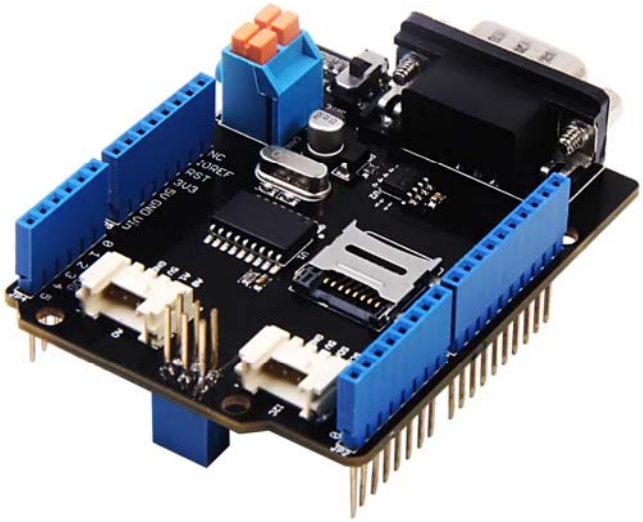
EN: This Datasheet/Document is presented by the manufacturer.

Please visit our website for pricing and availability at www.hestore.hu.

CAN-BUS Shield V2.0

Introduction



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, such 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 capability. 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.

Previously we have made two versions of CAN-BUS Shield, the V1.0 and V1.2. They are all awesome shields that widely liked by our users. In order to make it better, several months ago we conducted a survey about CAN-BUS Shield V1.2 and received many valuable advices (Thanks to all the users who replied to us), so we decided to make an update and here it is - CAN-BUS Shield V2.

Version

This document applies to the following version of products:

Version	Released Date	
CAN BUS Shield V1.0	Oct 14, 2012	
CAN BUS Shield V1.1	Aug 10, 2013	
CAN BUS Shield V1.2	Jan 05, 2015	
CAN BUS Shield V2.0	Aug 01,2017	

Version Tracker

Features	V1.2	V2.0
CAN-BUS Controller	MCP2515	MCP2515
CAN Transceiver	MCP2551	MCP2551
Default OBD Pinout	OBD-II Standard	OBD-II Standard
CAN Standard Pinout	Not compatible	Compatible (jumper)
INT Pin	Not changeable	D2 or D3 (jumper)
CS pin for TF card slot	No TF card slot	D4 or D5 (jumper)

Features	V1.2	V2.0
Serial Grove	D0/D1	A0/A1
I2C Grove	A4/A5	SDA/SCL
Grove Orientation	Vertical	Horizontal
P1 pad	Front of the shied	Back of the shield

What's new in CAN BUS Shield V2.0

Hardware

- OBD-II or CAN standard pinout can be selected by switching jumpers on DB9 interface, the default pinout is OBD-II.
- Add a TF card slot for data storage and the CS pin can be either set to D4 or D5.
- The INT pin can be set to D2 or D3 by switching jumpers on the back of the shield.
- Moved the P1 pad from front to the back of the shield to make it easier to cut and solder.
- Consider that the D0/D1 pin are usually used for downloading code, we changed the serial Grove connector to pin A0/A1.
- The I2C grove connector is also changed to more reasonable standard SDA/SCL pin instead of previous A4/A5.
- The two grove connectors are both changed to horizontal rather than vertical to the shield so that it would be more convenient when connecting to other grove modules.

Software

- Add the function and example to access the data of your car.
- Add the function to read the SD card.
- Add the example to store the data of your car into the SD card.
- Fix some bugs and optimize some program.

D-Sub CANbus PinOut

pin#	Signal names	Signal Description
1	Reserved	Upgrade Path
2	CAN_L	Dominant Low
3	CAN_GND	Ground
4	Reserved	Upgrade Path
5	CAN_SHLD	Shiled, Optional
6	GND	Ground, Optional
7	CAN_H	Dominant High
8	Reserved	Upgrade Path
9	CAN_V+	Power, Optional

What if I want to connect this shield to my car

If you want to read data or control your car, there's an OBD>DB9 cable available for you, this cable make easier to connect to OBD-connector and DB9-connector. This cable will also work with anything that has a OBD-connector. Add a power switch makes such a satisfying click.



USB-CAN Analyzer

If you want a CAN Bus Analyzer to debug your CAN Bus, this USB-CAN Analyzer is recommended.



