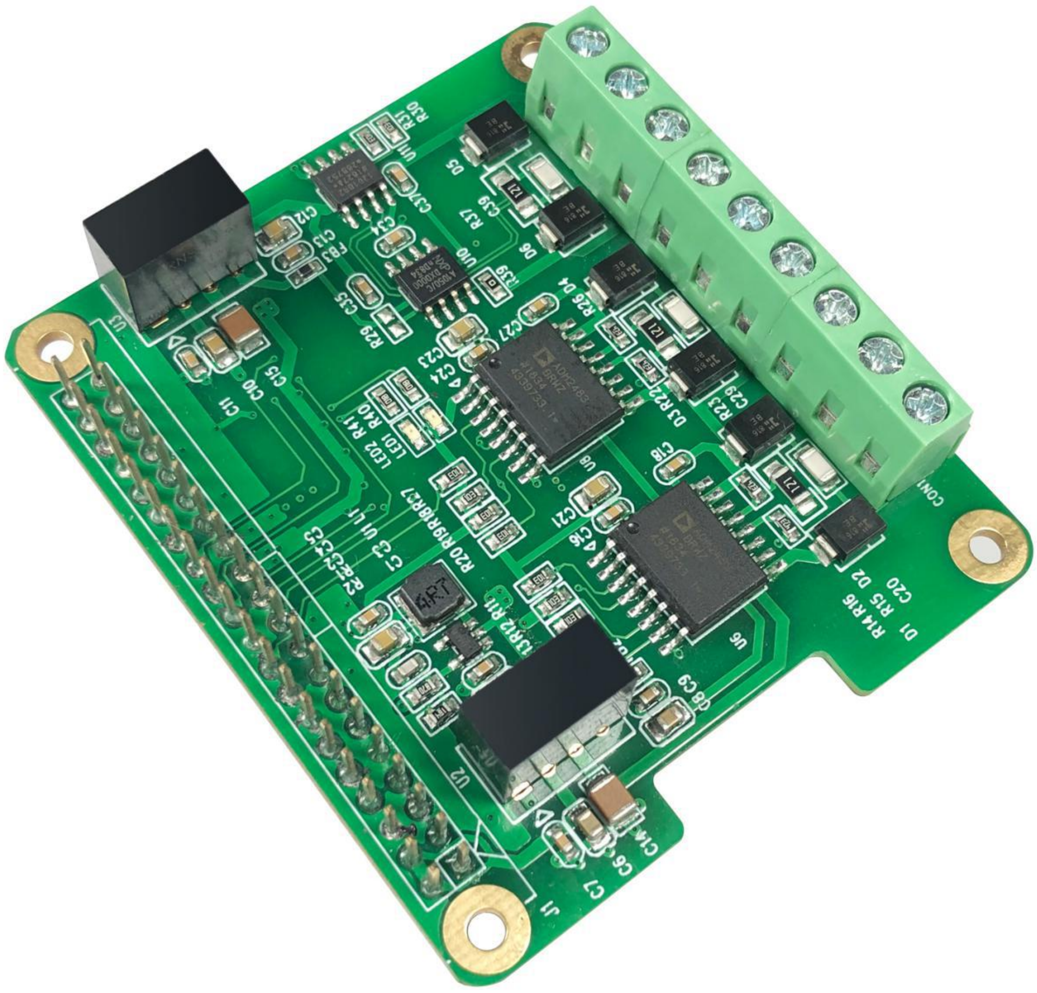
Raspberry Pi RS485&CAN Module User Manual



1.General Description:

RS485&CAN Module is an industrial communication module for Raspberry Pi, on board 2*RS485 Bus and 1*CAN Bus communication interface via SPI interface.

CAN bus and RS485 bus powered through separated isolation power module, signal between the transceiver and the controller is isolated , ESD protection for the communication port, ensure your Raspberry Pi can be used in more strictly industrial sites

2.Features:

2.1 Compatible with Raspberry Pi Zero/Zero W/2B/3B/3B+/4. Only a small amount of GPIO are used and the remaining pins allow to work for other extended function

