
case 'f':  
    control_byte=control_byte & 0xf7;  
    break;
```

Figure 12 - light conditions

Case C and D are for the left indicator and Hazzard buttons respectively. As discussed before they both turn on the same light which means the code is identical.

```
case 'C':  
    local_output+=0x10;  
    left=1;  
    break;  
  
case 'c':  
    local_output=local_output & 0xef;  
    left=0;  
    break;  
////////////////////////////////////  
case 'D':  
    local_output+=0x10;  
    left=1;  
    break;  
  
case 'd':  
    left=0;  
    break;  
  
void flash_control(void)  
{  
    if(left & flash)output_high(pin_b4);  
    else output_low(pin_b4);  
}
```

Figure 13 - hazard and Light indicator

As seen above the left variable is used for the flash control if the flash and left are both on then then B4 will go high and then low on repeat.

To facilitate the above code protocols using I2C for both reading in and out data had to be written.

I2C is a protocol that is used to move data simply and quickly from one device to another. It is a serial interface and is bi-directional. A serial interface is simply when data is transmitted as a single stream of bits [12]. Bi-directionality means that data can be sent both ways this is done by the acknowledge feature this allows for the chips to acknowledge that they have received the data, when the data is sent is controlled by the master to make sure that data isn't already being sent elsewhere in the system. The master also controls when it is read. In conclusion I2C mainly consists of two key principles the SDA – serial data bus and the SCL - serial clock. The serial data line transfers data to and from the master whereas the Serial clock controls when data is sent and when it is read and is controlled by the master[12].

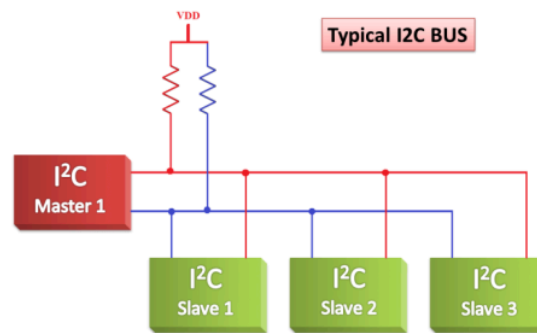


Figure 14 - I2C Flow Diagram [A]

A subroutine that controls the switching between the switch cases was also needed to confirm if any of the switch conditions were met and needed to turn on certain outputs. To write to a specific location on the chip it not only needs to know where it is writing too but also what is being stored there, this is why in the same line as the subroutine it states the two outputs, there is delays added in to allow time for the data to be written and saved in the desired location.

```

void ep_w(char where,char what)
{
    while(!input(hs));
    output_low(hs);
    i2c_start();
    i2c_write(0xA0);
    i2c_write(where);
    i2c_write(what);
    delay_ms(20);
    i2c_stop();
    delay_ms(20);
    input(hs);
}

```

Figure 15 - I2C write command

For the reading subroutine it is much simpler it simply requires the system to look for the value in the specified location and return the value present there.

```

char ep_r(char where)
{
    char data=0;
    while(!input(hs));
    output_low(hs);
    i2c_start();
    i2c_write(0xA0);
    i2c_write(where);
    delay_ms(20);
    i2c_start();
    delay_ms(20);
    i2c_write(0xA1);
    data=i2c_read();
    delay_ms(20);
    i2c_stop();
    delay_ms(20);
    input(hs);
    return(data);
}

```

Figure 16 - I2C character read

With the above strings of code, a fully working system was able to put together producing the below outputs in accordance with real car lighting.

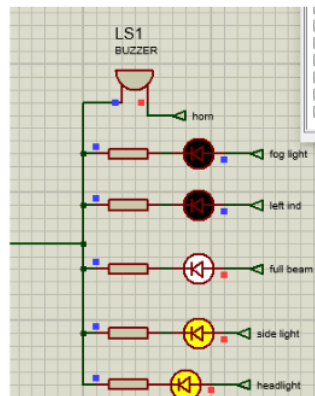


Figure 17 - outputs

For the switch cases it is simply, reads in the character value checks if the old button is equal to that of the last value if it is it keeps checking in a loop till it is different. Then once it is different prints that Character to the vertical terminal and then writes the value to a specific location and then runs the main script to get the script to action the input command.

