USB2CAN MODULE User Manual

1. General Description:

USB2CAN module is a ‘plug and play’ and bi-directional port powered USB to CAN converter which realizes long-distance communication between your Raspberry Pi/SBC/PC and other devices stably though CAN- Bus connection.

With small size and convenient operation, It’s a cost-effective solution that are safe and reliable for all your data-conversion / device-protection applications for any experienced engineer interfacing to expensive industrial equipment yet simple enough for home use by an amateur hobbyist.

USB2CAN can also be applied to obtain the data of car via the OBD connector, but you need to configured and secondary development by yourself.

2. Features

1. Compatible with Raspberry Pi Zero(W)/Pi3B+/PI4/Beaglebone/Tinker Board and any single board computer .

2. Plug and Play USB device. No external power required. Support wider CAN baud rate, From 20Kbps to 1Mbps can be programmed arbitrarily.

3. On board STM32F0 microcontroller, high speed data transfer with DMA technique. Support for CAN bus 2.0A and 2.0B specification.

4. Power supply isolation, signal input/output isolation, Built-in surge and static protection. 120 Ohm resistor selectable jumper feature.

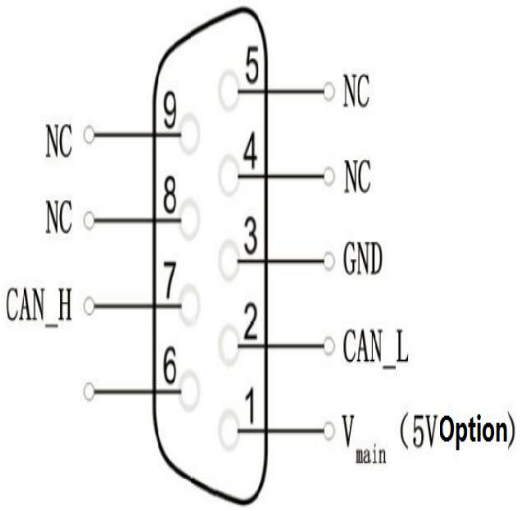
5. Comes with C/Python demos of Socket-CAN, detailed user manual and friendly technology support.

3. Technical Specification

Connector	
CAN	D-SUB, 9 pins
USB	USB 2.0 Full-Speed, Micro USB
CAN Features	
Specification	2.0A (standard format) and 2.0B (extended format), ISO 11898-2 High-speed CAN
Data Rate	From 20kbps to 1Mbps can be programmed arbitrarily.
Isolation Voltage	1.5K VDC/min, 3K VDC/1s
Microcontroller	STM32F0, 48MHz
Termination	120 Ohm resistor selectable jumper
CAN Transceiver	ISO1050DUBR ,Texas Instruments
Other	
Work Temperature	-40° ~ 85°
Relative humidity	15-90%, not condensing
PCBA Size (L * W * H)	56.50mm * 31.20mm * 14.20mm
Weight	15.5 g

4. Hardware Description

4.1 CAN connector Pinout



Pin	Description
1	5V/150ma output . Weld 0Ω resistor on R9 to enable this function(close to the jumper).
2	CANL bus line (dominant low)
3	CAN_GND
4	NC
5	NC
6	NC
7	CANH bus line (dominant high)
8	NC
9	NC

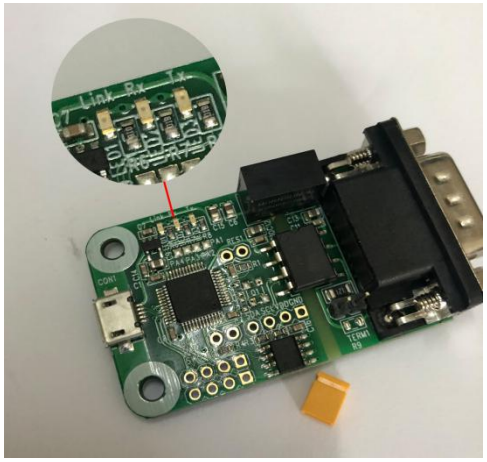
4.2 120 Ohm Resistor Setting.

A High-speed CAN bus (ISO 11898-2) must be terminated on both ends with 120 Ohms. The USB2CAN module with a on-board 120Ω selectable jumer.

Disable 120 Ohm Resistor.

Enable120 Ohm Resistor.

4.3 LED Indicate



LED Name	Description
Link	Red led is normally on to indicate the module is started successfully
Tx	Red led flash to indicate send data.
Rx	Red led flash to indicate receive data.