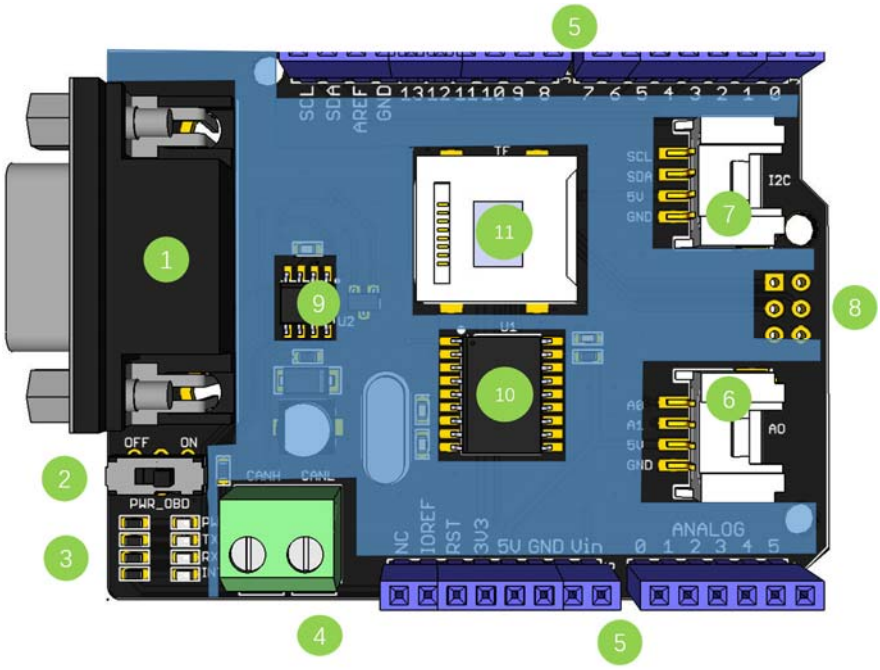
Features

- Implements CAN V2.0B at up to 1 Mb/s
- Industrial standard 9 pin sub-D connector
- OBD-II and CAN standard pinout selectable.
- Changeable chip select pin
- Changeable CS pin for TF card slot
- Changeable INT pin
- Screw terminal that easily to connect CAN_H and CAN_L
- Arduino Uno pin headers
- 2 Grove connectors (I2C and UART)
- SPI Interface up to 10 MHz
- Standard (11 bit) and extended (29 bit) data and remote frames
- Two receive buffers with prioritized message storage

Note

CAN BUS Shield Work well with Arduino UNO (ATmega328), Arduino Mega (ATmega1280/2560) as well as Arduino Leonardo (ATmega32U4).

Hardware Overview

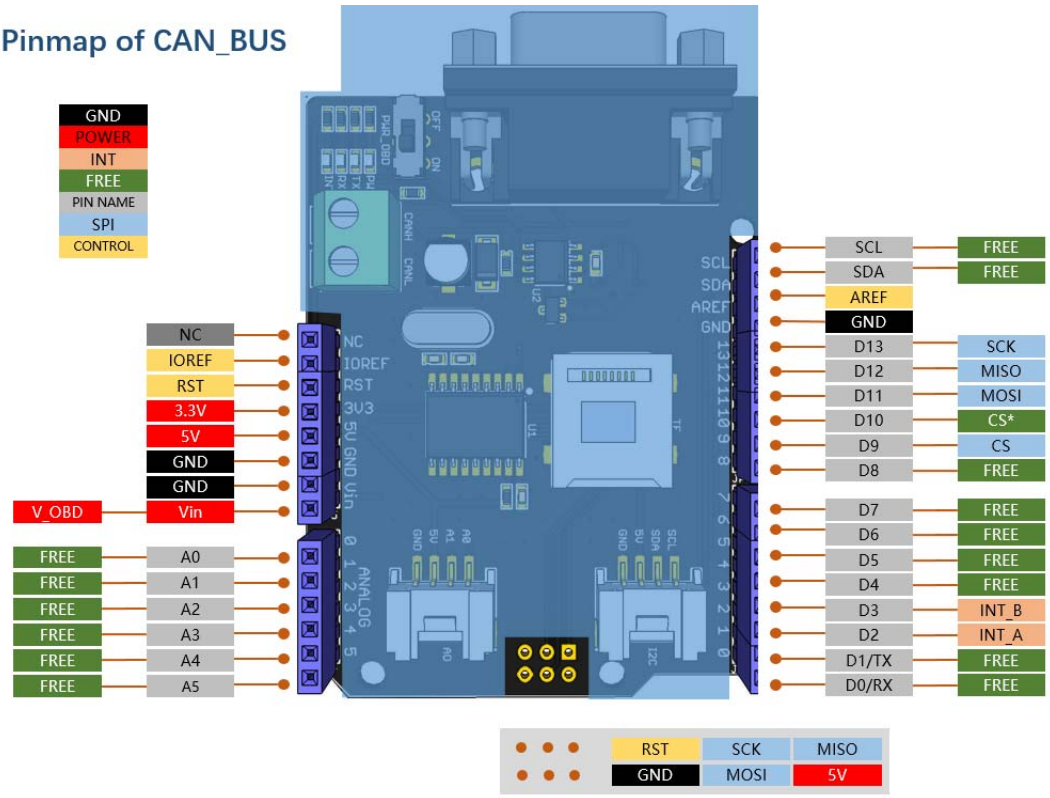


1. **DB9 Interface** - to connect to OBDII Interface via a DBG-OBD Cable.
2. **V_OBD** - It gets power from OBDII Interface (from DB9)
3. **Led Indicator**:
   - **PWR**: power
   - **TX**: blink when the data is sending
   - **RX**: blink when there's data receiving
   - **INT**: data interrupt
4. **Terminal** - CAN_H and CAN_L
5. **Arduino UNO pin out**
6. **Serial Grove connector**
7. **I2C Grove connector**
8. **ICSP pins**
9. **IC** - MCP2551, a high-speed CAN transceiver (datasheet)
10. **IC** - MCP2515, stand-alone CAN controller with SPI interface (datasheet)
11. **SD card slot**