2.2 Power supply and signal transmission isolation, Build-in surge and ESD protection.

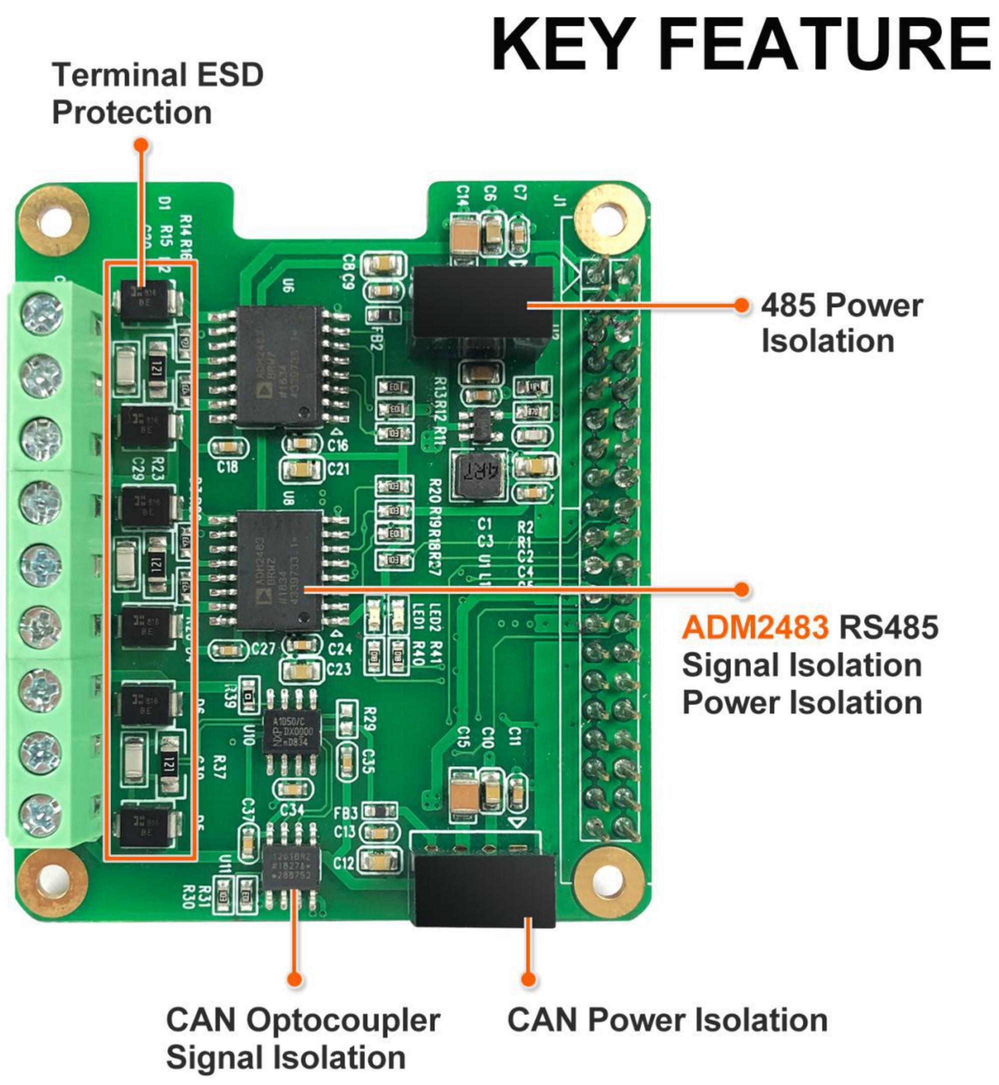
2.3 CAN function: on-board CAN controller MCP2515 via SPI interface, high speed CAN transceiver, digital isolation ADUM1201BRZ, and communication Rates 20Kbps-1Mpbs can be programmed arbitrarily.

2.4 RS485 function: on-board controlled via UART, half-duplex communication, supports automatically TX/RX switch without extra programming, on-board SPI to RS485 SC16IS1752 Electrical data isolation with ADI ADM2483.

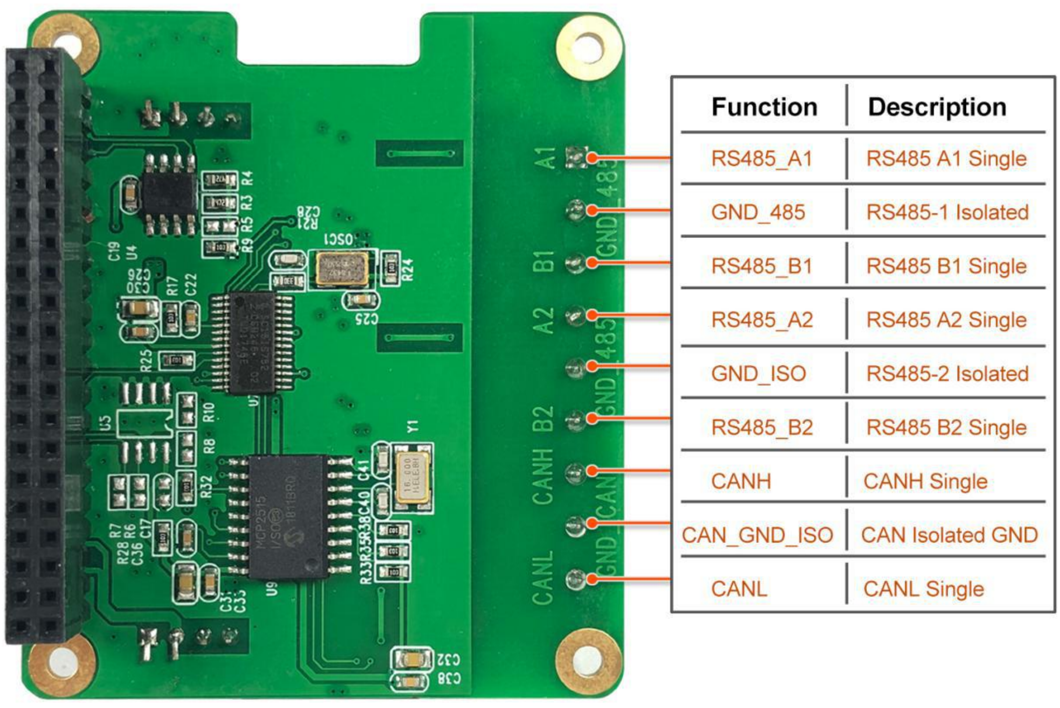
2.5 On-board individual 120 Ohm terminal resistance, Impedance matching and guarantee the ability to drive. On-board User ATC24C32 EEPROM,

