# Controller Area Network (CAN) and QFSAE

Ethan Peterson

November 30, 2019

## 1 CAN Bus Protocol

### 1.1 Background

Controller Area Network (CAN) is a communications protocol that is standard for electronics in the automotive industry. The protocol was introduced by the Society of Automotive engineers (SAE) in 1986. CAN is primarily used for communication between different electronics systems in different locations on the car. For instance, sensor systems on the back of vehicle would be able to read data from the Engine Control Unit (ECU). In the case of the team's car, the ECU is the primary CAN device that other systems communicate with. The ECU provides a variety of metrics about the engine such as RPM, throttle position, ignition angle and many more. These metrics are read from the ECU over CAN by using their CAN ID. Each message must have a unique ID from the IDs in use by other devices on the car. It is also important to note that only one message can be on a CAN bus at a time. Messages with more "dominant" IDs will take precedence on the bus. As a result, there must be careful consideration when selecting an ID for a certain message depending on its priority overall.

### 1.2 Data Transmission and Addressing

CAN is two wire half-duplex serial based communication protocol. Half duplex means that the CAN bus can only be used to send or receive messages in an alternating fashion and not at the same time as with full-duplex protocols. CAN messages consist of an identifier (CAN ID) and a data frame. There are several standards for the size of the message frames. The ECU on the QFSAE car implements the Society of Automotive Engineers (SAE) J1939 standard. This standard uses 29 bit message identifiers and 64 bit data frames. Since CAN is half-duplex, only one message can be on the bus at a time. However, CAN can still be used to create large networks of CAN devices despite this through its implementation of a priority bus. CAN defines "dominant" and "recessive" bits where logical 0 is dominant. This allows to CAN to resolve conflicts on the bus. The table below summarizes this process.

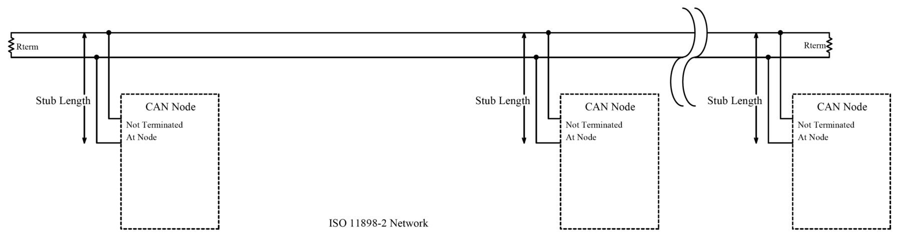
|  | CAN ID Bits |  |  |  |  |
|---|---|---|---|---|---|
|  | Start Bit | 3 | 2 | 1 | 0 |
| Node 3 | 0 | 0 | 0 | 0 | 1 |
| Node 4 | 0 | 0 | 1 | Stopped Transmitting | N/A |
| CAN Bus Data | 0 | 0 | 0 | 0 | 1 |

Table 1: Demonstrating how CAN chooses the most "dominant" ID on the bus in the case of a conflict between messages.

As a result of 0 being the dominant bit on the bus it is the smallest CAN IDs that have the highest priority on the bus. This property of the bus must be considered when selecting CAN IDs for custom messages outside of those broadcast by the ECU. For example, a message indicating to the dash that a gear shift happened is likely of a higher priority than updating the RPM.

### 1.3 Wiring

CAN consists of a two wire interface where the first two nodes connected to one another should have two 120Ω terminating resistors connected on either end of the bus. Nodes are defined as devices connected to the bus which can send or receive CAN messages. The diagram below is an example CAN network with three nodes.



## 2 CAN on the Q19 Car

In the case of the Formula car, The ECU has a terminating resistor and so does the end of the car's CAN line where the CAN to USB dongle is found. Thus, additional CAN peripheral should omit the terminating resistor to keep the bus functioning properly.

### 2.1 Using the Testbench

Given below is Arduino code for a CAN test bench. The Arduino INO file can be found here.

```
// CAN Testbench
// runs between two CAN shields and adjusts the
   brightness of an LED
// connected to the receiver based off potentiometer
   readings from the sender

#include <SPI.h>
#include "mcp_can.h"

#define SPI_CS_PIN 9

// Set to 0 to run receiver code
#define SENDING 1

// LED and Potentiometer Definitions
#define LED_CATHODE 7
#define LED_PWM 6

#define POT_VCC A0
#define POT_WIPER A1
#define POT_GND A2

MCP_CAN CAN(SPI_CS_PIN);

void setup() {
  Serial.begin(115200);

  // SETUP LED FOR OUTPUT
  pinMode(LED_CATHODE, OUTPUT);
  pinMode(LED_PWM, OUTPUT);
  digitalWrite(LED_CATHODE, LOW);

  //SETUP POTENTIOMETER FOR BRIGHTNESS READ
  pinMode(POT_VCC, OUTPUT);
  pinMode(POT_WIPER, INPUT); // WIPER PIN
  pinMode(POT_GND, OUTPUT);
  digitalWrite(POT_VCC, HIGH);
  digitalWrite(POT_GND, LOW);

  while (CAN.begin(CAN_500KBPS) != CAN_OK) {
    Serial.println("CAN BUS init failure");
    Serial.println("Trying again");
    delay(100);
  }
  Serial.println("CAN Bus Initialized!");
}

void loop() {
  unsigned char len = 0;

  // messages have max length of 8 bytes
  unsigned char buf[8];

  // Check if there is a message available
  if (!SENDING && CAN_MSGAVAIL == CAN.checkReceive())
   {
    CAN.readMsgBuf(&len, buf);
    unsigned long id = CAN.getCanId();
    Serial.print("Getting Data from ID: ");
    Serial.println(id, HEX);

    for (int i = 0; i < len; i++) {
      Serial.print(buf[i]);
      Serial.print("\t");
    }
    // Brightness of LED depends on potentiometer
   position on the sending CAN node
    Serial.print("writing to LED: ");
    Serial.println(buf[0]);
    analogWrite(LED_PWM, buf[0]);
    Serial.print("\n");
    Serial.println("END OF MESSAGE");
  } else {
    unsigned char message[8] = {0, 0, 0, 0, 0, 0, 0,
   0};
    unsigned long sendingID = 0x00;
    unsigned char reading = map(analogRead(POT_WIPER),
    0, 1023, 0, 255);
    Serial.println(reading);
    message[0] = reading;
    CAN.sendMsgBuf(sendingID, 0, 8, message);
    delay(100);
  }
}
```

CAN Testbench