```
while(true)                                // allows a
{
    buts=ep_r(0x00);
    if(old_but!=but)
    {
        printf("%c",but);
        ep_w(0x10,but);
        delay_ms(20);
        sc();
    }
    old_but=but;
}
```

Figure 18 - switch condition code

Review

During the build there was a number of problems from hardware to software. One such issue occurred when the schematic was made, due to the pins being moved round on numerous occasions to decide the order they were often written on top of old pins. However, on a few occasions it was forgotten to rename the LED's that matched. This meant that some LED's could never have been spoken too, luckily once problem solving began it was noticed that the light which shows the pin is on and off wasn't in fact there showing it wasn't connected to the input at all as can be seen below.

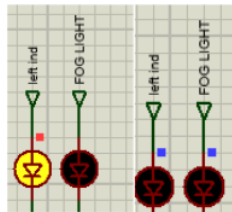


Figure 19 - led without a connected output pin

Figure 20 - Led with a connect output pin

Another issue that occurred during the development process was the difference between semi-colon placement as well as the importance of bracket placement. When a subroutine code is written it is placed between semi-colons to denote that it is attached to that subroutine so it can easily be called if needed. However, if you are not careful you can easily include values or aspects you didn't mean too. During the development an off and on condition was included within the same brackets when they should have been in two separate sets this meant that the light would never turn on because as soon as it was turned on it was instantly turned off. This shows just how crucial proper alignment in code is so it can clearly be seen what is associated with what.

Another key piece of syntax that caused delays to the development process was the placement of semi colons especially when in a loop, if you have a conditional statement you can place one line of code after it and then can place a semi colon it will then only execute that one line and not the following lines under this condition, to avoid this it was opted to expand loops with a full set of brackets and then use semi-colons unless it was very clear to avoid confusion.

Another problem that occurred was the use of variables for condition statements, initially each case had that variable equal 0 even if it wasn't relevant for example, lights = 0;. This was done to make sure that a variable wasn't accidentally triggering values however this made the code bulky and difficult to problem solve so these were removed and only kept where they were direct necessarily.

If I was to take this design further I would firstly add in the other lights within the car to really show the impact that the hazard condition has on the other lights. I would also add in redundancy capability into the system. This would mean if one of the functions wasn't working that another function would do something different to its usual function. For instance, if the brake lights stopped working the horn would beep three times to audibly alert other drivers the car is slowing. Another example of this could be if the main beam failed having the side lights, head lights and break lights come on adding maximum illumination to the car. I would also add in dashboard lights to let the user know that a failure has occurred on their lights, which one has failed and which light have gone on instead, further to this I would additionally add in a start up sequence which flashes all the bulbs once to check they all work before sending a message to the dashboard to say they are all okay.

Conclusion

In conclusion the reader should now have a firm understanding of point to point, point to multi-point as well as multi-master transmission as well as understand the I2C protocol. The reader should also be aware of how to create a car light system as well of some of the problems that can occur and further functionality that can be added.

Appendix

[1] "I2C Protocol." *Embedded Wala*. Accessed: 04/11/2024. [Online]. Available: [I2C Protocol | Embedded Wala](#)

[2] "I2C." Sparkfun. Accessed: 04/11/2024. [Online]. Available: [I2C - SparkFun Learn](#)

[3] S.Camball. "Basics of the I2C Communication Protocol." *Circuit Basics*. Accessed: 06/11/2024. [Online]. Available: [Basics of the I2C Communication Protocol](#)

- [4] J.Wu. "A Basic Guide to I2C." Texas Instruments. Accessed: 06/11/2024. [Online]. Available: [A Basic Guide to I2C](#)
- [5] "What is Handshaking?." Tutorials point . Accessed: 06/11/2024. [Online]. Available: [What is Handshaking?](#)
- [6] "Title of web page." Everything you need to know about point-to-point networking. Accessed: 06/11/24. [Online]. Available: [Everything you need to know about point-to-point networking](#)
- [7] A. Froehlich. "Point-to-Point Protocol (PPP)." Tech Target . Accessed: 06/11/2024. [Online]. Available: [What is PPP \(Point-to-Point Protocol\) and How Does it Work?](#)
- [8] B. Feng . "Point to Point Communication Comprehensive Guide." Sanny Telecom. Accessed: 06/11/24. [Online]. Available: [Point to Point Communication Comprehensive Guide - Sanny Telecom](#)
- [9] M.Rouse. "Point-to-Multipoint Communication" Techopedia. Accessed: 06/11/24. [Online]. Available: [What is Point-to-Multipoint Communication \(PMP\)? - Definition from Techopedia](#)
- [10] "Star Topology : Working, Features, Diagram, Fault detection & Its Applications." Elprocus. Accessed: 06/11/2024. [Online]. Available: [Star Topology : Working, Differences, Advantages & Its Applications](#)
- [11] "PIC18F2525-I/SP." Farnell. Accessed: 06/11/2024. [Online]. Available: [PIC18F2525-I/SP MICROCHIP, 8 Bit MCU, Flash, PIC18 Family PIC18F25xx Series Microcontrollers, PIC18, 40 MHz, 48 KB, 28 Pins | Farnell UK](#)
- [12] M. Rouse. "Serial Interface." Techopedia. Accessed: 07/11/24. [Online]. Available: [What is a Serial Interface? - Definition from Techopedia](#)
- [13] "Quality Policy ISO 9001:2015 Clause 5.2 and Clause 6.2." Engineering Council. Accessed: 11/11/24. [Online]. Available: [Engineering Council](#)

[14] "Quality vs. Safety: Understanding the Difference and How to Achieve Both." Qooling.
Accessed: 11/11/24. [Online]. Available: <https://blog.qooling.com/quality-vs-safety-understanding-the-difference-and-how-to-achieve-both/>

Code

Master code

