

# Final Submission Distributed Intelligent systems

*by* Joseph Gillespie

---

**Submission date:** 11-Nov-2024 10:39AM (UTC+0000)

**Submission ID:** 242278280

**File name:**

42359\_Joseph\_Gillespie\_Final\_Submission\_Distributed\_Intelligent\_systems\_523660\_985686580.docx (1.33M)

**Word count:** 3319

**Character count:** 17640

2024

# Distributed Intelligence Systems

ACDA5018

JOSEPH GILLESPIE (2214885)

THE UNIVERSITY OF WAIYSEL TRINATEY ST DAVIDS | SA1 Waterfront Campus, IQ  
Building

# Abstract

This report demonstrates the operation of a Master to slave electronic transmission system via a demonstration produced within the simulation program Proteus8. The simulation successfully demonstrates the use of tokens to turn on and off multiple LEDs that are representative of lights on a vehicle. Each of the buttons represents a capital letter that is an activation token for its respective light, once the button is pressed a second time a lowercase token of the capital letter is sent back to the master device resulting in its connecting LED turning off.

The following sections in this report will explain the simulation construction, the code construction, the overall simulation performance/code performance, and a brief explanation of electronic transmission systems.

## Table of Contents

Abstract..... 1

|                                              |    |
|----------------------------------------------|----|
| Table of Figures .....                       | 2  |
| Introduction .....                           | 3  |
| (1:0) Quality Management Systems (QMS) ..... | 4  |
| (2.0) Transmission Systems.....              | 5  |
| (2.1) Point to Point Transmission .....      | 5  |
| (2.2) Point to Multipoint Transmission ..... | 6  |
| (2.3) Multi-Master Transmission .....        | 7  |
| (3.0) The Proteus System .....               | 8  |
| (3.1) The Input.....                         | 8  |
| (3.2) The Main Operation .....               | 9  |
| (3.3) The Output.....                        | 10 |
| (4.0) The Code.....                          | 11 |
| (4.1) Master .....                           | 11 |
| (4.2) Slave .....                            | 12 |
| Conclusion .....                             | 15 |
| References & Bibliography .....              | 16 |
| Appendices .....                             | 17 |

## Table of Figures

|                                                                   |   |
|-------------------------------------------------------------------|---|
| Figure 1, Basic Front Car Lights [3] .....                        | 4 |
| Figure 2, Wireless Point-to-Point Transmission [6] .....          | 6 |
| Figure 3, Example of Multipoint [7] .....                         | 7 |
| Figure 4, Block diagram of a Multi-Master transmission [10] ..... | 8 |

|                                                   |    |
|---------------------------------------------------|----|
| Figure 5, System inputs .....                     | 9  |
| Figure 6, Main Operation .....                    | 10 |
| Figure 7, System Output .....                     | 10 |
| Figure 8, Variables .....                         | 11 |
| Figure 9, Maine Function .....                    | 11 |
| Figure 10, Writing and reading to the slave ..... | 12 |
| Figure 11, Button reading.....                    | 12 |
| Figure 12, Slaves variables.....                  | 12 |
| Figure 13, Writing to the Virtual Machine .....   | 13 |
| Figure 14, Maine function .....                   | 13 |
| Figure 15, Writing to the master.....             | 14 |
| Figure 16, Indicator function .....               | 14 |
| Figure 17, Error watch .....                      | 14 |
| Figure 18, Lights error functions .....           | 15 |
| Figure 19, Indicator error function .....         | 15 |

## Introduction

Distributed intelligent systems improved upon the centralised systems that lacked the capabilities of large-scale integration, the ability to scale with development, and the ability to handle high-complexity systems alongside their low robustness. Distributed intelligence systems achieve the prevention of these issues due to their capability of splitting operations between cooperating systems that specialise in a section of the overall task. It is due to this capability of splitting tasks that allows the distributed intelligent system to achieve greater feats of robustness, scalability and reliability [1]. These systems can be produced in hardware, software or a combination of both, and can be prominently represented within autonyms vehicles and lights [2].

It is within this report that the master and slave transmission system variation of a distributed intelligence system will be investigated. The system has been simulated/constructed within Proteus8 with the accompanying code being written in C. This system will focus on the front lights on a standard UK car, specifically focusing on those seen on the left-hand side to give a visual example of the distributed intelligence system working. The construction of the simulation will be explained within section 2.0 before a deep look into the operation of the code driving the master and slave transmission system within section 3.0.