Warning

When you use more than two CAN Bus Shield in one net, you should take the impedance into consideration. You should either cut P1 in the PCB with a knife, or just remove R3 on the PCB.

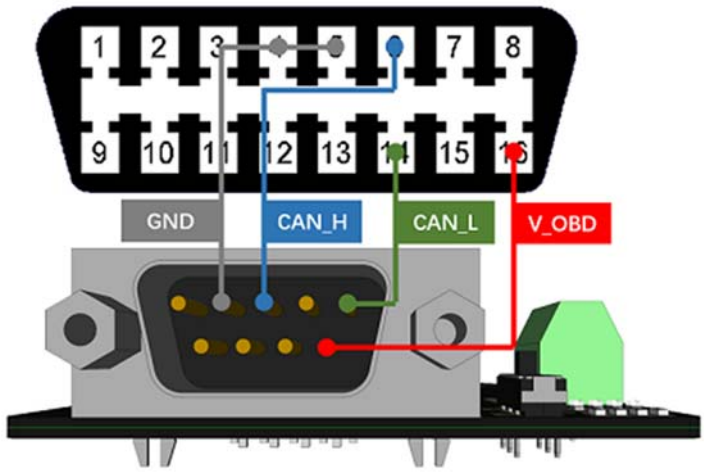
Pin map



Note

- The FREE pin is available for the other usages.

