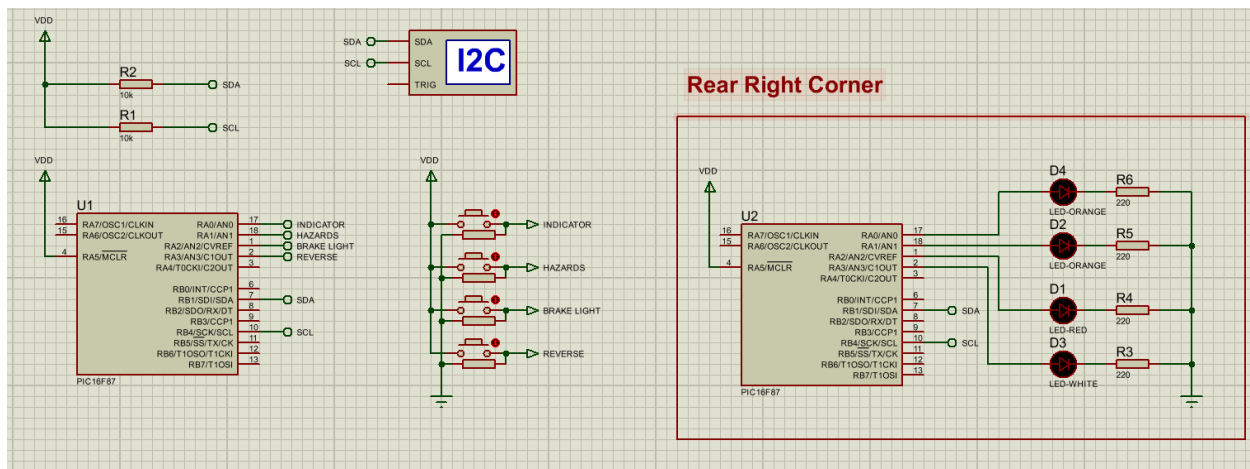


Distributed Intelligence Systems – ACDA5018

A simple token based distributed network



Jayden Fox – 2316329

Abstract

This report presents the design, implementation and evaluation of a simple token-based distributed network for lighting control on a vehicle. The system uses two PIC16F87 microcontrollers configured in a point-to-point topology and communicating via the I²C protocol to operate the rear-right corner lights of a vehicle. The system was developed and simulated using the Proteus software, with a real-world breadboard prototype also being created. Results show that the system operates reliably with stable timing and accurate output behaviour, as per the input to the master microcontroller. Quality management principles were also applied throughout development to support the reliability and safety of the system. The project demonstrates how distributed microcontroller architectures can simplify vehicle lighting control and explores the possibilities to expand the system.

Table of Contents

1. Introduction	5
2. Background Research.....	6
2.1 Point-to-Point Transmission System	6
2.2 Point-to-Multipoint Transmission System	6
2.3 Multi-Master Transmission System	6
2.4 I ² C Communication in Distributed Intelligence Systems	7
3. Design.....	8
3.1 Component List	8
3.2 Microcontroller Justification.....	9
4. Development.....	11
5. Analysis	13
5.1 Master Program Operation.....	13
5.2 Slave Program Operation.....	15
5.3 Quality management systems and continuous improvement	18
6. Results	19
7. Conclusion	23
Appendix A.....	26

Table of Figures

Figure 1 - Whole circuit schematic	8
Figure 2 – Master microcontroller setup.	9
Figure 3 – Slave PIC and output LEDs.....	10
Figure 4 – Master PIC ‘main.c’ code	11
Figure 5 – Slave PIC ‘main.c’ code.....	12
Figure 6 – Master microcontroller operation flowchart	14
Figure 7 – Interrupt Service Routine (ISR) code.....	15
Figure 8 – Main program loop	16
Figure 9 – Indicator light code	16
Figure 10 – Hazard light code	16
Figure 11 – Brake and reverse light code	16
Figure 12 – Slave microcontroller operation flowchart.....	17
Figure 13 – Indicator LED blinking when button pressed.....	20
Figure 14 – Hazards LED blinking when button pressed.	21
Figure 15 – Brake and reverse LEDs lit when respective buttons pressed	22
Figure 16 – Breadboard Circuit	26

1. Introduction

Modern electrical systems rely heavily on distributed intelligence systems to provide a reliable, low complexity and scalable system. Instead of using individual point-to-point wiring from every switch to every load, distributed systems use microcontrollers that communicate over shared networks to coordinate the behaviour of a system.

This report presents the design and implementation of a simple token-based distribution network using PIC microcontrollers for the rear right corner of a vehicle. This system is implemented using two PIC microcontrollers communicating via the inter-integrated circuit (I²C) protocol and simulated using Proteus. It uses a point-to-point master to slave transmission topology where control tokens are issued by the master to operate different lighting functions on the rear-right vehicle corner.

2. Background Research

Distributed intelligence systems rely on efficient communication architectures to operate multiple microcontrollers that control multiple outputs. The choice of transmission system affects the scalability and complexity of a system, and what abilities microcontrollers have to do with communication. This section focuses on the key transmission systems relevant in distributed intelligence systems, such as point-to-point, point-to-multipoint and multi-master systems.

2.1 Point-to-Point Transmission System

A point-to-point transmission system contains a dedicated communication link between exactly two devices. The data sent by the transmitter is received only by one receiver, with nothing else involved in the communication channel [1]. Point-to-point communication is described as the most simple and predictable form of communication, and this approach is most suited to smaller systems [1, 2]. On the other hand, if the size of a system increases, the point-to-point method would lead to increase wiring complexity, due to a larger amount of required physical connections and other transmission methods should be used.

