Arduino CAN Bus Module 1st Network Tutorial

**Contents**[[show](http://henrysbench.capnfatz.com/henrys-bench/arduino-projects-tips-and-more/arduino-can-bus-module-1st-network-tutorial/)]

An Arduino Based CAN Bus Network

CAN (Controller Area Network) bus networks are found everywhere.  They are found in vehicles, farm equipment, and in industrial environments.   These networks allow for control and data acquisition.  Depending on the application, they can be formed around a stringent set of standards (such as J1939) or in a ‘get it done’ approach suitable for an Arduino DIY project.

This tutorial introduces you to some very basic CAN bus principles and guides to building your first CAN bus network using the readily available Arduino CAN bus modules.

Required Material for the Tutorial

You’re going to need two Arduino’s for this tutorial.  You’re also going to need to 2 Arduino CAN Bus Modules.  These can be purchased at any of the sites below:

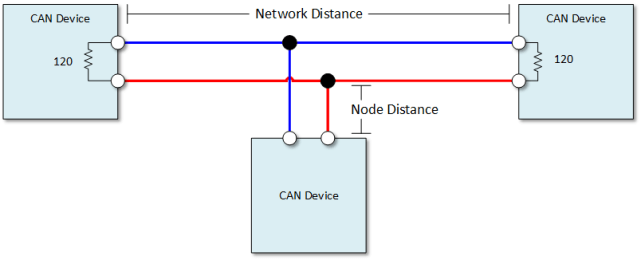
[Amazon](https://www.amazon.com/gp/search/ref=as_li_qf_sp_sr_il_tl?ie=UTF8&tag=leaacicarbatf-20&keywords=Arduino%20CAN%20Bus%20Module&index=aps&camp=1789&creative=9325&linkCode=xm2&linkId=e560efc64860aaff56565574d24da4ef)     [eBay](http://rover.ebay.com/rover/1/711-53200-19255-0/1?icep_ff3=9&pub=5575103433&toolid=10001&campid=5337702195&customid=&icep_uq=Arduino+CAN+Bus+Module&icep_sellerId=&icep_ex_kw=&icep_sortBy=12&icep_catId=&icep_minPrice=&icep_maxPrice=&ipn=psmain&icep_vectorid=229466&kwid=902099&mtid=824&kw=lg)     [Ali-Express](http://www.anrdoezrs.net/links/8535047/type/dlg/https:/www.aliexpress.com/wholesale?catId=0&initiative_id=SB_20180210105630&SearchText=can+bus+module)

The Arduino CAN Bus Module pin outs and schematics can be found [HERE](http://henrysbench.capnfatz.com/henrys-bench/arduino-projects-tips-and-more/arduino-can-bus-module-pin-outs-and-schematics/).

Brief CAN Intro

In this description,  I’m going to use the wordk ‘general’ a lot.  There is a reason for that.  CAN is a very flexible means of communications and has been adapted and defined to meet MANY standards.  What is a firm rule for one may not be for another.   What’s important here is that you get enough of an understanding to use CAN in your DIY project.

The image below is thus explained in very general terms.

[](https://i1.wp.com/henrysbench.capnfatz.com/wp-content/uploads/2017/01/Arduino-CAN-Bus-Basics.png)

CAN Wires

There are two CAN wires.   They are called CAN High and CAN Low.  In applications that are designed to be robust,  the wires are normally shielded twisted pairs.   The shield is normally landed (or grounded) as one end.

CAN Speed and Distance

Communication speeds generally range from 50kpbs to 1Mbps.   This maximum distance is drive by the selected speed.  It can range from 40 meters at 1Mbps and 1000 meters at 50kpbs.

The general rule of thumb is that the shorter and slower the bus is, the more robust the communications.

Node distance is generally specified to be no more than 0.3 meters ( 1 foot).

The CAN Message

With CAN, we send and respond to messages over the CAN bus.    A message contains an identifier and data.

The identifier is also known as a CAN ID or is sometimes to referred to as a PGN.    The length of the identifier is either 11 or 29 bits in length.

The data can be anywhere from 0 to 8 bytes in length.

CAN Termination

A single 120 ohm resistor is generally used at the two ends of the CAN network.  Nodes that hang of of the network can be up to 0.3 meters (1 foot) in length.

Arduino CAN Network Tutorial

This tutorial is very simple.   Using our CAN transmitter we will create a CAN message and repetitively broadcast it.   We will receive this CAN message using our CAN receiver.  We will then display the received output in our serial monitor.

We will create messages from Arduino input and respond to those messages in future tutorials.

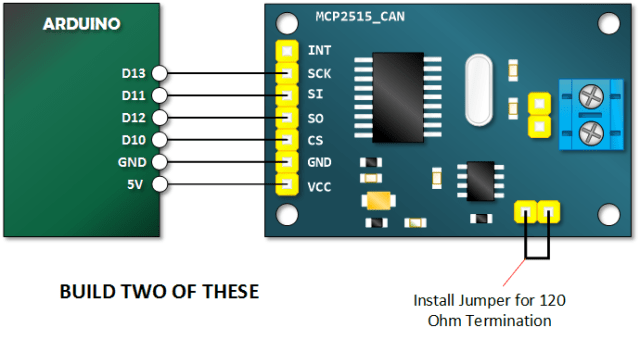
Get the Required Library

You are going to need the mcp\_can.h library in order to run this tutorial.  You can find that library [HERE](https://github.com/Seeed-Studio/CAN_BUS_Shield).

If you’re not familiar with installing libraries,  you can read about it [HERE](https://www.arduino.cc/en/Guide/Libraries).

Build Arduino CAN Transmitter and Receiver

The receiver and transmitter are wired identically.  Build **two** of these.

[](https://i0.wp.com/henrysbench.capnfatz.com/wp-content/uploads/2017/01/Arduino-CAN-Bus-Module-Tutorial-Schematic.png)

Copy, Paste and Upload the CAN Bus Module Receive Sketch

afds

// Henry's Bench

// 1st CAN Network - CAN RECEIVE

#include <**SPI**.h>

#include "mcp\_can.h"

long unsigned int rxId;

unsigned long rcvTime;

unsigned char len = 0;

unsigned char buf[8];

const int SPI\_CS\_PIN = 10;

**MCP\_CAN** CAN(SPI\_CS\_PIN);                                    // Set CS pin

void setup()

{

**Serial**.begin(115200);

   while (CAN\_OK != CAN.begin(CAN\_250KBPS))              // init can bus : baudrate = 500k

   {

**Serial**.println("CAN BUS Module Failed to Initialized");

**Serial**.println("Retrying....");

       delay(200);

   }

**Serial**.println("CAN BUS Module Initialized!");

**Serial**.println("Time\t\tPGN\t\tByte0\tByte1\tByte2\tByte3\tByte4\tByte5\tByte6\tByte7");

}

void loop()

{

   if(CAN\_MSGAVAIL == CAN.checkReceive())            // check if data coming

   {

       rcvTime = millis();

       CAN.readMsgBuf(&len, buf);    // read data,  len: data length, buf: data buf

       rxId= CAN.getCanId();

**Serial**.print(rcvTime);

**Serial**.print("\t\t");

**Serial**.print("0x");

**Serial**.print(rxId, HEX);

**Serial**.print("\t");

       for(int i = 0; i<len; i++)    // print the data

       {

           if(buf[i] > 15){

**Serial**.print("0x");

**Serial**.print(buf[i], HEX);

           }

         else{

**Serial**.print("0x0");

**Serial**.print(buf[i], HEX);

         }

           //Serial.print("0x");

           //Serial.print(buf[i], HEX);

**Serial**.print("\t");

       }

**Serial**.println();

   }

}

Copy, Paste and Upload the CAN Bus Module Transmit Sketch

asdf

// Henry's Bench

// 1st CAN Network - CAN TRANSMIT

#include <**mcp\_can**.h>

#include <**SPI**.h>

const int SPI\_CS\_PIN = 10;

// Build an ID or PGN

long unsigned int txID = 0x1881ABBA; // This format is typical of a 29 bit identifier.. the most significant digit is never greater than one.

unsigned char stmp[8] = {0x0E, 0x00, 0xFF, 0x22, 0xE9, 0xFA, 0xDD, 0x51};

//Construct a MCP\_CAN Object and set Chip Select to 10.