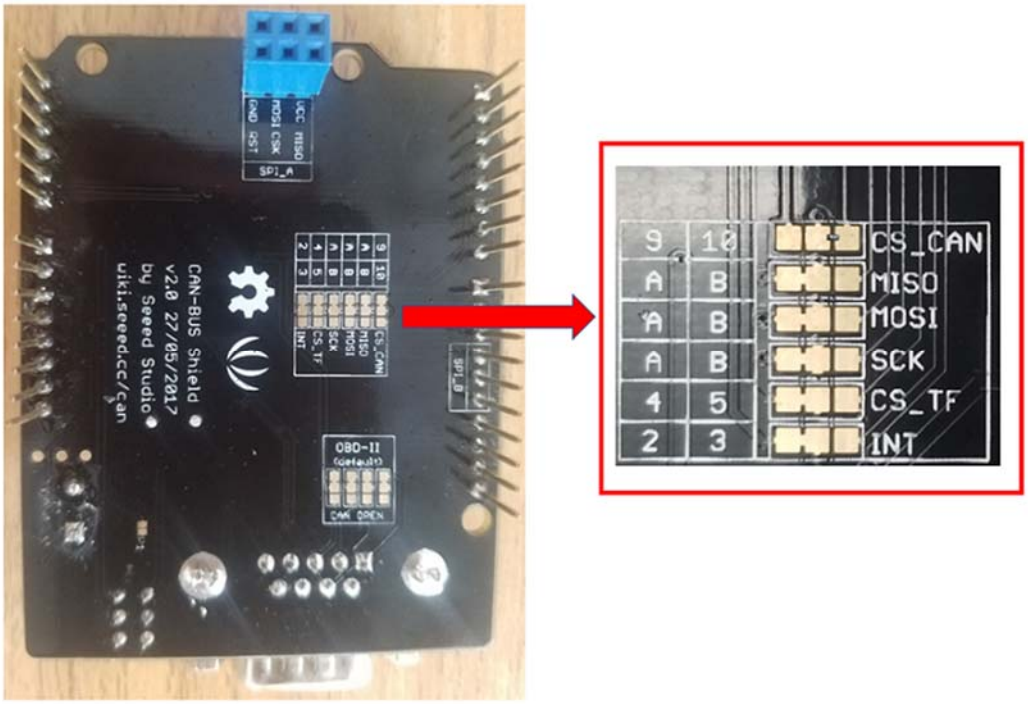
DB9&OBDii Interface



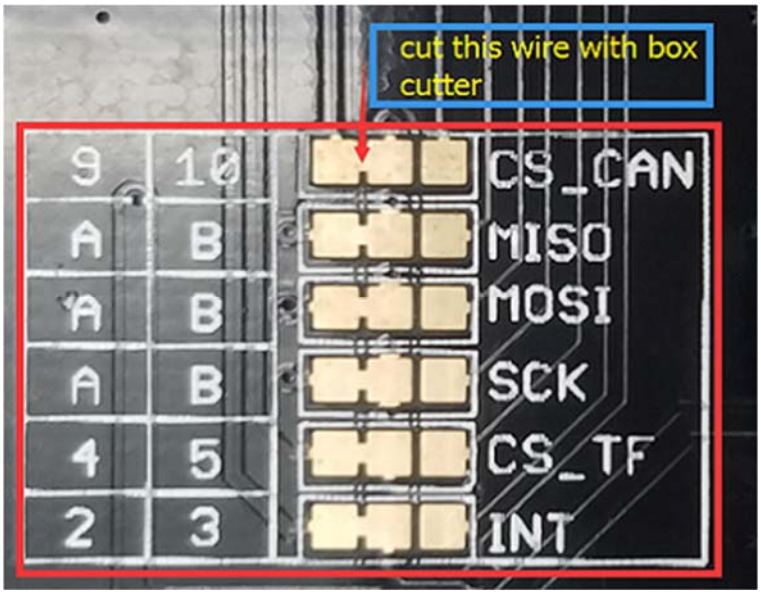
CS_CAN pin

SPI_CS pin of V2.0 is connected to **D9** by default. If you want to change to **D10**, please follow below instructions.

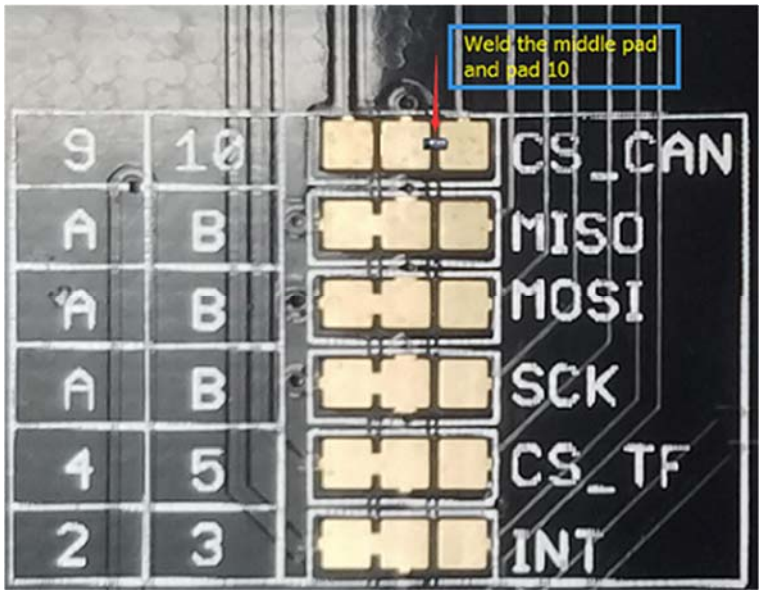
- Step1: Take a look at the backside of the PCBA, you will find a pad named CS_CAN.



- Step2: Cut the wire between pad9 and the middle pad.



- Step3:Weld the middle pad and pad 10.



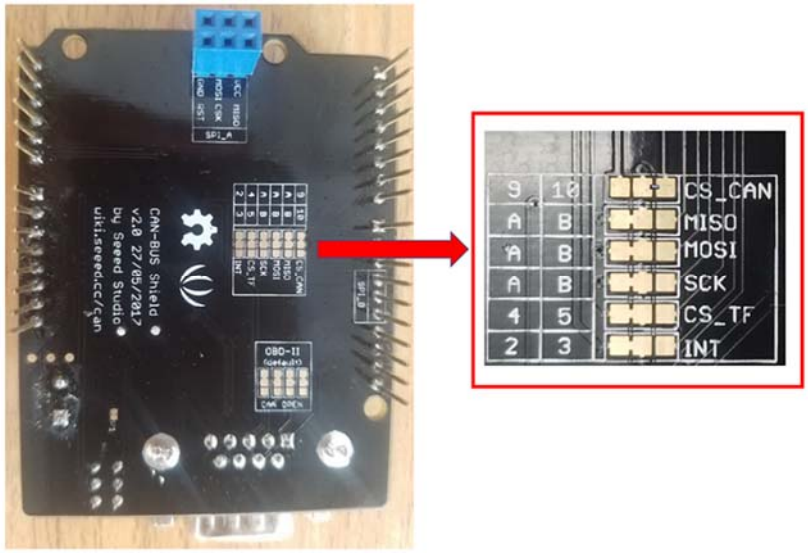
Warning

Be careful with the box cutter, it's easy to hurt yourself or the PCBA.

