**Communication 3: CAN Bus**



*By: Johnson Domacasse (#4471709)*

*December 2023*

# Abstract:

*A controller area network bus (CAN bus) is a communication network protocol that is used in a variety of sectors around the world, primarily automotives. Once configured properly, it can act as a reliable, robust communication system. It offers protection against electronic noise.*

Contents

[Abstract: 1](#_Toc153784296)

[List of Figures: 2](#_Toc153784297)

[List of tables: 2](#_Toc153784298)

[Introduction: 3](#_Toc153784299)

[Procedure: 3](#_Toc153784300)

[1. Circuit: 3](#_Toc153784301)

[2. System behaviour: 5](#_Toc153784302)

[3. Message protocol 7](#_Toc153784303)

[4. Implementation 8](#_Toc153784304)

[4.1 Sender: 8](#_Toc153784305)

[4.2 Receivers: 9](#_Toc153784306)

[Conclusion 9](#_Toc153784307)

[References: 10](#_Toc153784308)

# List of Figures:

Figure 1: inspiration circuit………………………………………………………………………………………………………………..3

Figure 2: example of an additional device in a CAN network……………………………………………………………..4

Figure 3: CAN module WITH jumper………………………………………………………………………..……………….……….4

Figure 4: Complete system circuit……………………………………………………………………………………….……..……..5

Figure 5: Sender state machine………………………………………………………………………………………………….………6

Figure 6: receiver high beam state machine………………………………………………………………………………………6

Figure 7: receiver blinking state machine…………………………………………………………………………………………..7

Figure 8: CAN message packet…………………………………………………………………………………………………………..7

# List of tables:

Table 1: Messaging protocol……………………………………………………………………………………………………………..8

Table 2: messages’ byte values………………………………………………………………………………………………………….8

# Introduction:

A controller area network (CAN) bus is one of the many communication protocols that exist today. CAN is primarily used in the automotive industry but they can be found in other sectors as well. The can protocol is serial type communication, It has a bus topology, it is half duplex, and is asynchronous. All of these will be discussed within this report. **[1]**

# Procedure:

*In this section of the report, the steps taken towards the implementation and the actual implementation will be discussed. This section consists of the circuits that were used to make the implementation. The behaviour of the system which will be shown and explained in the form of flowcharts. The messaging protocol and why this was so important. Finally, the actual implementation.*

***Note:*** *This implementation was built around the use of 4 buttons instead of the joystick. The logic should stay the same. The implementation is built around 1 sender a 1 receiver due to the lack of CAN devices. Again the logic shouldn’t matter since there will be 2 receiver codes provided.*

## Circuit:

The circuit that was used as inspiration is the following circuit.**[3]** See figure **1**.

A diagram of a circuit board

Description automatically generated

*Figure 1: Inspiration circuit.*

There are 2 important things that can be determined aside from the wiring from the module to the MCUs. These two things are the jumpers that need to be placed on each modules, and the H and L lines being connected to each other.

The jumpers that are placed on each module is to ensure that the CAN network has proper termination. The jumpers absorbs the CAN signal energy so that it is not reflected from the end of the cables back along the network to cause interference**[1][4].** These jumpers do not need to be placed in a CAN network with multiple devices. Take figure **2** for example. What is important, is that if there is a CAN network with multiple modules attached to the H and L lines, then the jumpers need to be placed on each end of the network. Figure 3 depicts on how the jumpers on J1 were used inside this project.

A diagram of a circuit board

Description automatically generated

*Figure 2: Example of an additional device in a CAN network.*

*A hand holding a small blue circuit board

Description automatically generated*

*Figure 3: CAN module WITH jumper.*

As mentioned before, this implementation was built without the use of a joystick. The functionality of this was replaced by buttons. See figure **4** for the complete circuit of the implementation.

A circuit board with wires and colorful buttons

Description automatically generated

*Figure 4: Complete system circuit.*

What is important to note in this circuit is that the LEDs are not controlled by the sender device. In this case the Redboard. They are controlled by the Uno. These two MCUs are connected to each other via a CAN bus. The button inputs are processed by the sender device. The same as if there was a joystick. The button layout is as follows:

* Red button: simulates holding the joystick left.
* Green button: simulates holding the joystick right.
* blue button: simulates holding the joystick up.
* yellow button: simulates holding the joystick down.

## System behaviour:

*This section shows the behaviour of the system in the form of state machine. You will find one state machine for each implementation. figure* ***5*** *shows the sender. figure* ***6*** *shows the receiver blinking. figure* ***7*** *shows the receiver high beam.*

*A diagram of a software company

Description automatically generated*

*Figure 5: Sender state machine.*

*A diagram of a computer

Description automatically generated*

*Figure 6: receiver high beam state machine.*

*A diagram of a computer flowchart

Description automatically generated*

*Figure 7: receiver blinking state machine*

## Message protocol

CAN is a message based protocol. Meaning that the messages are sent into the form of frames to the other devices in the CAN network. These frames include bytes like the ID, the length of the data, the data etc.  
As previously mentioned, Can is an asynchronous, half-duplex communication protocol. This means that the messages can only be sent one at a time**[1].** A typical can message can look something like what can be seen in figure 5.