**MCP\_CAN** CAN(SPI\_CS\_PIN);

void setup()

{

**Serial**.begin(115200);

   while (CAN\_OK != CAN.begin(CAN\_250KBPS))              // init can bus : baudrate = 250K

   {

**Serial**.println("CAN BUS Module Failed to Initialized");

**Serial**.println("Retrying....");

       delay(200)

   }

**Serial**.println("CAN BUS Shield init ok!");

}

void loop()

{   **Serial**.println("In loop");

   // send the data:  id = 0x00, Extended Frame, data len = 8, stmp: data buf

   // Extended Frame = 1.

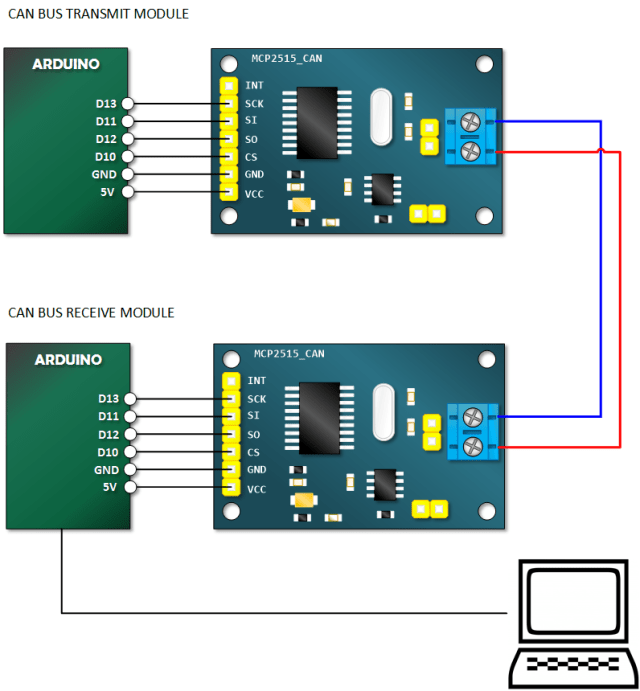
   CAN.sendMsgBuf(txID,1, 8, stmp);

   delay(25);    // send data every 25mS

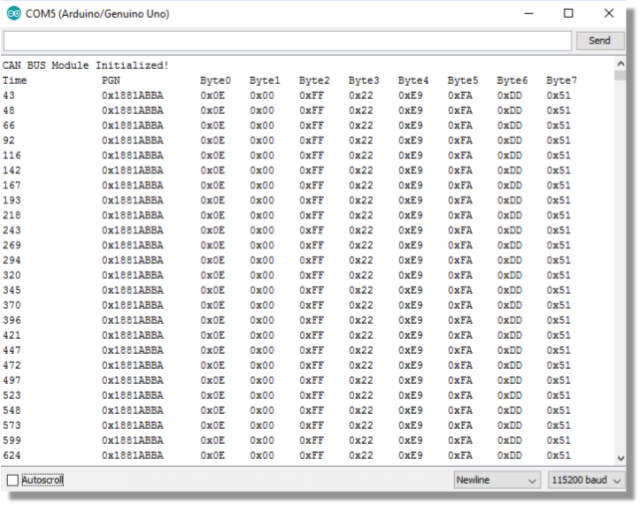
}

Connect the Transmitter and Receiver into a Network

You will need to power the transmitter with its own power source.  The receiver will connect to your computer and will receive its power via USB.

[](https://i1.wp.com/henrysbench.capnfatz.com/wp-content/uploads/2017/01/Arduino-CAN-Bus-Network-Connection.png)

Open Serial Monitor and Verify Receiving of Data

[](https://i1.wp.com/henrysbench.capnfatz.com/wp-content/uploads/2017/01/Arduino-CAN-Bus-Module-Tutorial-Output.png)

# Arduino MCP2515 CAN Bus Interface Tutorial

AUGUST 23, 2018 BY [RAVI](https://www.electronicshub.org/author/raviteja/) [1 COMMENT](https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/#comments)

In this project, we will learn about the MCP2515 CAN Controller Module, how to interface the MCP2515 CAN Bus Controller with Arduino and finally how to enable communication between two Arduino board with the help of two MCP2515 CAN Controllers and the CAN Protocol.

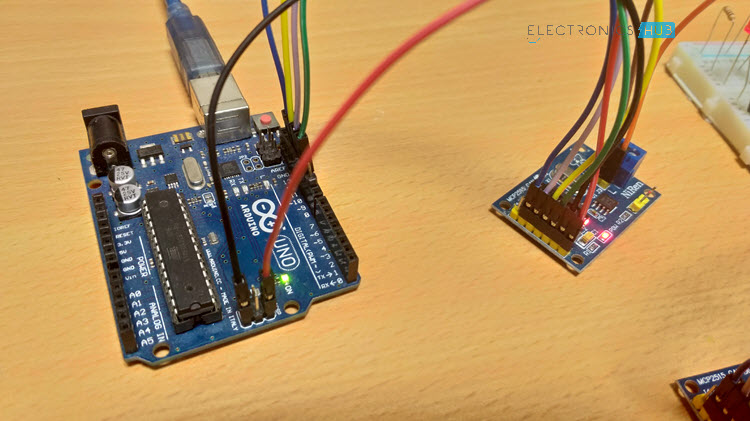


Table of Contents

* [Introduction](https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/#Introduction)
* [A Brief Note on MCP2515 CAN Bus Controller Module](https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/#A_Brief_Note_on_MCP2515_CAN_Bus_Controller_Module)
* [Schematic of MCP2515 CAN Bus Module](https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/#Schematic_of_MCP2515_CAN_Bus_Module)
* [Circuit Diagram for Interfacing MCP2515 with Arduino](https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/#Circuit_Diagram_for_Interfacing_MCP2515_with_Arduino)
  + [Components Required](https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/#Components_Required)
  + [Circuit Design](https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/#Circuit_Design)
* [Code](https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/#Code)
  + [Transmitter Code](https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/#Transmitter_Code)
  + [Receiver Code](https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/#Receiver_Code)
* [Working](https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/#Working)
* [Applications](https://www.electronicshub.org/arduino-mcp2515-can-bus-tutorial/#Applications)

### Introduction

Controlled Area Network of simple CAN is a bus standard that allows a Microcontroller and its peripheral devices to communicate without the need of a host device or a computer.

Developed by Robert Bosch GmbH, CAN is protocol is main used in automobiles for communication between a control unit and its components.

