FOSSE - REPORT

Name: Chirag Rathore
Reg no: 19BEE1025

IMPLEMENTATION OF CAN BUS PROTOCOL

ABTRACT

CAN stands for Controller Area Network, it is used to allow microcontrollers 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.

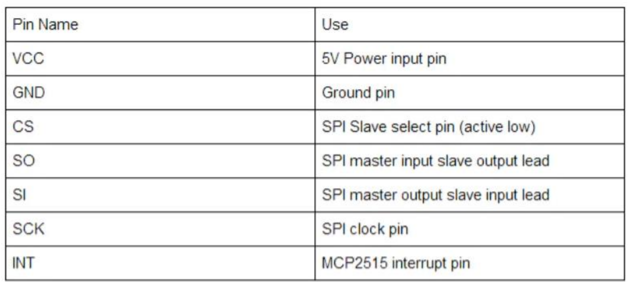
CAN is a serial communication bus designed for industrial and automotive applications. For example, they are found in vehicles, farming equipment, industrial environments, etc. The real example of CAN Bus application can be found in speed car data sensor that can be transferred to the rpm indicator.

Circuit Details

1) 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) .

The pin configuration of MCP2515 module is shown in fig(b)

MCP2515 CAN Module:

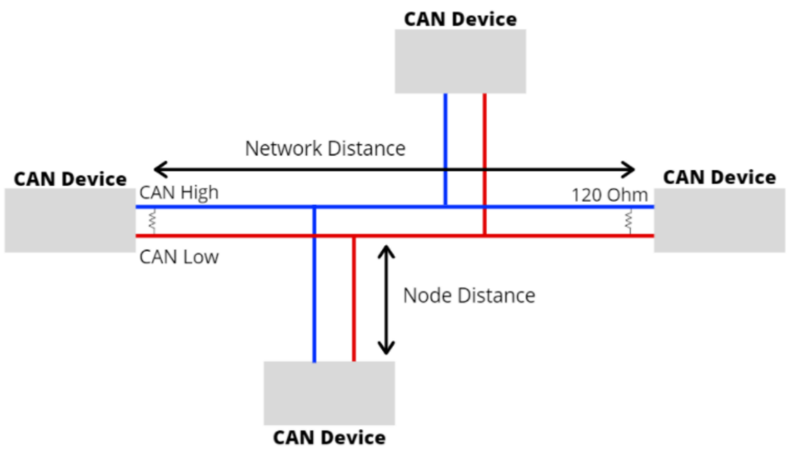


Pin Name	Use
VCC	5V Power input pin
GND	Ground pin
CS	SPI Slave select pin (active low)
SO	SPI master input slave output lead
SI	SPI master output slave input lead
SCK	SPI clock pin
INT	MCP2515 interrupt pin

Figure A

Figure B

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.

Hardware part

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.

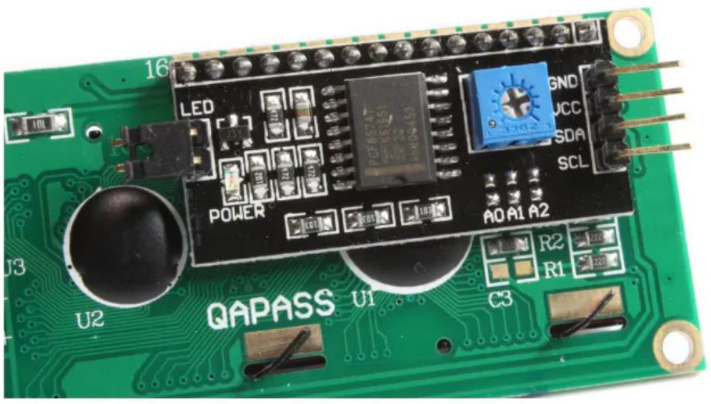
1. Connecting MCP2515 with Arduino

- 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
- Now do the same with other mcp2515 module and Arduino
- Connect the CAN_H and CAN_L of first mcp2515 to CAN_H and CAN_L of second mcp2515 for communication

