

OW DSP

by Owen Wild

Submission date: 13-Nov-2024 04:52PM (UTC+0000)

Submission ID: 242561238

File name: 42279_Owen_Wild_OW_DSP_523660_905979099.docx (857.9K)

Word count: 2888

Character count: 13406

Distributed Intelligence systems

ACDA5018

A simple token-based
distributed network.

Student ID: 2214461

Word count:2450

Author: Owen Wild

Acknowledgement

Proteus and CCS compiler have been used to code and produce this work.

Table of Contents

Acknowledgement	2
Table of Figures.....	2
Abstract.....	3
Introduction	3
Background	3
Methods.....	4
Design.....	4
Code	6
Master code	8
Slave code	12
Conclusion and Recommendation	15
References	15
Appendix.....	16

Table of Figures

Figure 1 - Circuit design overview.....	5
Figure 2 - flowchart.....	7
Figure 3 - Master code variables.....	8
Figure 4 - Master Switch case	9
Figure 5 - Master Switch case (2).....	9
Figure 6 - Master Switch case (3).....	9
Figure 7 - Master code write.....	10
Figure 8 - Master code read.....	11
Figure 9 - Master code read button.....	11
Figure 10 - Slave code variables	12
Figure 11 - Slave code setup	13
Figure 12 - Slave code payload	14
Figure 13 - Slave code payload (2)	14
Figure 14 - Slave code flash.....	15
Figure 15 - Buttons Overview	16
Figure 16 – Tokens	16
Figure 17 - I/O pins and PIC layout.....	16
Figure 18 - Main.h code	17
Figure 19 - Memory location of A	17

Abstract

This report showcases how a multi-master system can be used to demonstrate a simple token-based network using a master and a slave PIC18F2525. Also demonstrates the code for this system in embedded C. The system works to the standard of a rear corner of a car lights system.

Introduction

The structure of this report is to showcase from background researchers the design of the simple token-based transmission system using PIC18F2525s and a memory unit. The code behind the working of this multi-master system can transmit and receive an inputted value and transfer it to the output using a separate device from the input one. The main conclusion is the system works and demonstrates how a multi-master system can be implemented into a simple token-based setup.

Background

Quality management systems, point-to-point, point-multipoint, and multi-master transmission systems were examined for this assignment and lab report.

Industries use quality management systems to formalise goals, objectives, goods and services. Quality management systems should be regularly maintained to fulfil orders. Some basic requirements of a quality management system are the quality of the products so they meet the company standards, as well as country or international standards. Another is maintaining and observing existing and new market trends to aid the industries to adapt and be fit for their specific markets. [1]. Quality management systems influence safety standards as when developing and manufacturing products safety should be a high priority therefore the quality management system will help identify safety improvement when and before a safety incident occur.

Point-to-point transmission is a network protocol that connects two networks without having any nodes and only one channel. [2]. An example of this type of transmission is rs323 which will connect two devices via the transmitting and receiving terminals of those devices. [3]. Point-to-multipoint is similar to point-to-point communications but the main difference is a single transmitter will transmit to multiple devices. [4]. How it works is that the dater is ready to be transmitted from the transmitter over a medium like satellite or wireless to multiple receivers that all will get the same dater. The multi-master transmission system uses a master or several masters connected to a Slave or multiple slaves to transfer and receive dater. How this system transfers dater is with a CAN bus which is a controlled area network therefore links all the masters and slaves to a dater line. If more than one devise is trying to transmit dater on the CAN bus the highest priority dater will be transmitted first. [5]. A handshake mechanism is one way to help with the priority of the dater and whether one device is and wants to use the dater bus.

Methods

PIC18F2525 is used as it has the number of I/O pins required and the ability to be used in a multi-master transmission system to fulfil the assignment brief using I2C as the communication and transmissions.

Design

The design and configuration of the multi-master transmission is laid out to be the back left corner of a standard car. The lights are layout to be the brake lights, reversing lights, left indicator, rear light and fog light all with separate LEDs connected to a resistor and ground. There are two PIC18F2525 that are used. The left PIC is the master and the right one is the slave. These paramount are defined in the code.

A button array has been made consistent of 5 buttons connected to Vcc (power supply). The button outputs have all been labelled A to E and these are our tokens: Figure 15 - Buttons Overview, Figure 16 – Tokens. The output from the buttons is connected to the I/O pins of the master PIC. Using terminals makes it clearer as they replace wire connection so easier to read whilst doing the same job just have to define whether it's an input or output terminal. This process is similar to the slave I/O pins connecting to the lights.