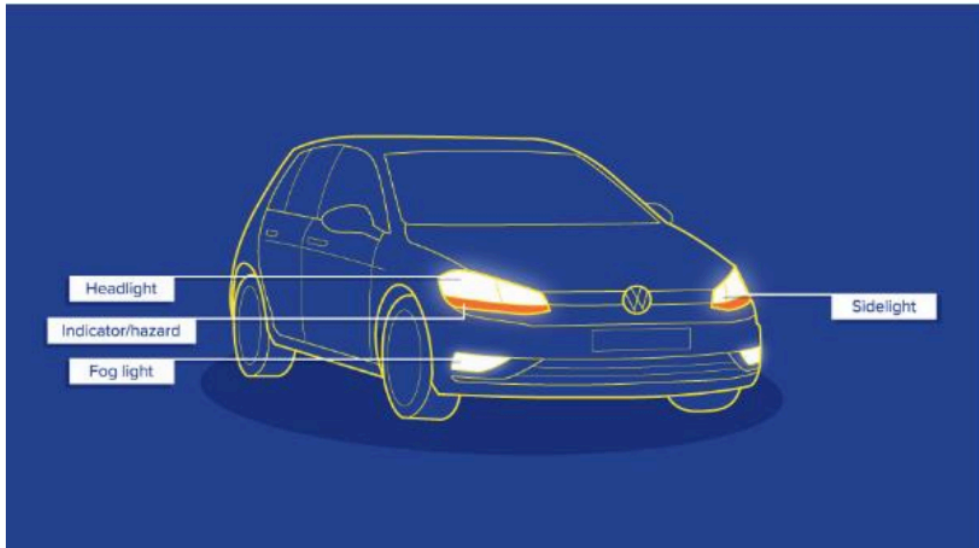


*Figure 1, Basic Front Car Lights [3]*

## (1:0) Quality Management Systems (QMS)

The purpose of having QMS in place throughout the development, production and sales of engineering products is to allow for further innovation and to keep the product to a structured standard. This is accomplished in many ways, aver by helping engineering organisations structure their development procedure with the guide of regulations, all the way to customer/test feedback to aid in future improvements and developments.

Without a QMS in place, products will not be consistent in their quality with some being dangerous to users due to a lack of safety standards being followed, and others performing far above their required purpose resulting in over-engineering that proved detrimental to an organisation. With regards to the absence of user/test feedback, development will stagnate or progress slowly due to repeating mistakes and impractical operations.

## (2.0) Transmission Systems

In terms of distributed intelligence, an electronic transmission system is the method by which the distributed intelligence system communicates with all of its cooperating parts. These transmission systems come in different varieties depending on the overall requirement/setup of the system in which they operate.

The main content of this report demonstrates the operation of the Master and Slave transmission system, but this section explores three more transmission variations.

### (2.1) Point to Point Transmission

A point-to-point transmission system transfers and receives information to two designated points by wireless or wired methods. Due to the direct connection between the two points, the point-to-point transmission method is seen as one of the most secure methods of communication between systems all though limited to two points [4].

The wired point-to-point transmission method can be achieved by using fibre optic cables, ethernet cables, wires and even copper tracks on printed circuit boards (PCB) attached between systems. Due to their direct connection, wired transmission is the most secure method of transmission but is only effective over short distances such as in-office communication or neighbouring systems.

When it comes to long-distance point-to-point, wireless transmission is favoured. This method of transmission can be achieved by many means like air fibre, laser, miller metre wave and microwave frequency. During the set-up of any of these transmission types, paired antennas will be selected and placed at each point. This antenna will be installed up high to allow for a line of sight between the two points and to prevent loss from obstacles. Each antenna will be configured to its pair, resulting in frequency, power output, modulation and other parameters being the same. Once this is done, a test transmission will be carried out before the point-to-point system is operational [5].



*Figure 2, Wireless Point-to-Point Transmission [6]*

## (2.2) Point to Multipoint Transmission

In its basic form, point-to-multipoint transmission operates exactly like point-to-point with the differentiating factor of multipoint being that there are multiple receivers linked to a single transmission point. This allows a central transmission point, such as a telecom tower or a home router, to transmit data to multiple different receivers simultaneously. An example of this within the public will be that of television broadcasting and WiFi access points [7].

Multipoint transmissions can be set up in two different ways wired and wireless with the most common set-up being that of wirelessly due to ease of installation and the transmitter not requiring a direct line of sight to its receivers. Within these set-ups, multipoint can operate in unicast and multicast configurations.

Unicast communication is when the transmission points send individual pieces of data to each of its connected receivers. This requires the receivers to have the capability to receive and process the data sent to it via the transmitter as an individual. This is like point-to-point transmission due to its direct individual communication [7].

Multicast communication occurs when a single piece of data is sent from the transmission point to multiple connected receivers. An example of this will be a live news broadcast [7].

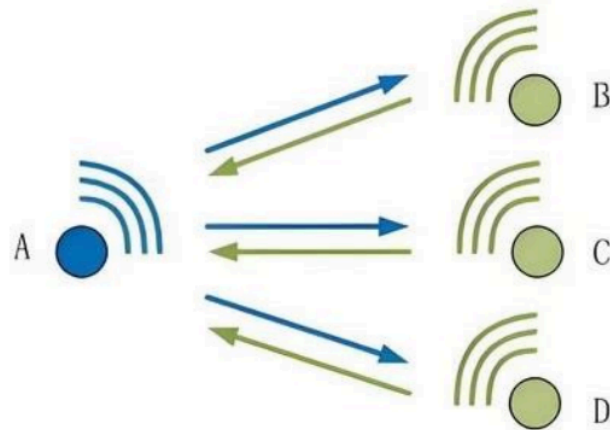


Figure 3, Example of Multipoint [7]

### (2.3) Multi-Master Transmission

The multi-master transmission system can be seen as the evolution of the slave and master transmission system explored later in this report. This is due to multi-master utilising more than one master device within its system to send information to the connected slave devices. Due to this configuration, data can be sent to multiple master devices at the same time with each master device capable of changing the data sent to them to its required need before replicating that data and sending it to its required slave device [8].

For this transmission system to operate effectively the system utilises I2C, which is a communications protocol, that utilises a dedicated serial data line (SDA) and a serial clock line (SCL) when transferring data between devices. The data is sent in byte form with an attached unique address so that the data is sent to the correct receiver. It is due to this structure that the I2C protocol can handle multiple devices that can receive and send commands within a system at the same time [9].

When two master devices transmit the same data to the same slave, all devices will stop there for not transmitting or receiving the information. When multiple masters send data to the same slave at the same time, the master sending a trigger high (1) will take priority with the rest waiting for when the bus is clear to send a new transmission. If multiple masters are sending multiple different data to different slaves at the same time, the first master to transmit a low (0) will take the lead resulting in the remaining masters waiting on the bus to become free once again [10].