5. Run USB2CAN Test Demo

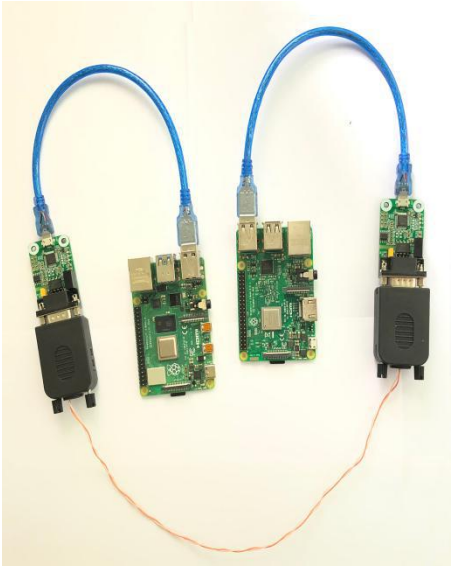
USB2CAN module can run properly without any additional driver request on all Linux system since version 3.9. such as Ubuntu, Debian and Raspbian. If you meet problems in older system, You need to reconfigure the kernel drivers. Enable ‘gs_usb.c’ and install ‘gs_usb.ko’ into system. So notice that if you only compile this drivers, It may fail to load in system. At this time, compile fully with new configure.

You can test the USB2CAN module any single board computer or PC with the right Linux version. We take Raspberry Pi 4 as an example to show you how to run the C/Python and can-utils demo.

5.1 Preparatory work

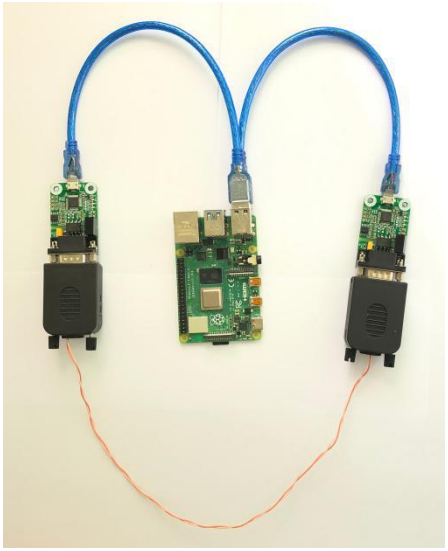
5.1.1 Connection

There are two way to test the USB2CAN module. One is plug two USB2CAN module into the USB Host of each Raspberry Pi. The other is plug two USB2CAN into one Raspberry Pi. But the codes and commands are a little bit different. And then connect the CAN_H pin and CAN_L pin to each other. No GND pin connection requirement.



