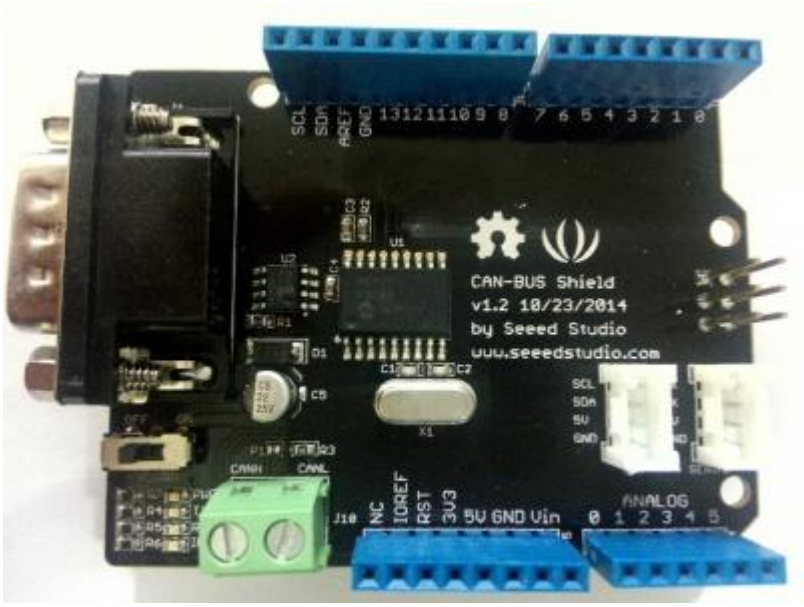
CAN-BUS Shield V1.2



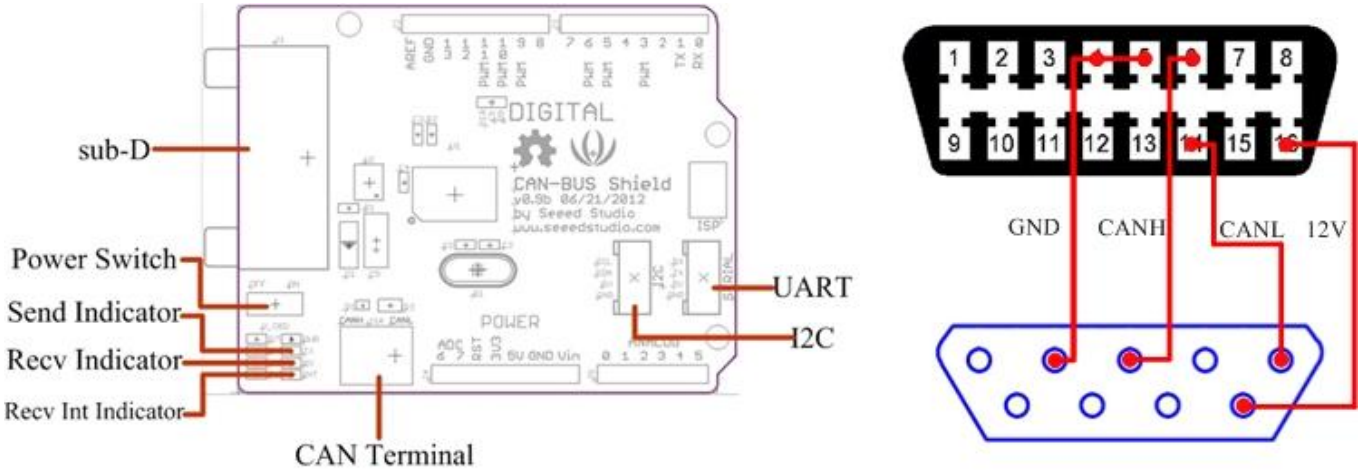
CAN-BUS is a common industrial bus because of its long travel distance, medium communication speed and high reliability. It is commonly found on modern machine tools and as an automotive diagnostic bus. This CAN-BUS Shield adopts MCP2515 CAN Bus controller with SPI interface and MCP2551 CAN transceiver to give your Arduino/Seeeduino CAN-BUS capibility. With an OBD-II converter cable added on and the OBD-II library imported, you are ready to build an onboard diagnostic device or data logger.