For example, the Engine Control Unit is a major control using in a car. This unit is connected to many sensors and actuators like air flow, pressure, temperature, valve control, motors for air control etc. The communication between these modules and the control unit is through CAN Bus.

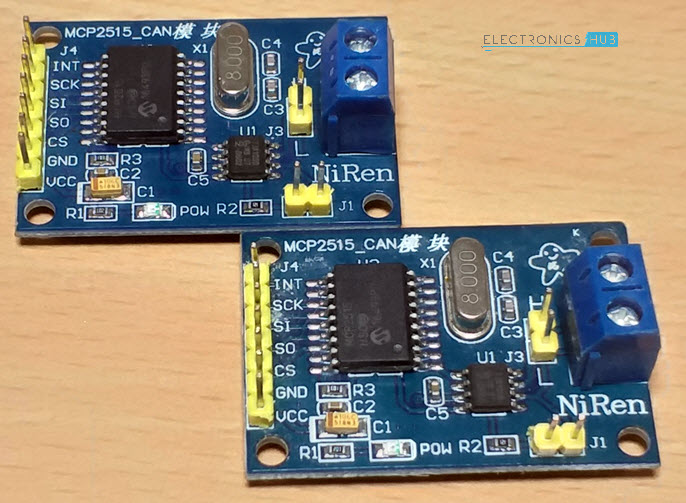
In order to understand a little bit more about CAN Bus, CAN Controller and other important aspects, the MCP2515 CAN Bus Controller Module is very helpful.

Also read: [**BASICS OF SPI COMMUNICATION**](https://www.electronicshub.org/basics-serial-peripheral-interface-spi/).

### A Brief Note on MCP2515 CAN Bus Controller Module

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.

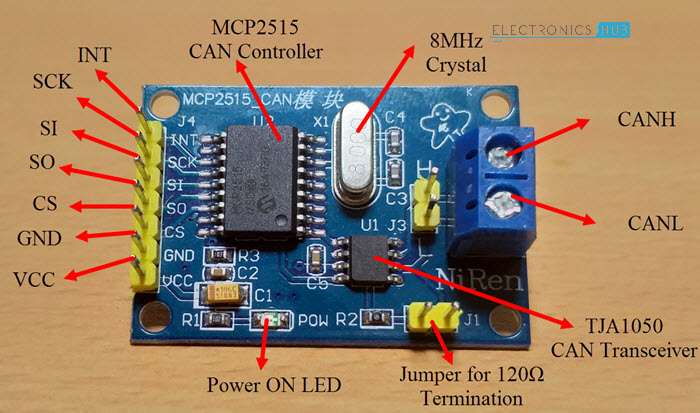
The module used in the project is shown in the image below.



This particular module is based on MCP2515 CAN Controller IC and TJA1050 CAN Transceiver IC. The MCP2515 IC is a standalone CAN Controller and has integrated SPI Interface for communication with microcontrollers.

Coming to the TJA1050 IC, it acts as an interface between the MCP2515 CAN Controller IC and the Physical CAN Bus.

The following image shows the components and pins on a typical MCP2515 Module.



### Schematic of MCP2515 CAN Bus Module

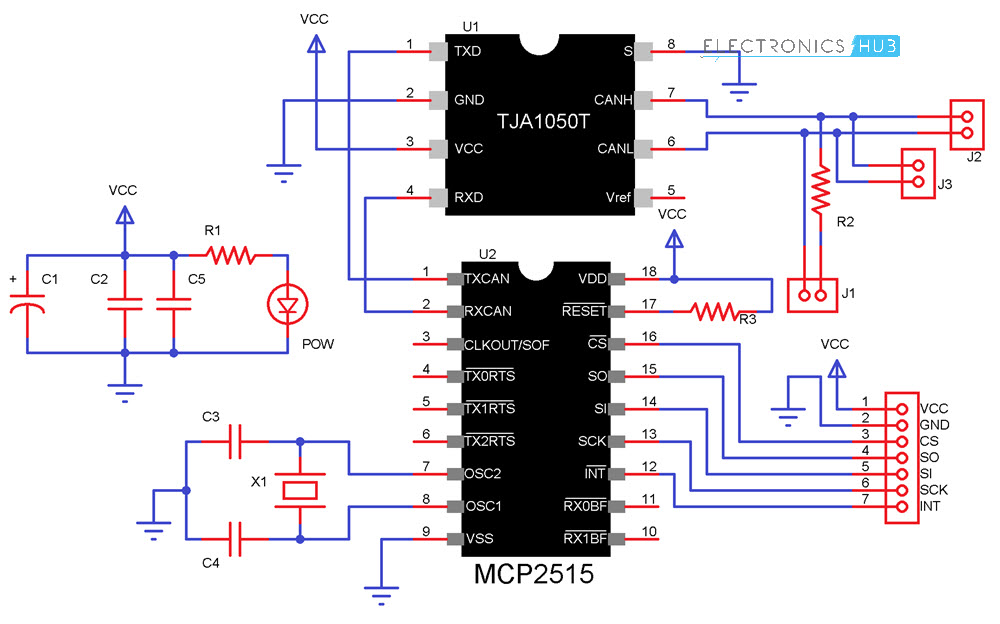
Before seeing the schematic of the module, you need to understand a couple of things about both the ICs i.e. MCP2515 and TJA1050.

MCP2515 IC is the main controller that internally consists of three main subcomponents: The CAN Module, the Control Logic and the SPI Block.

CAN Module is responsible for transmitting and receiving messages on the CAN Bus. Control Logic handles the setup and operation of the MCP2515 by interfacing all the blocks. The SPI Block is responsible for the SPI Communication interface.

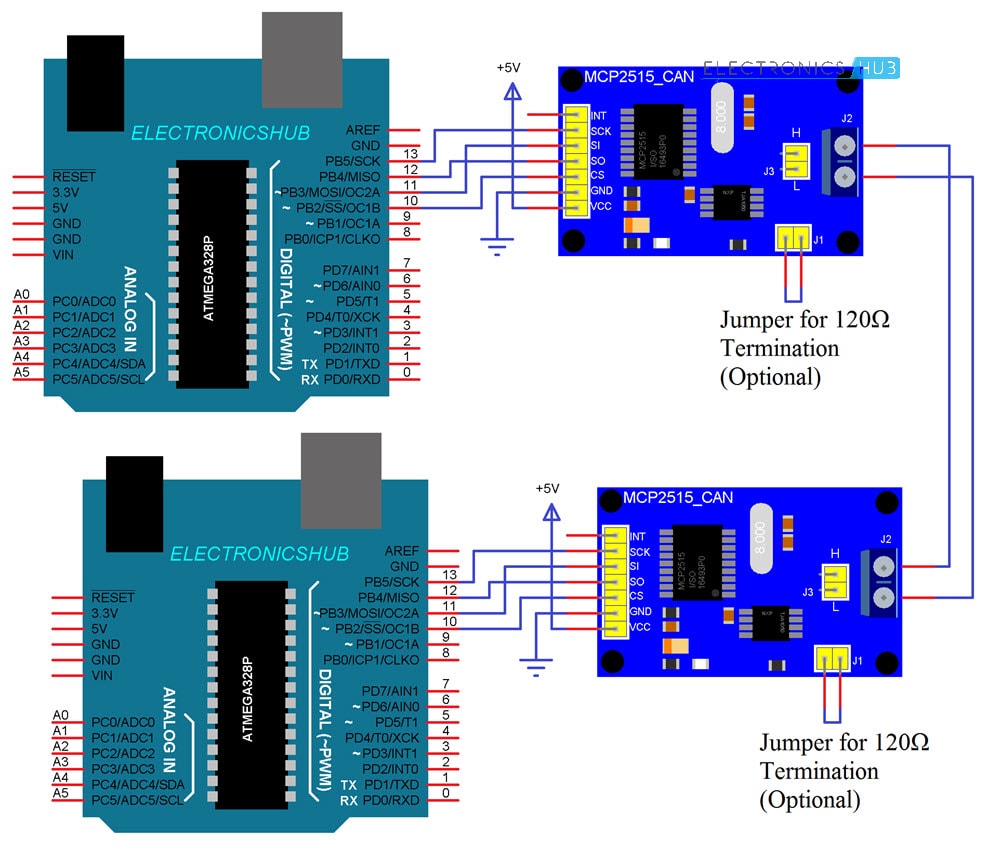
Coming to the TJA1050 IC, since it acts as an interface between MCP2515 CAN Controller and the physical CAN Bus, this IC is responsible for taking the data from the controller and relaying it on to the bus.