2.2 Point-to-Multipoint Transmission System

Point-to-multipoint systems involve a single master device communicating with multiple slave devices [3]. In embedded distributed systems, point-to-multipoint communication reduces the complex wiring requirements mentioned in point-to-point communication as multiple nodes share the same communication lines. This allows for centralised logic, where the master controls when and how each slave is addressed [4].

Point-to-multipoint systems are widely used in distributed applications where multiple nodes need to be coordinated by a primary controller, such as fixed wireless broadband, sensor networks and lighting systems [5].

2.3 Multi-Master Transmission System

A multi-master transmission system allows more than one device to initiate communication on the network [6]. In this architecture, multiple controllers can transmit data independently. These systems increase flexibility and fault tolerance, however, they tend to bring complexity as they require a need for collision detection and priority handling [7]. Multi-master transmission systems are most commonly found in advanced automotive and industrial networks. The most popular example is the Control Area Network (CAN), which is multi-master based and allows any control unit to initiate communication [8, 9].

2.4 I²C Communication in Distributed Intelligence Systems

I²C is a synchronous serial communication protocol that uses two shared lines called the Serial Data (SDA) and Serial Clock (SCL) [10, 11]. The SCL line is used to synchronously clock data in or out of the target device and the SDA line is used to send data to or from target devices [10]. I²C supports addressed communication so can operate in point-to-point, point-to-multipoint and multi-master configurations, which makes it well suited to short-distance communication between microcontrollers and other devices in embedded systems.

3. Design

This section focuses on the design of the distributed system. The system will focus on the rear-right corner of a car and will control an indicator light, hazard light, reverse light and a brake light. The indicator and hazard lights will blink at 500ms intervals, as each indicator should flash constantly at a rate of not less than 60 and no more than 120 flashes per minute [12].

The system will utilise 2 PIC16F87 microcontrollers, one as a master controller and one as a slave controller. The microcontrollers are configured in a point-to-point topology, with communication handled through the I²C protocol. The master controller will be used to read the simulated switch inputs and then generate and transmit control tokens over the I²C bus [11]. Then, the slave controller will receive these tokens, verify the address, and control the assigned outputs [10, 11]. An I2C debug probe was used, as shown in Figure 1, to monitor the bus traffic between the two microcontrollers. This token-based approach ensures that all lighting commands are synchronised and consistently updated to provide a functioning, real-time system.

3.1 Component List

The system's schematic, shown in Figure 1, consists of:

- 2 x PIC16F87 (1 as master controller, 1 as slave controller)
- 1 x I²C debug probe.
- 4 x Button (1 for each LED output)
- 6 x 10k Ω resistor (2 for SDA/SCL, 4 as pull-down resistors for switch inputs)
- 4 x 220 Ω resistor
- 2 x Orange LED
- 1 x Red LED
- 1 x White LED

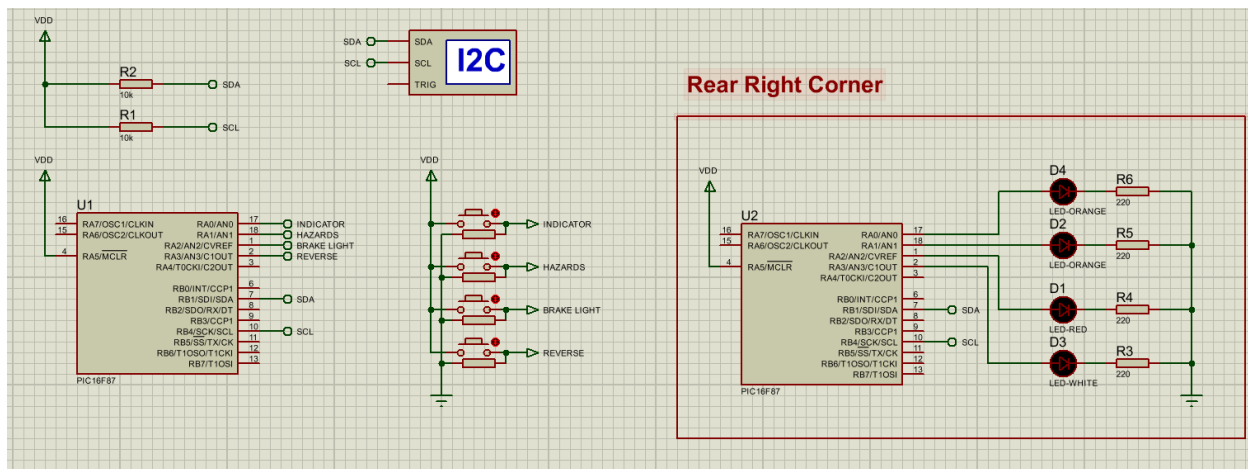


Figure 1 - Whole circuit schematic

3.2 Microcontroller Justification

The PIC16F87 microcontroller was selected as it meets the requirements of the system well. The microcontroller has a Synchronous Serial Port (SSP) module that fully supports I²C communication by handling data transmission/reception and clock generation in the background to allow the processor to perform other tasks [13, 14]. This allowed the system to achieve better timing accuracy for other tasks.

The PIC16F87 also offers an ideal number of digital I/O pins to support both the master and slave microcontrollers in the system [14]. Each button input has its own pin, and each LED output has its own pin, as well as dedicated SDA and SCL pins [14]. A larger microcontroller with more I/O pins would be impractical in this system as more pins are simply not required. Selecting the PIC16F87 microcontroller meant the system could stay as concise as possible whilst still providing enough pins for the system to operate.