Features

- Implements CAN V2.0B at up to 1 Mb/s
- SPI Interface up to 10 MHz
- Standard (11 bit) and extended (29 bit) data and remote frames
- Two receive buffers with prioritized message storage
- Industrial standard 9 pin sub-D connector
- Two LED indicators

Specifications

Voltage：5V
Dimensions： 68x53mm
Net Weight：50g

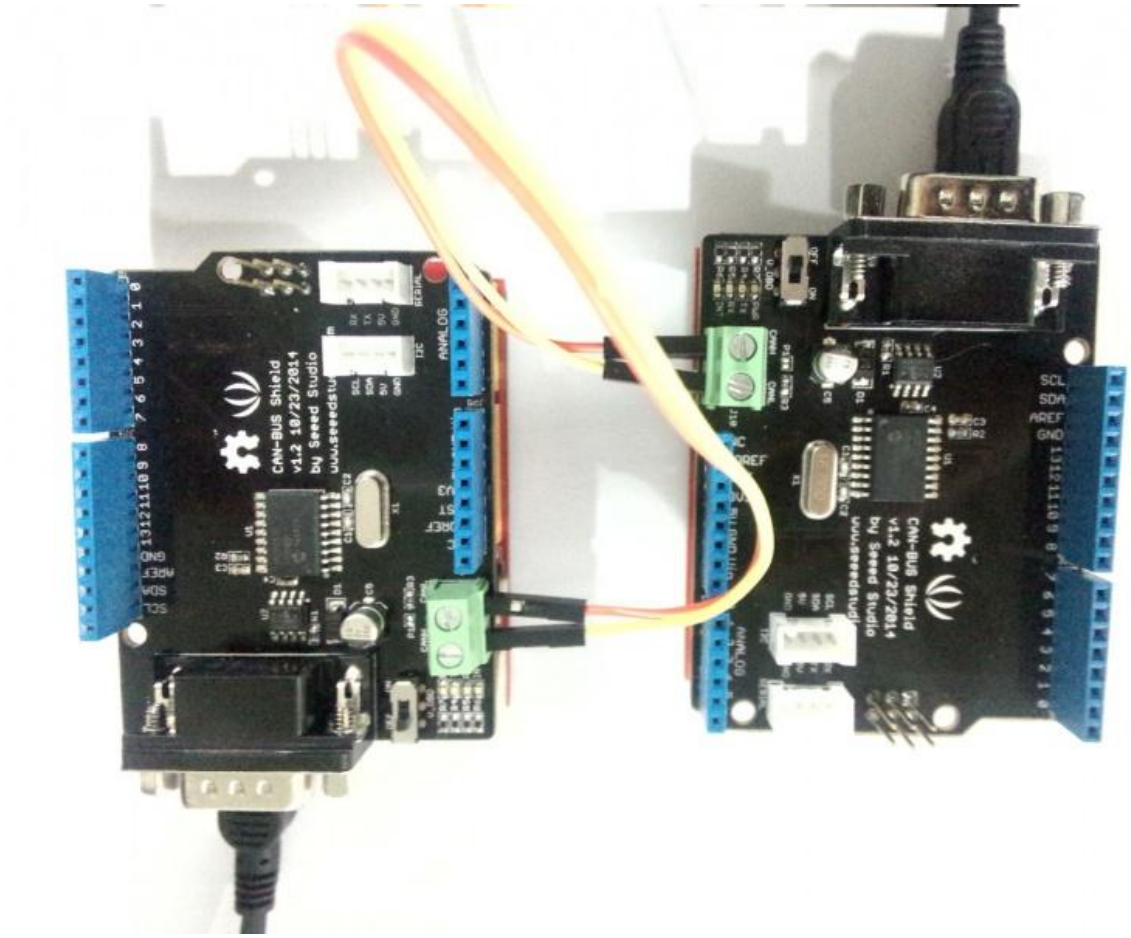


Note:

When you use more than two CAN Bus Shield in one net, you should concern about the impedance.