Methods A

Use two Raspberry Pi to test USB2CAN

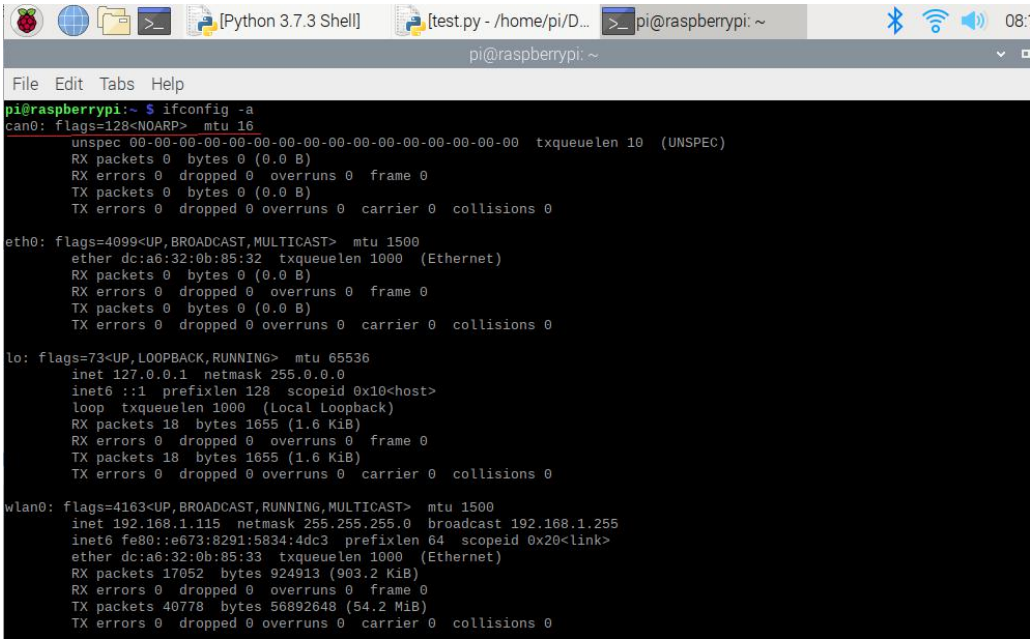


Methods B

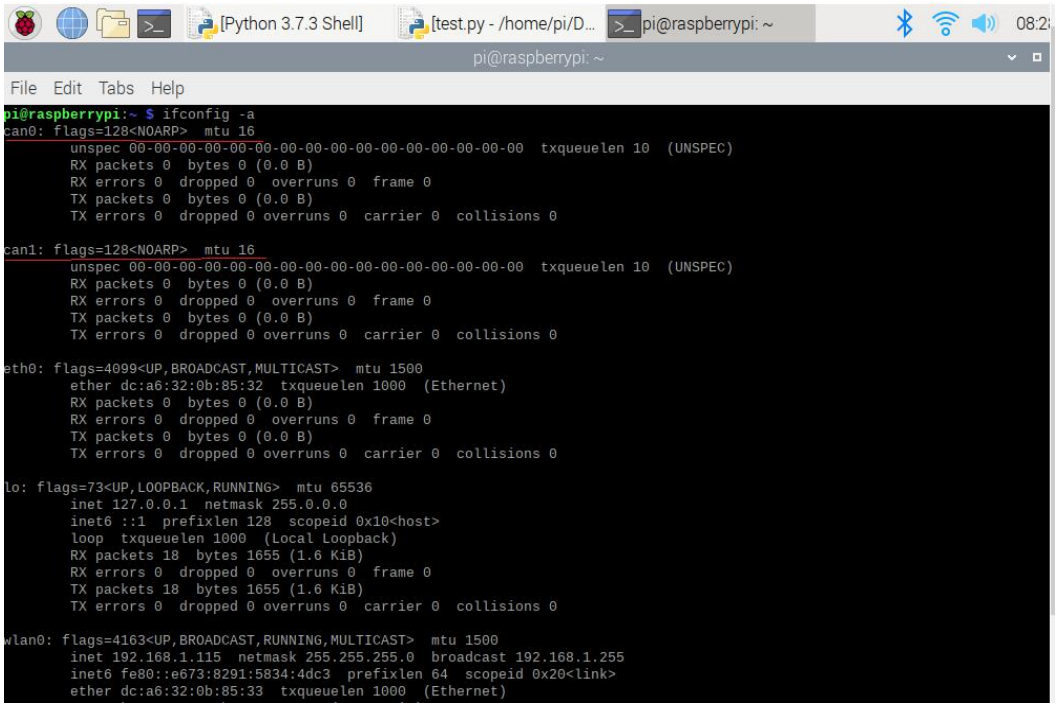
Use one Raspberry Pi to test USB2CAN

5.1.2 ifconfig -a

Type command ‘ifconfig -a’ to check ‘can0’ device is available in system.

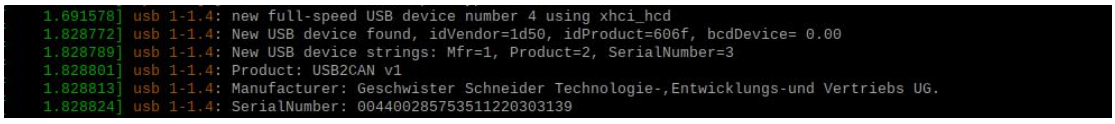


If you plug two USB2CAN into one Raspberry PI, You will see one more ‘can1’ device.



5.1.3 demsg

You can type command ‘demsg’ for see more information about USB2CAN module at the bottom.



5.2 Use can-utils tool

This tool is a very easy way to test CAN commuincation by only type two command.