The features of the PIC16F87 allowed the system to stay concise and efficient, making it ideal for this distributed system. A closer look at the setup of the two PIC16F87 microcontrollers in this system can be seen in Figure 2 and Figure 3.

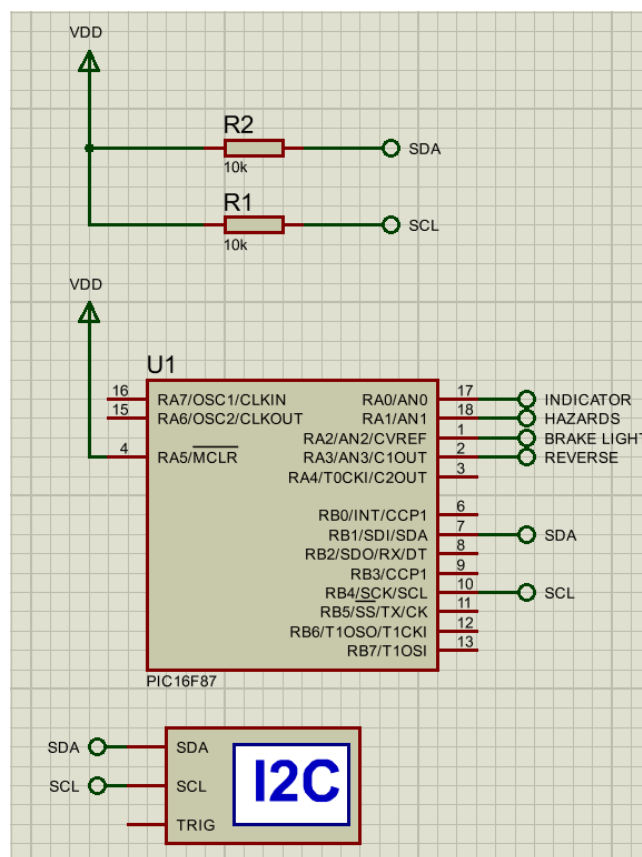


Figure 2 – Master microcontroller setup.

Rear Right Corner

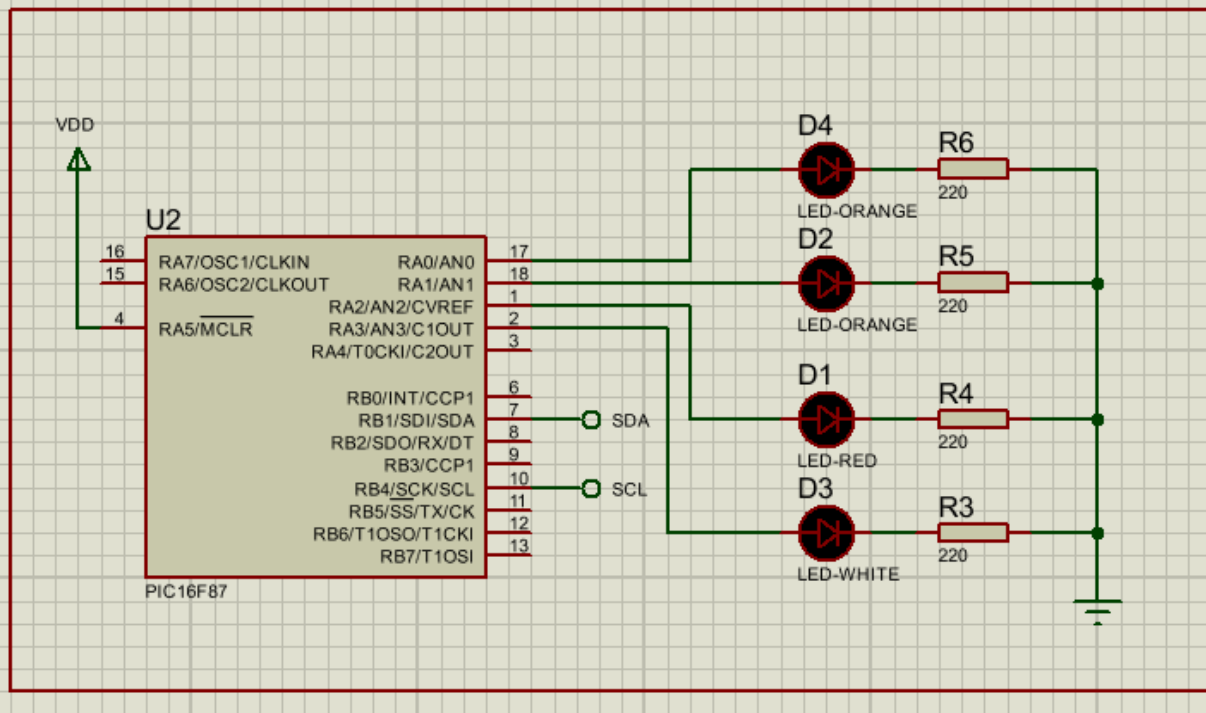


Figure 3 – Slave PIC and output LEDs

4. Development

This section shows the documented programs created to run on each microcontroller. The Proteus software was used to build and simulate the programs. The whole program used for the master microcontroller can be seen in Figure 4 and the whole program of the slave controller can be seen in Figure 5.

```
1 #include <main.h>
2
3
4 //MASTER
5
6 void main()
7 {
8
9     while(TRUE)
10    {
11
12        i2c_start();           //sends a START condition on the I2C bus
13        i2c_write(0x10);       //sends the slave device address
14        i2c_write(input_a());  //reads PORT A input and sends to I2C device
15        i2c_stop();            //sends a STOP condition to end the transmission
16        delay_ms(100);         //waits 100ms before completing
17
18    }
19
20 }
21
```