The master-slave PICs are designed and laid out to take I2C as its form of multi-master transmission system. The Master and slave are connected with three wires and two of the lines are connected to a memory component. This is so the master and slave can read and write, to store the HEX-decimal number in one storage location. This HEX-decimal value correlates to which button has been pressed along with the token it needs to transmit to get the Hex value. There are two pull-up resistors connected to two transmission lines. These pull-up resistors keep the line high till the master or slave transmits a signal then it goes low to be able to send the signal. In the design, the memory component acts as a buffer as well as the storing location the PICs can read and write to. The last pull-down resistor is connected to the Hs line or Handshake line. This line helps the PICs not to send a signal at the same time as the other one. The pull-down resistor aids this line by keeping it low so the PICs know each other isn't transmission so it's free to send a signal. Once a signal is sent the line is high so the other PIC knows it can't send a signal. Altogether these three resistors and transmission lines help the master and slave PICs to communicate with each other within a memory storage, with no cross signal and double transmission from both at the same time.

Finally, in this circuit design, there are four different scopes added to help debug the code if errors occur and to aid. Two virtual terminals are added to show what character the master is storing in the memory and the other what the slave is reading and putting out the corresponding lights. The second one is an oscilloscope just to see what the signal looks like storing and taking the data within the memory and what's outputted from the slave. This acts as a sanity check to see visually if all the signals are the same. Lastly, the I2C debugger shows the HEX-decimal value being written and read from the memory. This benefits us as we can see if the values are the correct ones.

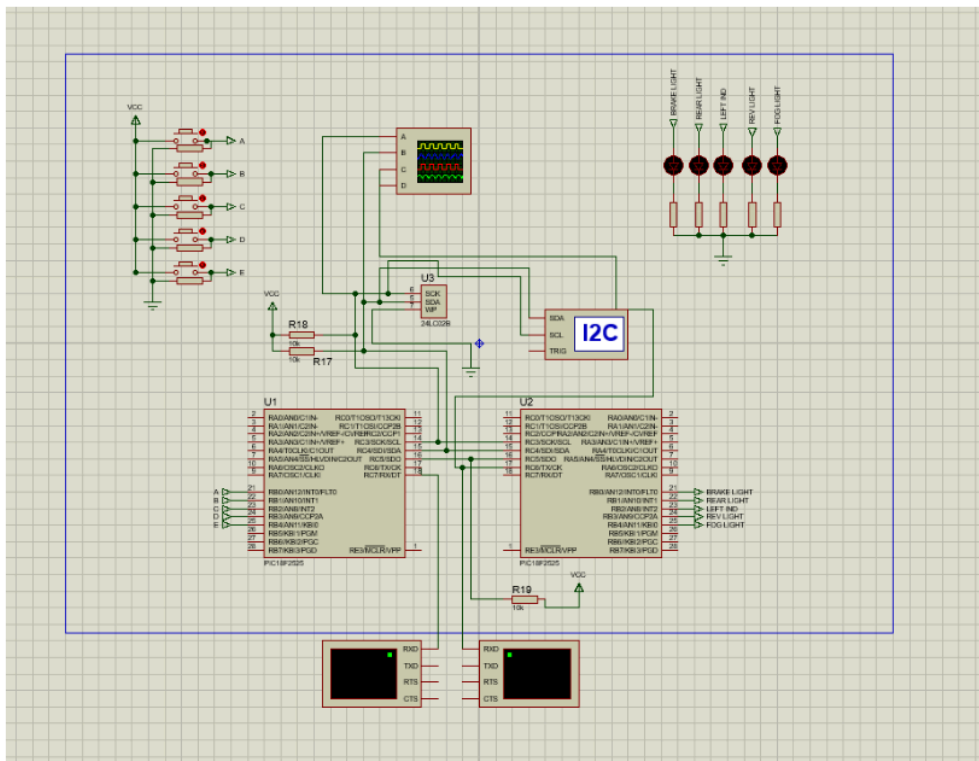


Figure 1 - Circuit design overview.

Code

In this section, the code of this multi-master is shown and discussed. There are two documents of code as there are two pics one for the master and one for the slave. The main.h file is included on both master and slave code as it sets up the PICs fuses and the communication type (I2C) it needs to do. Global variables in this code can be used in any part of the code so can be used outside the subroutine and within the void main like the local one has to.