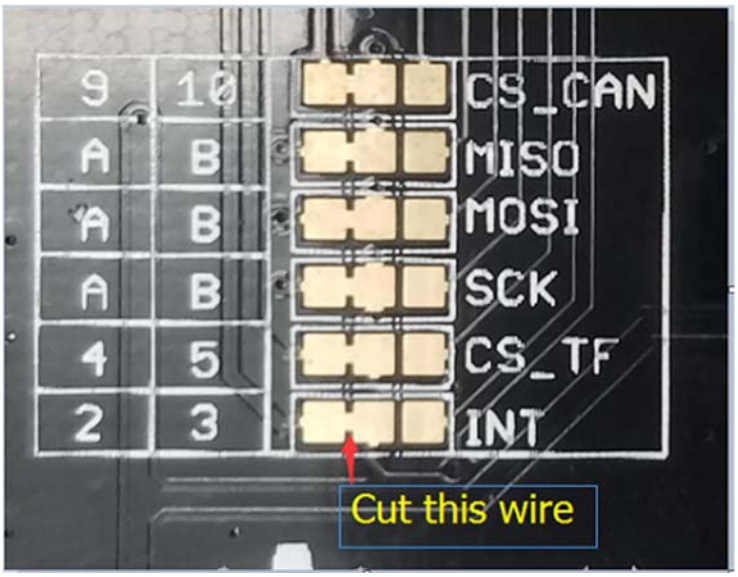
INT pin

INT pin of V2.0 is connected to **D2** by default. If you want to change to **D3**, please follow below instructions.

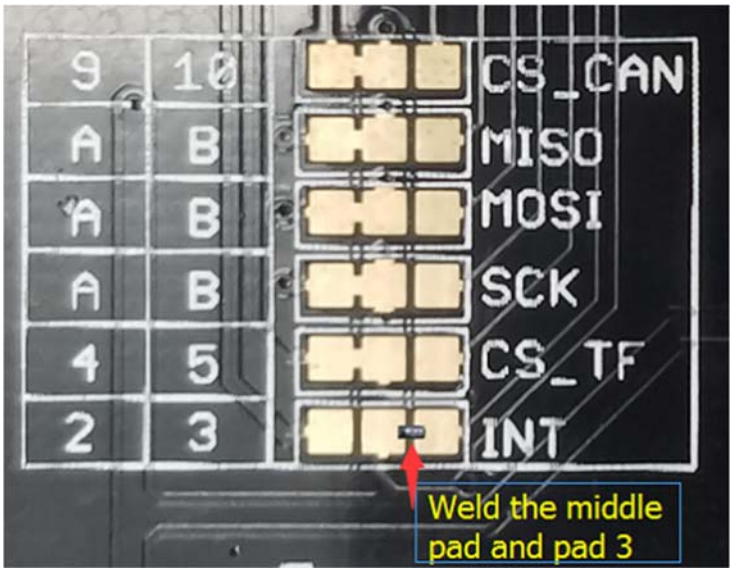
- Step1: Take a look at the backside of the PCBA, you will find a pad named INT.



- Step2: Cut the wire between pad 2 and the middle pad.



- Step3:Weld the middle pad and pad 3.



SPI pins

The SPI pins (SCK, MISO, MOSI) are routed to the ICSP pins by default. But for some boards, the SPI pins are located at D11~D13. if this happens, you need make some change to the PCBA. Take a look at the backside of the PCBA, there're three pads, MOSI, MISO and SCK, they are connected to A by default. You can change them to B if needed.

Note

For Arduino UNO, Arduino Mega, Arduino Leonardo and any others AVR based Arduino boards, it works well by default setting.

Warning

Be careful when you are going to change SPI pins, it's easy to hurt yourself or the PCBA.

Getting Started

Here's a simple example to show you how CAN-BUS Shield works. In this example we need 2 pieces of CAN-BUS Shields as well as Arduino or Seeeduino.

Note

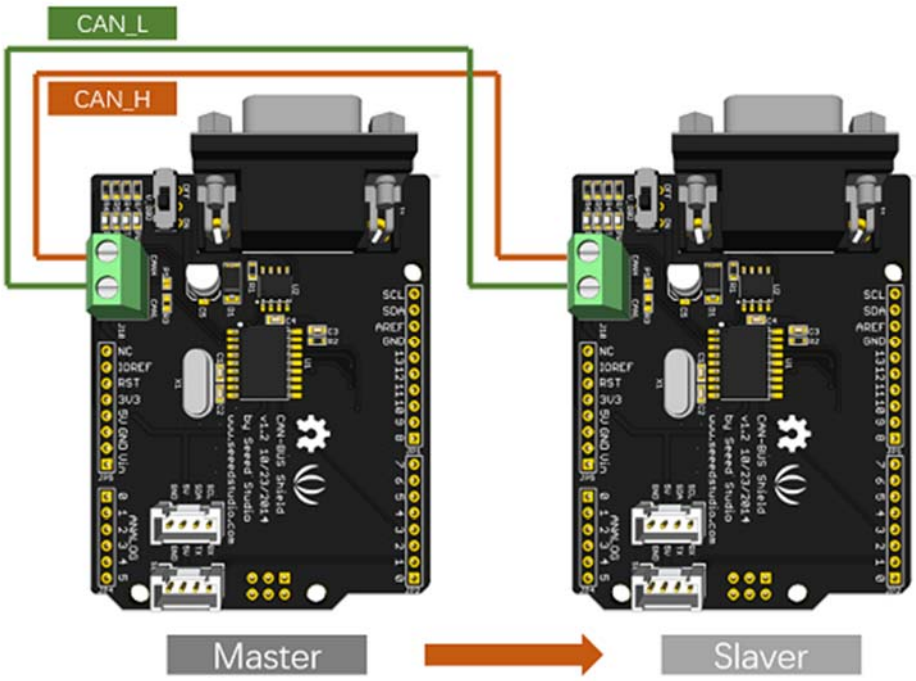
This example is built under Arduino IDE version 1.6.9.