Figure 4 – Master PIC 'main.c' code

```

1 #include <main.h>
2 //SLAVE
3
4 volatile int RXByte = 0x00; //holds the most recent byte received from the master
5
6
7 #INT_SSP
8 void SSP_isr(void)
9 {
10
11     int8 I2CState = i2c_isr_state(); //reads the current I2C state
12     if(I2CState == 0) //if 0, then address received
13     {
14         i2c_read(); //used to clear the buffer after receiving the address byte
15     }
16     else if(I2CState == 1) //if 1, then data byte received
17     {
18         RXByte = i2c_read(); //read actual data byte sent by master
19     }
20 }
21 void main()
22 {
23     enable_interrupts(INT_SSP); //enable I2C interrupt
24     enable_interrupts(GLOBAL); //enable global interrupt handling
25
26     while(TRUE)
27     {
28         if(RXByte & 0B00000001) //Looking at indicator LED pin (Slave - A0)
29         {
30             output_high(INDICATOR_LED); //Indicator LED ON
31             delay_ms(500); //wait 500ms
32             output_low(INDICATOR_LED); //Indicator LED OFF
33             delay_ms(500); //wait 500ms
34         }
35         else if(RXByte & 0B00000010) //Looking at hazards LED pin (Slave - A1)
36         {
37             output_high(HAZARDS_LED); //Hazards LED ON
38             delay_ms(500); //wait 500ms
39             output_low(HAZARDS_LED); //Hazards LED OFF
40             delay_ms(500); //wait 500ms
41         }
42         output_a(RXByte & 0B00001100); //Looks at Brake and Reverse LED pins (A2 & A3)
43                                         //written directly to port A as they do not need to blink
44     }
45 }
46

```

Figure 5 – Slave PIC ‘main.c’ code

5. Analysis

This section focuses on the operation of the programs in the design and discusses the need for quality management systems to further develop operation. The programs will be broken down into key parts to further explain their function and importance.

5.1 Master Program Operation

The master microcontroller's program places a start condition on the I²C bus to signal the beginning of a new transmission. In a point-to-multipoint or multi-master system, all slaves would listen for this event. The master then sends the 7-bit slave address (in this case, 0x10) and only the slave with the address will respond, all others would ignore this message. This address check ensures the desired point-to-point communication.

The master then reads the state of 'PORT A', where each bit corresponds to one lighting switch as shown in Table 1. This byte is then transmitted to the slave as the command token. The 'i2c_stop()' line is then used to end the transmission and a 100ms delay takes place before the next token is sent to ensure stable communication and predictable update timings. A flowchart to display the operation of the master program can be seen in Figure 6.

Table 1 – Port A setup.

Bit	Pin	Car light
0	A0	Indicator
1	A1	Hazard
2	A2	Reverse
3	A3	Brake

MASTER

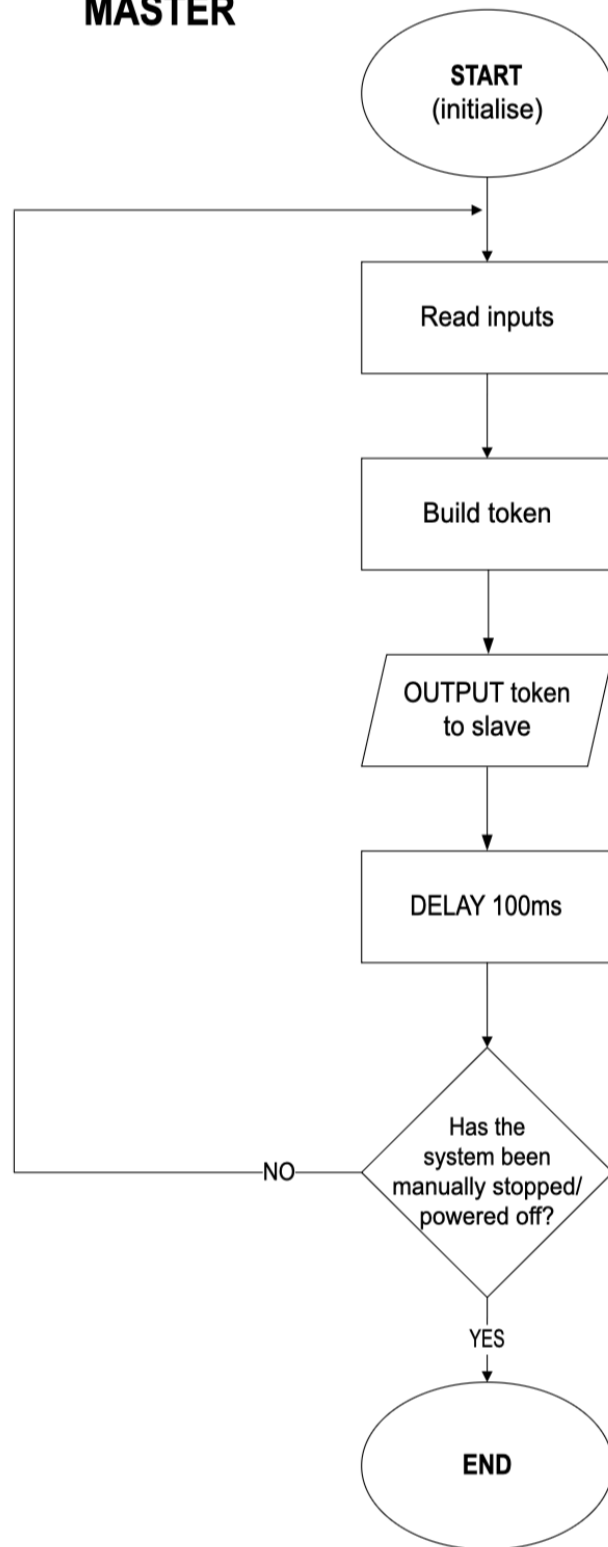


Figure 6 – Master microcontroller operation flowchart

5.2 Slave Program Operation

In terms of the slave program operation, the flowchart seen in Figure 12 displays the full operation. The slave program receives the transmitted byte from the master and controls the lighting outputs accordingly. The program uses an interrupt-driven approach to ensure reliable I²C communication.