```
1 #include <main.h>
2 // master
3 // variable and subroutine creation and assignment ///////////////////////////////////////////////////
4 void ep_w(char, char); // creating a subroutine - which outputs two characters in an array (x,y) (where, what)
5 char read_buttons(void); // creating a variable to store values later
6 char ep_r(char); // creating a variable to store values for later categorised as a character
7
8 char local_output; // creating a variable to store values later
9 char txbyte; // creating a variable to store values later
10 char q=0; // creating a variable and assigning it an initial start value
11 unsigned int value=0x00; // creating a variable and assigning it a start value
12
13 //main body of code ///////////////////////////////////////////////////
14 void main() // starting the subroutine which holds the main body of the code
15 {
16     char but=0; // Assigning initial button value
17     while(TRUE) // starting a while true this means this will run in an endless loop
18     {
19         but=read_buttons(); // get the value of but as a character
20         switch (but) // switches to the case selected when the button is pressed
21         {
22             case 1: if(local_output & 0x01) // case 1 of the switch conditions which will later be used to toggle
23                 // the brake lights on and off
24                 {
25                     local_output--; // set bit off
26                     txbyte='a'; // set the token as lower case a to show the button has been pressed twice for the off
27                 }
28             else // if the above condition doesn't apply do this
29             {
30                 local_output++; // set bit on
31                 txbyte='A'; // upper case token to be used to show the user which option they selected
32             }
33         }
34         break; // breaks the code out of that case
35     }
36     case 2: if(local_output & 0x02) // case 2 of the switch conditions which will later be used to toggle the side lights
37     {
38         local_output--; // set bit off / low
39         txbyte='b'; // lower case token to show the button has been pressed twice and therefore is off
40     }
41     else // if the above case condition doesn't apply do this
42     {
43         local_output++; // set local output high
44         txbyte='B'; // token is capital B to show which button has been pressed
45     }
46     break; // breaks the code out of that case
47     case 4: if(local_output & 0x04) // toggle the left indicator on and off
48     {
49         local_output--; // sets bit low / off
50         txbyte='c'; // token is shown as lower c
51     }
52     else // if the above case doesn't apply do this instead
53     {
54         local_output++; // sets bit high
55         txbyte='C'; // token upper case C is used to show which letter and which state
56     }
57     break; // breaks the code out of the case
58 }
59
60 // breaks the code out of the case
61 case 8: if(local_output & 0x08) // toggle the Hazard lights on and off
62 {
63     local_output--; // sets the bit low because it was left on it sets it low
64     txbyte='d'; // outputs a lower case d to show which state and which button was used
65 }
66 else // if the above case doesn't apply do this instead
67 {
68     local_output++; // turns the bit high
69     txbyte='D'; // output capital d to show which state and which button has been pressed
70 }
71 break; // breaks you out of that case condition
72 case 32: if(local_output & 0x20) // toggle the main/dip ie. main and dip is on
73 {
74     local_output--; // turns the bit low
75     txbyte='f'; // outputs a lower case f to show it is low
76 }
77 else // if the above condition doesn't apply then do this instead
78 {
79     local_output++; // turns the bit high
80     txbyte='F'; // outputs an upper case f to show it is on and which button was pressed
81 }
82 break; // breaks you out of that switch condition
83 case 64: if(local_output & 0x40) // toggle the fog lights on and off
84 {
85     local_output--; // turns the bit low
86     txbyte='g'; // tells the user what button and what status
87 }
88 }
```