3.Hardware Description

3.1 Overview


KEY FEATURE
Terminal ESD Protection
485 Power Isolation
ADM2483 RS485 Signal Isolation Power Isolation
CAN Optocoupler Signal Isolation
CAN Power Isolation

3.2 Output Innterface Desceiption

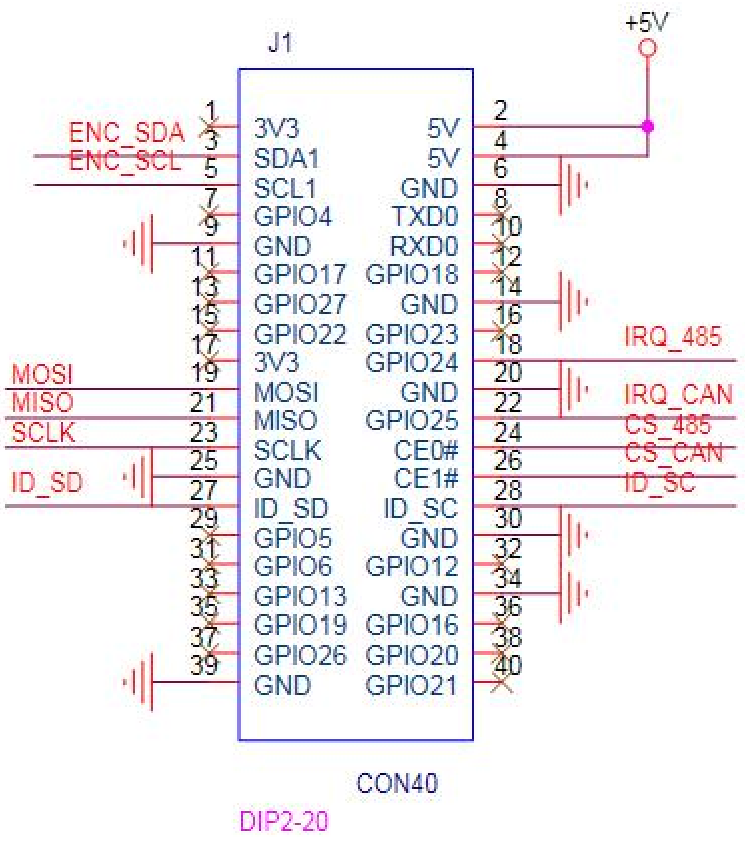


Function	Description
RS485_A1	RS485 A1 Single
GND_485	RS485-1 Isolated
RS485_B1	RS485 B1 Single
RS485_A2	RS485 A2 Single
GND_ISO	RS485-2 Isolated
RS485_B2	RS485 B2 Single
CANH	CANH Single
CAN_GND_ISO	CAN Isolated GND
CANL	CANL Single

Onboard SPI to CAN Controller MCP2515
Onboard SPI to RS485 SC16IS1752

Item	Function	Description
1	RS485_A1	RS485-1 A Signal
2	GND_485	RS485-1 GND
3	RS485_B1	RS485-1 B Signal
4	RS485_A2	RS485-2 A Signal
5	GND_ISO	RS485-2 GND
6	RS485_B2	RS485-2 B Signal
7	CANH	CAN-H Signal
8	CAN_GND_ISO	CAN Isolated GND
9	CANL	CAN-L Signal

3.3 Pins map of GPIO header on the Raspberry Pi

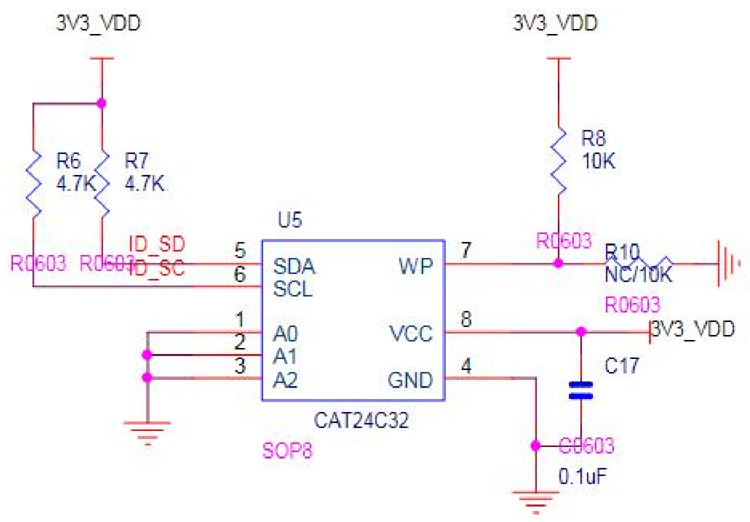


PIN	Symbol	Description
2, 4	+5V	+5V Supply Pin, connected to the main 5V supply of the Raspberry Pi
3, 5	IIC_SDA and IIC_SCL	IIC Used for EEPROM(U4 AT24C32)
19	GPIO_10	SPI0_MOSI
21	GPIO_9	SPI0_MISO
23	GPIO_11	SPI0 _SCLK
24	GPIO_8	RS485 Transfer Chip Selection
26	GPIO_7	CAN Transfer Chip

		Selection
18	GPIO_24	RS485 Interrupt
22	GPIO_25	CAN Interrupt
27, 28	ID SCL and ID SDA	Reserved for an ID EEPROM on the Raspberry Pi. These pins are always reserved and should never be used to connect external components
6, 9, 14, 20, 25, 30, 34, 39	GND	Ground Pin, connected to the main system Ground of the Raspberry Pi
The remaining pins are unused, You can use them for your other hardware boards.		