A flowchart has been produced to show the basic operation of the system without looking at the code and to show how both the master and slave interact. Shown in Figure 2 - flowchart

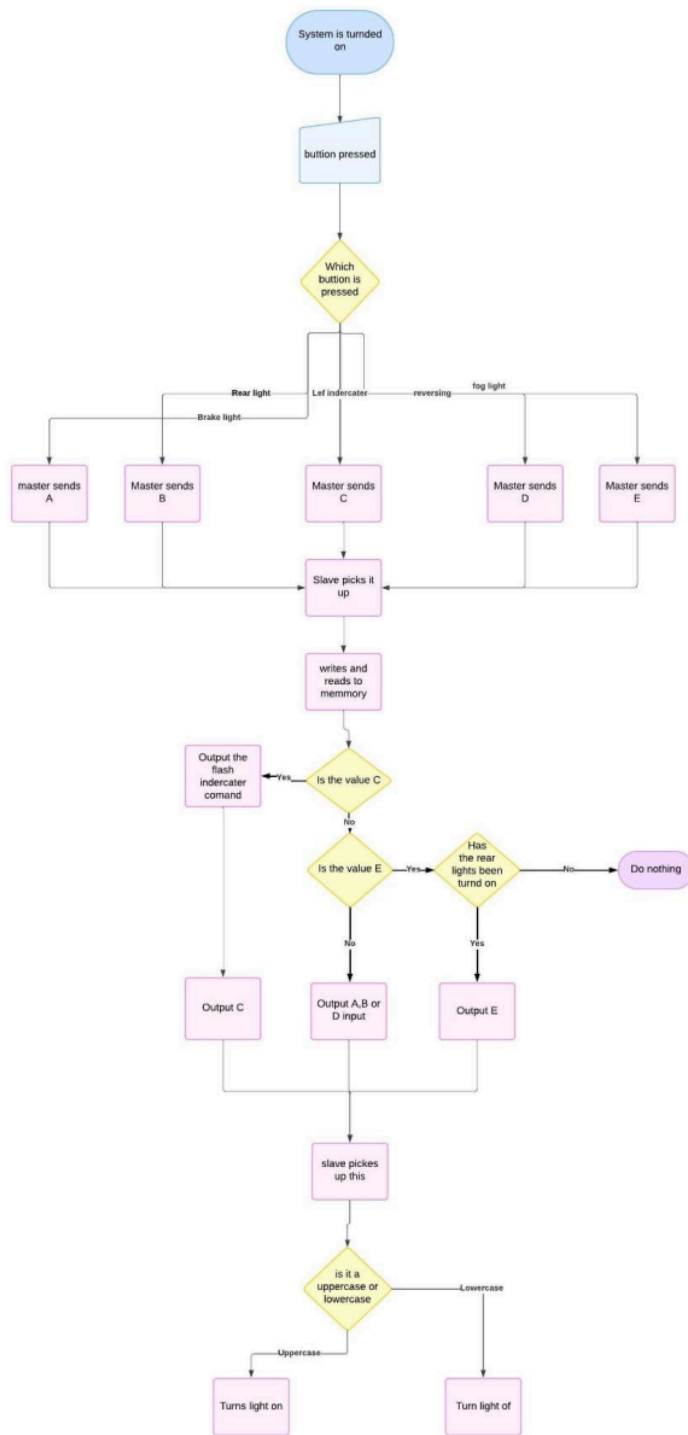
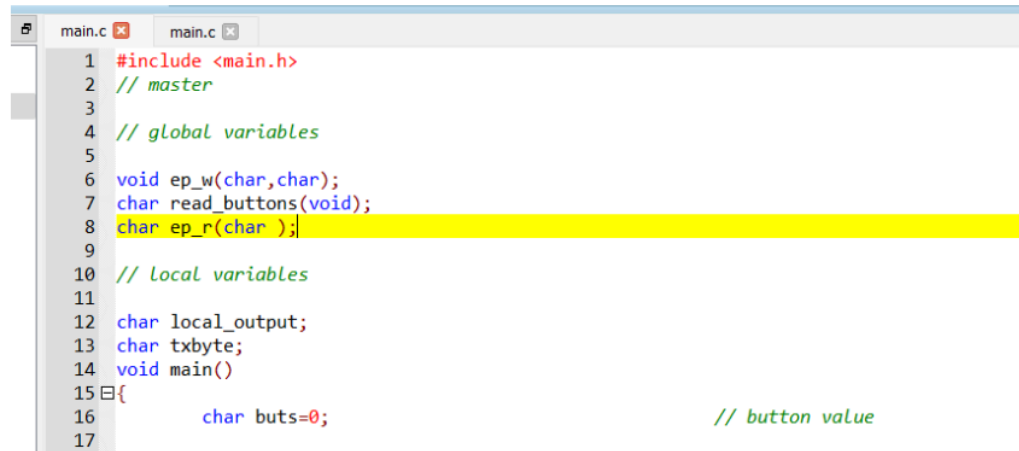