```

134
135 char ep_r(char where) // name of subroutine and its expected output
136 {
137     char data=0;
138     while(input(hs)); // wait for mem chip to be available
139     output_low(hs); // set hs line low (mem busy)
140     i2c_start(); // issue i2c start pulse
141     i2c_write(0x40); // issue chip type and number
142     i2c_write(where); // send address to be read from
143     delay_ms(20); // wait for chip to set its for read
144     i2c_start(); // issue another start pulse
145     delay_ms(20); // wait for chip to be ready
146     i2c_write(0x41); // issue chip type and number read command
147     data=i2c_read(); // read data from the chip
148     delay_ms(20);
149     i2c_stop();
150     delay_ms(20);
151     input(hs);
152     return(data);
153 }
154
155
156
157 char read_buttons(void) // names the subroutine so it can be called later
158 {
159     char value=0;
160     while(input_b(0)); // setup a variable for the buttons
161     value=input_B(); // wait for button press
162     delay_ms(500); // get the value of port b
163     return (value); // time button release
164     // return the value of the buttons pressed
165 }
166

```

Slave code

```

1 //include <main.h>
2 // slave
3 //variable and subroutine creation and assignment //////////////////////////////////////
4 char local_output=0; // assigning a character variable an initial value
5 char control_byte=0; // assigning a character variable an initial value
6 unsigned int lights=0; // setting up an initial value to use later for if statements i.e. if lights is on do blank
7 unsigned int counter=0; // setting up an initial value to use later for if statements i.e. if counter is on do blank
8 unsigned int flash=0; // setting up an initial value to use later for a while loop for the flash statement
9 unsigned int left=0; // setting up an initial value to use later for a if statement for the left flash condition
10 // if you wanted to implement a right indicator you would need to set us a variable for that too here
11 #WIRBUTS=0; // creating the variable butts (buttons) to use later for switch conditions
12 char old_but=0; // creating a value to store the old button chosen for use in switch conditions
13
14 char ep_r(char); // ep_r or epram read is the variable that will later used to read data in and out from a specific location
15 void ep_w(char where, char what); // ep-w or epram write is the variable that is used to write characters into specific memory locations on the chip (where it is stored)
16 void flash_control(void); // creating a subroutine for the flashing instructions for the indicators and hazards lights to be stored
17 void sc(void); // sc or switch conditions is the subroutine that holds the switch conditions and there operations
18
19 #INT_TIMER2
20 void TIMER2_isr(void) // the timer subroutine
21 {
22     counter++; // adds one to the counter each run through
23     if(counter==40) // if the counter equals 40 then do the following counter is set initially above
24     {
25         counter=0; // sets the counter to zero
26         if(flash) flash=0; // if flash is equal to zero it sets flash to 0
27         else flash=1; // if flash is one its set as one
28         flash_control(); // then the flash control subroutine is called this controls when it is on or off
29     }
30 }
31
32
33 void main() // calling the subroutine which holds the main section of code
34 {
35     char q=0; // assigning the character q to 0
36     setup_timer_2(T2_DIV_BY_16,249,15); // 1.0 ms overflow, 15.0 ms interrupt
37     enable_interrupts(INT_TIMER2); // enables the interrupts to be able to be used later if needed
38     enable_interrupts(GLOBAL); // enable the interrupt for later use
39
40     while(true) // allows a series of actions to take place whilst a condition is true
41     {
42         buts=ep_r(0x00); // assigns the first read location to buttons
43         if(old_but!=but) // if buts doesn't equal the old condition do the following actions for other buttons to be used
44         {
45             printf("%c",but); // print the character of the button pressed A-H a-h
46             ep_w(0x10,but); // write the value recieved into the memory @ address 16 (0x10)
47             delay_ms(20); // wait 20ms
48             sc(); // run the switch condition subroutine
49
50             old_but=but; // copy the value of but to old_but (becomes the previous value) allowing for a change of state to
51
52         }
53     }
54
55     void sc(void) // carry out the payload operations.
56     {
57         switch(buts) // switch between the conditions when relevant
58         {
59             //////////////////////////////////////
60             case 'A': control_byte=0x01; // head light - outputs 0x01 which will turn my headlight led high
61                     break; // breaks out of that case
62
63             case 'a': control_byte=control_byte & 0xfe; // AND the control byte with 0xfe to clear and turn off the LED
64                     break; // breaks out of that case
65             //////////////////////////////////////