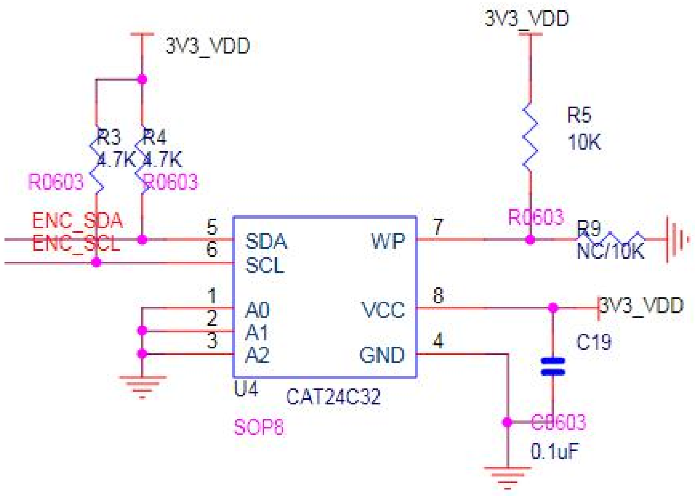
3.4 Extended Function

(1) ID EEPROM: (U5, No soldering on-board)

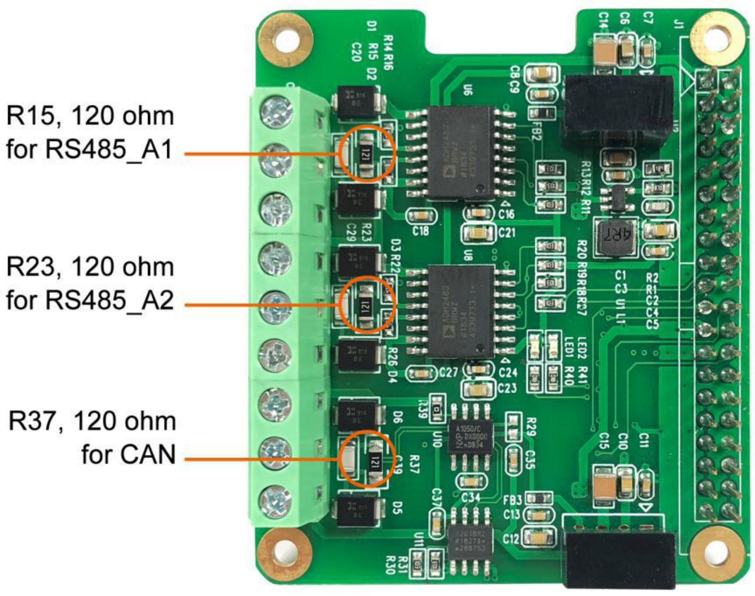


Pin 27 and 28 are always reserved for an ID EEPROM on the Raspberry Pi. Independently which card you use. It’s useless for most application. If you want to use this function, you need to solder the IC, resistance and capacitance by yourself.

(2) USER EEPROM: (U4)



(3) On-board 120 Ohm Termination Resistor



For more information about GPIO of Raspberry PI, please refer to below link:

https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/#prettyPhoto

4. Run 485&CAN Test Demo

There are two ways to run 485&CAN test demo. One is download our prepared image from our wiki page:

wiki.inno-maker.com/display/RS485CANMODULE

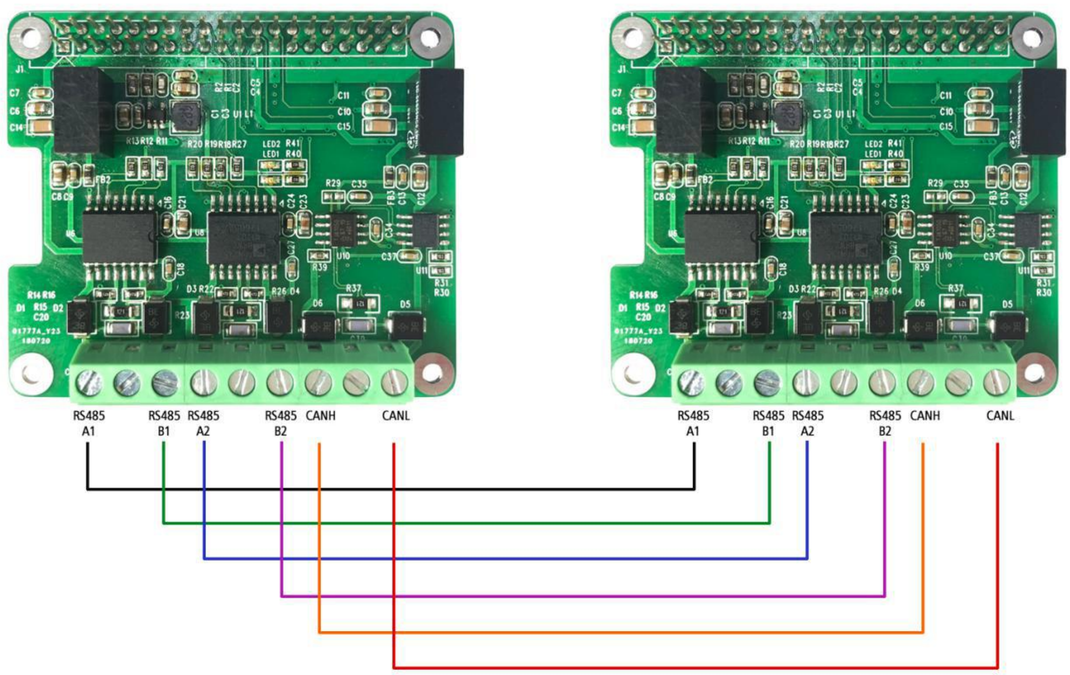
or https://www.jianguoyun.com/p/DTxr8-EQpdSrBxi0irsB