2. Connecting potentiometer with 1st Arduino

- 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

3. Connecting I2c module and lcd to 2nd Arduino



You can adjust the contrast of lcd with the potentiometer in I2c module

- Now 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

Interfacing the mcp2515 module in Arduino ide

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

Transmitting side

1) First we include the <mcp2515.h> library and now declare an object
   ```
   struct can_frame canMsg;
   ```

2) structure can_frame data is: -
   ```
   struct can_frame {
      uint32_t can_id;  /* 32 bit CAN_ID + EFF/RTR/ERR flags */
      uint8_t  can_dlc;
      uint8_t  data[8] };
   ```
3) To create connection with MCP2515 provide pin number where SPI CS is connected (10 by default), baudrate and mode

   ```
   MCP2515 mcp2515(10); // initializes the mcp2515
   ```

- Set baud rate and oscillator frequency

  ```
  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

- Set modes.

  ```
  mcp2515.setNormalMode();
  ```

4) Now we have to initialize the can_id and packet size it can be of maximum upto 8 byte
   ```
   canMsg.can_id=0x036;
   canMsg.can_dlc=1;
   ```

5) 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

Receiver 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) A struct data type is declared for storing CAN message format
   `struct can_frame canMsg;`
   Set the pin number where SPI CS is connected (10 by default)
   `MCP2515 mcp2515(10);`

3) now initialize the lcd
   `LiquidCrystal_I2C lcd(0x27,16,2);`
   Where 0x27 is the address of i2c module and 16,2 is the type of lcd

   ```
   lcd.init()
   lcd.backlight()
   ```
   commands are used to setup the lcd

4) like transmitting side set the bitrate and mode of mcp module
5) SPI.begin() is used to begin the spi communication
6) The following statement is used to receive the message from the CAN bus. If message is received it gets into the if condition.

7) `if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK)`

   In the if condition the data is received and stored in canMsg, the data [0] that has value and. Values is stored in an integer x

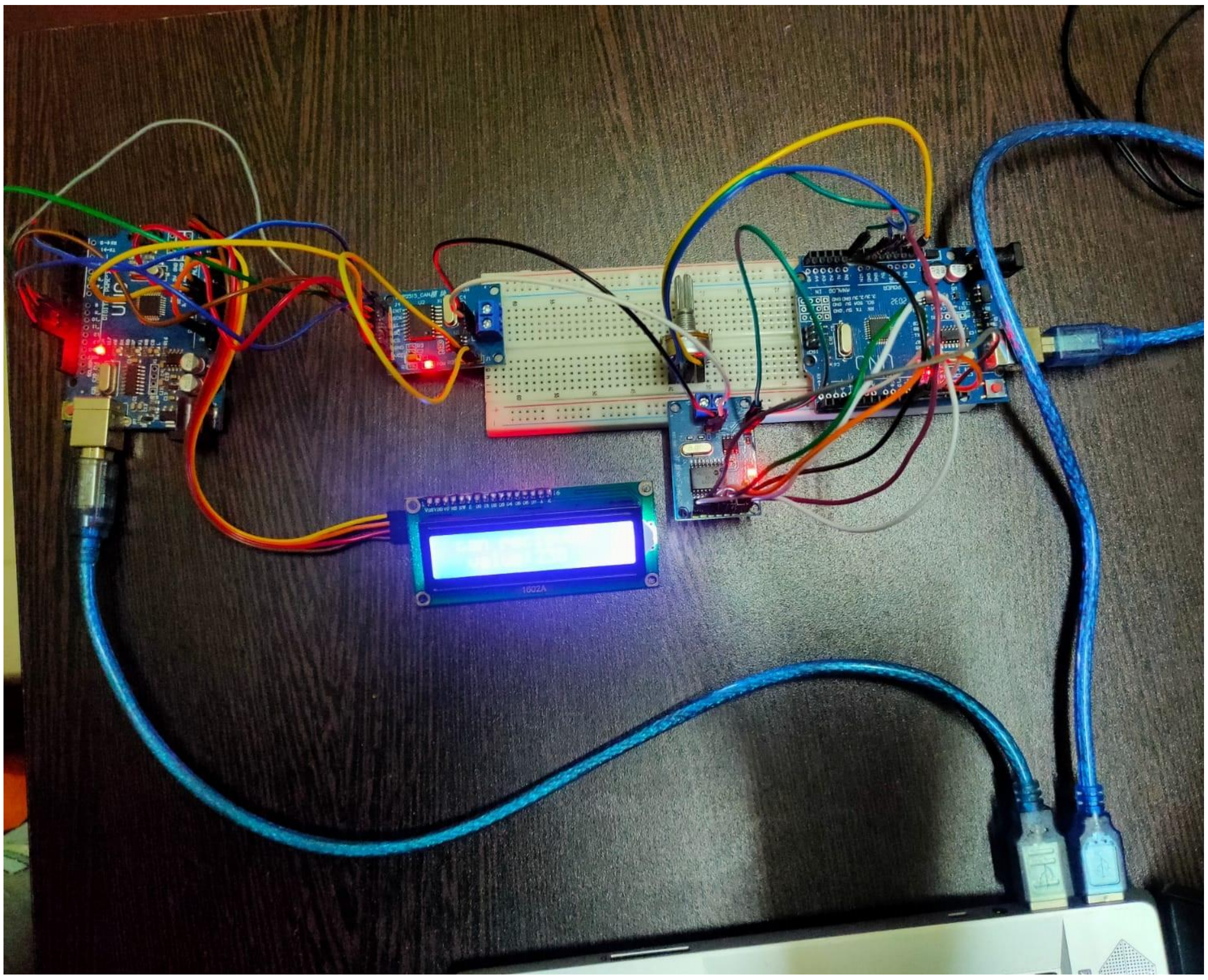
   ```
   int x = canMsg.data[0];
   ```

After receiving the value, the value is displayed in 16x2 LCD display using following statement.

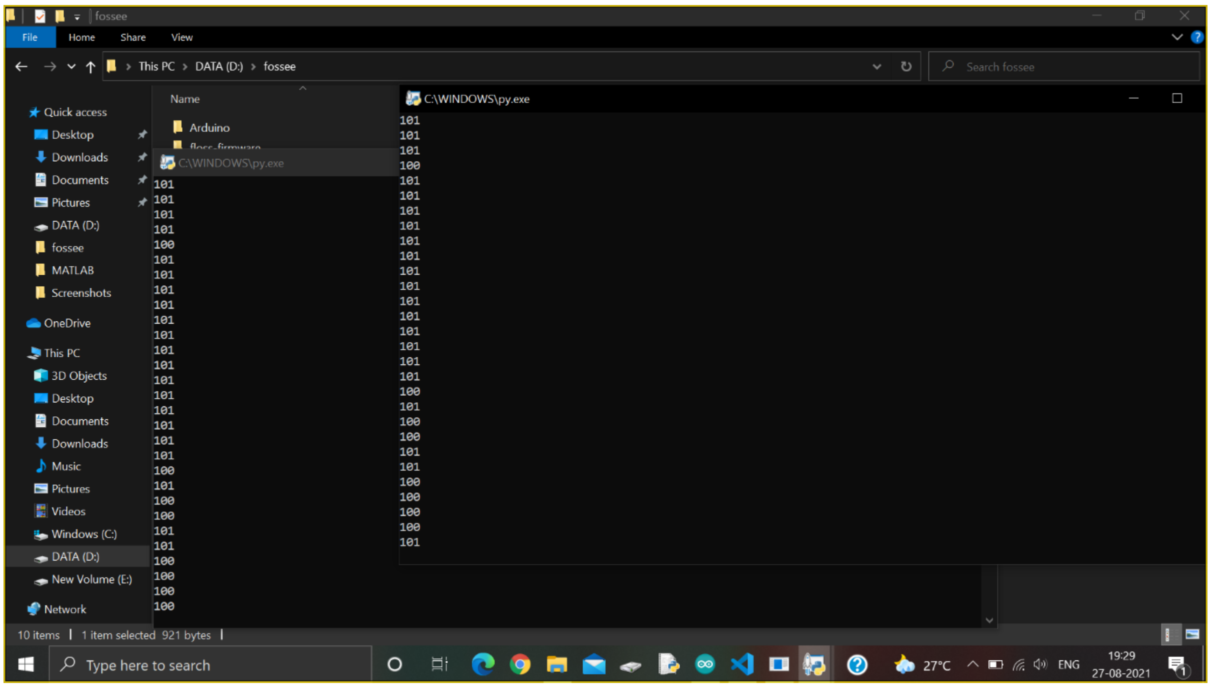
```
lcd.setCursor(1,0);