The following image shows the schematic of the MCP2515 CAN Module and it shows how MCP2515 IC and TJA1050 IC are connected on the Module.

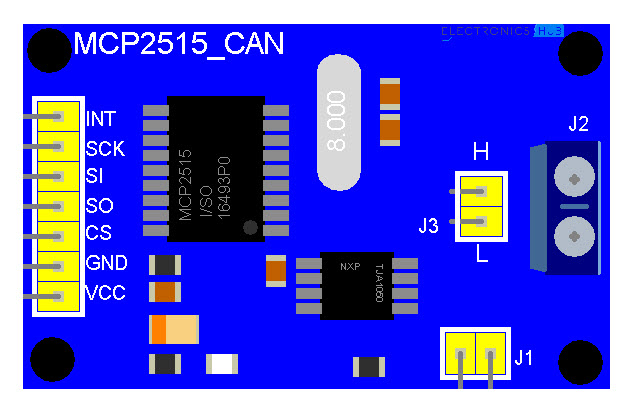


### Circuit Diagram for Interfacing MCP2515 with Arduino

The following image shows the circuit diagram of interfacing MCP2515 CAN Module with Arduino and possible communication between two Arduino over CAN Protocol.



If the pins of the MCP2515 Module are not clear, the following image might be useful.

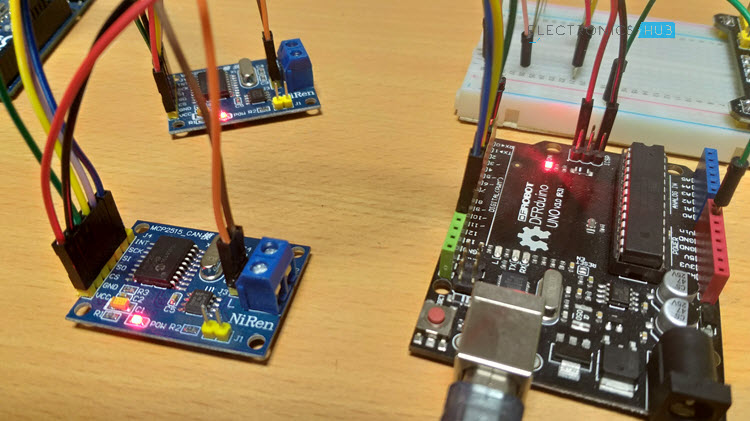


#### Components Required

* Arduino UNO x 2  [[Buy Here](https://amzn.to/2FFyQfc?tag=eh04e-21" \t "_blank)]
* MCP2515 x 2  [[Buy Here](https://amzn.to/2T5KpiX?tag=eh04e-21" \t "_blank)]
* USB Cable x 2
* Connecting Wires  [[Buy Here](https://amzn.to/2CBz0jN?tag=eh04e-21" \t "_blank)]

#### Circuit Design

As mentioned earlier, the CAN Controller IC facilitates SPI Communication Protocol for interfacing with any Microcontroller. Hence, connect the SPI Pin i.e. SCK, MOSI (SI), MISO (SO) and CS of the MCP2515 Module to corresponding SPI Pins of Arduino (see circuit diagram).



Make two such connections: one pair acts as a transmitter and the other as a receiver. Now for the communication between this transmitter and receiver, connect CANH and CANL pins of each MCP2515 Module.

### Code

Before going into the code, you need to download a library for the MCP2515 Module. There are many libraries but I have used [**this**](https://github.com/Seeed-Studio/CAN_BUS_Shield) particular one.

Download it and place the extracted contents in the libraries directory of Arduino.

Since the communication involves a Transmitter Module and a Receiver Module, the code is also divided into Transmitter Code and Receiver Code.

#### Transmitter Code

|  |  |
| --- | --- |
|  | #include <SPI.h> |
|  | #include <mcp\_can.h> |
|  |  |
|  | const int spiCSPin = 10; |
|  | int ledHIGH = 1; |
|  | int ledLOW = 0; |
|  |  |
|  | MCP\_CAN CAN(spiCSPin); |
|  |  |
|  | void setup() |
|  | { |
|  | Serial.begin(115200); |
|  |  |
|  | while (CAN\_OK != CAN.begin(CAN\_500KBPS)) |
|  | { |
|  | Serial.println("CAN BUS init Failed"); |
|  | delay(100); |
|  | } |
|  | Serial.println("CAN BUS Shield Init OK!"); |
|  | } |
|  |  |
|  | unsigned char stmp[8] = {ledHIGH, 1, 2, 3, ledLOW, 5, 6, 7}; |
|  |  |
|  | void loop() |
|  | { |
|  | Serial.println("In loop"); |
|  | CAN.sendMsgBuf(0x43, 0, 8, stmp); |
|  | delay(1000); |
|  | } |

[**view raw**](https://gist.github.com/elktros/d7a64f7becc6b067e2d0dcb145392acf/raw/35a4e3e19f83a49c5b4d4766c1180ff01f6eb3ba/Arduino_MCP_2515_CAN_Tutorial_Tran.ino)[**Arduino\_MCP\_2515\_CAN\_Tutorial\_Tran.ino**](https://gist.github.com/elktros/d7a64f7becc6b067e2d0dcb145392acf#file-arduino_mcp_2515_can_tutorial_tran-ino) hosted with  by [**GitHub**](https://github.com/)