Figure 7 shows the start of the program, containing the 'RXByte' global variable and the code for the interrupt service routine (ISR). The 'RXByte' global variable stores the most recent command byte received from the master, it is declared as volatile to prevent the compiler from potentially altering it, as it is modified inside the ISR. The ISR is triggered every time the slave receives a transmission on the I2C bus, and the 'i2c_isr_state()' variable is used to tell the slave what data has arrived.

When the 'I2CState' variable is equal to 0, this means the master has sent the slave address. When the slave detects that the received byte is an address, it simply reads and clears the buffer so it can acknowledge the master and get ready for the next byte. When the 'I2CState' variable becomes 1, this means the master has sent the command byte. The slave reads this byte and stores it in the 'RXByte' variable, where this byte will contain the lighting states in bits 0 to 3. As this process is handled through an interrupt, the slave is able to capture every incoming byte from the master, and this ensures no data is lost.

```
4 volatile int RXByte = 0x00;           //holds the most recent byte received from the master
5
6
7 #INT_SSP
8 void SSP_isr(void)
9 {
10
11     int8 I2CState = i2c_isr_state();   //reads the current I2C state
12     if(I2CState == 0)                 //if 0, then address received
13     {
14         i2c_read();                   //used to clear the buffer after receiving the address byte
15     }
16     else if(I2CState == 1)            //if 1, then data byte received
17     {
18         RXByte = i2c_read();          //read actual data byte sent by master
19     }
20 }
```

Figure 7 – Interrupt Service Routine (ISR) code

The main program loop can be seen in Figure 8. This enables the I²C interrupt and global interrupt handling to allow the slave to automatically receive data whenever the master transmits.

```

21 void main()
22 {
23     enable_interrupts(INT_SSP);           //enable I2C interrupt
24     enable_interrupts(GLOBAL);           //enable global interrupt handling

```

Figure 8 – Main program loop

The code to control the indicator light can be seen in Figure 9 and if bit 0 is set, the indicator light will blink in 500ms intervals. The blinking is handled locally and not by the master to reduce I²C traffic and make the system more efficient. Figure 10 shows that a similar template is followed for the hazard light where if bit 1 is set, the hazard LED blinks at the same rate.

```

27     {
28         if(RXByte & 0B00000001)           //looking at indicator LED pin (Slave - A0)
29         {
30             output_high(INDICATOR_LED);     //Indicator LED ON
31             delay_ms(500);                  //wait 500ms
32             output_low(INDICATOR_LED);      //Indicator LED OFF
33             delay_ms(500);                  //wait 500ms
34         }

```

Figure 9 – Indicator light code

```

35         else if(RXByte & 0B00000010)       //looking at hazards LED pin (Slave - A1)
36         {
37             output_high(HAZARDS_LED);       //Hazards LED ON
38             delay_ms(500);                  //wait 500ms
39             output_low(HAZARDS_LED);        //Hazards LED OFF
40             delay_ms(500);                  //wait 500ms
41         }

```

Figure 10 – Hazard light code

Finally, Figure 11 shows the code to control the brake and reverse lights (bits 2 and 3). These lights do not blink, so the program writes them directly to PORTA and only light when the switch inputs are pressed.

```

42     output_a(RXByte & 0B00001100);         //looks at Brake and Reverse LED pins (A2 & A3)
43                                           //written directly to port A as they do not need to blink