lcd.print(“value”:);

lcd.print(x);

Delay(100);
```

Delay of 100 ms before displaying the original values

Implemented Circuit:



Output

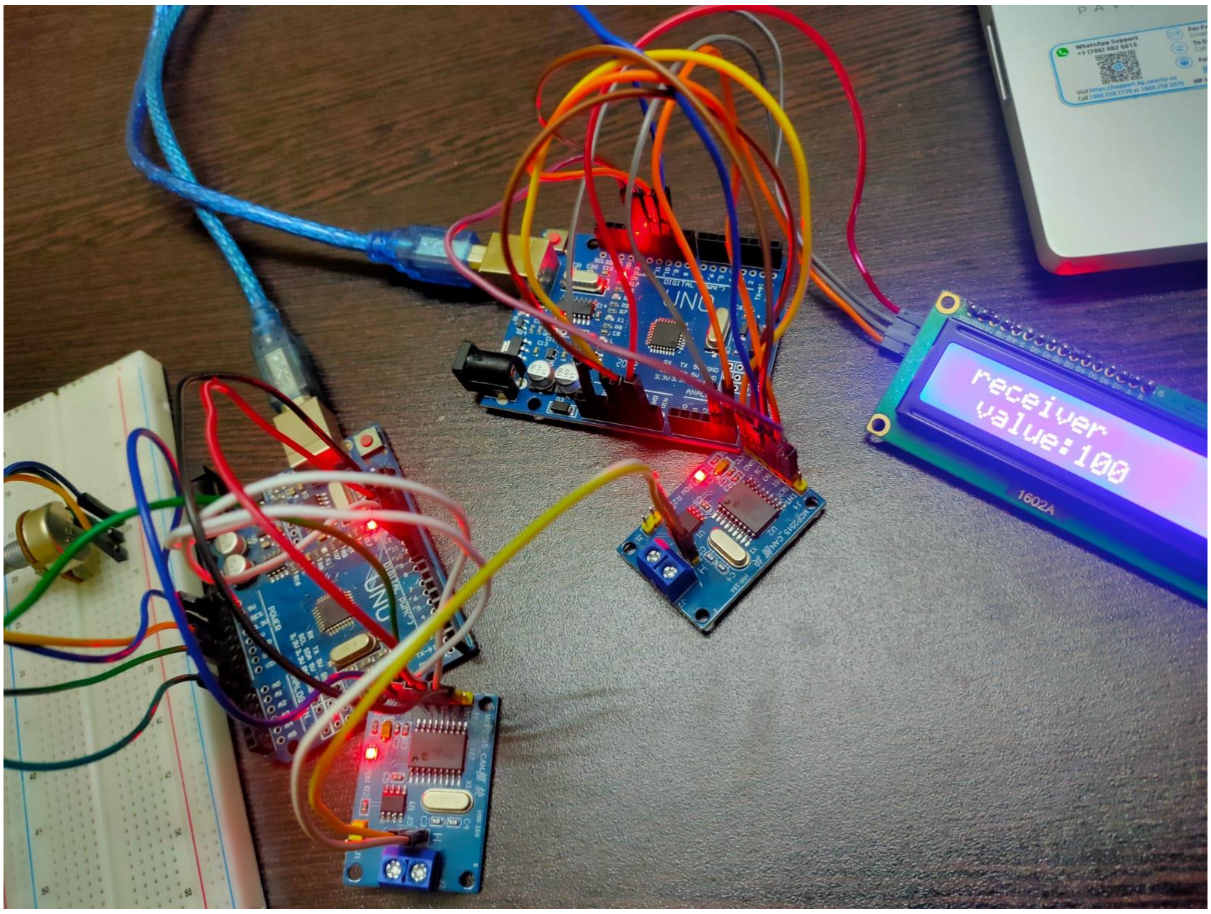