#### Receiver Code

|  |  |
| --- | --- |
|  | #include <SPI.h> |
|  | #include "mcp\_can.h" |
|  |  |
|  | const int spiCSPin = 10; |
|  | const int ledPin = 2; |
|  | boolean ledON = 1; |
|  |  |
|  | MCP\_CAN CAN(spiCSPin); |
|  |  |
|  | void setup() |
|  | { |
|  | Serial.begin(115200); |
|  | pinMode(ledPin,OUTPUT); |
|  |  |
|  | while (CAN\_OK != CAN.begin(CAN\_500KBPS)) |
|  | { |
|  | Serial.println("CAN BUS Init Failed"); |
|  | delay(100); |
|  | } |
|  | Serial.println("CAN BUS Init OK!"); |
|  | } |
|  |  |
|  |  |
|  | void loop() |
|  | { |
|  | unsigned char len = 0; |
|  | unsigned char buf[8]; |
|  |  |
|  | if(CAN\_MSGAVAIL == CAN.checkReceive()) |
|  | { |
|  | CAN.readMsgBuf(&len, buf); |
|  |  |
|  | unsigned long canId = CAN.getCanId(); |
|  |  |
|  | Serial.println("-----------------------------"); |
|  | Serial.print("Data from ID: 0x"); |
|  | Serial.println(canId, HEX); |
|  |  |
|  | for(int i = 0; i<len; i++) |
|  | { |
|  | Serial.print(buf[i]); |
|  | Serial.print("\t"); |
|  | if(ledON && i==0) |
|  | { |
|  |  |
|  | digitalWrite(ledPin, buf[i]); |
|  | ledON = 0; |
|  | delay(500); |
|  | } |
|  | else if((!(ledON)) && i==4) |
|  | { |
|  |  |
|  | digitalWrite(ledPin, buf[i]); |
|  | ledON = 1; |
|  | } |
|  | } |
|  | Serial.println(); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/elktros/87625c133d8fecbafc02aceca20e51a4/raw/9d9d570556a95cd618c1c50aec034d9326a30c8f/Arduino_MCP_2515_CAN_Tutorial_Recv.ino)[**Arduino\_MCP\_2515\_CAN\_Tutorial\_Recv.ino**](https://gist.github.com/elktros/87625c133d8fecbafc02aceca20e51a4#file-arduino_mcp_2515_can_tutorial_recv-ino) hosted with  by [**GitHub**](https://github.com/)

### Working

Working of this project is very simple as all the work is done by the libraries (SPI and CAN). Since CAN is message-based communication, you need to send a message anywhere between 0 and 8 bytes.

In this project, the transmitter is sending a message as 1 1 2 3 0 5 6 7. This message is transmitted over CAN Bus and the receiver receives this message and is displayed on its serial monitor.

Additionally, the 0th and 4th bit i.e. 1 and 0 in the above sequence are extracted separately by the receiver and turns ON and OFF the LED connected to Pin 2 of Arduino.

### Applications

As mentioned in the introduction, CAN is widely used in the field of automobiles. Some of the applications include:

* Electronic Gear Shift System
* Main Interface in Automation (like industrial)
* Medical Equipment
* Robotics
* Auto Start/Stop of Car Engine

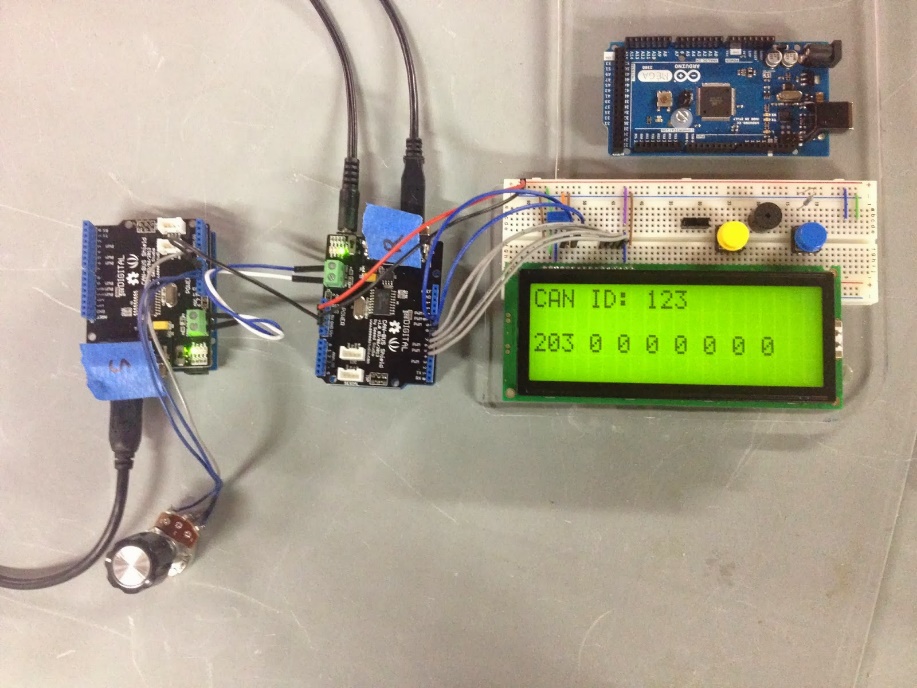
### Arduino - Sending data over a CAN bus

[https://2.bp.blogspot.com/-Xw8-zmtODuU/Ulb0f9QiiPI/AAAAAAAADR4/ZUkDxEwd5I8/s1600/arduino_logo.jpg](http://2.bp.blogspot.com/-Xw8-zmtODuU/Ulb0f9QiiPI/AAAAAAAADR4/ZUkDxEwd5I8/s1600/arduino_logo.jpg)

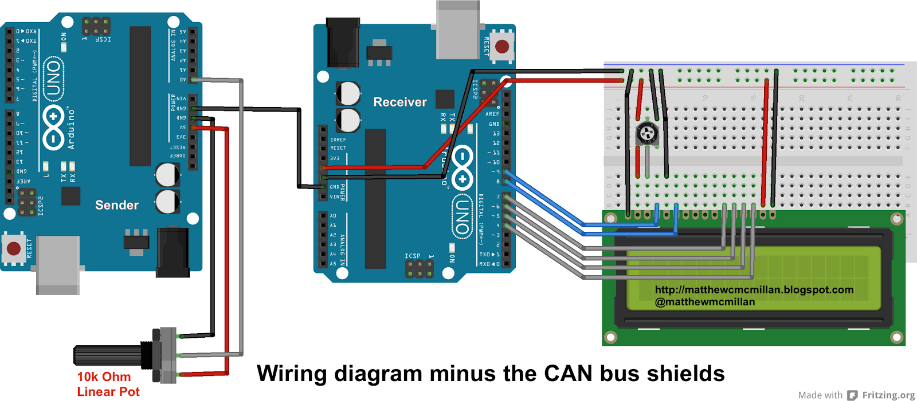
I have been tinkering with CAN buses due to my interest in cars. It's fascinating to me that packets are flying around a modern vehicle controlling nearly everything. Gauges, lights, locks, engine sensors, etc. To have a better understanding of the basics of a CAN bus I wanted to build the simplest possible setup to send and receive CAN messages. I chose two Arduino Uno's with a [Seeed Studio CAN-BUS shield](http://www.seeedstudio.com/depot/canbus-shield-p-1240.html" \t "_blank) attached to each Uno. The Seeed shield is very straight forward and inexpensive. The [Sparkfun CAN-BUS shield](https://www.sparkfun.com/products/10039" \t "_blank) has an SD card slot, LCD connector and GPS connector. All of which are cool but drive up the price and complexity. The Seeed shield only does CAN bus and includes screw terminals which are handy for testing.

|  |  |
| --- | --- |
| [https://4.bp.blogspot.com/-M2db1RAvyEU/Ulb463HQrHI/AAAAAAAADSY/W5vsdysGnFA/s1600/arduino_uno_r3.jpg](http://4.bp.blogspot.com/-M2db1RAvyEU/Ulb463HQrHI/AAAAAAAADSY/W5vsdysGnFA/s1600/arduino_uno_r3.jpg) | [https://1.bp.blogspot.com/-0gcVnv5m4Lk/Ulb3CYt9eMI/AAAAAAAADSM/y8SRFPlUbQQ/s1600/canbus+shield.jpg](http://1.bp.blogspot.com/-0gcVnv5m4Lk/Ulb3CYt9eMI/AAAAAAAADSM/y8SRFPlUbQQ/s1600/canbus+shield.jpg) |
| Arduino Uno R3 | Seeed CAN-BUS Shield |

What I wanted to do with this experiment was transmit the value of an analog pin hooked up to a linear potentiometer. The data would be sent from one Arduino to another over a CAN bus and then display that value on an LCD connected to the second Arduino. Here is a picture of my setup. (Ignore the Mega2560 above the LCD. It's not used here.)

