**Communication Challenge 3: CAN Bus**



*By: Johnson Domacasse (#4471709)*

*Oktober 2023*

# Abstract:

*A controller area network bus (CAN bus) is a communication network protocol that is used in a variety of sector around the world. Like any protocol it has its advantages and drawbacks. Great aspects revolving around CAN bus is its reliability and easy usage.*

Contents

[Abstract: 1](#_Toc149686064)

[Introduction: 3](#_Toc149686065)

[Procedure: 3](#_Toc149686066)

[Part 1: 3](#_Toc149686067)

[Part 2: 5](#_Toc149686068)

[Results: 6](#_Toc149686069)

[Conclusions: 7](#_Toc149686070)

[References: 8](#_Toc149686071)

# Introduction:

The focus of this document is to hold all of the information that was gathered and processed regarding the CAN bus. CAN stands for Controller Area Network. common digital data network used in automotive, industrial, medical and scientific systems. The CAN bus is used for routing sensor data between pieces of equipment. The main advantages are high resilience to noise, reliability, low cost, simple wiring and ease of use. The disadvantages are that the data packet lengths are small, transmission rates are low and the message transmission cycle time can vary. **[1]**

*NOTE: For the sake of this assignment, I will not be using the provided link code as this is outdated. The only thing I will keep is the circuit that was used.* **[2]**

# Procedure:

## Part 1:

In this section our primary goal is to make 2 CAN bus devices communicate with each other. From here we need to decide if we are going to use jumper J1. Once we have the communication ready, all we have to do is make the programs so that the sender can switch an LED on and off on the receiver side. This is done simply by typing in the serial monitor.

### Termination:

In this assignment we will not be using jumper j1 because we are working with just 2 modules. However if we are working with multiple modules then we need to consider applying the jumper j1 so that it can function as the resistor. The CAN bus needs to have a termination resistor at each end. The termination resistor has a value of around 120 ohms. The resistor absorbs the CAN signal energy so that it is not reflected from the end of the cables back along the network to cause interference.   
 Considering that the only thing we need to make sure in our system is if the jumper j1 is in place or not, it is fairly easy to connect them. In our case since we are using just 2 modules, they need to be in place.**[3]**

For this we will not be needing 8 data bytes so 1 is sufficient. See figure 1 for the circuit.

A diagram of a circuit board

Description automatically generated

*Image 1: Can circuit.*

For this part, a simple state machine was developed to toggle an LED on and off based on the message that is received on the receiver side. See figures 2 and 3 for both state machines.

A diagram of a computer flow

Description automatically generated  
*Figure 2: transmitter state machine.*

A diagram of a computer

Description automatically generated

*Figure 2: receiver state machine*

## 

## Part 2:

For this part of the assignment the goal is to change the state of the LEDs depending on what is received from the transmitter. See figure 4 for the circuit that was used. There are some constraints we are working with. The first being that we do not have 3 devices available to us so it is difficult.

### CAN ID:

*Before we begin with this implementation, there is a section in the programming that I would like to address. The id of the can module. Previously the explanation and the importance of it was not given properly so this section is to give a brief description of what its purpose and importance is.*

When working with 2 modules connected to each other in a CAN network it may seem unimportant. Yet it still is. In a large-scale CAN network the purpose of the ID becomes more significant. Each CAN module/node must have a specific ID. This is the ensure that when the data is received from one node to another it can be easily determined from where the message came from. For example if you have the ID 0x100 that generally sends messages like “on” and “off”, then you can easily check whether that node works fine by just checking the id. It can also work to our benefit in software. We first have a check whether the node ID matches the ID we expect to receive from, then we can work out what the message says.

This is one of the 3 important data that needs to be sent from node to node **[1].**

A diagram of a circuit board

Description automatically generated

*Figure 4: extra components on circuit.*

### Implementation:

The same idea we had in part 1 now goes into part 2. We simply added a joystick that will change the messages that are being sent to the receiver, in this case our Arduino uno. On the other side, you will have 2 LEDs that will be controlled based on the message that is being sent.

The plan here is to implement a switch case to handle all of the different states that are being sent and received. For example, the four states on the receiver end would be to blink either the left or right side of the LED, to turn both LEDs on and to turn them off.

A state machine to simulate this behaviour will be provided. You may notice that it follows the same principal of reading on the receiver end and then going into a specific state based on the message it received. On the transmitter size, we made some changes to the button reading. Now it works with reading a joystick. See figure 5 below.

A diagram of a computer

Description automatically generated

What I will also provide is a messaging table for the messages that are being transmitted to the can modules. Based on what the message is, the send message function from the can library will change. What we need to send is an id, the size of the message and the actual message. I also send the mode of the external interrupt of the module (which is just 0 since we don’t use interrupts in our implementation). Since we are using only 1 module, only 1 ID was needed. The messages that are being sent then to the receiver are byte values and on the receiver size they can be interpreted to turn the LEDs off or on. See table 1 bellow.

|  |  |  |
| --- | --- | --- |
| **Description** | **Address** | **Message** |
| Blink the left LED. | 0x100 | sendMsgBuf(ID, externalintmode, msgSize, blinkleft) |
| Blink the right LED. | 0x100 | sendMsgBuf(ID, externalintmode, msgSize, blinkRight) |
| Turn both LEDs on. | 0x100 | sendMsgBuf(ID, externalintmode, msgSize, LEDsOn) |
| Turn both LEDs off. | 0x100 | sendMsgBuf(ID, externalintmode, msgSize, LEDsOff) |

*Table 1: messaging table*

# Results:

Initially the link that was provided to us is outdated as stated in the library readme file itself. Once some issues were resolved, we were able to send 8 bytes to the receiver. For the purpose of part A, we only needed to send one. The state machine that was made, made it so that if the transmitter sends a 1, the LED will turn on and if the transmitter sends a 0, it will turn it off.

# Conclusions:

In the end the conclusion is somewhat satisfactory. All the requirements are met.

In terms of efficiency, the only thing that could’ve been done differently is the use of delays. Instead we would opt for MILLIS. However, during testing this did not seem to have much of an effect on the actual working of the system. Performance wise we think this is a good implementation.

If others wish to extend this software it can also be done quite easily. Since we are using a switch case to handle all of the states, adding a new state would simply be creating a new case on each side of the CAN bus.

# References:

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

**[2] -** Administrator. (2023, March 13). *MCP2515 can bus interface tutorial*. ElectronicsHub. <https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/>

**[3] -** *MCP2515 CAN bus interface module*. ProtoSupplies. (2023, November 9). <https://protosupplies.com/product/mcp2515-can-bus-interface-module/>