You can just cut P1 in the PCB with a knife, or just remove R3 on the PCB.

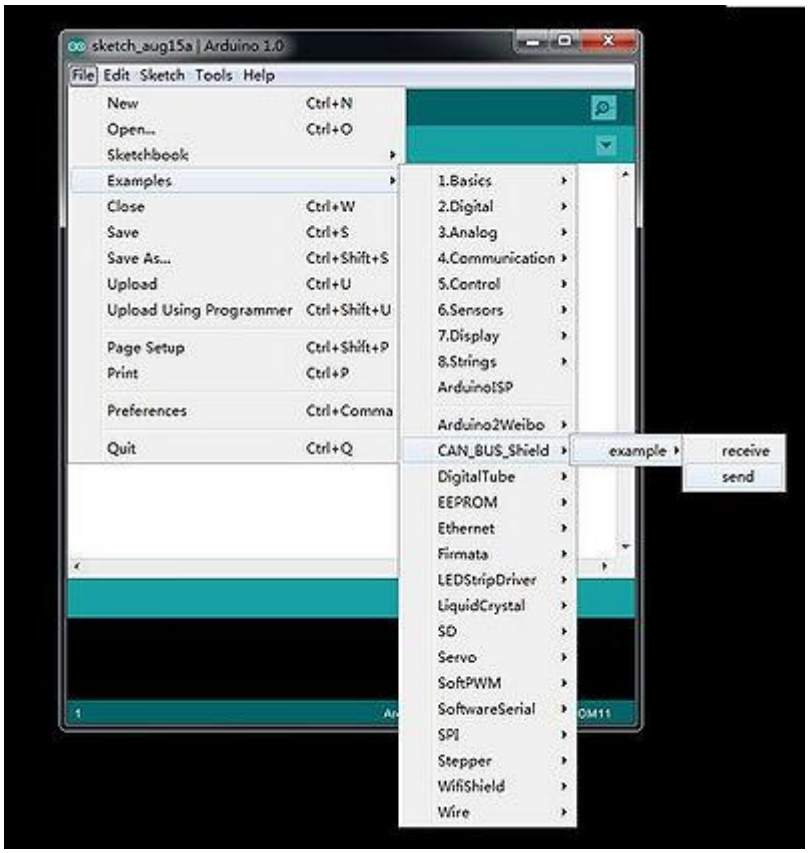
Demonstration



1. Download the CAN-BUS Source code file for Arduino 1.0 and release it in the libraries file in the Arduino-1.0 program.: ..\arduino-1.0\libraries.

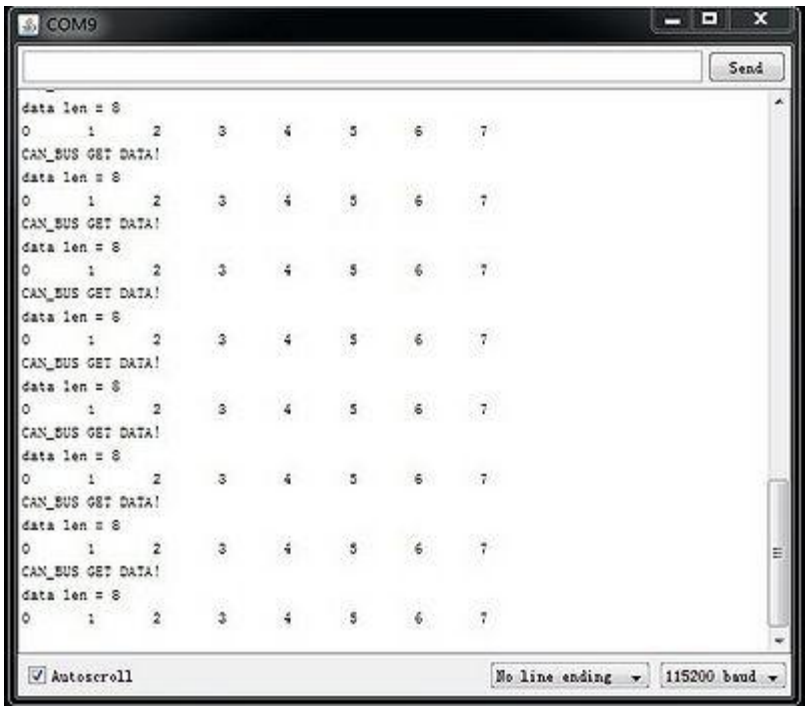
If the folder name include "-master", just remove it.