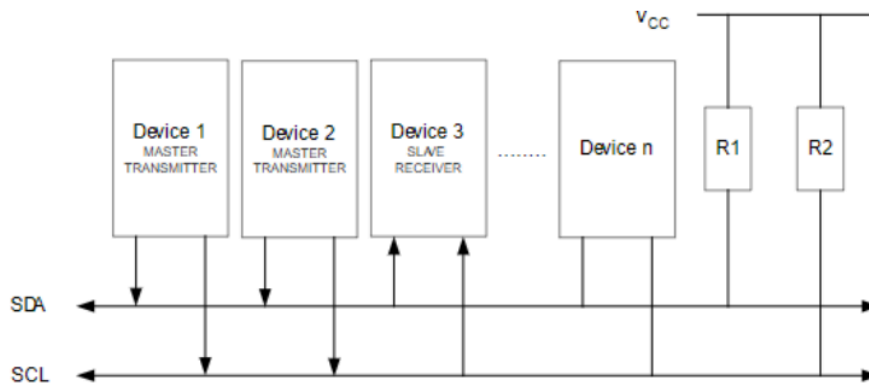


Figure 4, Block diagram of a Multi-Master transmission [10]

### (3.0) The Proteus System

To visually show a master and slave transmission system in operation, a simple car lights system was made within Proteus 8. This system is made up of three operation areas the input, the main operation and the output. It is within this section that each of these areas will be explained.

#### (3.1) The Input

The input of the system consists of four push-to-make (PTM) buttons, four 10k ohms (10k) resistors, four outputs, a VCC connection, and a ground. Each of the push-to-make buttons is assigned to activate one of the LEDs used to represent the Headlights, Full Beas, Fog lights or the indicator that would be typically seen on the front left-hand side of a car. When the user presses one of the push-to-makes, a signal high-coursed by the connection made to VCC is sent to the main operation section of the system. When the user presses the same PTM a second time, the same process of linking VCC to the main operational system occurs, but the related LED switches off.

Figure 5 shows the construction of the input section of the system alongside each push-to-make button linked to its respective lights.

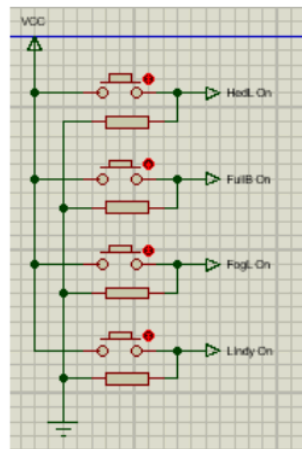


Figure 5, System inputs

### (3.2) The Main Operation

Consisting of two Pic18f2525s, three 10k resistors, an I2C controller in conjunction with a 24LC02B chip, and two virtual terminals, make up the main operation section of the system. These components, plus the c code elaborated on in section 3.0, take inputs given to them via both the input section and the output and produce the appropriate output to their given commands.

The leading components within this operation are that of the two pics which have been split into a slave and master configuration. The left-most pic, seen in Figure 6, is that of the master within the code. This pic uses pins RB0 to RB3 as its receiving pins from the input, whereas pins RC3, 4 and 5 are being used as the pic's communication bus to the second pic, the slave. The slave utilises most of the same pins as its master counterpart except for four additional inputs located at RA0 to RA3 and a fifth output at RB4. These additional inputs are that of the triggers to activate the fifth output.

Seen between the two pics are the I2C controller with chip and the two virtual terminals, located at the top and bottom of Figure 6 respectively. The operation and purpose of the I2C configuration is the same as what was mentioned in section 1.3, to prevent conflict of information between the two pics. As for the two virtual terminals, these are operating as visual aids for the operator so that they can see the tokens being sent between the two pics when they receive inputs.

The configuration of the main input can be seen as a hole in Figure 6.

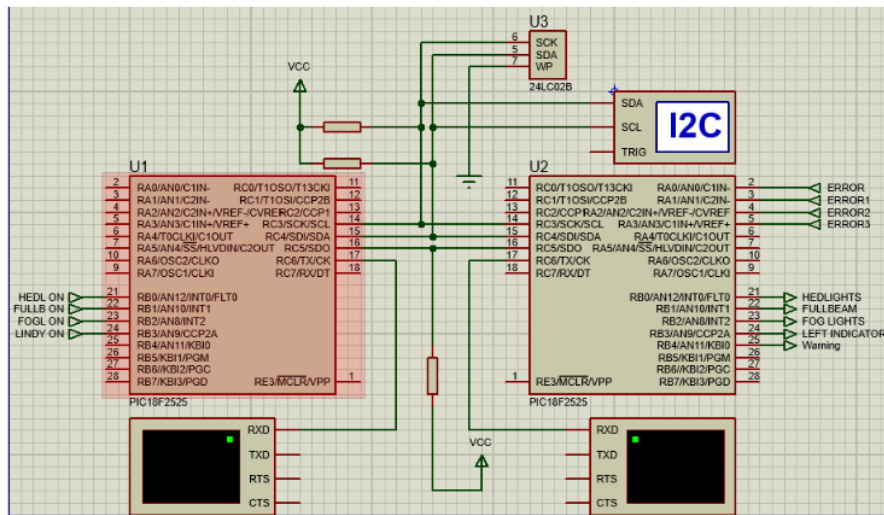


Figure 6, Main Operation

### (3.3) The Output

As the visual representation of the code, the output consists of five LEDs, four switches, and five 240-ohm resistors. Each of the LEDs represents one of the following lights: headlights, full-beam, fog lights, left indicator and a red warning light. These are all triggered on and off once a signal high is sent from the input through the system, but this excludes that of the warning light. This is only triggered by any of the four switches that act as errors within the system.