```

Figure 11 – Brake and reverse light code

SLAVE

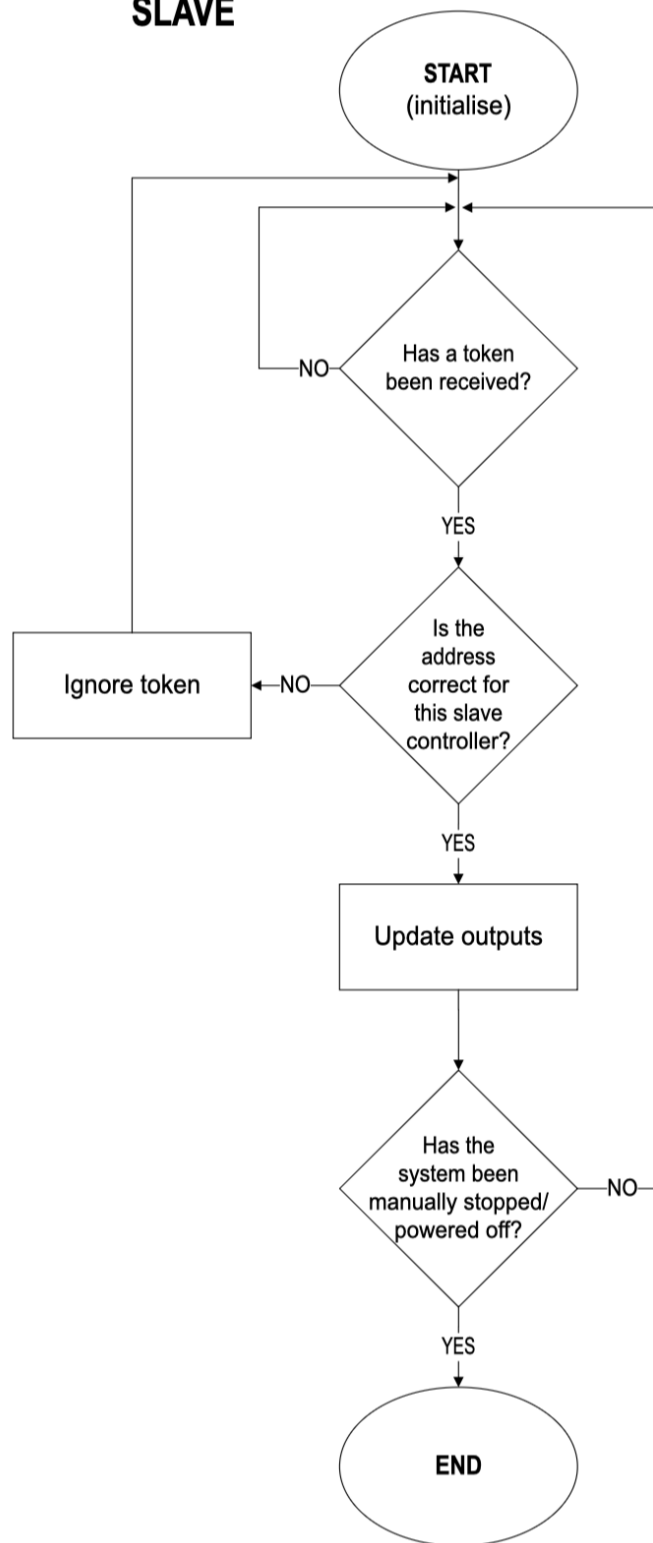


Figure 12 – Slave microcontroller operation flowchart

5.3 Quality management systems and continuous improvement

Quality management systems (QMS) are very important in vehicle distributed systems as they are typically related to safety-critical vehicle lighting and signalling functions. A structured and thorough quality management approach makes sure that systems are reliable, as easy as possible to maintain, and that they are compliant with engineering standards throughout their lifecycle [15].

Using a QMS framework like ISO 9001 in the system would ensure that all development steps follow documented procedures, such as code version control, the validation of circuit schematics and thorough testing of I²C communication [16]. This was done through the multiple simulations and tests the system went through throughout its creation. The circuit was also replicated onto a breadboard to gain an understanding of its real-world application, as shown in Appendix A. The ISO 9001 framework mentions risk-based thinking and process control, and both of these are important in distributed systems as failures or incorrect outputs could lead to unsafe scenarios that cause harm.

Another core element of QMS and system safety is continuous improvement, which can help to form strategies that focus on danger prevention and improvement [17, 18]. After the system is first put into use, it should be reviewed to identify areas of weakness and opportunities for improvements [18]. As an example for this system, the blinking logic is currently handled in software using delays, and this could be improved by using hardware timers to increase efficiency and responsiveness of the system.

By following these QMS principles during the system's design and development stages, it is shown that it allows the system to be more reliable and easier to maintain, whilst also aligning with industrial engineering standards.

6. Results

Section 6 shows the simulation results obtained from the system. The system was shown to function correctly as when a switch input is pressed, the related pins on each microcontroller are high and the correct output LED is lit. The indicator and hazard LEDs also blink in 500ms intervals when the switch input is pressed, as shown in Figure 13 and Figure 14. The results of pressing the brake and reverse switch inputs can also be seen in Figure 15. Table 2 shows the full results of the simulation.

Table 2 – System simulation results

Switch pressed	Pin	Expected Result	Actual Result
Indicator	A0	Output RA0 high, indicator LED blinking in 500ms intervals.	Output RA0 high, indicator LED blinking in 500ms intervals.
Hazard	A1	Output RA1 high, hazard LED blinking in 500ms intervals.	Output RA1 high, hazard LED blinking in 500ms intervals.
Brake	A2	Output RA2 high, brake LED lit.	Output RA2 high, brake LED lit.
Reverse	A3	Output RA3 high, reverse LED lit.	Output RA3 high, reverse LED lit.