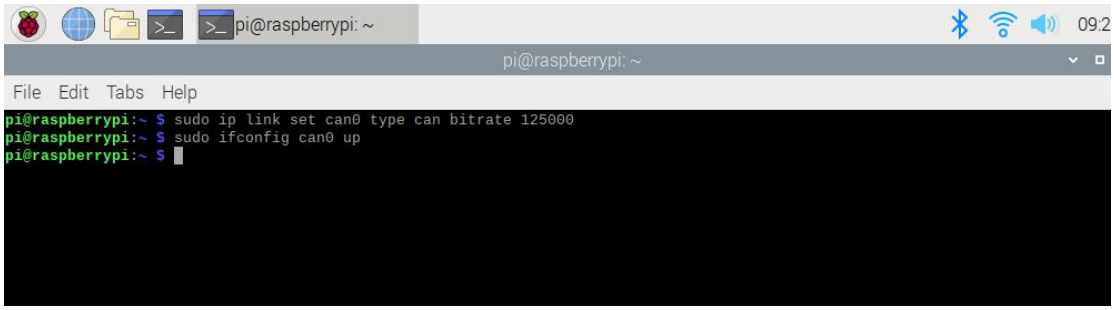
5.2.1 Install Tools

sudo apt-get install can-utils

5.2.2 Initialize CAN port

sudo ip link set can0 type can bitrate 125000

sudo ifconfig can0 up



5.2.3 Set one as receiver

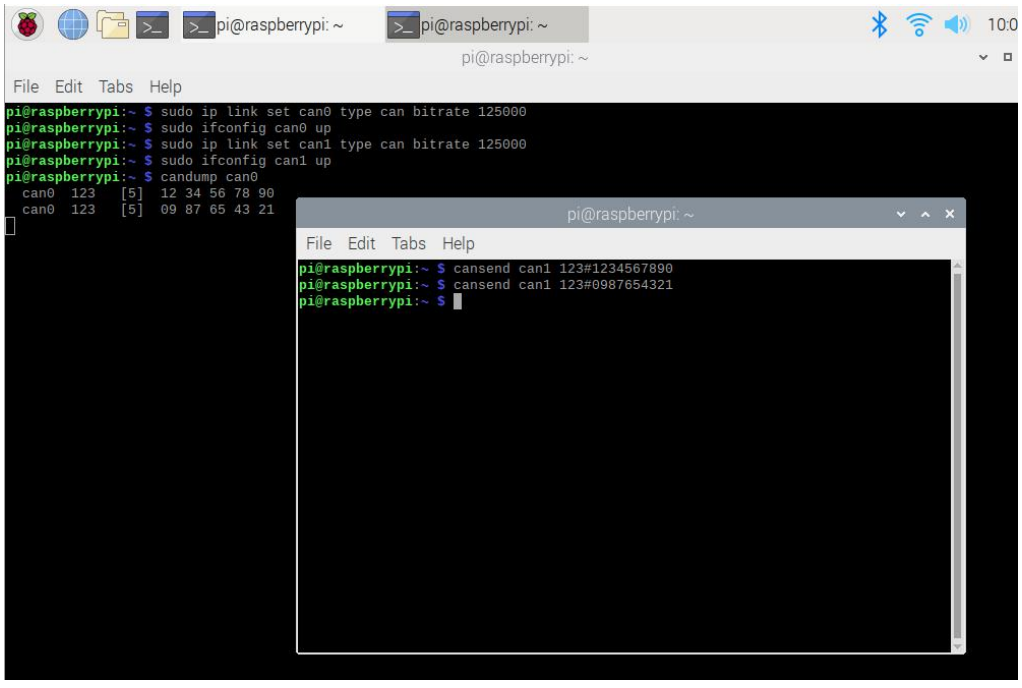
candump can0

5.2.4 Set the other as sender

cansend can0 123#1234567890

5.2.5 Test two module on one Raspberry Pi

Do as above steps, set ‘can0’ as receiver and ‘can1’ as sender.



5.3 Run C Demo

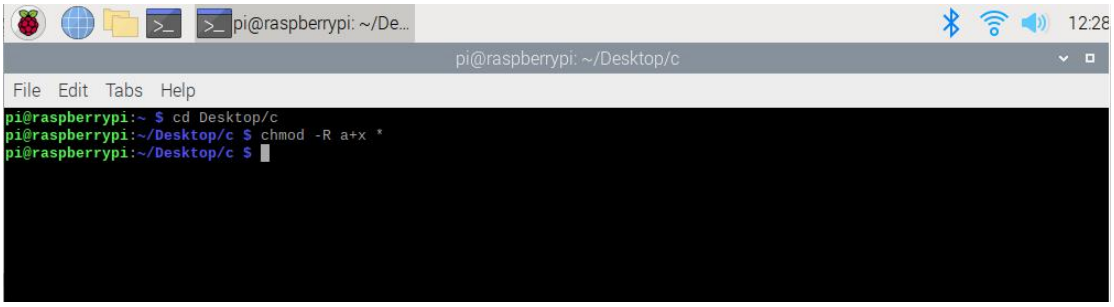
(1) Load C Demo named ‘usb2cantest’ from our Wiki and up-zip it to the desktop of Rasbian.