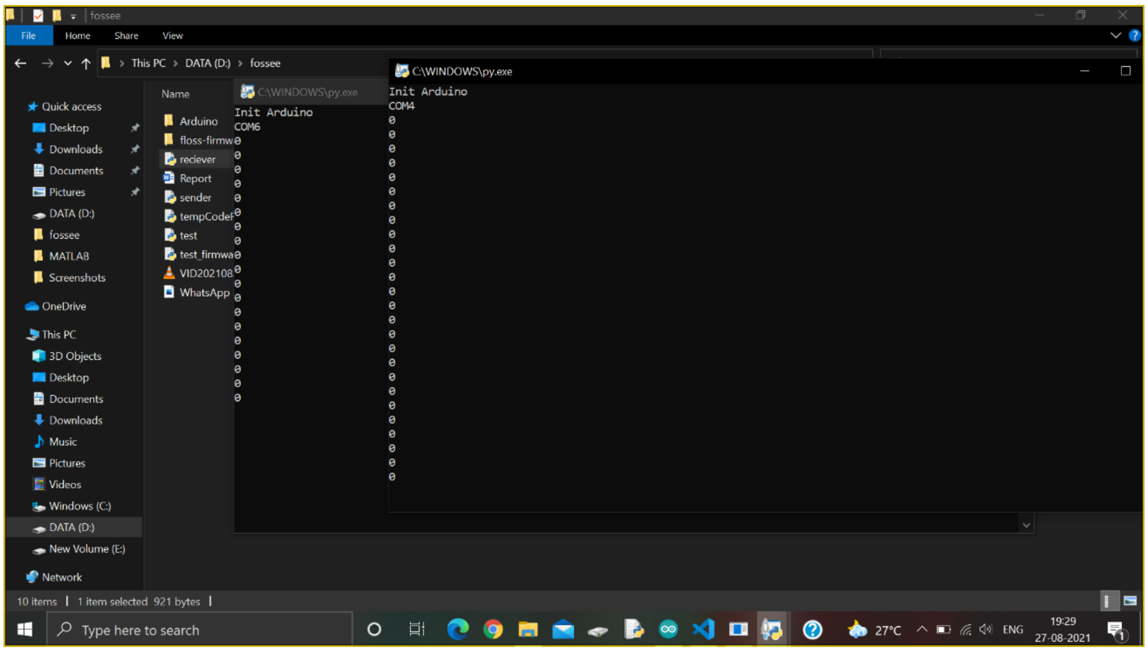
C:\WINDOWS\py.exe

```
101
101
101
101
100
101
101
101
101
101
101
101
101
101
101
101
101
101
101
100
101
100
100
101
101
100
100
100
100
```

C:\WINDOWS\py.exe