Figure 2 - flowchart

Master code

The master codes have very few global and local variables as all its doing is reading which buttons are pressed and storing them in the memory location. The global variables set up the master to be able to write to the external memory location and define the read button to be able to code with it later. The last one is to be able to read the character that's in the memory. Local variables are set up to read the characters of the local output and txbyte and store them in the variable.



```
1 #include <main.h>
2 // master
3
4 // global variables
5
6 void ep_w(char, char);
7 char read_buttons(void);
8 char ep_r(char );
9
10 // local variables
11
12 char local_output;
13 char txbyte;
14 void main()
15 {
16     char buts=0; // button value
17 }
```

Figure 3 - Master code variables

The master switch case outlines the code to which the master needs to read which buttons are and what character to the external memory unit. The case number is the binary number which goes up in powers of 2. Local_output stores the character that is sent from the button an uppercase character for when it's pressed and a lowercase character for when it's released. Local_output is assigned to its binary number and gives its character. The outline for this master switch case is below in Figure 4 - Master Switch case, Figure 5 - Master Switch case (2), and Figure 6 - Master Switch case (3).

```

14 void main()
15 {
16     char buts=0;           // button value
17
18     while(TRUE)
19     {
20
21         buts=read_buttons(); // get the value of buts as a character
22         switch (buts)        // allows the pic to read the button presses and releases and producing a signal for them
23         {
24             case 1: if(local_output & 1) // toggle the brake Lights Lights on and off
25             {
26                 local_output-=1; // set bit off
27                 txbyte='a';      // set the token
28             }
29             else
30             {
31                 local_output+=1; // set bit on
32                 txbyte='A';      // set the token
33             }
34             break;
35             case 2: if(local_output & 2) // toggle the rear Lights on and off
36             {
37                 local_output-=2;
38                 txbyte='b';
39             }
40             else
41             {
42                 local_output+=2;
43                 txbyte='B';
44             }
45         }

```

Figure 4 - Master Switch case

```

        break;
        case 4: if(local_output & 4) // toggle the Left indicator on and off
        {
            local_output-=4;
            txbyte='c';
        }
        else
        {
            local_output+=4;
            txbyte='C';
        }
        break;
        case 8: if(local_output & 8) // toggle the reversing light on and off
        {
            local_output-=8;
            txbyte='d';
        }
        else
        {
            local_output+=8;
            txbyte='D';
        }
        break;
        case 16: if(local_output & 0x10) // toggle the fog Lights on and off
        {
            local_output-=16;
            txbyte='e';
        }
        else

```

Figure 5 - Master Switch case (2)

```

        {
            local_output+=16;
            txbyte='E';
        }
        break;
        default: break;
    }
    ep_w(0,txbyte); // store the data in mem Location 0;
}

```

Figure 6 - Master Switch case (3)

The Master has to be able to read and write as we are using I2C, external memory and slave PIC which has to be written and read into using eeprom. Writing shows how the master is going to write to the external memory so the slave can read and follow the instruction the master has defined it to do. To write to the slave the master needs to see if the hs line is low before it can think and start transmitting. Once this is confirmed it sends out a low signal to the hs line which will get pulled low by the pull-down resistor. This makes the hs line high so the slave knows the memory is in use. 0xA0 is the storage location the master writes to, this allows the character from the button to be stored as a HEX-decimal number, This is part of the what and where defined. The delays help it to wait before sending and receiving signals to allow it to get all the data so it's as accurate as possible.