http://www.inno-maker.com/wiki/doku.php?id=usb_can

Or **http://www.inno-maker.com/wiki/doku.php**

(2) Go to folder named ‘c’ and change the permissions.

```
chmod -R a+x *
```

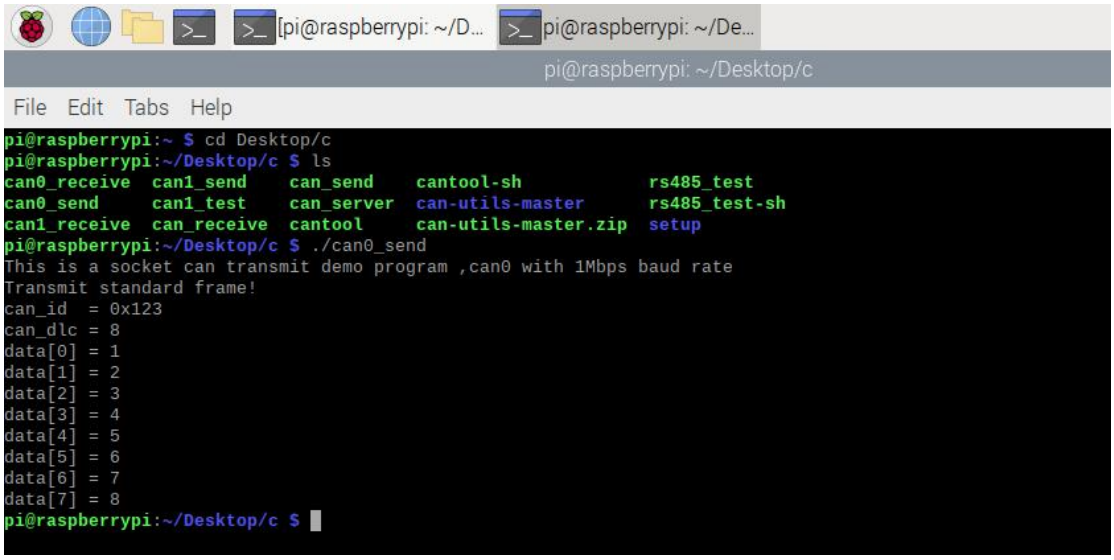


(3) Set one as receiver, execute following commands in serial terminal. Now this Raspberry pi is blocked.

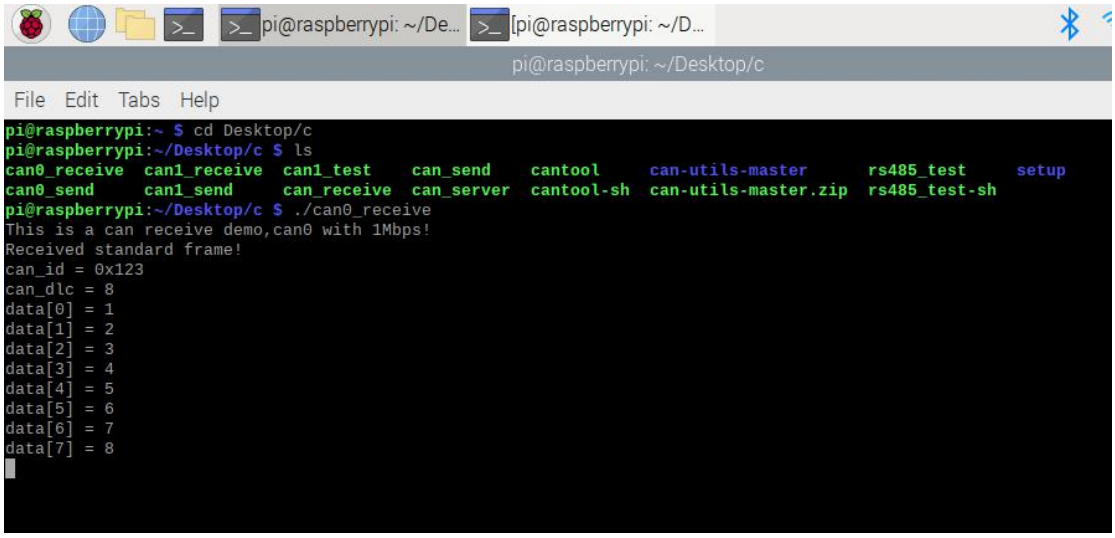
```
./can0_receive
```

(4) Set the other Pi as sender, execute following commands.