```
101
101
101
100
101
101
101
101
101
101
101
101
101
101
101
101
101
101
100
101
100
100
101
101
100
100
100
100
101
```



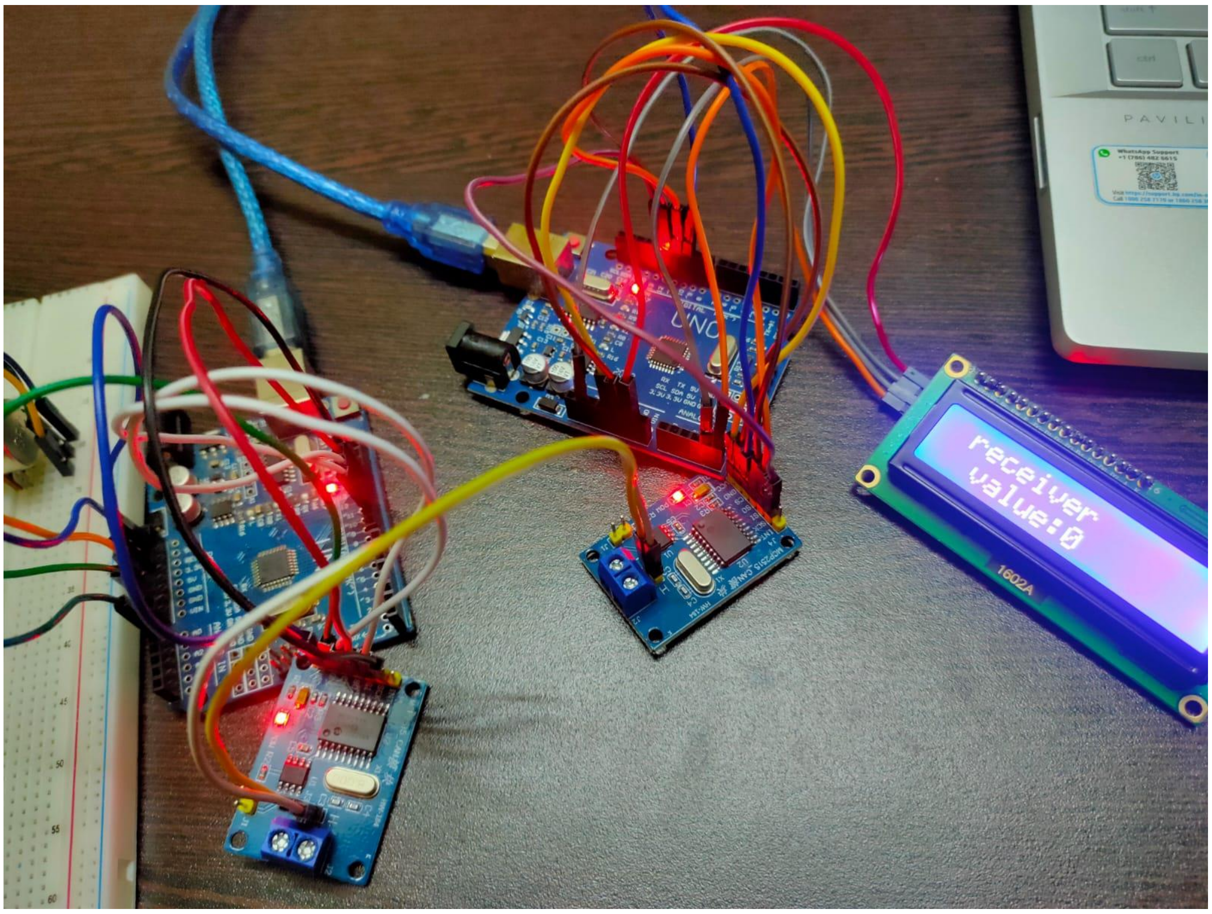


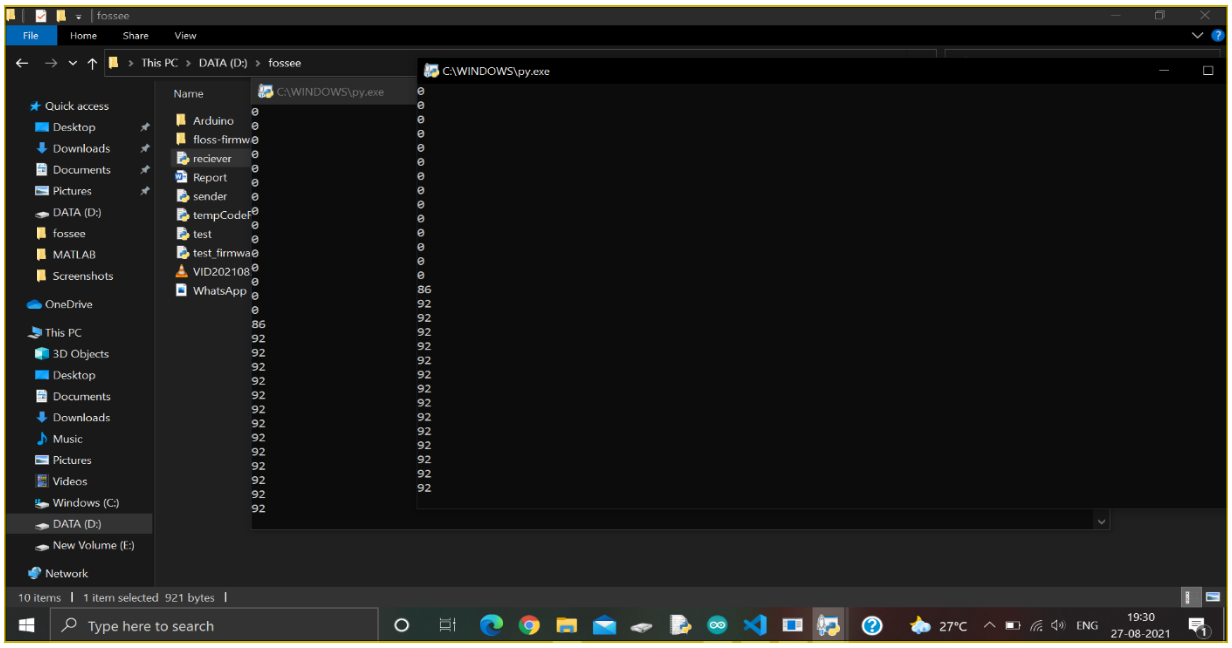
C:\WINDOWS\py.exe