The other way is update the drive file to your image.Please refer to the 4.6 Chapter.

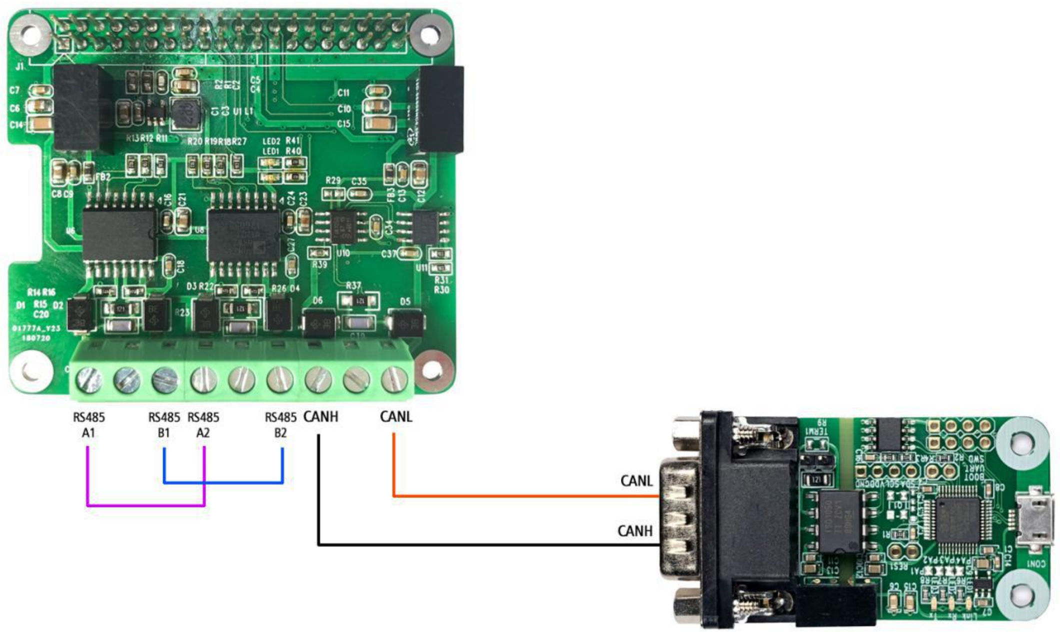
4.1 Preparatory Work.

(1) Load the image file onto a SD card, using the instructions provided on the Raspberry Pi website for Linux, Mac or PC:
https://www.raspberrypi.org/documentation/installation/installing-images/README.md

(2) Prepare two RS485&CAN modules, two TF cards and two Raspberry Pi boards. Connect the two connectors in proper order. Connect the UART debug port of Raspberry Pi to your computer (or using remote login).

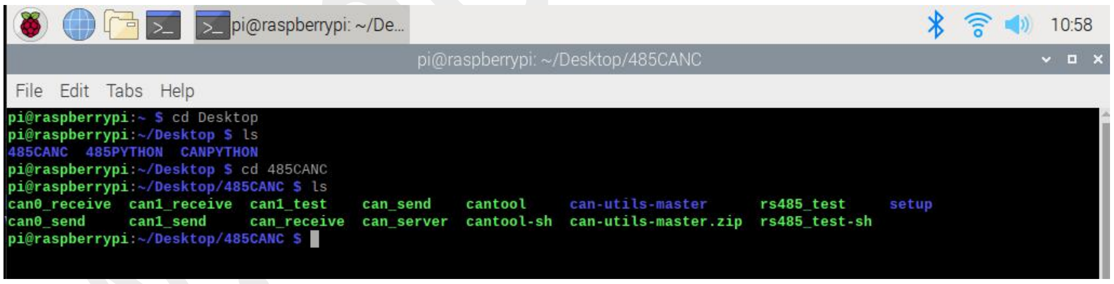


(3) You also can connect two RS485 port directly of one module and connect the CAN port to our USB2CAN module(http://www.inno-maker.com/product/usb-can/) for testing.



