Fosse report

Chirag Rathore 19BEE1025

Abstract

In this we have implemented a simple Can bus protocol in python CAN stands for Controller Area Network, it is used to allow micro controllers and devices to communicate with each other within a vehicle without a host computer which allows for control and data acquisition. These devices are also called Electronic Control Units (ECU) and they enable communication between all parts of a vehicle. CAN is a serial communication bus designed for industrial and automotive applications. For example, they are found in vehicles, farming equipment, industrial environments, etc. The real example of CAN Bus application can be found in speed car data sensor that can be transferred to the rpm indicator. .

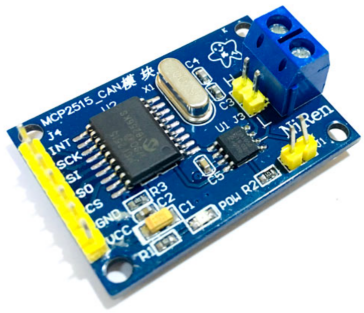
Chapter 1

Implementation of can bus protocol

1.1 preliminaries

A typical can bus is shown in fig (a) . Can bus is a message based protocol that can be used for multiple device communication. Can communication range is in range 50kbps to 1mbps,with the distance range of 40 meters(at 1mbps) to 1000 meters (at 50kbps) .

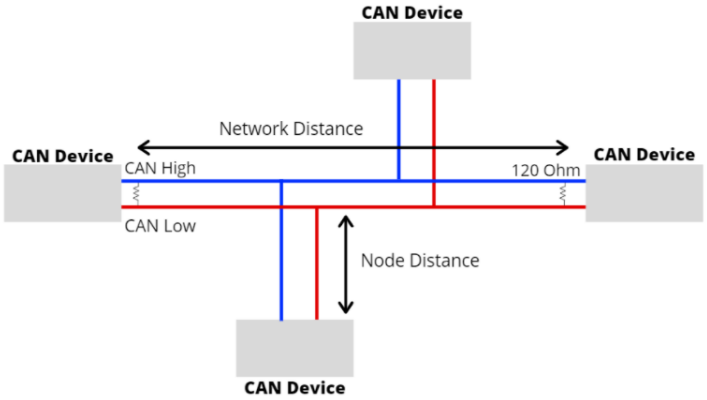
MCP2515 CAN Module:



The pin configuration of MCP2515 module is shown in below.

Pin Name	Use
VCC	5V Power input pin
GND	Ground pin
CS	SPI Slave select pin (active low)
SO	SPI master input slave output lead
SI	SPI master output slave input lead
SCK	SPI clock pin
INT	MCP2515 interrupt pin

CAN BUS WIRING SEQUENCE.



Can protocol consist of two wires CAN-H and CAN-L to send and receive message. This two wires act as a different line where CAN signal is represented with the potential difference between them. If the difference is positive and larger than a certain minimum voltage, then the signal is 1, and if the difference between them is negative, it will be 0.

CAN MESSAGE

CAN message contains many segment, The 2 main segments ,identifier and data, will be the ones transmitting the data. The identifier is used to identify the can devices in a can network while data will be the sensor or control data that have to be sent from one device to another.
The identifier or CAN-ID is either 11 (standard can) or 29(extended can) bits in length depending on the type of can protocol is used.
While the data can be anywhere from 0 to 8 bytes

1.2 connections

The MCP2515 CAN Bus Controller is a simple Module that supports CAN Protocol version 2.0B and can be used for communication at 1Mbps. In order to setup a complete communication system, you will need two CAN Bus Module.