The configuration of the output can be seen in Figure 7.

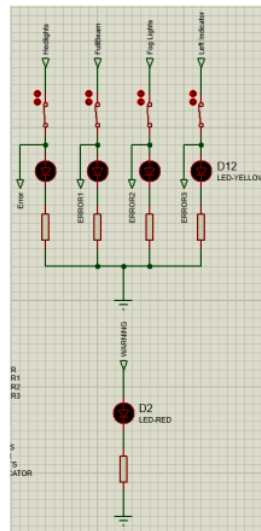


Figure 7, System Output

## (4.0) The Code

The code behind the system's operation, previously examined in section 2.0, will be explored in the following section. For ease of explanation, it has been broken down into its Master and slave structure.

### (4.1) Master

At the start of the master's program, five variables are defined. These variables are used throughout the program and led into their own functions, which will be explored later. It should also be noted that due to the positioning of the variable defining position, the five variables seen in Figure 8 can be used anywhere within the following program.

```
1 #include <main.h>
2 // master
3
4
5 void ep_w(char, char); // The following Void and Char commands are defining later commands
6 char read_buttons(void);
7 char ep_r(char);
8 char local_output;
9 char txbyte;
```

Figure 8, Variables

Following from the previous section is the “main” function seen in Figure 9. Within this function, the process of sending the correct token to the bus relating to its triggering input takes place.

This function works by checking whether or not the buttons within the input have been pressed via the command “buts=read\_buttons()”. If a button press has been detected, one of the four “if” statements will be activated depending on the Hex value that has been triggered at the pin-high location. Once this has been identified a token defined by the “txbyte” variable is sent via the connecting bus to the slave pic enacting the output function.

```
16 {
17     buts=read_buttons(); // Calls for the function that allows button operation (line 114)
18     switch (buts) // Sets the correct button press to the following cases.
19     {
20         case 1: if(local_output & 0x01) // Activates the tern on command for the headlights
21         {
22             local_output-=1; // set bit off
23             txbyte='a'; // set the letter
24         }
25         else
26         {
27             local_output+=1; // set bit on
28             txbyte='A'; // set the letter
29         }
30         break;
31         case 2: if(local_output & 0x02) // Activates the tern on command for the fullbeam
32         {
33             local_output-=2;
34             txbyte='b';
35         }
36         else
37         {
38             local_output+=2;
39             txbyte='B';
40         }
41         break;
42         case 4: if(local_output & 0x04) // Activates the tern on command for the Fog lights
43         {
44             local_output-=4;
45             txbyte='c';
46         }
47         else
48         {
49             local_output+=4;
50             txbyte='C';
51         }
52         break;
53         case 8: if(local_output & 0x08) // Activates the tern on command for the indicator left
54         {
55             local_output-=8;
56             txbyte='d';
57         }
58         else
59         {
60             local_output+=8;
61             txbyte='D';
62         }
63         break;
64         default: break;
65     }
66 }
```

Figure 9, Main Function

To allow for the transmission of the activated token to be sent via the I2C bus, the function “ep\_w” comes into effect alongside the “ep\_r” function. The purpose of both of these functions is to transfer data between the two pics and to present a conflict of data being sent at the same time vior checking the bus to see if it is clear before transmitting. When sending data the write function is active, seen first in Figure 10. When receiving data back from the slave, the read function comes into effect.

```

76 void ep_w(char where, char what) // This section writes information to the slave adding pauses to allow for brakes.
77 {
78     while(!input(hs)); // wait for mem chip to be available
79     output_low(hs); // set hs line low
80     i2c_start(); // issue i2c start pulse
81     i2c_write(0x00); // issue chip type and number
82     i2c_write(where); // send the address to write to
83     i2c_write(what); // send the data to be written
84     delay_ms(20);
85     i2c_stop(); // issue an i2c stop pulse
86     delay_ms(20); // allow i2c chip to store data
87     input(hs); // release hs line
88 }
89
90
91 char ep_r(char where) // This section is waiting for data from the slave befor sending any more information.
92 {
93     char data=0;
94     while(!input(hs)); // wait for mem chip to be available
95     output_low(hs); // set hs line low (mem busy)
96     i2c_start(); // issue i2c start pulse
97     i2c_write(0x00); // issue chip type and number
98     i2c_write(where); // send address to be read from
99     delay_ms(20); // wait for chip to set its for read
100    i2c_start(); // issue another start pulse
101    delay_ms(20); // wait for chip to be ready
102    i2c_write(0x01); // issue chip type and number read command
103    data=i2c_read(); // read data from the chip
104    delay_ms(20);
105    i2c_stop();
106    delay_ms(20);
107    input(hs);
108    return(data);
109 }
110
111

```

Figure 10, Writing and reading to the slave

Referring back to the “buts=read\_buttons();” command seen within the “main” function. This command calls to a function used later within the program. This function activates the pics B pins as inputs for the system. When one of these pins receives a trigger high, the system responds as the signal being a button press resulting in the program carrying out its function.

```

113
114 char read_buttons(void) // This is the main button operation, this section looks for the button being pressed
115 {
116     char value=0; // setup a variable for the buttons
117     while(!input_b()); // wait for button press
118     value=input_b(); // get the value of port b
119     delay_ms(500); // time button release
120     return (value); // return the value of the buttons pressed
121 }
122