4.2 Run C test Demo for RS485

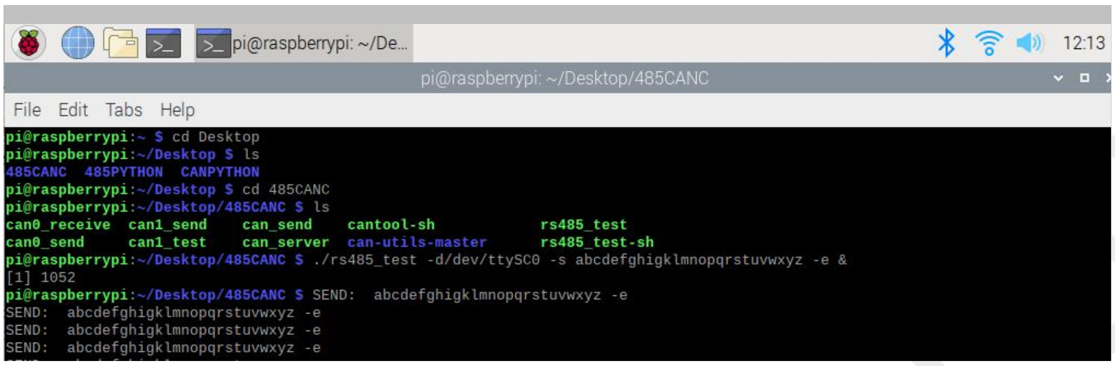
(1) Visit the folder named '485CANC' on the desktop.



```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ls
485CANC  485PYTHON  CANPYTHON
pi@raspberrypi:~/Desktop $ cd 485CANC
pi@raspberrypi:~/Desktop/485CANC $ 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/485CANC $ 
```

(2) Execute following commands on both Raspberry Pi, And you will see the data interaction.

```
./rs485_test -d/dev/ttySC0 -s abcdefghigklmnopqrstuvwxyz -e &
```

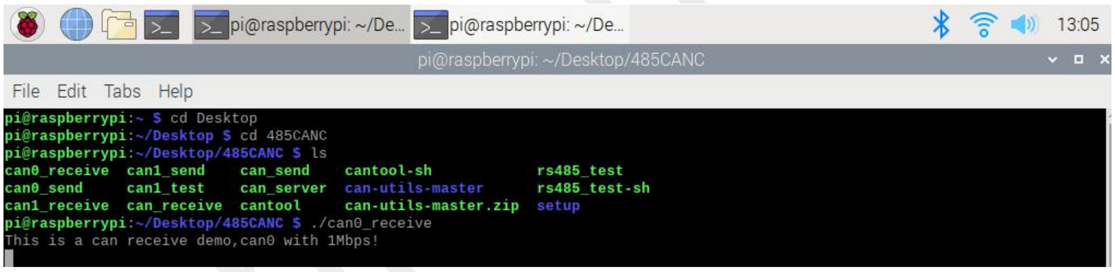


```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ls
485CANC  485PYTHON  CANPYTHON
pi@raspberrypi:~/Desktop $ cd 485CANC
pi@raspberrypi:~/Desktop/485CANC $ ls
can0_receive  can1_send    can_send    cantool-sh           rs485_test
can0_send     can1_test    can_server  can-utils-master     rs485_test-sh
pi@raspberrypi:~/Desktop/485CANC $ ./rs485_test -d/dev/ttySC0 -s abcdefghigklmnopqrstuvwxyz -e &
[1] 1052
pi@raspberrypi:~/Desktop/485CANC $ SEND:  abcdefghigklmnopqrstuvwxyz -e
SEND:  abcdefghigklmnopqrstuvwxyz -e
SEND:  abcdefghigklmnopqrstuvwxyz -e
SEND:  abcdefghigklmnopqrstuvwxyz -e
```

4.3 Run C test Demo for CAN

(1) Visit the folder named '485CANC' on the desktop.

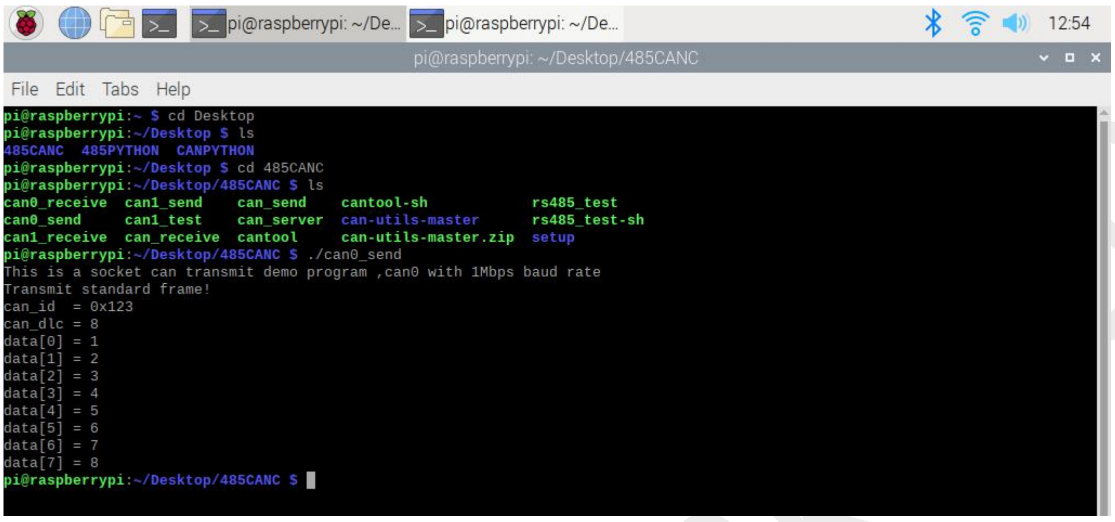
(2)Set one Raspberry Pi as receiver. Now this Raspberry pi is blocked.



```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ cd 485CANC
pi@raspberrypi:~/Desktop/485CANC $ 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/485CANC $ ./can0_receive
This is a can receive demo,can0 with 1Mbps!
```

(3) Set the other one as senderand You should see that the receiver has received the packet.