STEP1: What do we need

Name	Function	Qty	
CAN-BUS Shield	CAN Bus communication	2	
Seeeduino V4.2	Controller	2	
Jumper Wire	connection	2	

STEP2: Hardware Connection

Insert each CAN-BUS Shield into Seeeduino V4.2, and connect the 2 CAN-BUS Shield together via 2 jumper wires. Shown as below images.



Note

CAN_H to CAN_H, CAN_L to CAN_L

STEP3: Software

Please follow how to install an arduino library procedures to install CAN BUS shield library.

Click on below button to download the library.
http://wiki.seeed.cc/How_to_install_Arduino_Library/

Download CAN BUS Shield Library

https://github.com/Seeed-Studio/CAN_BUS_Shield

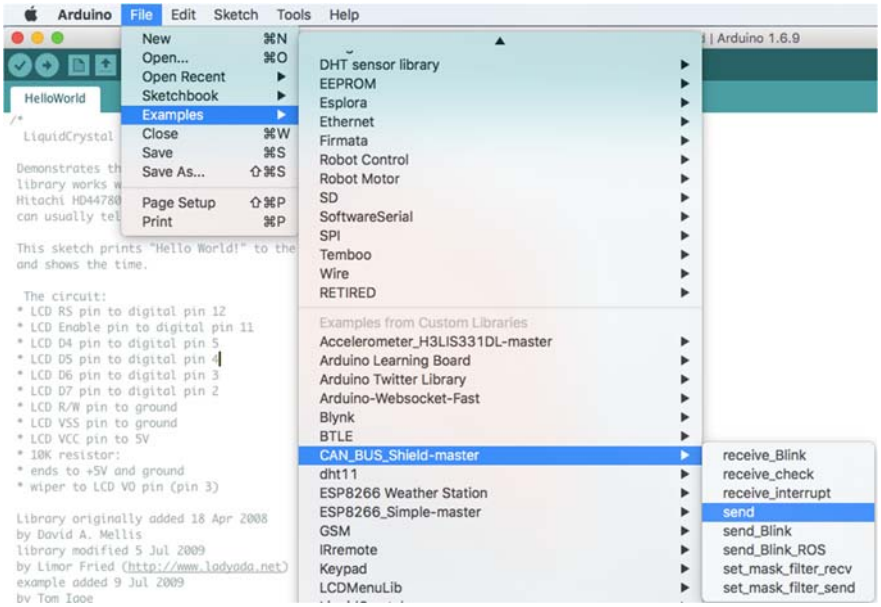
Install the library to your Arduino IDE when it is downloaded.

One of the node (a node means Seeeduino + CAN_BUS Shield) acts as master, the other acts as slaver. The master will send data to slaver constantly.

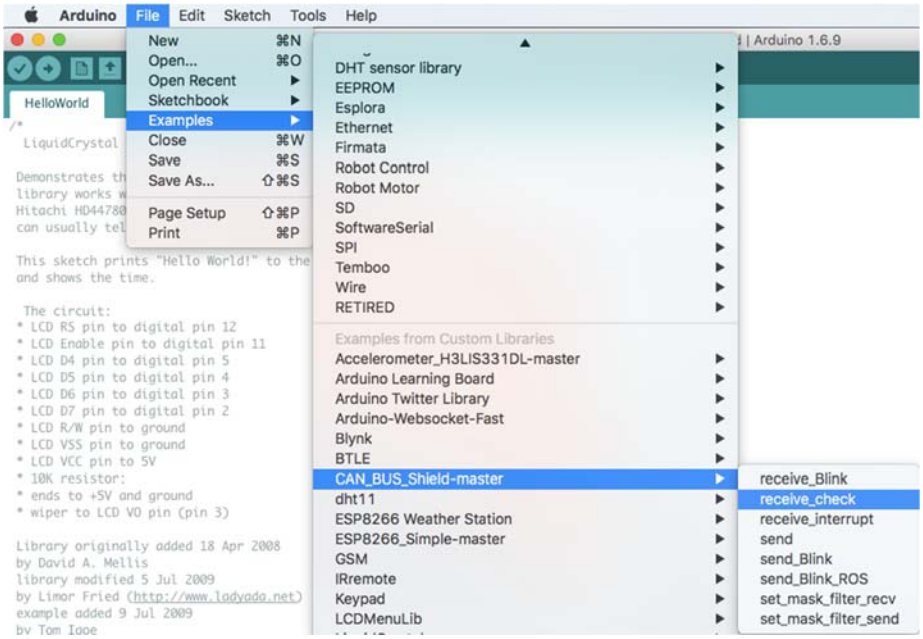
Note

Each node can act as master before the code being uploaded.