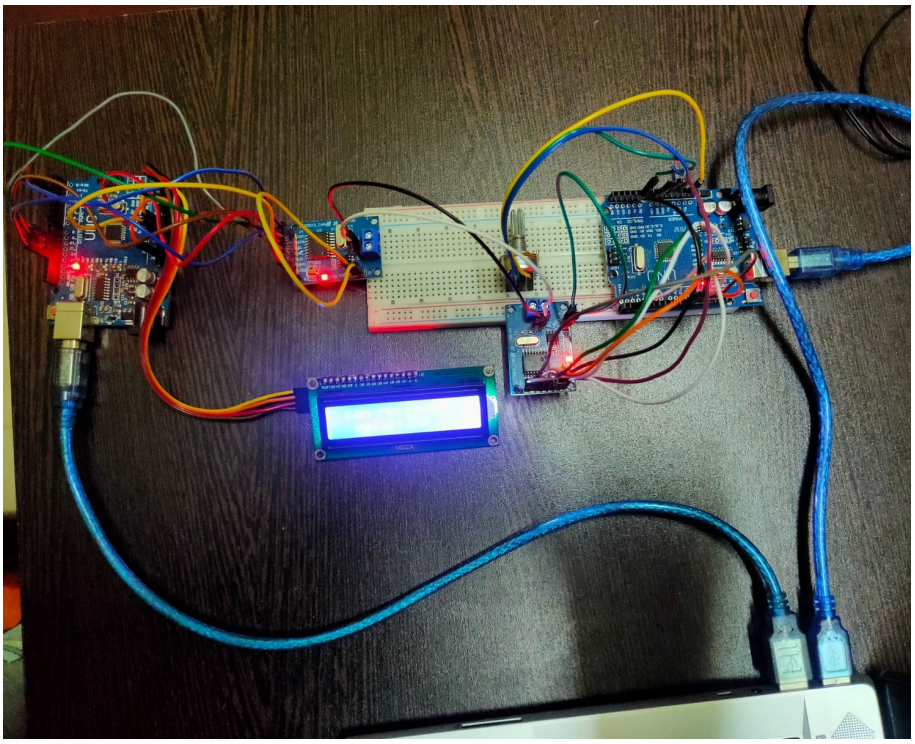
1. Connect SCK,SI,SO,CSGND,VCC of mcp2515 module to digital pin 13,11,12,10 and GND and +5v pin of Arduino.(note mcp2515 library by default initializes cs pin = 10) , so it is advisable to connect the pins in that exact same order

2. Now do the same with other mcp2515 module and Arduino

3. Connect the CAN-H and CAN-L of first mcp2515 to CAN-H and CAN-L of second mcp2515 for communication

4. Place the 10K potentiometer in bread bored now connect the centre pin of potentiometer to A0 analog pin of Arduino and the other two pins of potentiometer to GND and 5v pin of Arduino

5. Connect the SCL and SDA pin of I2c module to A5 and A4 analog pin of Arduino and GND and VCC of I2c module to GND and +5v pin of Arduino



1.3 implementation in Arduino

Sending msg (transmitting side
This code makes use of ¡mcp2515.h¿ library this library needs to be installed separately, you can find the .zip library in the code folder
listings

1. This code makes use of mcp2515 library this library needs to be installed separately, you can find the .zip library in the code folder and create a can object

```
#include<mcp2515.h>
 struct can_frame canMsg;
```

2. To create connection with MCP2515 provide pin number where SPI CS is connected (10 by default), bitrate and mode

```
MCP2515 mcp2515(10); // initializes the mcp2515
mcp2515.setBitrate(CAN_125KBPS, MCP_8MHZ);
//Available Baud Rates:
CAN_5KBPS, CAN_10KBPS, CAN_20KBPS, CAN_31K25BPS, CAN_33KBPS,
   CAN_40KBPS, CAN_50KBPS, CAN_80KBPS, CAN_83K3BPS, CAN_95KBPS,
   CAN_100KBPS, CAN_125KBPS, CAN_200KBPS, CAN_250KBPS, CAN_500KBPS,
   CAN_1000KBPS.
//Available Clock Speeds:
MCP_20MHZ, MCP_16MHZ, MCP_8MHZ
mcp2515.setNormalMode();
```

3. To send the message we have to first initialize the data

```
canMsg.data[0] = potentiometer value or any sensor data;
mcp2515.sendMessage(&canMsg);
delay(200);
```

where sendMessage will send the data ,next there is delay of 200 ms before sending the next packet

Reciever side

1. First the required libraries are included, SPI Library for using SPI Communication, MCP2515 Library for using CAN Communication and LiquidCrsytal Library for using 16x2 LCD with Arduino.

```
#include <SPI.h
#include <mcp2515.h>
#include <LiquidCrystal.h>
```

2. like sender side code here also we have to initialize can object and set bit rate and mode

3. To use 16x2 lcd with i2c module first we have to intialize and set up the lcd //

```
LiquidCrystal_I2C lcd(0x27,16,2);
```

Where 0x27 is the address of i2c module and 16,2 is the type of lcd and the following commands are used to setup the lcd

```
lcd.init()
lcd.backlight()
```

4

```
int x = canMsg.data[0];
```

The following statement is used to receive the message from the CAN bus , x can be any value or sensor data