[](http://3.bp.blogspot.com/-I11jC0VMsEQ/Ulb-XMonC8I/AAAAAAAADSo/cJ32Dyfq72g/s1600/IMG_3365.JPG)

And here is a Fritzing diagram minus the CAN-BUS shields.

[](http://1.bp.blogspot.com/-AdFFnPyqkDk/U4I3wGg2sgI/AAAAAAAADg4/8g57aNRxoew/s1600/can_test_bb.png)

#### CAN bus termination

A CAN bus requires 120 Ohm termination resistors at each end of the bus. The Seeed Studio shields have built in termination resistors. When you connect two Seeed CAN bus shields togther like I did in this example you will have a properly terminated CAN bus. If you plan on connecting into an existing CAN bus that already has termination you can disable the built in termination resistors. To disable termination you can cut trace P1 or you can desolder resistor R1.

|  |
| --- |
| [https://3.bp.blogspot.com/-UFLm09G1m9k/U817w1XLykI/AAAAAAAADoI/DSPa9HCwc9I/s1600/seeed_term_resistor.jpg](http://3.bp.blogspot.com/-UFLm09G1m9k/U817w1XLykI/AAAAAAAADoI/DSPa9HCwc9I/s1600/seeed_term_resistor.jpg) |
| Close up view of the Seeed CAN bus shield  termination resistors. |

\*\*Note: I have recently discovered the Seeed Studio CAN-BUS shield v1.0 uses a 60 ohm termination resistor for R3. While that worked for this small demo I later ran into issues when trying to use this shield with other nodes on a CAN bus. This 60 ohm resistor caused me many hours of frustration. If you are going to use this shield with on a bus with multiple nodes I would recommend desoldering R3 and using the correct 120 ohm resistance at the ends of your bus.

#### Connecting into an existing CAN bus

If you are planning on connecting into an existing CAN bus (like in a car) you need to remove/disable the termination resistor on the shield as explained above. The CAN bus in a vehicle already has termination resistors. Adding a new node with a termination resistor will cause errors and disrupt communication on the bus.  
  
Another important step is to connect a common ground between your Arduino board and the vehicle. If you are connecting at the OBD2 port pin 5 provides a signal ground. If you can't find a signal ground wire a chassis ground will suffice.

#### CAN bus messages

So I should probably explain a bit about CAN bus messages. Each message is made up of an id and some data. The id's in hex start at 0x000 and go to 0x7FF or 0 to 2047 in decimal. In most systems lower id values are considered more important. The bus handles collisions by letting the lower id win the collision. The data can be between 1 and 8 bytes for each message. Each byte can have a value from 0 to 255 or in hex 0x00 to 0xFF. When you send a CAN bus message you transmit the id, how many bytes you are sending (this is called DLC) and the actual data. The receiver will only read the number of bytes you said should be in the message. So if you send a DLC of 4 but the message contains 8 bytes the receiver will only read the first 4 bytes. Eight bytes per message is a bit limiting but the tradeoff is the high reliability of the bus. So sometimes you have to be creative with stuffing data into those bytes. If the value you are sending is less than 255 you can just use a single byte. Larger numbers will require using multiple bytes. Ascii codes can be sent but only eight characters per message. Whatever method you use to stuff the data in will also have to be used to un-stuff the data on the receiver. In my simple example here I did some math to limit the range of values to 0-255. An analog pin produces values between 0-1024. I simply divided the result by four to give me data I could send in a single byte.

CAN buses can operate at several different speeds up to 1 Mbit/s. Typical rates are 100 kbit/s, 125 kbit/s and 500 kbit/s. Slower rates allow for longer length buses. All devices on a bus must transmit at the same speed. The [CAN bus wikipedia page](http://en.wikipedia.org/wiki/CAN_bus) is a good place to start if you want to learn more about the CAN protocol.

#### Code

I started with the example code provided by Seeed and modified it to add in the LCD output on the 'receiver' device and added reading of the potentiometer on A0 for the value that is transmitted. They have basic examples for send and receive. You can find some good info on [their wiki page](http://wiki.seeedstudio.com/CAN-BUS_Shield_V1.2/). Their [libraries are available here](https://github.com/Seeed-Studio/CAN_BUS_Shield). On my Mac I created the directory ~/Documents/Arduino/libraries/CAN\_BUS\_Shield for the library files. I unzipped the file and copied over the .h and .cpp files into that new directory. The zip file also contains the send and receive examples.  
  
Note that normally devices on a CAN bus are both receivers and transmitters of data. This is a simplified example where each device is only doing one task.