66             case 'B': control_byte=control_byte & 0x06; // turn on side light by outputting it to 0
67                     lights=1; // setting lights as 1 so it can be used as part of a condition
68                     break; // breaks out of the case
69             case 'b': control_byte=control_byte & 0xf1; // AND the local output byte with 0xfe to clear bits 1,2,3
70                     lights=0; // sets light to 0 so if lights are turned off the high beam on
71                     break;
72             //////////////////////////////////////
73             case 'C': local_output=0x10; // turn on left indicator
74                     left=1; // turns the variable left high to be used as a condition for the indicator
75                     break; // breaks the code out of that case so the code doesn't get stuck
76             case 'c': local_output=local_output & 0xef; // AND the local output byte with 0xfe to clear bit 4
77                     left=0; // turns left low so the indicator doesn't continue to flash
78                     break; // breaks the code out of that case
79             //////////////////////////////////////
80             case 'D': local_output=0x10; // turn on left indicator which are used for hazard lights
81                     left=1; // left is needed for the flash for the indicator
82                     break;
83             case 'd': left=0; // turning left off so it is reset for next use
84                     break;
85             //////////////////////////////////////
86             case 'E': left=1; // full hazard on
87             case 'e': left=0; // full hazard off
88             case 'F': left=1; // full hazard on
89             case 'f': left=0; // full hazard off
90             case 'G': left=1; // full hazard on
91             case 'g': left=0; // full hazard off
92             case 'H': left=1; // full hazard on
93             case 'h': left=0; // full hazard off
94             case 'I': left=1; // full hazard on
95             case 'i': left=0; // full hazard off
96             case 'J': left=1; // full hazard on
97             case 'j': left=0; // full hazard off
98             case 'K': left=1; // full hazard on
99             case 'k': left=0; // full hazard off
100            case 'L': left=1; // full hazard on
101            case 'l': left=0; // full hazard off
102            case 'M': left=1; // full hazard on
103            case 'm': left=0; // full hazard off
104            case 'N': left=1; // full hazard on
105            case 'n': left=0; // full hazard off
106            case 'O': left=1; // full hazard on
107            case 'o': left=0; // full hazard off
108            case 'P': left=1; // full hazard on
109            case 'p': left=0; // full hazard off
110            case 'Q': left=1; // full hazard on
111            case 'q': left=0; // full hazard off
112            case 'R': left=1; // full hazard on
113            case 'r': left=0; // full hazard off
114            case 'S': left=1; // full hazard on
115            case 's': left=0; // full hazard off
116            case 'T': left=1; // full hazard on
117            case 't': left=0; // full hazard off
118            case 'U': left=1; // full hazard on
119            case 'u': left=0; // full hazard off
120            case 'V': left=1; // full hazard on
121            case 'v': left=0; // full hazard off
122            case 'W': left=1; // full hazard on
123            case 'w': left=0; // full hazard off
124            case 'X': left=1; // full hazard on
125            case 'x': left=0; // full hazard off
126            case 'Y': left=1; // full hazard on
127            case 'y': left=0; // full hazard off
128            case 'Z': left=1; // full hazard on
129            case 'z': left=0; // full hazard off
130            case '[': left=1; // full hazard on
131            case ']': left=0; // full hazard off
132            case '^': left=1; // full hazard on
133            case '_': left=0; // full hazard off
134            case '`': left=1; // full hazard on
135            case '~': left=0; // full hazard off
136            case '{': left=1; // full hazard on
137            case '}': left=0; // full hazard off
138            case '|': left=1; // full hazard on
139            case '~': left=0; // full hazard off
140            case '~': left=1; // full hazard on
141            case '~': left=0; // full hazard off
142            case '~': left=1; // full hazard on
143            case '~': left=0; // full hazard off
144            case '~': left=1; // full hazard on
145            case '~': left=0; // full hazard off
146            case '~': left=1; // full hazard on
147            case '~': left=0; // full hazard off
148            case '~': left=1; // full hazard on
149            case '~': left=0; // full hazard off
150            case '~': left=1; // full hazard on
151            case '~': left=0; // full hazard off
152            case '~': left=1; // full hazard on
153            case '~': left=0; // full hazard off