```

Figure 11, Button reading

## (4.2) Slave

Similar to that seen at the beginning of the master program, the slave starts with the defining of seventeen variables used throughout the code. A point to note from this that differs from that seen within the master is the use of “char” and “unsigned int”.

```

1 #include <main.h>
2
3 // slave
4
5 char local_output=0; // This section is definin and setting variables that are used throught the code
6 char control_byte=0;
7 char buts=0;
8 char old_but=0;
9 char ep_r(char);
10
11 void ep_w(char where, char what);
12 void flash_control(void);
13 void payload(void);
14 void hedlight_fail(void);
15 void fullbeam_fail(void);
16 void foglight_fail(void);
17 void indicator_fail(void);
18 void ind_fail(void);
19
20 unsigned int lights=0;
21 unsigned int counter=0;
22 unsigned int flashod;
23 unsigned int left=0;

```

Figure 12, Slaves variables

The choice behind the use of these two commands is due to the variation of their function, where “char” is only capable of producing one byte of data, making it perfect for a fixed variable, “unsigned int” is capable of two to four bytes of data making it perfect for varying values within a variable.

Figure 13 shows a function within the main loop that processes data sent to the slave program from the master and prints the token value to the virtual terminals found on the system.

```

34         while(true)
35         {
36             buts=ep_r(0x00);
37             if(old_but!=but)
38             {
39                 printf("%c",but);
40                 ep_w(0x10,but);
41                 delay_ms(20);
42                 payload();
43             }

```

// read data byte from keyboard  
// If but < old\_but  
// print the character of but  
// write the value received into the memory @ address 16 (0x10)  
// wait 20ms  
// run the payload operations.

Figure 13, Writing to the Virtual Machine

Operating similarly to the main function within the master, the “payload” function is triggered when a token is sent from the master device to the slave. It is within this function that the output relating to the button press is activated with “case “B:.” and “case “C:.” being controlled by an “if” statement that depends on the trigger of “case “A:.” before they can carry out their operation.

All the lowercase token triggers excluding “case “a:.” are used to deactivate the output they are relating to but in the case of “case “a:.”, the deactivation includes token A as well as tokens B and C if they have been activated under their rules of there if statement requirements.

```

52 void payload(void)
53 {
54     switch(buts)
55     {
56         case 'A': local_output=0x01;
57                 control_byte=0x01;
58                 lights=1;
59                 break;
60         case 'a': local_output=local_output & 0xf8;
61                 control_byte=control_byte & 0xf8;
62                 lights=0;
63                 break;
64         case 'B': if(lights)
65                 {
66                     local_output=0x02;
67                     control_byte=control_byte & 0x02;
68                     break;
69                 }
70         case 'b': local_output=local_output & 0xf8;
71                 control_byte=control_byte & 0xf8;
72                 break;
73         case 'C': if(lights)
74                 {
75                     local_output=0x04;
76                     control_byte=control_byte & 0x04;
77                     break;
78                 }
79         case 'c': local_output=local_output & 0xf8;
80                 control_byte=control_byte & 0xf8;
81                 break;
82         case 'D': local_output=0x08;
83                 left=1;
84                 break;
85         case 'd': local_output=local_output & 0x08;
86                 left=0;
87                 break;
88     }
89     output_b(control_byte);
90 }

```

// carry out the payload operations.  
// The following section is the headlights  
// Using Hex, adds the output of the slave at 00  
// Sets variable of lights to 1 (High)  
// Using the Hex value to clear Headlights, Full beam and Fog lights when off  
// Sets lights variable to 0 (off)  
// This function activated the full beams only once the headlights are active.  
// Sets the output to 01  
// Clears only the fullbeam light when button pressed on second time  
// This function activated the fog lights only once the headlights are active.  
// Clears only the foglights  
// Activates the indicator  
// Sets the variable lights to high  
// Clears the indicator  
// Sets left variable to low  
// send the data to the b Port.

Figure 14, Main function

Figure 15 operates the same as the transmission to the I2C data bus as that within the master program with the only exception of the data being written to and received from the master device.

```

98 void ep_w(char where, char what)
99 {
100     while(!input(hs)); // wait for mem chip to be available
101     output_low(hs); // set hs line low
102     i2c_start(); // issue i2c start pulse
103     i2c_write(0xA0); // issue chip type and number
104     i2c_write(where); // send the address to write to
105     i2c_write(what); // send the data to be written
106     delay_ms(20);
107     i2c_stop(); // issue an i2c stop pulse
108     delay_ms(20); // allow i2c chip to store data
109     input(hs); // release hs line
110 }
111
112 char ep_r(char where)
113 {
114     char data=0;
115     while(!input(hs)); // wait for mem chip to be available
116     output_low(hs); // set hs line low (mem busy)
117     i2c_start(); // issue i2c start pulse
118     i2c_write(0xA0); // issue chip type and number
119     i2c_write(where); // send address to be read from
120     delay_ms(20); // wait for chip to set its for read
121     i2c_start(); // issue another start pulse
122     delay_ms(20); // wait for chip to be ready
123     i2c_write(0xA1); // issue chip type and number- read command
124     data=i2c_read(); // read data from the chip
125     delay_ms(20);
126     i2c_stop();
127     delay_ms(20);
128     input(hs);
129     return(data);
130 }
131
132

```

Figure 15, Writing to the master

Figure 16 shows the activation of the pic's timer function, which is built into the device. Once the timer has been set to its required pulse settings, the function "flash\_control", which follows, utilises this ability to operate the blinking of the indicator light once activated.

```

133 #INT_TIMER2
134 void TIMER2_isr(void) // This function is utilising the inbuilt timer within the Pic18F2525
135 {
136     counter++;
137     if(counter== 40)
138     {
139         counter=0;
140         if(flash) flash=0;
141         else flash=1;
142         flash_control();
143     }
144 }
145
146 void flash_control(void) // Activates the indicator so that it blinks
147 {
148     if(left & flash) output_high(pin_B3);
149     else output_low(pin_B3);
150 }
151