The read routine of the master has the same memory location and delay times. This routine is important as it checks whether the data has been inputted correctly and if the slave has received the data correctly. The read routine sets the character data as 0 and sets the hs line to busy. There are two steps that the master needs to go through before it can read the data. Firstly it needs to issue the chip type and number and the address to where it is located for 0xA0. Secondly, need to set a start pulse again to issue the chip type and number of the location of 0xA1. Finally, the chip can read what is located at the address it wants to read and retrieve the data. The delays help with not having a stack overflow where it will read the whole thing as multiple things would be transmitted at once. To end the sequence the pic stops reading and outputs the hs line high to show it's free.

```

92
93 void ep_w(char where, char what)
94 {
95     while(!input(hs));
96     output_low(hs);
97     i2c_start();
98     i2c_write(0xA0);
99     i2c_write(where);
100    i2c_write(what);
101    delay_ms(20);
102    i2c_stop();
103    delay_ms(20);
104    input(hs);
105
106 }
107
// allows to write to memory
// wait for mem chip to be available
// set hs line low
// issue i2c start pulse
// issue chip type and number
// send the address to write to
// send the data to be written
// waits for the data to be written
// issue an i2c stop pulse
// allow i2c chip to store data
// release hs line

```

Figure 7 - Master code write

```

109 char ep_r(char where) // Reading from the memory
110 {
111     char data=0;
112     while(!input(hs)); // wait for mem chip to be available
113     output_low(hs); // set hs line low (mem busy)
114     i2c_start(); // issue i2c start pulse
115     i2c_write(0xA0); // issue chip type and number
116     i2c_write(where); // send address to be read from
117     delay_ms(20); // wait for chip to set its for read
118     i2c_start(); // issue another start pulse
119     delay_ms(20); // wait for chip to be ready
120     i2c_write(0xA1); // issue chip type and number read command
121     data=i2c_read(); // read data from the chip
122     delay_ms(20); // waits for it to read the data
123     i2c_stop(); // stops transmission
124     delay_ms(20); // allows time to stop
125     input(hs); // inputs it on the hs line
126     return(data); // returns the data
127 }
128

```

Figure 8 - Master code read

The read button code for the master PIC allows in effect that the button to turn on and control the light can be read and interpreted by the master PIC. The value I set to 0 as a character then sets up a variable for the input of the buttons on the port B side of the master PIC. It then sits in an idle state till the button is pressed and then gets the value which the master gets and can deal with. A delay is set to allow time for the button to be pressed and released so only one value gets instead of multiple from the switch bouncing.

```

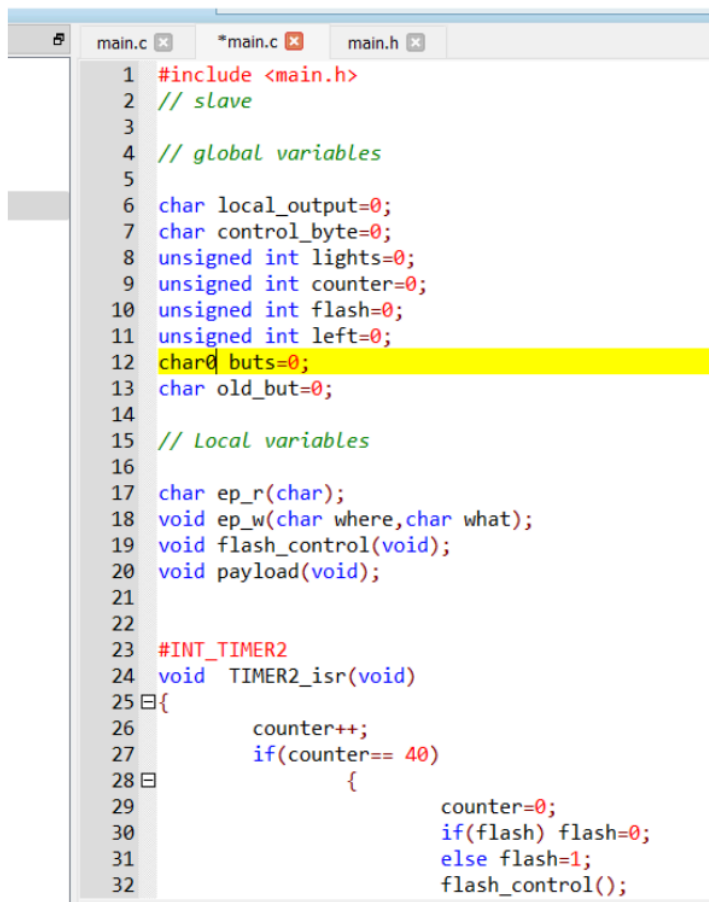
128
129
130
131 char read_buttons(void) // allows the buttons to be read
132 {
133     char value=0; // setup a variable for the buttons
134     while(!input_b()); // wait for button press
135     value=input_b(); // get the value of port b
136     while(input_b());
137     delay_ms(500); // time button release
138     return (value); // return the value of the buttons pressed
139 }
140