154            case '~': left=1; // full hazard on
155            case '~': left=0; // full hazard off
156            case '~': left=1; // full hazard on
157            case '~': left=0; // full hazard off
158            case '~': left=1; // full hazard on
159            case '~': left=0; // full hazard off
160            case '~': left=1; // full hazard on
161            case '~': left=0; // full hazard off
162            case '~': left=1; // full hazard on
163            case '~': left=0; // full hazard off
164            case '~': left=1; // full hazard on
165            case '~': left=0; // full hazard off
166            case '~': left=1; // full hazard on
167            case '~': left=0; // full hazard off
168            case '~': left=1; // full hazard on
169            case '~': left=0; // full hazard off
170            case '~': left=1; // full hazard on
171            case '~': left=0; // full hazard off
172            case '~': left=1; // full hazard on
173            case '~': left=0; // full hazard off
174            case '~': left=1; // full hazard on
175            case '~': left=0; // full hazard off
176            case '~': left=1; // full hazard on
177            case '~': left=0; // full hazard off
178            case '~': left=1; // full hazard on
179            case '~': left=0; // full hazard off
180            case '~': left=1; // full hazard on
181            case '~': left=0; // full hazard off
182            case '~': left=1; // full hazard on
183            case '~': left=0; // full hazard off
184            case '~': left=1; // full hazard on
185            case '~': left=0; // full hazard off
186            case '~': left=1; // full hazard on
187            case '~': left=0; // full hazard off
188            case '~': left=1; // full hazard on
189            case '~': left=0; // full hazard off
190            case '~': left=1; // full hazard on
191            case '~': left=0; // full hazard off
192            case '~': left=1; // full hazard on
193            case '~': left=0; // full hazard off
194            case '~': left=1; // full hazard on
195            case '~': left=0; // full hazard off
196            case '~': left=1; // full hazard on
197            case '~': left=0; // full hazard off
198            case '~': left=1; // full hazard on
199            case '~': left=0; // full hazard off
200            case '~': left=1; // full hazard on
201            case '~': left=0; // full hazard off
202            case '~': left=1; // full hazard on
203            case '~': left=0; // full hazard off
204            case '~': left=1; // full hazard on
205            case '~': left=0; // full hazard off
206            case '~': left=1; // full hazard on
207            case '~': left=0; // full hazard off
208            case '~': left=1; // full hazard on
209            case '~': left=0; // full hazard off
210            case '~': left=1; // full hazard on
211            case '~': left=0; // full hazard off
212            case '~': left=1; // full hazard on
213            case '~': left=0; // full hazard off
214            case '~': left=1; // full hazard on
215            case '~': left=0; // full hazard off
216            case '~': left=1; // full hazard on
217            case '~': left=0; // full hazard off
218            case '~': left=1; // full hazard on
219            case '~': left=0; // full hazard off
220            case '~': left=1; // full hazard on
221            case '~': left=0; // full hazard off
222            case '~': left=1; // full hazard on
223            case '~': left=0; // full hazard off
224            case '~': left=1; // full hazard on
225            case '~': left=0; // full hazard off
226            case '~': left=1; // full hazard on
227            case '~': left=0; // full hazard off
228            case '~': left=1; // full hazard on
229            case '~': left=0; // full hazard off
230            case '~': left=1; // full hazard on
231            case '~': left=0; // full hazard off
232            case '~': left=1; // full hazard on
233            case '~': left=0; // full hazard off
234            case '~': left=1; // full hazard on
235            case '~': left=0; // full hazard off
236            case '~': left=1; // full hazard on
237            case '~': left=0; // full hazard off
238            case '~': left=1; // full hazard on
239            case '~': left=0; // full hazard off
240            case '~': left=1; // full hazard on
241            case '~': left=0; // full hazard off
242            case '~': left=1; // full hazard on
243            case '~': left=0; // full hazard off
244            case '~': left=1; // full hazard on
245            case '~': left=0; // full hazard off
246            case '~': left=1; // full hazard on
247            case '~': left=0; // full hazard off
248            case '~': left=1; // full hazard on
249            case '~': left=0; // full hazard off
250            case '~': left=1; // full hazard on
251            case '~': left=0; // full hazard off
252            case '~': left=1; //
```