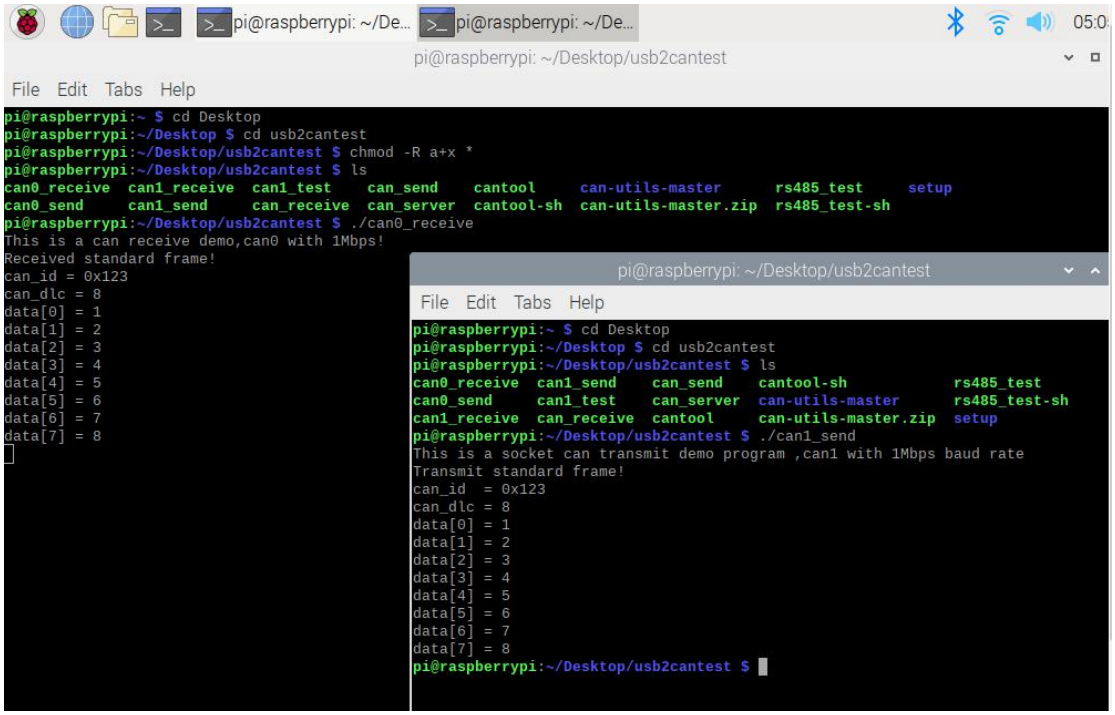
```
./can0_send
```



(5) You should see that the receiver has received the packet.



(6) You also can plug two USB2CAN module on one Raspberry PI board to test. You should see two can socket “can 0” and “can 1” devices. So notice that you need to change “can 0” to “can 1” when you use “can 1” device.


pi@raspberrypi: ~/De...
pi@raspberrypi: ~/De...
05:0
pi@raspberrypi: ~/Desktop/usb2cantest
File Edit Tabs Help
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ cd usb2cantest
pi@raspberrypi:~/Desktop/usb2cantest $ chmod -R a+x *
pi@raspberrypi:~/Desktop/usb2cantest $ ls
can0_receive can1_receive can1_test can_send cantool can-utils-master rs485_test setup
can0_send can1_send can_receive can_server cantool-sh can-utils-master.zip rs485_test-sh
pi@raspberrypi:~/Desktop/usb2cantest $ ./can0_receive
This is a can receive demo,can0 with 1Mbps!
Received standard frame!
can_id = 0x123
can_dlc = 8
data[0] = 1
data[1] = 2
data[2] = 3
data[3] = 4
data[4] = 5
data[5] = 6
data[6] = 7
data[7] = 8
pi@raspberrypi: ~/Desktop/usb2cantest
File Edit Tabs Help
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ cd usb2cantest
pi@raspberrypi:~/Desktop/usb2cantest $ ls
can0_receive can1_send can_send cantool-sh rs485_test
can0_send can1_test can_server can-utils-master rs485_test-sh
can1_receive can_receive cantool can-utils-master.zip setup
pi@raspberrypi:~/Desktop/usb2cantest $ ./can1_send
This is a socket can transmit demo program ,can1 with 1Mbps baud rate
Transmit standard frame!
can_id = 0x123
can_dlc = 8
data[0] = 1
data[1] = 2
data[2] = 3
data[3] = 4
data[4] = 5
data[5] = 6
data[6] = 7
data[7] = 8
pi@raspberrypi:~/Desktop/usb2cantest $

5.4 Run Python3 Demo

Download Python Demo named ‘python3’ from our Wiki and up-zip it to Desktop(or wherever you want put it).

http://www.inno-maker.com/wiki/doku.php?id=usb_can

Or **http://www.inno-maker.com/wiki/doku.php**

There are three files in the folder. ‘send.py’ and ‘receive.py’ is for you use two Raspberry Pi to test, and ‘test.py’ is for you use one Raspberry Pi and two USB2CAN module to test.

(1) Check the Python version of your Raspbian. Python 3.7.3 default in 2019-09-26-Raspbian.img. Our Demo can run on any Python3 version.

```
python3 -V
```

(2) If you can’t find the Python3 in system. Install the Python3

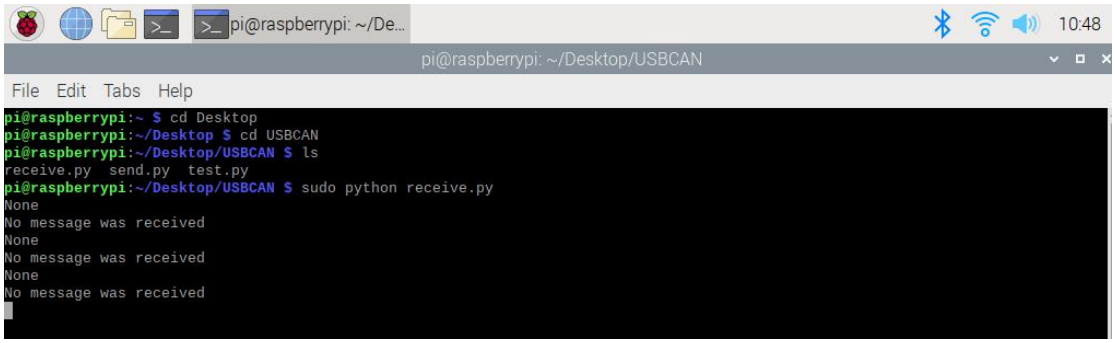
```
sudo apt-get install python3-pip
```