```
Init Arduino
COM6
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

C:\WINDOWS\py.exe

```
Init Arduino
COM4
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```



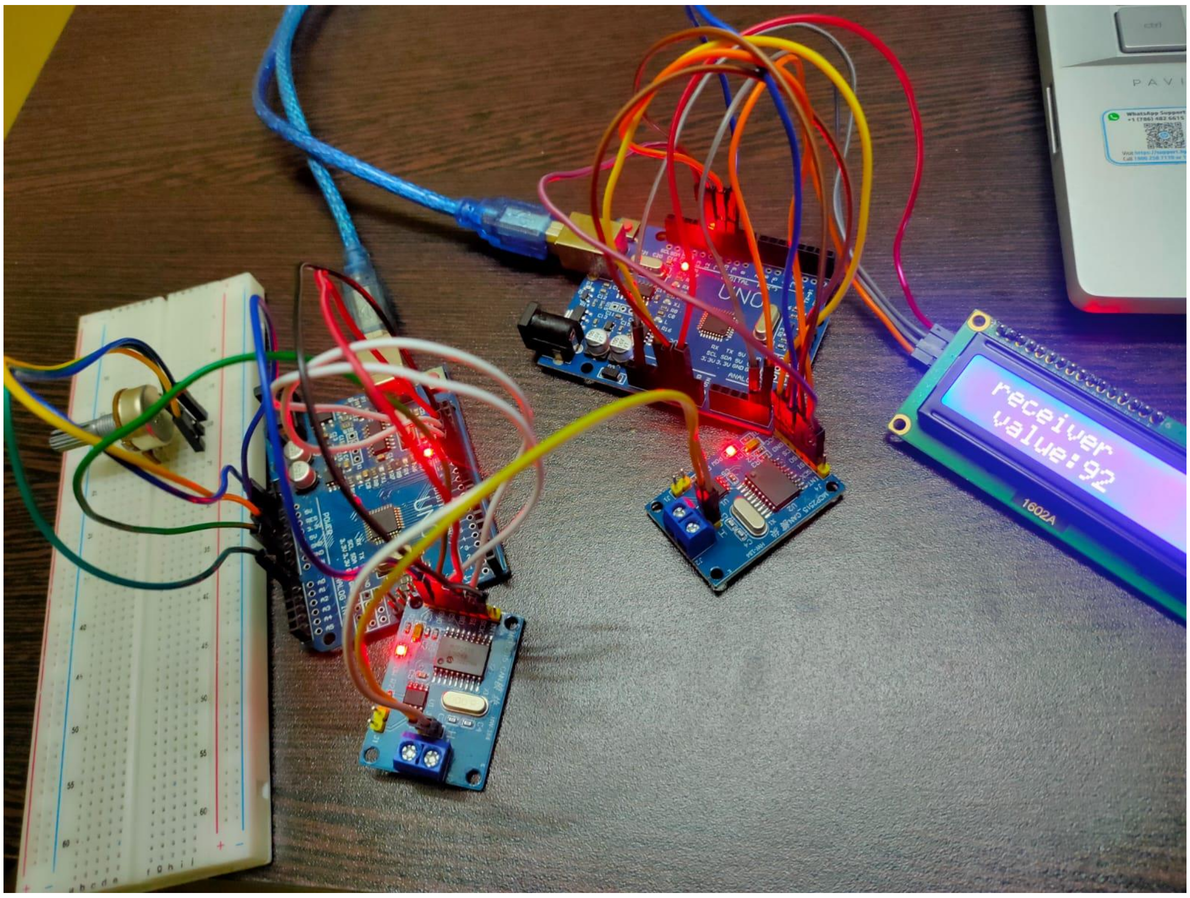


C:\WINDOWS\py.exe

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
86
92
92
92
92
92
92
92
92
92
92
92
92
92
```

C:\WINDOWS\py.exe

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
86
92
92
92
92
92
92
92
92
92
92
92
92
92
92
```



Arduino code:

Transmitter side:

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

#define potpin A0
int potValue = 0;

struct can_frame canMsg;
MCP2515 mcp2515(10);

void setup() {
  // put your setup code here, to run once:
  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);


}
```

Receiver side

```
#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() {
 // put your setup code here, to run once:
 lcd.init();
 lcd.backlight();
 delay(1000);

 SPI.begin();

 Serial.begin(9600);

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

}

void loop() {
 // put your main code here, to run repeatedly:
 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");
 }

}
```

Note: You need to first upload the floss-firmware.ino code included in the folder to Arduino before running the python code

Python code:

Transmitter 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
         val = self.obj_arduino._map(val,0,1023,0,255)   #mapping potentiometer value
         self.obj_arduino.send_msg(1,val) # sending potentiometer value
         print(val)
         sleep(0.5)

   def new_method(self, val):
      self.obj_arduino.send_msg(1,val)

   def exit(self):
      self.obj_arduino.close_serial()

def main():
   obj_send = transmitter(115200)

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

Receiver side:

```
import os
import sys

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

from Arduino.Arduino import Arduino
from time import sleep

class Receiver:
    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.locateport2()
        print(self.port)
        self.obj_arduino.open_serial(2,self.port,self.baudrate)

    def run(self):
        while True:
            val = self.obj_arduino.receive_msg(2)  # receiving potentiometer value
            self.obj_arduino.lcd_print(2,val)
            print(val)
            sleep(0.5)

    def exit(self):
        self.obj_arduino.close_serial()

def main():
    obj_rec = Receiver(115200)

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