1.4 Arduino code

1. Sender Side

```
#include <mcp2515.h>
#include<SPI.h>

#define potpin A0
int potValue = 0;

struct can_frame canMsg;
MCP2515 mcp2515(10);

void setup() {
  Serial.begin(9600);
  SPI.begin();

  mcp2515.reset();
  mcp2515.setBitrate(CAN_125KBPS,MCP_8MHZ);
  mcp2515.setNormalMode();

  canMsg.can_id = 0x036;
```

```
  canMsg.can_dlc = 1;

}

void loop() {
  potValue = analogRead(potpin);
  potValue = map(potValue,0,1023,0,255);
  Serial.println(potValue);
  canMsg.data[0] = potValue;
  mcp2515.sendMessage(&canMsg);
  delay(200);
}
```

2. Receiver side code

```
#include <SPI.h>
#include <mcp2515.h>

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27,16,2);

struct can_frame canMsg;
MCP2515 mcp2515(10);

void setup() {
  lcd.init();
  lcd.backlight();
  delay(1000);

  SPI.begin();

  Serial.begin(9600);
  mcp2515.reset();
  mcp2515.setBitrate(CAN_125KBPS,MCP_8MHZ);
  mcp2515.setNormalMode();
}

void loop() {
  if(mcp2515.readMessage(&canMsg)==MCP2515::ERROR_OK){
    if(canMsg.can_id == 0x036){
      int x = canMsg.data[0];
      Serial.println(x);
      lcd.setCursor(1,0);
      lcd.print(x);
      delay(200);
    }
  }
  else{
    Serial.println("ERROR");
  }
}
```

1.5 implementation in python

to use the python code first u need to upload the floss-firmware available in tools/python in arduino

1. sender side first we begin with importing necessary modules followed by setting up the serial port.Next to read the and map the potentiometer value we use the following pyhton code

```
    val = self.obj_arduino.cmd_analog_in(1,self.potpin)  #reading
    potentiometer value
    val = self.obj_arduino._map(val,0,1023,0,255)   #mapping
    potentiometer value

```

   to send the msg we use the following command followed by a delay, where val is the value of potentiometer

```
    self.obj_arduino.send_msg(1,val) # sending potentiometer value
    sleep(0.5)

```

2. receiver side to receive and print the msg in lcd we use following commands

```
    val = self.obj_arduino.receive_msg(2)  # receiving potentiometer
    value
    self.obj_arduino.lcd_print(2,val)

```

   where 2 is port no

1.6 Python code

1. Sender side

```

import os
import sys
cwd = os.getcwd()
(setpath,examples) = os.path.split(cwd)
sys.path.append(setpath)

from Arduino.Arduino import Arduino
from time import sleep

class transmitter:
    def __init__(self,baudrate):
        self.baudrate = baudrate
        self.setup()
        self.run()
        self.exit()

    def setup(self):
        self.obj_arduino = Arduino()
        self.port = self.obj_arduino.locateport1()
        print(self.port)
        self.obj_arduino.open_serial(1,self.port,self.baudrate)

    def run(self):
        self.potpin = 0  #initializing potentiometer pin
        val = 0          #decalring potentiometer value = 0
        while True:
            val = self.obj_arduino.cmd_analog_in(1,self.potpin)  #
    reading potentiometer value
```

```
29              val = self.obj_arduino._map(val,0,1023,0,255)    #mapping
    potentiometer value
30              self.obj_arduino.send_msg(1,val) # sending potentiometer
    value
31              print(val)
32              sleep(0.5)
33
34      def new_method(self, val):
35          self.obj_arduino.send_msg(1,val)
36
37      def exit(self):
38          self.obj_arduino.close_serial()
39
40  def main():
41      obj_send = transmitter(115200)
42
43  if __name__ == '__main__':
44      main()
45
46
```

2. Reiceiver side

```
1   import os
2   import sys
3
4   import serial
5   cwd = os.getcwd()
6   (setpath,examples) = os.path.split(cwd)
7   sys.path.append(setpath)
8
9   from Arduino.Arduino import Arduino
10  from time import sleep
11
12  class Receiver:
13      def __init__(self,baudrate):
14          self.baudrate = baudrate
15          self.setup()
16          self.run()
17          self.exit()
18
19      def setup(self):
20          self.obj_arduino = Arduino()
21          self.port = self.obj_arduino.locateport2()
22          print(self.port)
23          self.obj_arduino.open_serial(2,self.port,self.baudrate)
24
25      def run(self):
26          while True:
27              val = self.obj_arduino.receive_msg(2)  # receiving
    potentiometer value
28              self.obj_arduino.lcd_print(2,val)
29              print(val)
30              sleep(0.5)
31
32      def exit(self):
33          self.obj_arduino.close_serial()
34
35  def main():
36      obj_rec = Receiver(115200)
37
```

```
38 if __name__ == '__main__':
39     main()
```