A diagram of software level

Description automatically generated

*Figure 8: CAN message packet****[4]****.*

For the purpose of this implementation, we use the “mcp\_can” library to make this message easier to send. What is important is that when we use the built in sending function to send the predefined ID, the length of the data that we want to send and the actual data. See table **1** for further insight.

Another important aspect of the message protocol is the ID. The ID is primarily based on the messages that you are sending from one CAN device to another CAN device. The ID should not be confused as an ID per **device.** The ID also defines the priority of the messages in a CAN bus. The lower the ID, the higher the message’s priority**[2][4].** The way it was implemented in this case is there are two different IDs for two different sections. This is one of the options, another would be to have every message having their own ID so that the implementation can be more modular. For the sake of simplicity and understanding, only 2 different IDs will be used. One for the blinking messages and the others for the high-beam messages. See table **1.**

*Table 1: Messaging protocol*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **ID** | **Message**  **(in byte array)** | **Length**  **(in Bytes)** | **Frame** | **Description.** |
| 1. | 0x100 | “LEFT” | 1 | (0x100, 1, LEFT) | This message turns the left blinker on. |
| 2. | 0x100 | “RIGHT” | 1 | (0x100, 1, RIGHT) | This message turns the right blinker on. |
| 3. | 0x100 | “OFF” | 1 | (0x100, 1, OFF) | This message turns both the right and the left blinkers off. |
| 4. | 0x200 | “ON” | 1 | (0x200, 1, ON) | This message turns both high beam LEDs on. |
| 5. | 0x200 | “OFF” | 1 | (0x200, 1, OFF) | This message turns both the high beam LEDs off |

**Note:** The “messages” in the table each represent a different byte value that has been assign in the implementation. See table **2** below to see the values of each message.

*Table 2: Messages’ byte values.*

|  |  |  |
| --- | --- | --- |
|  | **Message** | **Value (in hexadecimal bytes)** |
| 1. | “RIGHT” | 0x01 |
| 2. | “LEFT” | 0x02 |
| 3. | “ON” | 0x03 |
| 4. | “OFF” | 0x04 |

## Implementation

*This section will cover a majority of the decisions that have been taken in order to produce an implementation. This section will show how the implementations are modular, extendable, and efficient.*

### 4.1 Sender:

The modularity of the sender application can be found in the functions that are declared and implemented to perform different tasks. Take the UpdateState() and the SendMessage() functions. There is also additional functions like the debounce button functions that offer easy understanding.

The implementation is extendable due to the use of the switch case and the enumerations. If the program were to be improved upon with additional states, this can be done easily by adding this state into the enumeration and then adding that case into the switch-case.

The implementation is efficient because of the spam guard within the send message function. Instead of constantly spamming the network with messages, you only send it once and then change the previous state to the current state. This ensures that if the implementation stays within one state, it will not send it again but only after the state changes. See figure **5.**

### 4.2 Receivers:

The modularity of the receivers can both be found in the functions that were declared and implemented in order to perform the specific tasks like turning the high beam on and off or blinking an LED. This made the loops more readable as well.

The implementations here can be extended to control additional LEDs for example because of the variables that were used to indicate the state of each LED. If the program were to be extended, than its an additional variable, which can then be used in the switch-case.

The implementation is efficient because before it actually execute any of the message frames that was meant for it, it checks for 2 things, if the message ID matches the ID within the implementation and the size. If these two things matches, then it performs the tasks based on the message it has received in the switch-case.

All three implementations are also error proofed, by adding a check to see if the communication has failed or not. Additionally, inside of the switch case, you can have situations where the ID and the length match but the message is unknown. Error cases are also implemented to throw these exceptions. Finally, all of the buttons were debounced, to ensure that the system works as smooth as possible.

# Conclusion

In conclusion, the sender implementation and the receiver implementation that are connected to each other via a CAN bus network can properly showcase the functionality of what CAN bus is supposed to be. There are some aspects of this assignment I would like to do differently in a large scale project compared to what was done in this assignment. Take the ID for example. In this implementation, the priority of the IDs were not considered. Along with this the IDs were not based on specific tasks but specific sections that perform the same/similar tasks. In a large scale project this could be done differently to have modularity and extendibility (more IDs for more tasks).

In summary, the CAN bus implementation presented in this report serves as a robust, modular, and extendable solution for communication between devices. This assignment can serve as a good source to come back to in case another CAN implementation needs to be made.

# References:

**[1]** - YouTube. (2023, July 9). *Can bus: Serial communication - how it works?*. YouTube. <https://www.youtube.com/watch?v=JZSCzRT9TTo&t=502s>

**[2] -** YouTube. (2017, July 21). *Can bus explained - a simple intro [v1.0 | 2019]*. YouTube. <https://www.youtube.com/watch?v=FqLDpHsxvf8&t=119s>

**[3] -** Staff, L. E. (2023, April 25). *Create your own can network with MCP2515 modules and Arduino*. Last Minute Engineers. <https://lastminuteengineers.com/mcp2515-can-module-arduino-tutorial/#google_vignette>

**[4] -** Fowler, D. S. (2020, August 18). *CAN Bus Wiring Diagram, a Basics Tutorial*. CAN Bus Wiring Diagram, a Basics Tutorial | Tek Eye. <https://tekeye.uk/automotive/can-bus-cable-wiring>