2. Open the Arduino-1.0, and you will find 3 examples: receive_check ,send and receive_interrupt. Here we'll use send and receive_check, open it then you should get two programming windows now.



3. Upload two examples to two boards separately. Choose the board via the path: Tools -->Serial Port-->COMX. Note down which board is assigned as a "send" node and which board is assigned as a "receive" node.

4. Open the "Serial Monitor" on the "receive" COM, you will get message sent from the "send" node. Here we have the preset message "0 1 2 3 4 5 6 7" showing in the following picture.



Reference

1. Set the BaudRate

This function is used to initialize the baudrate of the CAN Bus system.

The available baudrates are listed as follws:

CAN_5KBPS, CAN_10KBPS, CAN_20KBPS, CAN_40KBPS, CAN_50KBPS, CAN_80KBPS, CAN_100KBPS, CAN_125KBPS, CAN_200KBPS, CAN_250KBPS, CAN_500KBPS and CAN_1000KBPS

2. Set Receive Mask and Filter

There are 2 receive mask registers and 5 filter registers on the controller chip that guarantee you get data from the target device. They are useful especially in a large network consisting of numerous nodes.

We provide two functions for you to utilize these mask and filter registers. They are:

init_Mask(unsigned char num, unsigned char ext, unsigned char ulData); & init_Filt(unsigned char num, unsigned char ext, unsigned char ulData);

"num" represents which register to use. You can fill 0 or 1 for mask and 0 to 5 for filter.

"ext" represents the status of the frame. 0 means it's a mask or filter for a standard frame. 1 means it's for a extended frame.

"ulData" represents the content of the mask of filter.

3. Check Receive

The MCP2515 can operate in either a polled mode, where the software checks for a received frame, or using additional pins to signal that a frame has been received or transmit completed. Use the following function to poll for received frames.

INT8U MCP_CAN::checkReceive(void);

The function will return 1 if a frame arrives, and 0 if nothing arrives.

4. Get CAN ID

When some data arrive, you can use the following function to get the CAN ID of the "send" node.

INT32U MCP_CAN::getCanId(void)

5. Send Data

CAN.sendMsgBuf(INT8U id, INT8U ext, INT8U len, data_buf);

is a function to send data onto the bus. In which:

"id" represents where the data come from.

"ext" represents the status of the frame. '0' means standard frame. '1' means extended frame.

"len" represents the length of this frame.

"data_buf" is the content of this message.

For example, In the 'send' example, we have:

```
unsigned char stmp[8] = {0, 1, 2, 3, 4, 5, 6, 7};

CAN.sendMsgBuf(0x00, 0, 8, stmp); //send out the message 'stmp' to the bus and tell
other devices this is a standard frame from 0x00.
```

6. Receive Data

The following function is used to receive data on the 'receive' node:

CAN.readMsgBuf(unsigned char len, unsigned char buf);

In conditions that masks and filters have been set. This function can only get frames that meet the requirements of masks and filters.

"len" represents the data length.