```

Figure 9 - Master code read button

Slave code

The slave code variables are a lot more as the slave has to do a lot more work as it reads writes and has to take action on whatever the master tells it to do. Variables are set to hold a character or a value that is set to 0n at first. In the void main, there is a timer added to allow the left indicator to flash on and off repeatedly till turned off. The counter is set up to count up and store the flash as a 1 or 0.



```
1  #include <main.h>
2  // slave
3
4  // global variables
5
6  char local_output=0;
7  char control_byte=0;
8  unsigned int lights=0;
9  unsigned int counter=0;
10 unsigned int flash=0;
11 unsigned int left=0;
12 char0 buts=0;
13 char old_but=0;
14
15 // Local variables
16
17 char ep_r(char);
18 void ep_w(char where,char what);
19 void flash_control(void);
20 void payload(void);
21
22
23 #INT_TIMER2
24 void TIMER2_isr(void)
25 {
26     counter++;
27     if(counter== 40)
28     {
29         counter=0;
30         if(flash) flash=0;
31         else flash=1;
32         flash_control();
33     }
```

Figure 10 - Slave code variables

Within the void main, there are a few routines that are being used here. A timer is added to allow for an interrupt to be used and to do a 1ms overflow. Within the while true loop, it allows the slave to be able to read the data that the master is telling it and to move it to the external memory storage. A sanity check is done through old_buts= buts. This line of code checks if the old data is the same as the new data. If this passes the slave writes the received data in the memory address and carries out the payload function where it will turn on the corresponding lights.

```
void main()
{
    char q=0;
    setup_timer_2(T2_DIV_BY_16,249,15);           //1.0 ms overflow, 15.0 ms interrupt
    enable_interrupts(INT_TIMER2);
    enable_interrupts(GLOBAL);

    while(true)
    {
        buts=ep_r(0);                             // read data byte from keyboard
        if(old_but!=but)                          // if but <> old_but
        {
            printf("%c",but);                      // print the character of but
            ep_w(16,but);                          // write the value recieved into the memory @ address 16 (0x10)
            delay_ms(20);                          // wait 20ms
            payload();                             // run the payload operations.
        }
        old_but=but;                             // copy the value of but into old_but (becomes the previous value)
    }
}
```

Figure 11 - Slave code setup

The payload function of the slave code is to carry out what the master wants it to. This is done by the master telling the slave what button is pressed what it needs to process and the pins it needs to output on so the right lights turn on. Each case is set to upper- or lower-case letters so the slave knows which section it needs to execute. To turn a light on and off via the variable controlled_output a mask is needed to turn the light off. Therefore, to turn it on the value stays the same as local_output. To turn the light off with the second button press the control byte that was sent out from the local_output needs to be times with the opposite binary sequences. Therefore a binary sequence of 0000 0001 or 1 needs to be masked with a binary sequence of 1111 1110 or fe. This is the same case for all except the left indicator and the fog light. Since the left indicator flashes we have to add extra steps to enable it to flash. All that is added is the lights variable that's tied to 1 for on and 0 for off. For the turning of the left indicator case, a direct command to pin B4 to output low is made. This makes sure it turns off. The Fog light is coded to turn on only when the rear lights are on and to turn off when the rear lights are turned off. The first case change for the fog lights is the mask for the rear lights needs to be for the combined values of the rear button pressed and for the fog light press. Therefore it allows the fog lights to be toggled with the rear ones. In the fog light individual case, an if loop is added to do it only when the lights are at a value of one. This case will only happen when the rear lights are on which will output high on pin B4.

```

main.c  main.c  main.h
59 void payload(void)                                // carry out the payload operations.
60 {
61     switch(buts)
62     {
63         case 'A': local_output++;
64                   control_byte++;
65                   break;
66         case 'a': local_output=local_output & 0xfe; // AND the local output byte with 0xfe to clear bit 0
67                   control_byte=control_byte & 0xfe;
68                   break;
69         case 'B': local_output++; //turn on rear light
70                   control_byte=control_byte+2;
71                   lights = 1; // makes the fog light available to turn on
72                   break;
73         case 'b': local_output=local_output & 0xed; // AND the local output byte with 0xed to clear bits 2,6
74                   control_byte=control_byte & 0xed;
75                   lights = 0;
76                   output_low(pin_B4); // turn the fog light off
77                   break;
78         case 'C': local_output++;
79                   control_byte=control_byte+4; //turn on left indicator
80                   left=1; // starts the left flash
81                   break;
82         case 'c': local_output=local_output & 0xef; // AND the local output byte with 0xfe to clear bit 4
83                   control_byte=control_byte & 0xfb;
84                   left=0; // stops the left flash
85                   break;
86         case 'D': local_output++; //turn on reversing light
87                   control_byte=control_byte+8;
88                   break;
89         case 'd': local_output=local_output & 0xf7; // AND the local output byte with 0xf7 to clear bit 5
90                   control_byte=control_byte & 0xf7;