./can0_send

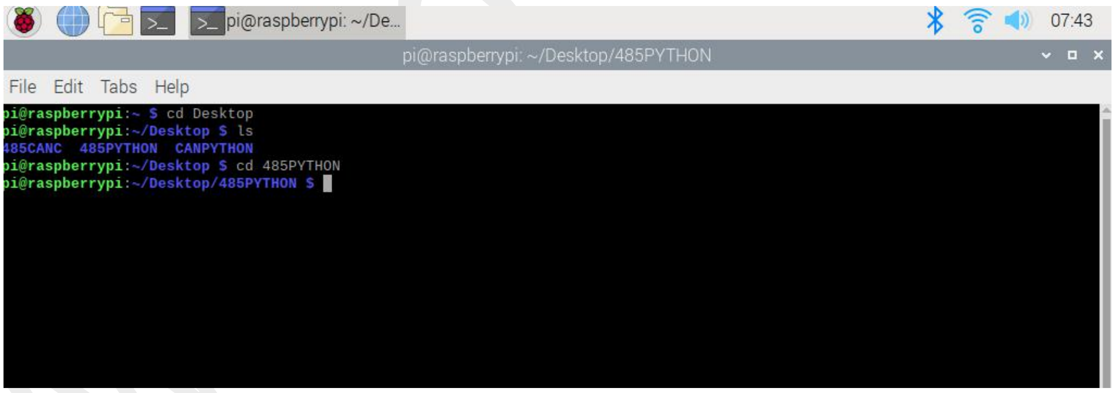


```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ls
485CANC  485PYTHON  CANPYTHON
pi@raspberrypi:~/Desktop $ cd 485CANC
pi@raspberrypi:~/Desktop/485CANC $ ls
can0_receive  can1_send    can_send    cantool-sh            rs485_test
can0_send     can1_test    can_server  can-utils-master      rs485_test-sh
can1_receive  can_receive  cantool     can-utils-master.zip  setup
pi@raspberrypi:~/Desktop/485CANC $ ./can0_send
This is a socket can transmit demo program ,can0 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/485CANC $
```

(4) Check whether receiver received the packet.

4.4 Run Python test Demo for RS485

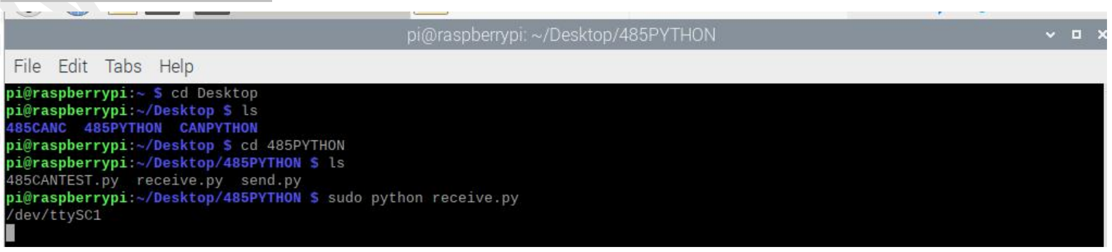
(1) Visit the folder named '485PYTHON' on the desktop.



```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ls
485CANC  485PYTHON  CANPYTHON
pi@raspberrypi:~/Desktop $ cd 485PYTHON
pi@raspberrypi:~/Desktop/485PYTHON $
```

(2) Set one Raspberry Pi as receiver

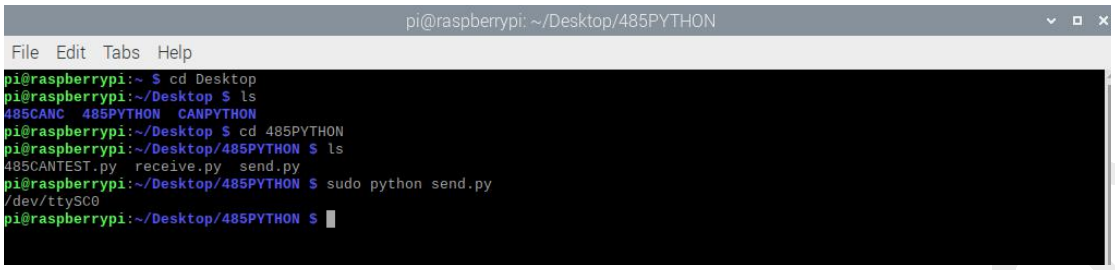
sudo python recieive.py



```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ls
485CANC  485PYTHON  CANPYTHON
pi@raspberrypi:~/Desktop $ cd 485PYTHON
pi@raspberrypi:~/Desktop/485PYTHON $ ls
485CANTEST.py  receive.py  send.py
pi@raspberrypi:~/Desktop/485PYTHON $ sudo python receive.py
/dev/ttySC1
```