```

Figure 16, Indicator function

At the end of the slave program, four functions operate as error responses in the case that an error is detected at the activated LED can be found. Figure 17 shows the code used to check the status of the activated LEDs and the commands that activate in response to an error detection. These lines of code are located within the "main" function of the program.

```

44     old_but=but; // copy the value of buts into old_but (becomes the previous value)
45     headlight_fail(); // Looked for a fault with the headlights
46     fullbeam_fail(); // Looked for a fault with the fullbeam
47     foglight_fail(); // Looked for a fault with the Foglights
48     indicator_fail(); // Looked for a fault with the Indicator
49 }

```

Figure 17, Error watch

When an error is detected due to a break in the circuit, resulting in a trigger low on the activated LEDs feedback line to the slave device pins A. The respective error function, shown in Figures 18 and 19, will be triggered. These functions work of a basic *if else* operation that depends on a trigger low being detected on the activated LEDs error feedback line to the slave. When this is detected, the *if else* statement triggers the fifth output being that of a red LED warning light. Once a triggered high is once again detected at pins A or the faulty output receives an off command, the warning LED will be deactivated.

```

152 void hedlight_fail(void) // Functon activates when a fault is detected on the hedlights
153 {
154     if(control_byte & 0x01)
155     {
156         if(!input(Pin_A0))
157         {
158             output_high(Pin_B4);
159         }
160         // Activates output B4
161     }
162     else
163     {
164         if(input(Pin_A0))output_low(Pin_B4);
165     }
166 }
167
168 void fullbeam_fail(void) // Functon activates when a fault is detected on the fullbeams
169 {
170     if(control_byte & 0x02)
171     {
172         if(!input(Pin_A1))
173         {
174             output_high(Pin_B4);
175         }
176         // If a input is not detected on A1 once the lights are on do following command
177     }
178     else
179     {
180         if(input(Pin_A1))output_low(Pin_B4);
181     }
182 }
183
184 void foglight_fail(void) // Functon activates when a fault is detected on the foglights
185 {
186     if(control_byte & 0x04)
187     {
188         if(!input(Pin_A2))
189         {
190             output_high(Pin_B4);
191         }
192         // If a input is not detected on A2 once the lights are on do following command
193     }
194     else
195     {
196         if(input(Pin_A2))output_low(Pin_B4);
197     }
198 }
199
200

```

Figure 18, Lights error functions

```

200 void Indicator_fail(void) // Functon activates when a fault is detected on the indicators
201 {
202     if(local_output & 0x08)
203     {
204         if(!input(Pin_A3))
205         {
206             if(!left & flash) output_high(Pin_B4);
207             // If a input is not detected on A3 once the Lights are on do following command
208         }
209     }
210     else
211     {
212         if(!left)output_low(Pin_B4);
213         // Stops LED from flassing with indicator once LED is sepost to be off
214         else output_low(Pin_B4);
215     }
216 }
217

```

Figure 19, Indicator error function

## Conclusion

In conclusion, this report has given an example of an operational master-to-slave transmission system while using the visual aid of a car's front left lights. The code, written in the C language, has used tokens to transmit information via the I2C bus when communicating between the master and slave devices further reinforcing the smooth operation of the system and aiding in the distribution of information during the running of the system.

All thou the system is fully operational, some minor errors within its operation are visible. This error are mainly focused on the warning light activation. This has resulted in certain light activations coursing the warning light to only activate when the last light to be triggered is broken. In other cases, if the indicator is broken after all other lights are activated, the warning light will blink like the indicator. All of these errors don't directly affect the operation of the system and do not disprove the effectiveness of the transmission system as a whole, they are matters that can be amended when further development time is given.

## References & Bibliography

- [1]W. A. Gruver, "Distributed Intelligence Systems: A New Paradigm for Systems Integration," pp. nil14–nil15, Aug. 2007, doi: <https://doi.org/10.1109/iri.2007.4296581>.
- [2]"Distributed AI: What it is and Why it Matters?," *Clanx.ai*, 2023.  
<https://clanx.ai/glossary/distributed-ai>
- [3]E. C. Parts, "How To Check Your Car Lights | Euro Car Parts," *Euro Car Parts Blog*, Oct. 07, 2020. <https://www.eurocarparts.com/blog/how-to-check-your-car-lights>
- [4]D. Jones, "What is a Point to Point Network?," *APC Solutions*, Nov. 23, 2020.  
<https://www.apcsolutionsuk.com/point-to-point-network/>
- [5]"Point to Point Communication Comprehensive Guide," *Sanny Telecom*, Apr. 27, 2024.  
<https://www.sannytelecom.com/point-to-point-communication-comprehensive-guide/>
- [6]adminr, "What is a point-to-point internet connection and what are the benefits?," □□□□□□, Jan. 21, 2021. <https://respina.net/blog/what-is-a-point-to-point-internet-connection-and-what-are-the-benefits/?lang=en>
- [7]"What is Point-to-Multipoint Communication?," *Sanny Telecom*, May 18, 2024.  
<https://www.sannytelecom.com/what-is-point-to-multipoint-communication/>
- [8]"Concepts of Multi-Master Replication (Oracle Fusion Middleware Deployment Planning Guide for Oracle Directory Server Enterprise Edition)," *docs.oracle.com*.  
<https://docs.oracle.com/cd/E19656-01/821-1502/aalfq/index.html>
- [9]J. Wu, "A Basic Guide to I 2 C." Accessed: Oct. 27, 2024. [Online]. Available:  
[https://www.ti.com/lit/an/sbaa565/sbaa565.pdf?ts=1730019928272&ref\\_url=https%253A%252F%252Fwww.google.com%252F](https://www.ti.com/lit/an/sbaa565/sbaa565.pdf?ts=1730019928272&ref_url=https%253A%252F%252Fwww.google.com%252F)
- [10]"25.7 Multi-master Systems and Arbitration," *Microchip.com*, 2022.  
<https://onlinedocs.microchip.com/oxy/GUID-80B1922D-872B-40C8-A8A5-0CBE009FD908-en-US-3/GUID-6AE16474-5C34-41C5-AA23-9C5519D30FDF.html> (accessed Oct. 27, 2024).