(3) Install Python CAN library.

```
sudo pip3 install python-can
```

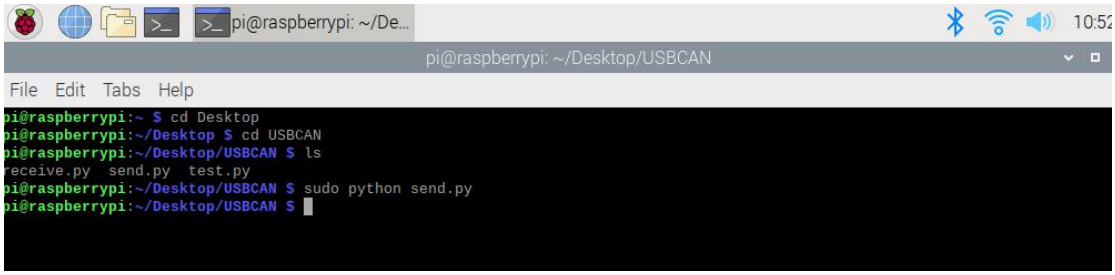
(4) Set one Raspberry Pi as receiver.

```
sudo python3 receive.py
```

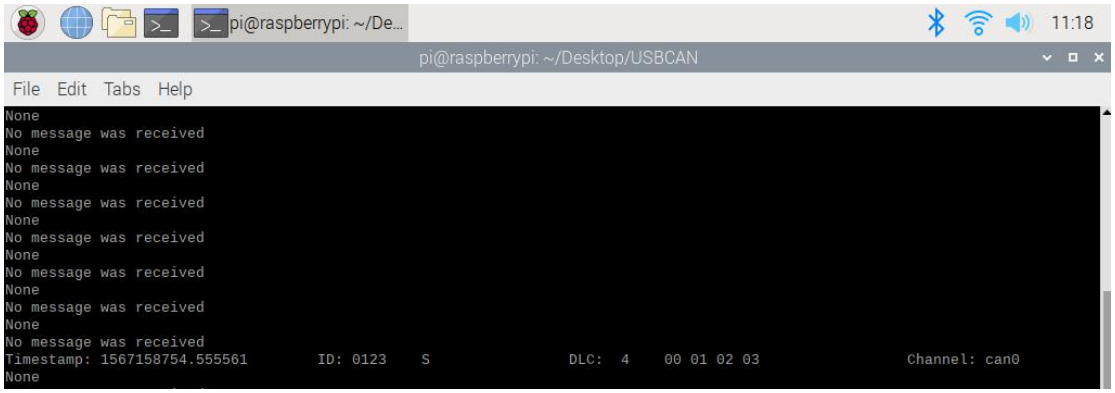


(5) Set the other as sender.

sudo python3 receive.py

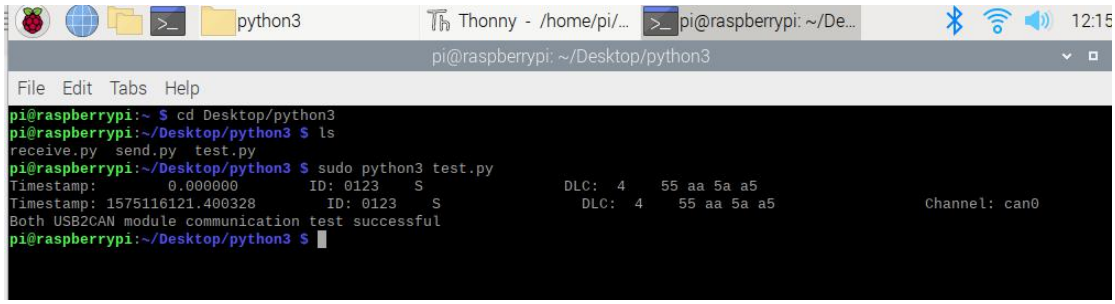


(6) You will see the data received.



(7) If you use one Raspberry Pi and two USB2CAN modeule for testing. Run 'test.py' and check the result.

sudo python3 test.py



6. Software Description

Now with previous demo's code to show you how to program socket can in Raspbian with C and Python . The socket can is an implementation of CAN protocols(Controller Area Network) for Linux. CAN is a networking technology which has widespread use in automation, embedded devices, and automotive fields. While there have been other CAN implementations for Linux based on character devices, Socket CAN uses the Berkeley socket API, the Linux network stack and implements the CAN device drivers as network interfaces. The CAN socket API has been designed as similar as possible to the TCP/IP protocols to allow programmers, familiar with network programming, to easily learn how to use CAN sockets.