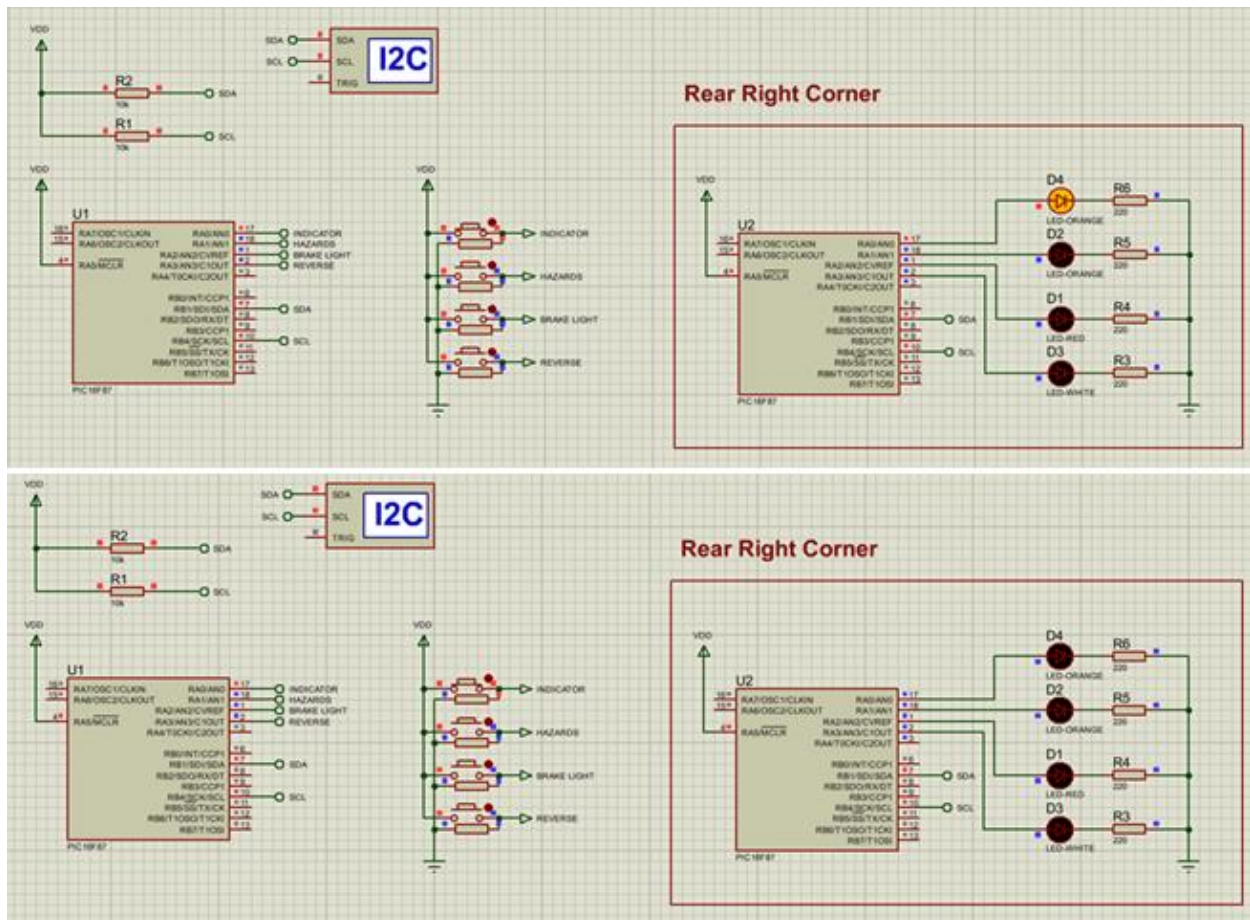


Figure 13 – Indicator LED blinking when button pressed.

7. Conclusion

This project demonstrated a successful token-based distributed network using two PIC16F87 microcontrollers, which communicated through the I²C protocol and point-to-point transmission. The system met all functional requirements for the rear right corner lighting of a vehicle, with it being able to control the indicator, hazard, brake, and reverse lights. The master microcontroller successfully read the switch inputs and sent control tokens, and the slave microcontroller received these tokens and activated the correct outputs. The use of an interrupt driven approach on the slave ensured that no data was missed, so the system operation could stay predictable and stable throughout simulation.

The design also highlighted the advantages of distributed intelligence in automotive applications. The wiring requirements of the system were not complex as inputs were handled by one microcontroller, and outputs were controlled by the other. The implementation also demonstrated how point-to-point I²C communication can be used to create a compact and efficient system suitable for real-time vehicle lighting control. This system was replicated onto a breadboard to provide a practical view of the system in a real-world application.

From a quality management perspective, the project emphasised the importance of a structured development process and continuous improvement. The system was able to remain reliable and maintainable throughout its development due to the application of ISO 9001 principles. Following these principles meant that the system aligned with engineering standards to ensure safety in vehicle use.

In terms of recommendations for future work, the system could be expanded to control all corners of a vehicle. This would require the use of point-to-multipoint transmission to allow a single master controller to communicate with multiple slave controllers. This would expand the system to better reflect real world vehicle lighting control. There is also a possibility to improve the components used, such as introducing a 3-position switch to control a left and right indicator to boost the system's similarity to real-world applications. As mentioned previously, the blinking logic currently relies on software delays. This could be improved by replacing the software delays with hardware timers, which would provide more accurate timing performance.

Overall, the project successfully demonstrated how a simple token-based distributed network can deliver reliable real-time vehicle lighting control, whilst also providing a strong foundation for future expansion and continuous improvement.

References

- [1] A. S. Tanenbaum, Computer Networks, Pearson Education India, 2011.
- [2] W. Stallings, Data and Computer Communications, Pearson Education India, 2007.
- [3] T. S. Rappaport, "Wireless Communications: Principles and Practice," *Microwave Journal*, vol. 45, no. 12, 2002.
- [4] Z. Hocenski, T. Keser and K. Nenadic, "Distributed intelligent control of car lighting system with fault detection," in *IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society*, Taipei, Taiwan, 2007.
- [5] J. K. Sastry, J. V. Ganesh and J. S. Bhanu, "I2C based Networking for Implementing Heterogeneous Microcontroller based Distributed Embedded Systems," *Indian Journal of Science and Technology*, vol. 8, no. 15, pp. 1-10, 2015.
- [6] A. Anagha and M. Mathurakani, "Prototyping of dual master I2C bus controller," in *2016 International Conference on Communication and Signal Processing (ICCSP)*, Melmaruvathur, India, 2016.
- [7] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications, Boston: Springer, 1997.
- [8] S. Corrigan, "Introduction to the Controller Area Network (CAN)," Texas Instruments, 2002.
- [9] Tindell, Hansson and Wellings, "Analysing real-time communications: controller area network (CAN)," in *1994 Proceedings Real-Time Systems Symposium*, San Juan, USA, 1994.
- [10] J. Wu, "A Basic Guide to I2C," 2022. [Online]. Available: <https://www.ti.com/lit/an/sbaa565/sbaa565.pdf>. [Accessed 29th April 2026].
- [11] Microchip, "I²C Master Mode," 2018. [Online]. Available: <https://ww1.microchip.com/downloads/en/AppNotes/I2C-Master-Mode-30003191A.pdf>. [Accessed 26th April 2026].
- [12] U. Government, "The Road Vehicles Lighting Regulations 1989," 1989. [Online]. Available:

[https://www.legislation.gov.uk/ukxi/1989/1796/schedule/7#:~:text=\(a\)%20Every%20indicator%20\(other,the%20speed%20of%20the%20vehicle..](https://www.legislation.gov.uk/ukxi/1989/1796/schedule/7#:~:text=(a)%20Every%20indicator%20(other,the%20speed%20of%20the%20vehicle..) [Accessed 5th May 2026].