# Appendices

## Master

```
1 #include <main.h>
2 // master
3
4
5 void ep_w(char, char); // The following Void and Char commands are defining Later commands
6 char read_buttons(void);
7 char ep_r(char);
8 char local_output;
9 char txbyte;
10
11 void main()
12 {
13     char butts=0; // Sets all button connections to low (0)
14
15     while(TRUE)
16     {
17         butts=read_buttons(); // Calls for the function that allows button operation (Line 114)
18         switch (butts) // Sets the correct button press to the following cases.
19         {
20             case 1: if(local_output & 0x01) // Activates the tern on command for the headlights
21             {
22                 local_output-=1; // set bit off
23                 txbyte='a'; // set the letter
24             }
25             else
26             {
27                 local_output+=1; // set bit on
28                 txbyte='A'; // set the letter
29             }
30             break;
31             case 2: if(local_output & 0x02) // Activates the tern on command for the fullbeam
32             {
33                 local_output-=2;
34                 txbyte='b';
35             }
36             else
37             {
38                 local_output+=2;
39                 txbyte='B';
40             }
41             break;
42             case 4: if(local_output & 0x04) // Activates the tern on command for the Fog Lights
43             {
44                 local_output-=4;
45                 txbyte='c';
46             }
47             else
48             {
49                 local_output+=4;
50                 txbyte='C';
51             }
52             break;
53             case 8: if(local_output & 0x08) // Activates the tern on command for the indicator Left
54             {
55                 local_output-=8;
56                 txbyte='d';
57             }
58             else
59             {
60                 local_output+=8;
61                 txbyte='D';
62             }
63             break;
64             default: break;
65         }
66
67         printf("%c", txbyte); // print the letter character to the rs232 output
68         ep_w(0, txbyte); // store the input
69     }
70 }
71
72
73
74
75 void ep_w(char where, char what) // This section writes information to the slave adding pauses to allow for brakes.
76 {
77     while(!input(hs)); // wait for mem chip to be available
78     output_low(hs); // set hs line low
79     i2c_start(); // issue i2c start pulse
80     i2c_write(0xA0); // issue chip type and number
81     i2c_write(where); // send the address to write to
82     i2c_write(what); // send the data to be written
83     delay_ms(20);
84     i2c_stop(); // issue an i2c stop pulse
85     delay_ms(20); // allow i2c chip to store data
86     input(hs); // release hs line
87 }
88
89
90
91 char ep_r(char where) // This section is waiting for data from the slave before sending any more information.
92 {
93     char data=0;
94     while(!input(hs)); // wait for mem chip to be available
95     output_low(hs); // set hs line low (mem busy)
96     i2c_start(); // issue i2c start pulse
97     i2c_write(0xA0); // issue chip type and number
98     i2c_write(where); // send address to be read from
99     delay_ms(20); // wait for chip to set its for read
100     i2c_start(); // issue another start pulse
101     delay_ms(20); // wait for chip to be ready
102     i2c_write(0xA1); // issue chip type and number read command
103     data=i2c_read(); // read data from the chip
104     delay_ms(20);
105     i2c_stop();
106     delay_ms(20);
107     input(hs);
108     return(data);
109 }
110
111
112
```

```

113
114 char read_buttons(void)                                // This is the main button operation, this section looks for the button being pressed
115 {
116     char value=0;                                       // setup a variable for the buttons
117     while(!input_b());                                  // wait for button press
118     value=input_b();                                    // get the value of port b
119     delay_ms(500);                                      // time button release
120     return (value);                                    // return the value of the buttons pressed
121 }
122