```

91 ///////////////////////////////////////////////////
92
93 case 'f': if(lights) // full beam but only if lights are on the do the following
94 {
95     control_byte=control_byte+0x08; //turn main beam on if lights on
96     break; // breaks the code out of that case
97
98 case 'f': control_byte=control_byte & 0xf7; // AND the control byte with 0xf7 to clear bit 3
99 break; // breaks the code out of that case
100 ///////////////////////////////////////////////////
101 case 'G': control_byte=control_byte+0x08; //turn on fog lights by outputting 0x08 to b or the slave chip
102 break; // breaks the code out of that case
103
104 case 'g': control_byte=control_byte & 0xbf; // AND the control byte with 0xbf to clear bit 6
105 break; // breaks the code out of that case
106
107 ///////////////////////////////////////////////////
108 case 'H': //local_output+=0x08; //turn on horn
109 control_byte=control_byte+0x08;
110 break; // breaks the code out of that case
111
112 case 'h': control_byte=control_byte & 0x7f; // AND the control byte with 0x7f to clear bit 7
113 break; // breaks the code out of that case
114 }
115
116 output_b(control_byte); // send the data to the b Port on the peripheral or slave chip
117
118 }
119
120
121 void ep_w(char where, char what) // name of the subroutine and two specified outputs and there categorisation
122 {
123
124     // name of the subroutine and two specified outputs and there categorisation
125     // wait for men chip to be available
126     while(!input(hs));
127     output_low(hs); // set hs line low (now busy)
128     i2c_start(); // issue i2c start pulse -- starts the i2c stream
129     i2c_write(0xA0); // issue chip type and number -- writes the value in
130     i2c_write(where); // send the address to write to -- selects where to send the value
131     i2c_write(what); // send the data to be written -- and what to write to it
132     delay_ms(20); // delay of 20ms is added
133     i2c_stop(); // issue an i2c stop pulse
134     delay_ms(20); // allow i2c chip to store data
135     input(hs); // release hs line
136
137 char ep_r(char where) // name of the subroutine so it can be called elsewhere in the script
138 {
139     char data=0; // assigns the data a value
140     while(!input(hs)); // wait for men chip to be available - while loop set a conditional statement
141     output_low(hs); // set hs line low (now busy)
142     i2c_start(); // issue i2c start pulse
143     i2c_write(0xA0); // issue chip type and number
144     i2c_write(where); // send address to be read from
145     delay_ms(20); // wait for chip to set its for read
146     i2c_start(); // issue another start pulse
147     delay_ms(20); // wait for chip to be ready
148     i2c_write(0xA1); // issue chip type and number read command
149     data=i2c_read(); // read data from the chip
150     delay_ms(20); // adds a 20ms delay
151     i2c_stop(); // stops the i2c subroutine
152     delay_ms(20); // adds a 20 ms delay
153     input(hs); // allow the hs input back into play by allowing it to have another input
154     return(data); // returns the value in data
155 }
156
157
158 void flash_control(void) // name of subroutine so it can be called elsewhere
159 {
160     if(left & flash) output_high(pin_b4); // If both flash and left are true turn on pin_b4
161     else output_low(pin_b4); // else keep pin_b4 low
162 }
163
164
165
166

```

Cerys Versey - ACDA5018 - DIS assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

70/100

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

CERY'S VERSEY

70.40

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 but does not acknowledge that both pics speak to memory IC in multimaster mode and not directly to each other 62%

9.3

SKILLS FOR PROFESSIONAL PRAC.



The report is well presented with sections to explain the code and include a limited number of references, and some very good graphics and a good flowchart. 75%

11.25