```

Figure 12 - Slave code payload

```

main.c  main.c  main.h
85
86         case 'D': local_output++; //turn on reversing light
87                   control_byte=control_byte+8;
88                   break;
89         case 'd': local_output=local_output & 0xf7; // AND the local output byte with 0xf7 to clear bit 5
90                   control_byte=control_byte & 0xf7;
91                   break;
92         case 'E': if(lights)
93                   {
94                       local_output++;
95                       control_byte=control_byte+16;
96                       output_high(pin_B4);
97                   }
98                   break;
99         case 'e': local_output=local_output & 0xef; //AND the local output byte with 0xef to clear bit 6
100                  control_byte=control_byte & 0xef;
101
102                  break;
103
104     }
105
106     output_b(control_byte); // send the data to the b Port.
107
108 }
109
110
111

```

Figure 13 - Slave code payload (2)

Lastly, the flash control is set up. This allows the left indicator to flash. How it is coded is to AND the left and flash together with outputting high pin B2 then low repeatedly.

```

146         }
147
148
149
150 void flash_control(void)
151 {
152     if(left & flash)output_high(pin_b2);           // set the indicator to flash
153     else output_low(pin_b2);
154 }
155
156
157

```

Figure 14 - Slave code flash

Conclusion and Recommendation

To conclude this report shows how a multi-master transmissions system can be implemented on two identical PICs utilizing I2C and fully working. For a demonstration of how a corner of a car light system can be used on a master-slave system obeying how the standard car light system works. One recommendation is to store the value of the buttons pressed in multiple memory locations instead of overwriting the same one every time. This will make the system more efficient for the system and memory use.

References

- [1] Z. Y. V. a. S. O. Shaposhnikov, "Quality Assurance of Quality Management Systems," Sochi, Russia, IEEE, 2019, pp. 323-325.
- [2] M. Rouse, "Point-to-Point Protocol," Techopedia, 18 01 2017. [Online]. Available: <https://www.techopedia.com/definition/25315/point-to-point-protocol-ppp>. [Accessed 11 11 2024].
- [3] S. Dietrich, "Understanding RS232 Serial Communication Message Formats," 26 01 2023. [Online]. Available: <https://control.com/technical-articles/understanding-rs232-serial-communication-message-formats/#:~:text=When%20connecting%20two%20devices%20together%2C%20the%20transmitting%20terminal,there%20is%20no%20need%20for%20addressing%20or%20nodes..> [Accessed 11 11 2024].
- [4] NordVPN, "Point-to-multipoint communication," NordVPN, [Online]. Available: <https://nordvpn.com/cybersecurity/glossary/point-to-multipoint-communication/?msockid=0b747707e4ba6c2c1d3e64efe55a6d5c>. [Accessed 11 11 2024].
- [5] T. Denton, "How it works: CAN bus," Institute of the Motor Industry, 02 12 2020. [Online]. Available: <https://tide.theimi.org.uk/industry-latest/motorpro/how-it-works-can-bus>. [Accessed 12 11 2024].

Appendix

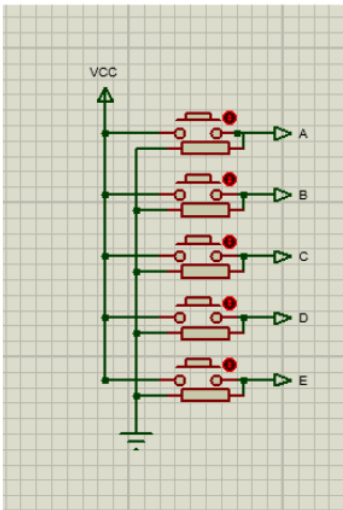


Figure 15 - Buttons Overview

ON	Off	
A	a	Brake lights
B	b	Rear light
C	c	indicator left
D	d	Reversing lights
E	e	Fog lights