```

## Slave

```

1 #include <main.h>
2
3 // slave
4
5 char local_output=0;
6 char control_byte=0;
7 char but=0;
8 char old_but=0;
9 char ep_r(char);
10
11 void ep_w(char where, char what);
12 void flash_control(void);
13 void payload(void);
14 void hedlight_fail(void);
15 void fullbeam_fail(void);
16 void foglight_fail(void);
17 void indicator_fail(void);
18 void ind_fail(void);
19
20 unsigned int lights=0;
21 unsigned int counter=0;
22 unsigned int flash=0;
23 unsigned int left=0;
24
25
26 void main()
27 {
28     char value=0xff;
29     char q=0;
30     setup_timer_2(T2_DIV_BY_16, 249, 15);
31     enable_interrupts(INT_TIMER2);
32     enable_interrupts(GLOBAL);
33
34     while(true)
35     {
36         but=ep_r(0x00);
37         if(old_but!=but)
38         {
39             printf("%c", but);
40             ep_w(0x0, but);
41             delay_ms(20);
42             payload();
43         }
44         old_but=but;
45         hedlight_fail();
46         fullbeam_fail();
47         foglight_fail();
48         indicator_fail();
49     }
50 }
51
52 void payload(void)
53 {
54     switch(buts)
55     {
56         case 'A': local_output=0x01;
57                  control_byte=0x01;
58                  lights=1;
59                  break;
60         case 'a': local_output=local_output & 0xf0;
61                  control_byte=control_byte & 0xf0;
62                  lights=0;
63                  break;
64         case 'B': if(lights)
65                  {
66                      local_output=0x02;
67                      control_byte=control_byte+0x02;
68                  }
69                  break;
70         case 'b': local_output=local_output & 0xf0;
71                  control_byte=control_byte & 0xf0;
72                  break;
73         case 'C': if(lights)
74                  {
75                      local_output=0x04;
76                      control_byte=control_byte+0x04;
77                  }
78                  break;
79         case 'c': local_output=local_output & 0xf0;
80                  control_byte=control_byte & 0xf0;
81                  break;
82         case 'D': local_output=0x00;
83                  left=1;
84                  break;
85         case 'd': local_output=local_output & 0x00;
86                  left=0;
87                  break;
88     }
89     output_b(control_byte);
90 }
91
92 // carry out the payload operations.
93 // The following section is the headlights
94 // Using Hex, adds the output of the slave at 80
95 // Sets variable of lights to 1 (High)
96 // Using the Hex value to clear headlights, Full beam and Fog lights when off
97 // Sets lights variable to 0 (off)
98 // This function activated the full beams only once the headlights are active.
99 // Sets the output to 81
100 // Clears only the fullbeam light when button pressed on second time
101 // This function activated the fog lights only once the headlights are active.
102 // Clears only the foglights
103 // Activates the indicator
104 // Sets the variable lights to high
105 // Clears the indicator
106 // Sets left variable to low
107 // send the data to the b Port.

```

```

97
98 void ep_w(char where, char what)
99 {
100     while(!input(hs));
101     output_low(hs);
102     i2c_start();
103     i2c_write(0x00);
104     i2c_write(where);
105     i2c_write(what);
106     delay_ms(20);
107     i2c_stop();
108     delay_ms(20);
109     input(hs);
110 }
111
112 char ep_r(char where)
113 {
114     char data=0;
115     while(!input(hs));
116     output_low(hs);
117     i2c_start();
118     i2c_write(0x00);
119     i2c_write(where);
120     delay_ms(20);
121     i2c_start();
122     delay_ms(20);
123     i2c_write(0xA1);
124     data=i2c_read();
125     delay_ms(20);
126     i2c_stop();
127     delay_ms(20);
128     input(hs);
129     return(data);
130 }
131
132 #INT_TIMER2
133 void TIMER2_isr(void)
134 {
135     counter++;
136     if(counter== 40)
137     {
138         counter=0;
139         if(flash) flash=0;
140         else flash=1;
141         flash_control();
142     }
143 }
144
145 void flash_control(void)
146 {
147     if(left & flash) output_high(pin_B3);
148     else output_low(pin_B3);
149 }
150
151
152 void hedlight_fail(void)
153 {
154     if(control_byte & 0x01)
155     {
156         if(!input(Pin_A0))
157         {
158             output_high(Pin_B4);
159         }
160         else
161         {
162             if(input(Pin_A0)) output_low(Pin_B4);
163         }
164     }
165 }
166
167 void fullbeam_fail(void)
168 {
169     if(control_byte & 0x02)
170     {
171         if(!input(Pin_A1))
172         {
173             output_high(Pin_B4);
174         }
175         else
176         {
177             if(input(Pin_A1)) output_low(Pin_B4);
178         }
179     }
180 }
181
182 void foglight_fail(void)
183 {
184     if(control_byte & 0x04)
185     {
186         if(!input(Pin_A2))
187         {
188             output_high(Pin_B4);
189         }
190         else
191         {
192             if(input(Pin_A2)) output_low(Pin_B4);
193         }
194     }
195 }
196
197
198 void indicator_fail(void)
199 {
200     if(local_output & 0x08)
201     {
202         if(!input(Pin_A3))
203         {
204             if(left & flash) output_high(Pin_B4);
205         }
206         else
207         {
208             if(!left) output_low(Pin_B4);
209             else output_low(Pin_B4);
210         }
211     }
212 }
213
214
215
216

```

// wait for mem chip to be available  
// set hs line low (mem busy)  
// issue i2c start pulse  
// issue chip type and number  
// send the address to write to  
// send the data to be written  
// issue an i2c stop pulse  
// allow i2c chip to store data  
// release hs line

// wait for mem chip to be available  
// set hs line low (mem busy)  
// issue i2c start pulse  
// issue chip type and number  
// send address to be read from  
// wait for chip to set its for read  
// issue another start pulse  
// wait for chip to be ready  
// issue chip type and number read command  
// read data from the chip

// This function is utilising the inbuilt timer within the Pic18F2525

// Activates the indicator so that it blinks

// Functan activates when a fault is detected on the headlights  
// If a input is not detected on A0 once the lights are on do following command  
// Activates output B4

// Functan activates when a fault is detected on the fullbeams  
// If a input is not detected on A1 once the lights are on do following command

// Functan activates when a fault is detected on the foglights  
// If a input is not detected on A2 once the lights are on do following command

// Functan activates when a fault is detected on the indicators  
// If a input is not detected on A3 once the lights are on do following command  
// Stops LED from flashing with indicator once LED is sepost to be off

# Final Submission Distributed Intelligent systems

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

69/100

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

JOSEPH GILLESPIE

68.75

**RESEARCH-INFORMED LITERATURE**



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

13

**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 reasonable bibliography only two of which are referenced.  
68%

**10.2**