Open the **send** example (**File > Examples > CAN_BUS_Shield-master > send**) and upload to the **master**.

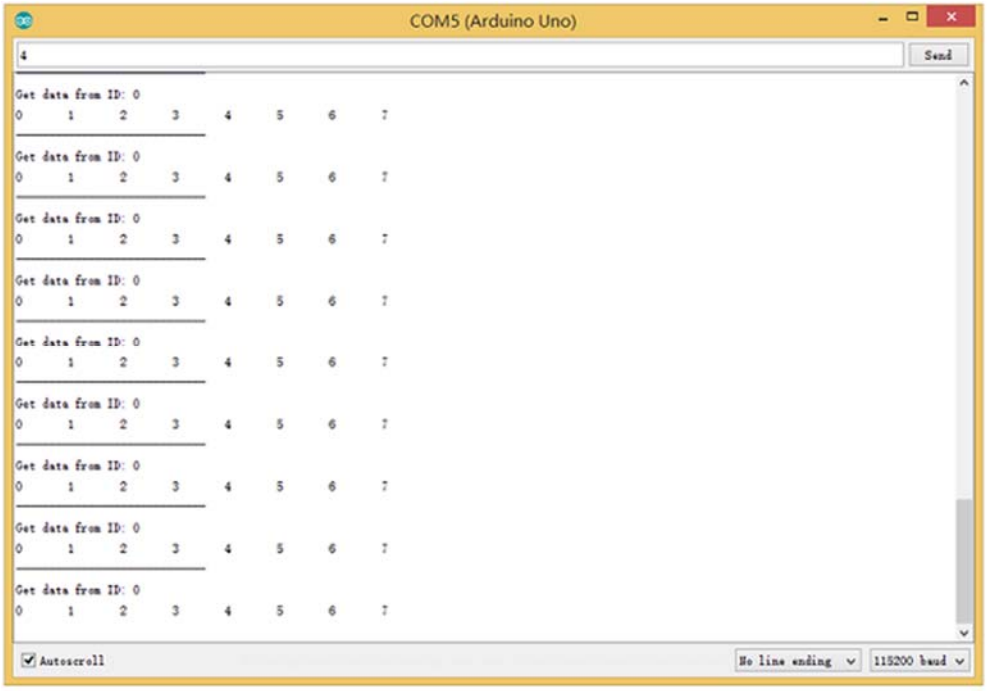


Open the **receive_check** example (**File > Examples > CAN_BUS_Shield-master > receive_check**) and upload to the **slaver**.



STEP4: View Result

Open the Serial Monitor of Arduino IDE(**slaver**), you will get the data sent from the master.



APIs

---

1. Set the Baud rate

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

The available baud rates are listed as follows:

```
#define CAN_5KBPS    1
#define CAN_10KBPS   2
#define CAN_20KBPS   3
#define CAN_25KBPS   4
#define CAN_31K25BPS 5
#define CAN_33KBPS   6
#define CAN_40KBPS   7
#define CAN_50KBPS   8
#define CAN_80KBPS   9
#define CAN_83K3BPS  10
#define CAN_95KBPS   11
#define CAN_100KBPS  12
```

```
#define CAN_125KBPS  13
#define CAN_200KBPS  14
#define CAN_250KBPS  15
#define CAN_500KBPS  16
#define CAN_666kbps  17
#define CAN_1000KBPS 18
```

2. Set Receive Mask and Filter

There are **2** receive mask registers and **5** filter registers on the controller chip that guarantee you getting 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:

Mask:

```
init_Mask(unsigned char num, unsigned char ext, unsigned char ulData);
```

Filter:

```
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);
```

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

- **id** represents where the data comes 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.

Generate a New BaudRate

We had provided many frequently-used baud rates, as below:

```
#define CAN_5KBPS    1
#define CAN_10KBPS   2
#define CAN_20KBPS   3
#define CAN_25KBPS   4
#define CAN_31KBPS   5
#define CAN_33KBPS   6
#define CAN_40KBPS   7
#define CAN_50KBPS   8
#define CAN_80KBPS   9
#define CAN_83KBPS   10
#define CAN_95KBPS   11
#define CAN_100KBPS  12
#define CAN_125KBPS  13
#define CAN_200KBPS  14
#define CAN_250KBPS  15
#define CAN_500KBPS  16
#define CAN_666KBPS  17
#define CAN_1000KBPS 18
```

Yet you may still can't find the rate you want. Here we provide a software to help you to calculate the baud rate you need.