For more Socket CAN detail please refer to below link:
https://www.kernel.org/doc/Documentation/networking/can.txt

6.1 Programming in C

6.1.1 For Sender's codes

(1): Create the socket, If an error occurs then the return result is -1.

```
/*Create socket*/
s = socket(PF_CAN, SOCK_RAW, CAN_RAW);
if (s < 0) {
    perror("Create socket PF_CAN failed");
    return 1;
}
```

(2): Locate the interface to "can0" or other name you wish to use. The name will show when you execute "./ifconfig –a".

```
/*Specify can0 device*/
strcpy(ifr.ifr_name, "can0");
ret = ioctl(s, SIOCGIFINDEX, &ifr);
if (ret < 0) {
    perror("ioctl interface index failed!");
    return 1;
}
```

(3): Bind the socket to “can0”.

```
/*Bind the socket to can0*/
addr.can_family = PF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
ret = bind(s, (struct sockaddr *)&addr, sizeof(addr));
if (ret < 0) {
    perror("bind failed");
    return 1;
}
```

(4): Disable sender’s filtering rules,this program only send message do not receive packets.

```
/*Disable filtering rules,this program only send message do not receive packets */
setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, NULL, 0);
```

(5): Assembly data to send.

```
/*assembly  message data! */
frame.can_id = 0x123;
frame.can_dlc = 8;
frame.data[0] = 1;
frame.data[1] = 2;
frame.data[2] = 3;
frame.data[3] = 4;
frame.data[4] = 5;
frame.data[5] = 6;
frame.data[6] = 7;
frame.data[7] = 8;
//if(frame.can_id&CAN_EFF_FLAG==0)
if(!(frame.can_id&CAN_EFF_FLAG))
    printf("Transmit standard frame!\n");
else
    printf("Transmit extended frame!\n");
```

(6): Send message to the can bus.You can use the return value of write() to check whether all data has been sent successfully .

```
/*Send message out */
nbytes = write(s, &frame, sizeof(frame));
if(nbytes != sizeof(frame)) {
    printf("Send  frame incompletely!\r\n");
    system("sudo ifconfig can0 down");
}
```

(7): Close can0 device and disable socket.

```
/*Close can0 device and destroy socket!*/
close(s);
```

6.2.3 For Receiver's codes

(1)step 1 and (2) is same as Sender's code.

(3):It's different from Sender's.

```
/*Bind the socket to can0*/
addr.can_family = PF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
ret = bind(s, (struct sockaddr *)&addr, sizeof(addr));
if (ret < 0) {
    perror("bind failed");
    return 1;
}
```

(4)： Define receive filter rules,we can set more than one filters rule。

```
/*Define receive filter rules,we can set more than one filter rule!*/
struct can_filter rfilter[2];
rfilter[0].can_id = 0x123;//Standard frame id !
rfilter[0].can_mask = CAN_SFF_MASK;
rfilter[1].can_id = 0x12345678;//extend frame id!
rfilter[1].can_mask = CAN_EFF_MASK;
```

(5): Read data back from can bus.

```
nbytes = read(s, &frame, sizeof(frame));
```

6.2 Programming in Python

6.2.1 Import

```
import os
```

The OS module in Python provides a way of using operating system dependent functionality. The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux. We usually use os.system() function to execute a shell command to set CAN.

```
import can
```

The python-can library provides Controller Area Network support for Python, providing common abstractions to different hardware devices, and a suite of utilities for sending and receiving messages on a CAN bus.

For more information about python-can, please to below link:

https://python-can.readthedocs.io/en/stable/index.html

6.2.1 Simple common functions

(1) Set bitrate and start up CAN device.

```
os.system('sudo ip link set can0 type can bitrate 1000000')
os.system('sudo ifconfig can0 up')
```

(2) Bind the socket to ‘can0’.

```
can0 = can.interface.Bus(channel = 'can0', bustype = 'socketcan_ctypes')
```

(3) Assembly data to send.

```
msg = can.Message(arbitration_id=0x123, data=[0, 1, 2, 3], extended_id=False)
```

(4) Send data.

```
can0.send(msg)
```

(5) Receive data.

```
msg = can0.recv(30.0)
```

(6) Close CAN device

```
os.system('sudo ifconfig can0 down')
```

7. User Manual Version Descriptions

Version	Description	Date	E-mail
V1.0.0.1		2019.02.02	support@inno-maker.com sales@inno-maker.com
V1.0.0.2	Add Python Add can-utils	2019.08.26	**calvin@inno-maker.com**
V1.0.0.3	Correct description	2019.08.26	**calvin@inno-maker.com**

If you have any suggestions, ideas, codes and tools please feel free to email to me. I will update the user manual and record your name and E-mail in list. Look forward to your letter and kindly share.