- [13] Microchip, "Section 17. Master Synchronous Serial Port (MSSP)," 1997. [Online]. Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/31017a.pdf>. [Accessed 4th May 2026].
- [14] Microchip, "PIC16F87/88 Data Sheet," 2005. [Online]. Available: <https://ww1.microchip.com/downloads/en/devicedoc/30487c.pdf>. [Accessed 4th May 2026].
- [15] R. Tricker, *Quality Management Systems*, Routledge, 2019.
- [16] I. Abuhav, *ISO 9001: A complete guide to quality management systems*, CRC Press, 2017.
- [17] L. Sanchez and B. Blanco, "Three decades of continuous improvements," *Total Quality Management & Business Excellence*, vol. 25, no. 9-10, pp. 986-1001, 2014.
- [18] G. D. Robson, *Continuous Process Improvement*, Simon and Schuster, 2010.

Appendix A

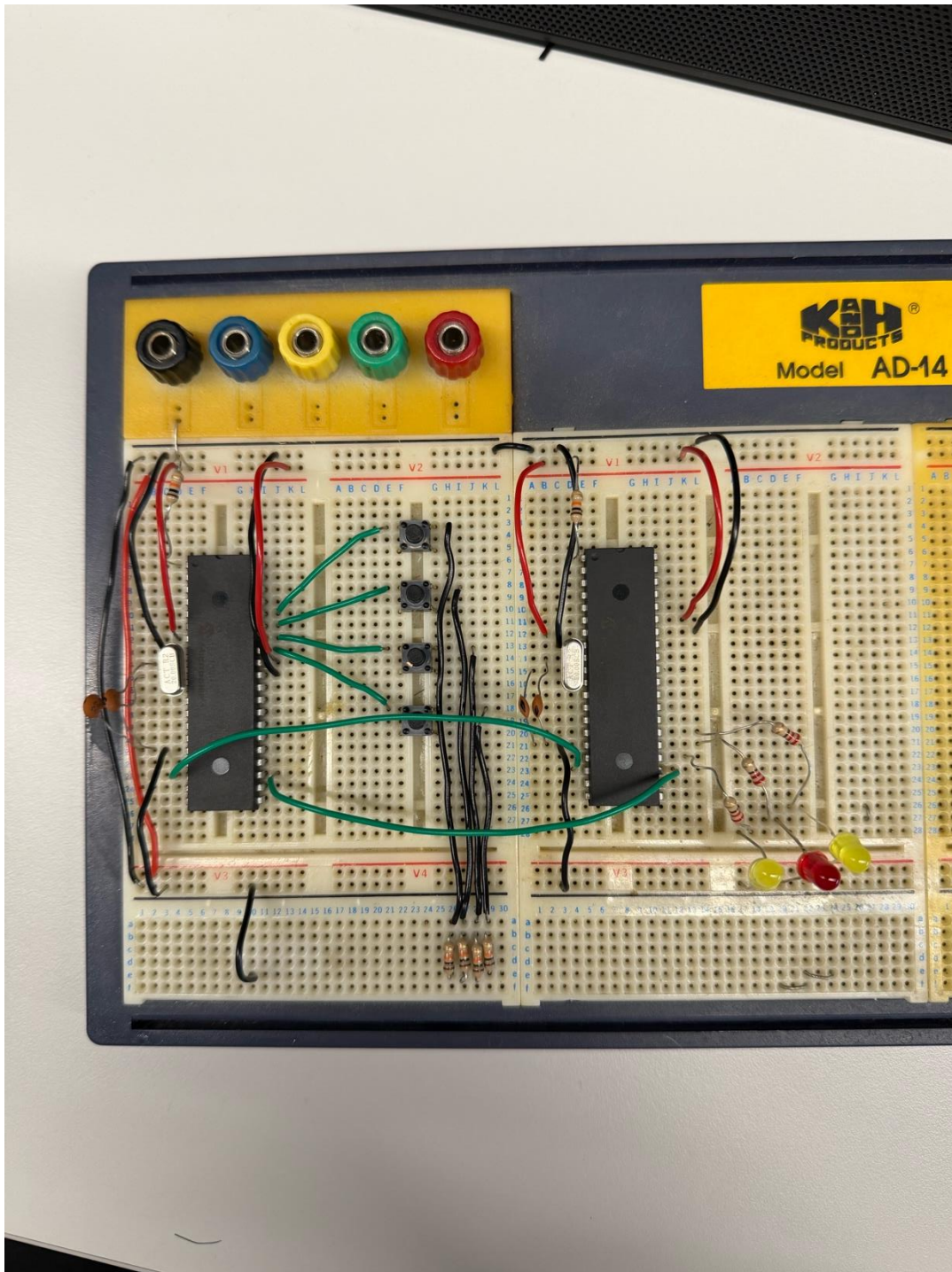


Figure 16 – Breadboard Circuit