Figure 16 – Tokens

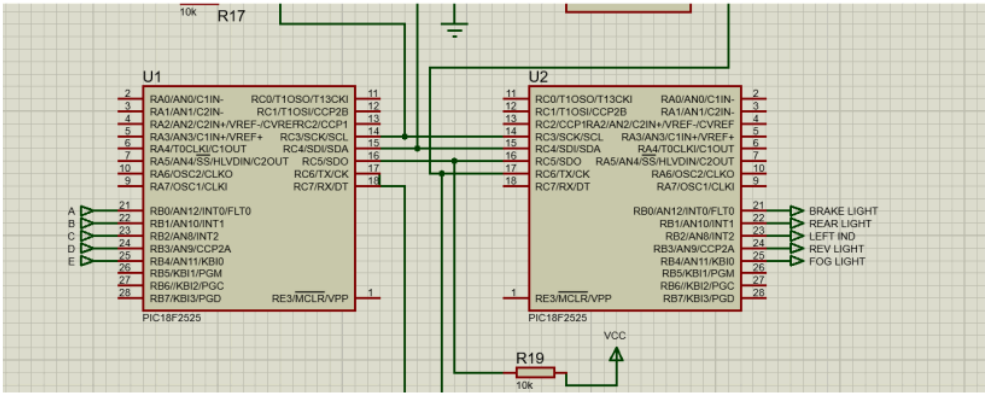


Figure 17 - I/O pins and PIC layout

```

1 #include <18F2525.h>
2 #device ADC=10
3
4 #FUSES FCMEN           //Fail-safe clock monitor enabled
5 #FUSES IESO            //Internal External Switch Over mode enabled
6 #FUSES PUT            //Power Up Timer
7 #FUSES BROWNOUT       //Reset when brownout detected
8 #FUSES BORV21         //Brownout reset at 2.1V
9 #FUSES NOPBADEN       //PORTB pins are configured as digital I/O on RESET
10 #FUSES LPT1OSC        //Timer1 configured for Low-power operation
11 #FUSES NOMCLR         //Master Clear pin used for I/O
12 #FUSES STVREN         //Stack full/underflow will cause reset
13 #FUSES NOLVP          //No Low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
14 #FUSES NOXINST        //Extended set extension and Indexed Addressing mode disabled (Legacy mode)
15 #FUSES NOPROTECT      //Code not protected from reading
16 #FUSES NOCPB          //No Boot Block code protection
17 #FUSES NOCPD          //No EE protection
18 #FUSES NOWRT          //Program memory not write protected
19 #FUSES NOWRTC         //Configuration registers not write protected
20 #FUSES NOWRTB         //Boot block not write protected
21 #FUSES NOWRTD         //Data EEPROM not write protected
22 #FUSES NOEBTR         //Memory not protected from table reads
23 #FUSES NOEBTRB        //Boot block not protected from table reads
24
25 #use delay(internal=16MHz)
26 #use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8,stream=PORT1,errors)
27 #use i2c(Master,Fast,sda=PIN_C4,scl=PIN_C3,force_hw)
28
29 #define LED PIN_A0
30 #define DELAY 500
31
32 #define hs Pin_c5

```

Figure 18 - Main.h code

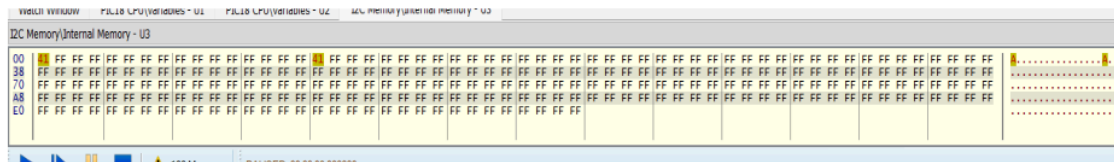


Figure 19 - Memory location of A

FINAL GRADE

GENERAL COMMENTS

71 /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

OWEN WILD**71.15****RESEARCH-INFORMED LITERATURE**

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

13.6**KNOWLEDGE AND UNDERSTANDING**

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

18.75**ANALYSIS**

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

17.5**PRACTICAL APPLICATION AND DEP.**

Your explanation is good and acknowledges that both pics speak to memory IC, only one of your rs232 virtual terminals displays the token 70%

10.5

SKILLS FOR PROFESSIONAL PRAC.



The report is well presented with sections to explain the code and include a limited number of references, and a good flowchart. 72%

10.8