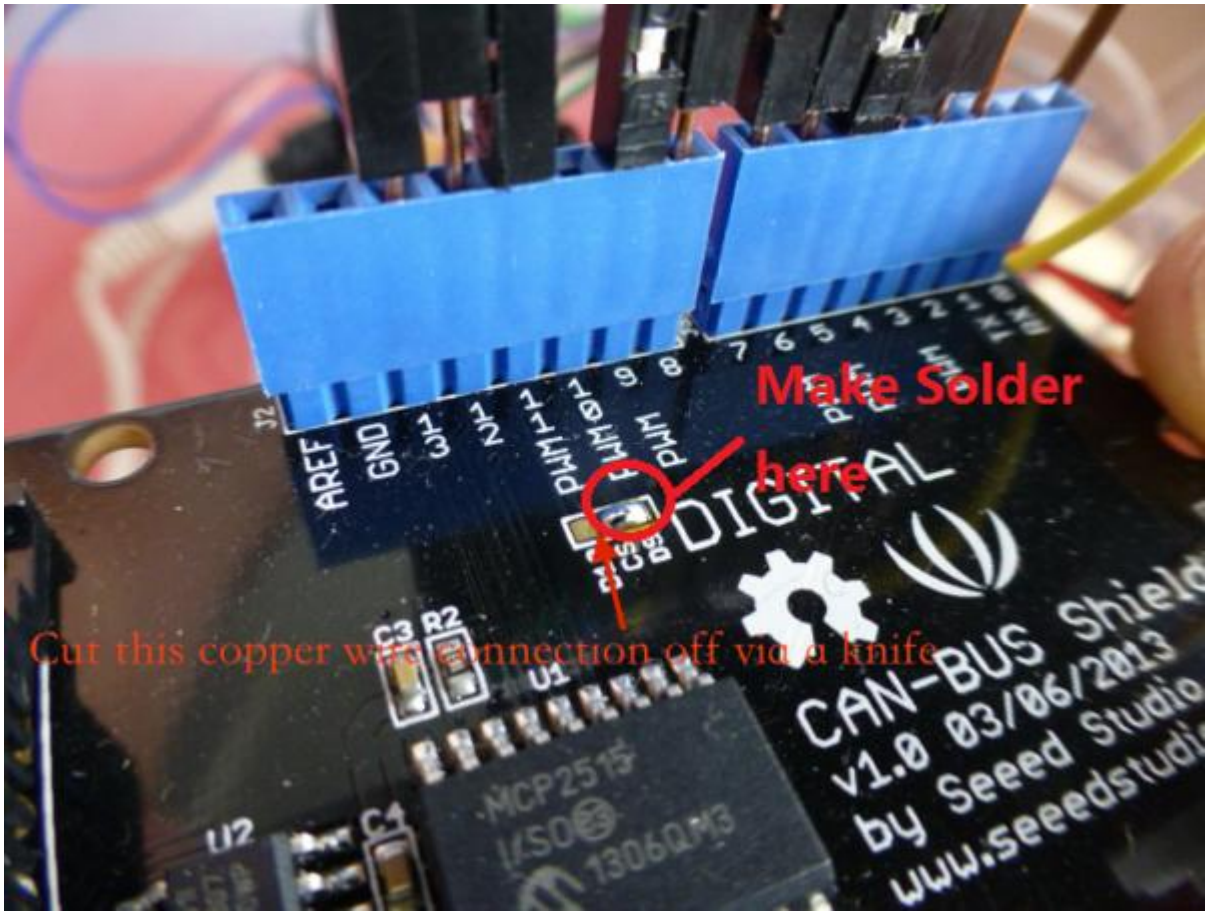
"buf" is where you store the data.

FAQ

How to Change SS Pin

The SPI SS pin is default D9, you can change it to D10 easily.

Firstly, cut off the copper wire between CS and digital 9 via a knife, then do some soldering just like the following image



Then, you need to change the SS Pin in the library.

You can refer to here

How to remove the terminal resistor

There's a 62 Ohm(120 Ohm in version 1.1 hardware) on CAN BUS Shield. Sometime it's not need. You can remove it easily, just as follow:

Resources

- CAN-BUS Shield V1.2 Schmatics
- CAN-BUS Shield V1.2 eagle file
- CAN-BUS Source code file for Arduino 1.0
- MCP2515 datasheet
- MCP2551 datasheet