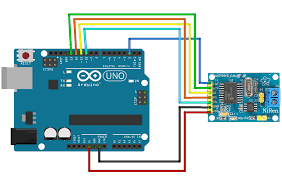
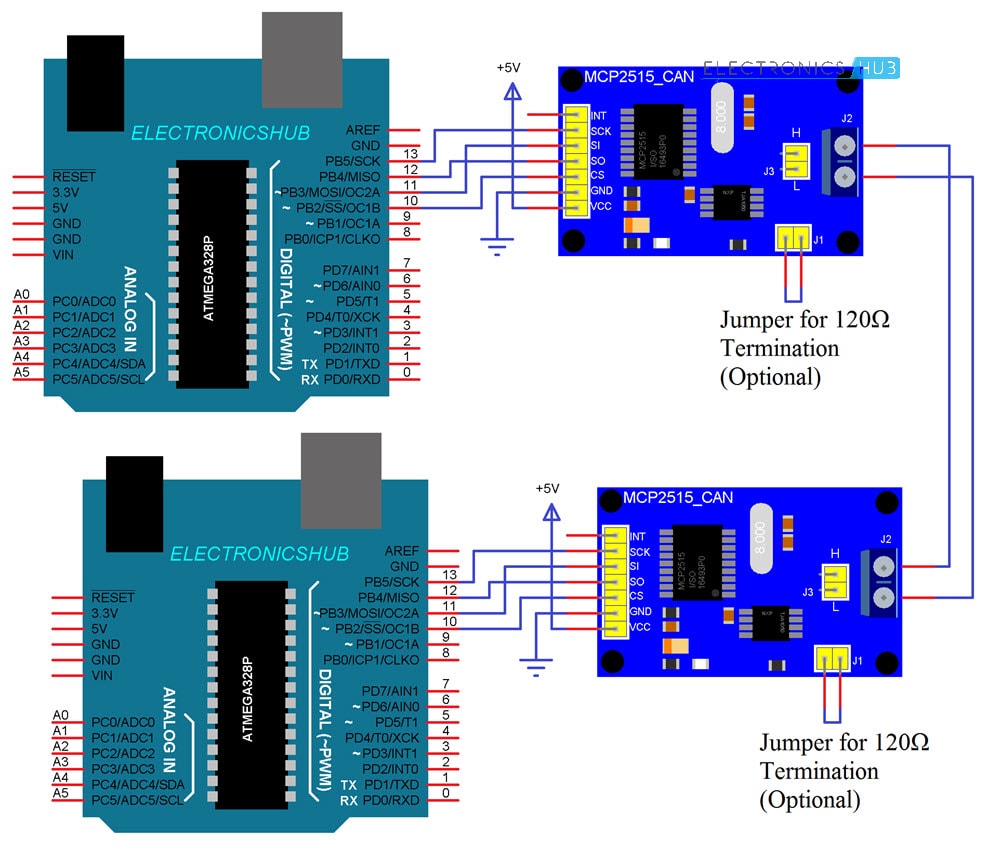
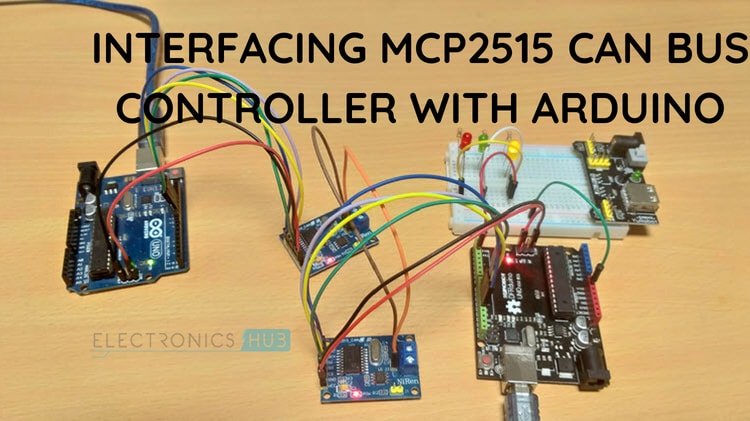
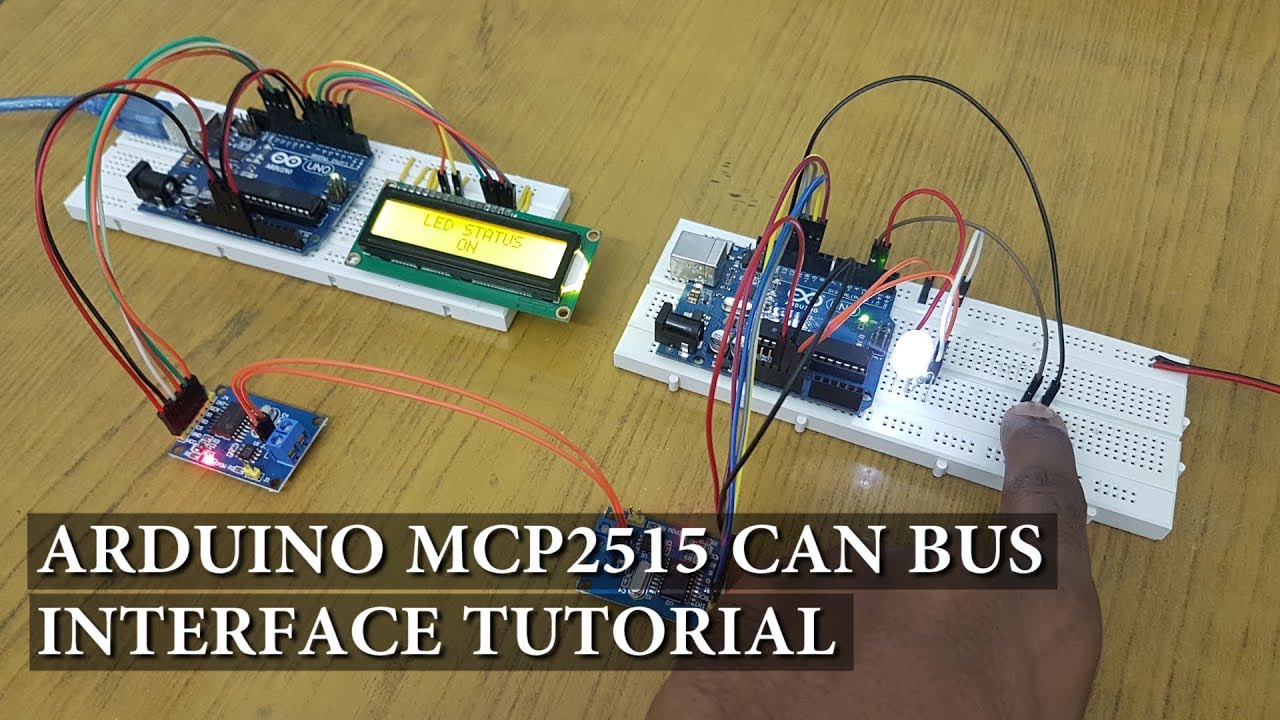
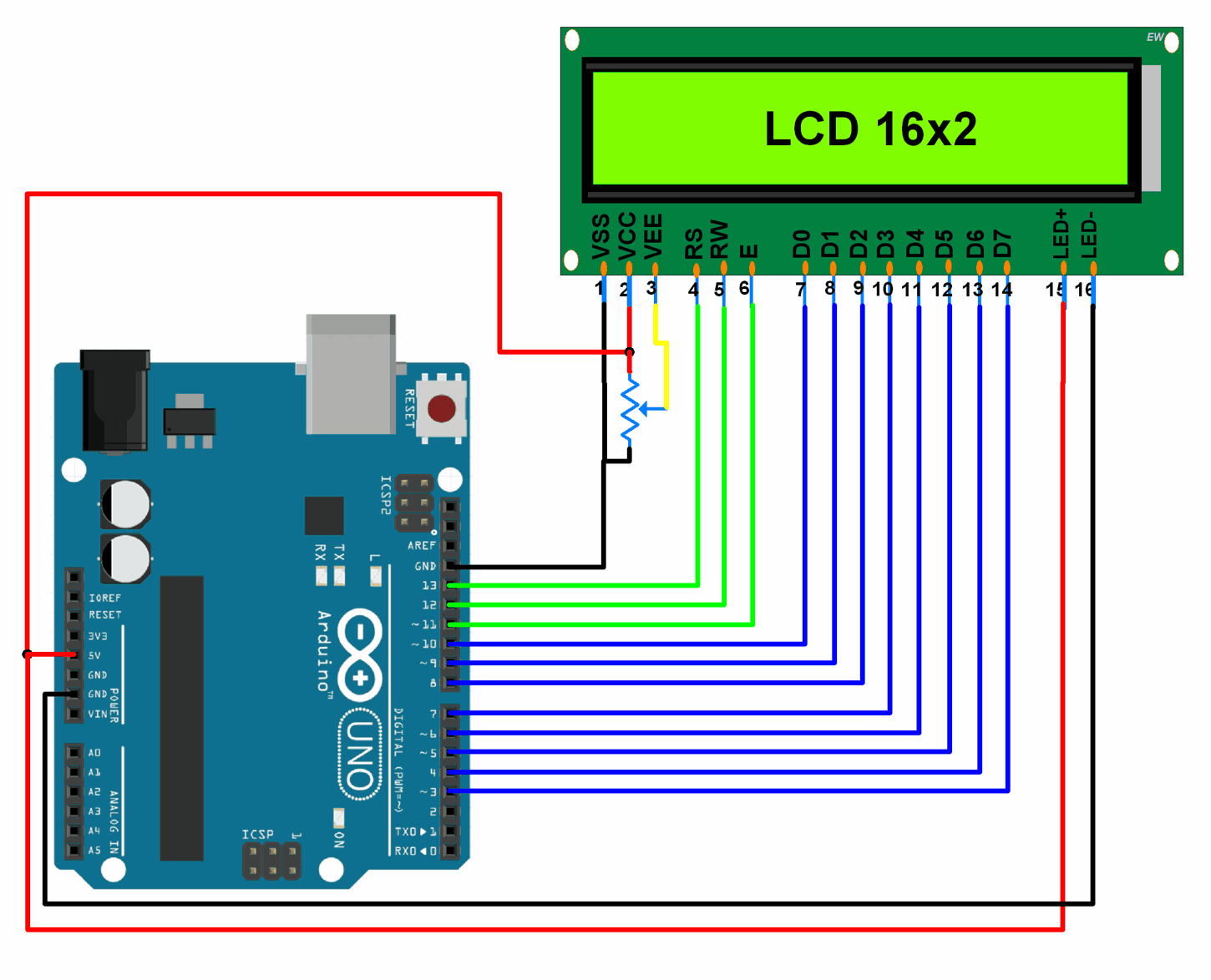
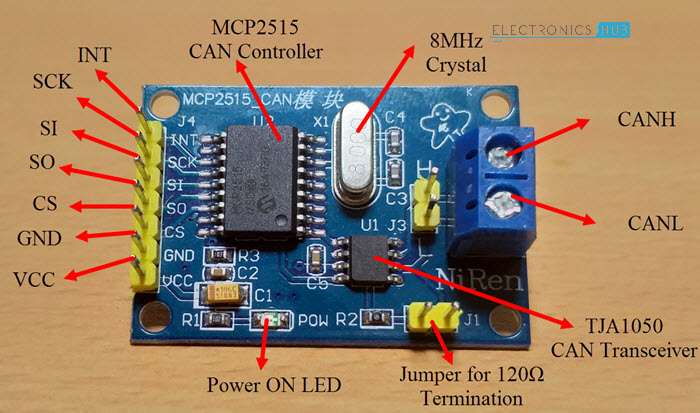
### Sender code