Click here to download the software, it's in Chinese, but never mind, it's easy to use.https://github.com/SeeedDocument/CAN_BUS_Shield/raw/master/resource/CAN_Baudrate_CalcV1.3.zip



Note

This software supports Windows system only. If you can't open it, please feel free to contact loovee@seeed.cc for support.

Open the software, what you need to do is to set the baud rate you want, and then do some simple setting, then click **calculate**.

Then you will get some data, cfg1, cfg2 and cfg3.

You need to add some code to the library.

Open **mcp_can_dfs.h**, you need to add below code at about line 272:

```
#define MCP_16MHz_xxxkBPS_CFG1 (cfg1)    // xxx is the baud rate you need
#define MCP_16MHz_xxxkBPS_CFG2 (cfg2)
#define MCP_16MHz_xxxkBPS_CFG3 (cfg2)
```

Then let's go to about line 390, add below code:

```
#define CAN_xxxKBPS NUM       // xxx is the baudrate you need, and NUM is a number, you need to get a different from the other rates.
```

Open **mcp_can.cpp**, goto the function **mcp2515_configRate**(at about line 190), then add below code:

```
case (CAN_xxxKBPS):
    cfg1 = MCP_16MHz_xxxkBPS_CFG1;
    cfg2 = MCP_16MHz_xxxkBPS_CFG2;
    cfg3 = MCP_16MHz_xxxkBPS_CFG3;
    break;
```

Then you can use the baud rate you need. And please give me a pull request at github when you use a new rate, so I can add it to the library to help other guys.

Projects

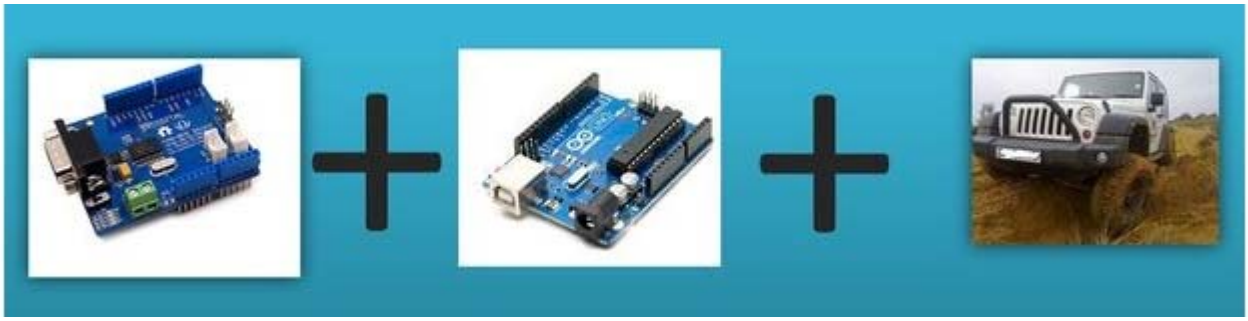
If you want to make some awesome projects with CAN-BUS shield, here are some projects for reference.

Volkswagen CAN BUS Gaming



Ever wanted to play a car/truck simulator with a real dashboard on your PC? Me too! I'm trying to control a VW Polo 6R dashboard via CAN Bus with an Arduino Uno and a Seeed CAN Bus Shield. Inspired by Silas Parker. Thanks Sepp and Is0-Mick for their great support!

Hack your vehicle CAN-BUS



Modern Vehicles all come equipped with a CAN-BUS Controller Area Network, Instead of having a million wires running back and forth from various devices in your car to the battery, its making use of a more clever system.

All electronic functions are connected to the TIPM, (Totally integrated Power Module), such as solenoids/relays to lock the doors or mini motors to wind the windows etc.

From each node (IE Switch pod that controls your windows or electric door locks) it broadcasts a message across the CAN. When the TIPM detects a valid message it will react accordingly like, lock the doors, switch on lights and so on.

FAQ

Q1: I can't get data from other CAN device.

- Check if the connection is right
- Check if the baud rate setting is right

Q2: The serial monitor print Init Fail.

- Check if the CS pin setting is matched with the code. For CAN Bus Shield V1.1/1.2, CS pin is connected to D9, others are to D10.

Q3. Where can I find technical support if I have some other issues.

- You can post a question to Seeed Forum or send an email to **techsupport@seeed.cc**.

Help us make it better

Welcome to the new documentation system of Seeed Studio. We have made a lot of progress comparing to the old wiki system and will continue to improve it to make it more user friendly and helpful. The improvement can't be done without your kindly feedback. If you have any suggestions or findings, you are most welcome to submit the amended version as our contributor via Github or give us suggestions in the survey below, it would be more appreciated if you could leave your email so that we can reply to you. Happy Hacking!