|  |  |
| --- | --- |
|  | // demo: CAN-BUS Shield, send data |
|  | #include <mcp\_can.h> |
|  | #include <SPI.h> |
|  |  |
|  | //Pot for adjusting value |
|  | int sensorPin = A0; |
|  | int sensorValue = 0; |
|  | int cantxValue = 0; |
|  |  |
|  | void setup() |
|  | { |
|  | Serial.begin(115200); |
|  | // init can bus, baudrate: 100k |
|  | if(CAN.begin(CAN\_100KBPS) ==CAN\_OK) Serial.print("can init ok!!\r\n"); |
|  | else Serial.print("Can init fail!!\r\n"); |
|  | } |
|  |  |
|  | //Some sample CAN messages |
|  | unsigned char msg1[8] = {0, 1, 2, 3, 4, 5, 6, 7}; |
|  | unsigned char msg2[8] = {0xFF, 0x01, 0x10, 0x0A, 0x00, 0x00, 0x00, 0x00}; |
|  | unsigned char msg3[4] = {0xFF, 0x01, 0x10, 0x0A}; |
|  |  |
|  |  |
|  | void loop() |
|  | { |
|  | //Read the value of the pot |
|  | sensorValue = analogRead(sensorPin); |
|  | //Each CAN bus byte can store a value between 0-255. |
|  | //Dividing sensorValue by 4 puts us in that range. |
|  | cantxValue = sensorValue / 4; |
|  | Serial.print("cantxValue: "); |
|  | Serial.print(cantxValue); |
|  | Serial.println(); |
|  | //Create data packet for CAN message |
|  | unsigned char canMsg[8] = {cantxValue, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}; |
|  | // send data: id = 0x123, standrad flame, data len = 8, stmp: data buf |
|  | CAN.sendMsgBuf(0x07B, 0, 8, canMsg); |
|  | delay(100); |
|  | } |

[**view raw**](https://gist.github.com/matt448/6925235/raw/d14be1d4977af4285297ad8bbd8b15e19854742f/can_sender.ino)[**can\_sender.ino**](https://gist.github.com/matt448/6925235#file-can_sender-ino) hosted with ❤ by [**GitHub**](https://github.com/)

### Receiver code

|  |  |
| --- | --- |
|  | // demo: CAN-BUS Shield, receive data |
|  | #include "mcp\_can.h" |
|  | #include <SPI.h> |
|  | #include <LiquidCrystal.h> |
|  | #include <stdio.h> |
|  | #define INT8U unsigned char |
|  |  |
|  | INT8U Flag\_Recv = 0; |
|  | INT8U len = 0; |
|  | INT8U buf[8]; |
|  | INT32U canId = 0x000; |
|  | char str[20]; |
|  | LiquidCrystal lcd(9, 8, 7, 6, 5, 4); |
|  |  |
|  | void setup() |
|  | { |
|  | lcd.begin(20, 4); |
|  | CAN.begin(CAN\_100KBPS); // init can bus : baudrate = 100k |
|  | attachInterrupt(0, MCP2515\_ISR, FALLING); // start interrupt |
|  | Serial.begin(115200); |
|  | } |
|  |  |
|  | void MCP2515\_ISR() |
|  | { |
|  | Flag\_Recv = 1; |
|  | } |
|  |  |
|  | void loop() |
|  | { |
|  | if(Flag\_Recv) // check if data was recieved |
|  | { |
|  | Flag\_Recv = 0; // clear flag |
|  | CAN.readMsgBuf(&len, buf); // read data, len: data length, buf: data buf |
|  | canId = CAN.getCanId(); |
|  | //Print data to the serial console |
|  | //and the LCD display |
|  | Serial.println("CAN\_BUS GET DATA!"); |
|  | Serial.print("CAN ID: "); |
|  | Serial.println(canId); |
|  | lcd.setCursor(0, 0); |
|  | lcd.print("CAN ID: "); |
|  | lcd.print(canId); |
|  | lcd.setCursor(0, 2); |
|  | Serial.print("data len = ");Serial.println(len); |
|  | //This loops through each byte of data and prints it |
|  | for(int i = 0; i<len; i++) // print the data |
|  | { |
|  | Serial.print(buf[i]);Serial.print("\t"); |
|  | lcd.print(buf[i]); |
|  | lcd.print(" "); |
|  | } |
|  | Serial.println(); |
|  | delay(50); |
|  | } |
|  | } |



#include <SPI.h>  
#include CAN.h>  
  
/\* This program is a CAN-bus monitor.  It listens on a  
\* CAN bus and prints out the IDs and data fields of  
\* any CAN messages it hears.  \*/  
  
CanMessage message;  
byte i;  
  
void setup()  
{  
 CAN.begin(CAN\_SPEED\_500000);  
 CAN.setMode (CAN\_MODE\_LISTEN\_ONLY);  
 Serial.begin(115200);  
}  
  
void loop()  
{  
 if (CAN.available()) {  
   message = CAN.getMessage ();  
   message.print (HEX);  
 }  
}