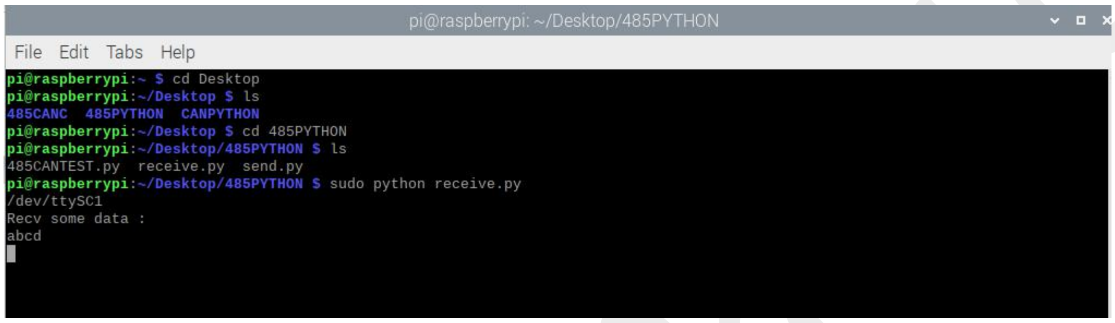
(3) Set the other Raspberry Pi as sender.

sudo python send.py



```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ls
485CANC  485PYTHON  CANPYTHON
pi@raspberrypi:~/Desktop $ cd 485PYTHON
pi@raspberrypi:~/Desktop/485PYTHON $ ls
485CANTEST.py  receive.py  send.py
pi@raspberrypi:~/Desktop/485PYTHON $ sudo python send.py
/dev/ttySC0
pi@raspberrypi:~/Desktop/485PYTHON $
```

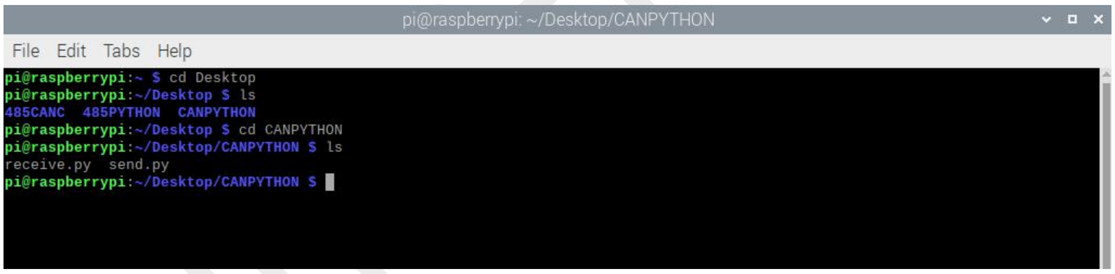
(4) View received result.



```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ls
485CANC  485PYTHON  CANPYTHON
pi@raspberrypi:~/Desktop $ cd 485PYTHON
pi@raspberrypi:~/Desktop/485PYTHON $ ls
485CANTEST.py  receive.py  send.py
pi@raspberrypi:~/Desktop/485PYTHON $ sudo python receive.py
/dev/ttySC1
Recv some data :
abcd
```

4.5 Run Python test Demo for CAN

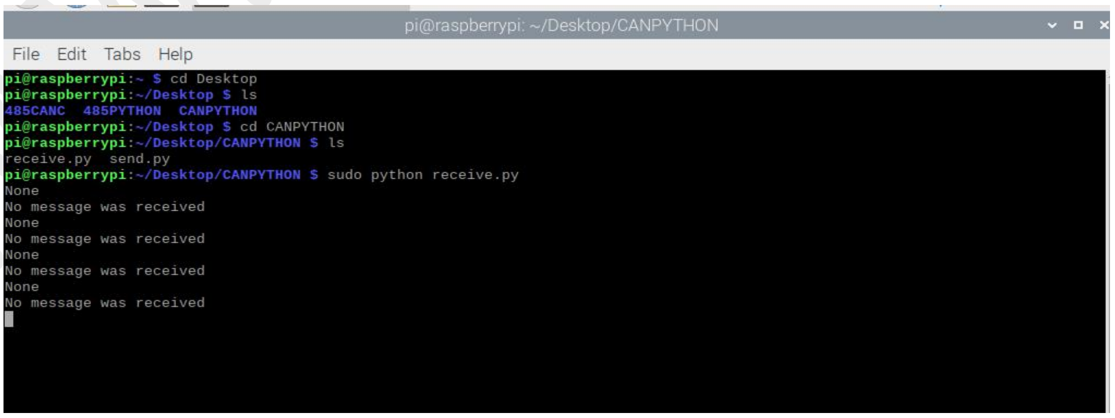
(1) Visit the folder named ‘CANPYTHONC’ on the desktop.



```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ls
485CANC  485PYTHON  CANPYTHON
pi@raspberrypi:~/Desktop $ cd CANPYTHON
pi@raspberrypi:~/Desktop/CANPYTHON $ ls
receive.py  send.py
pi@raspberrypi:~/Desktop/CANPYTHON $
```

(2) Set one Raspberry Pi as receiver.

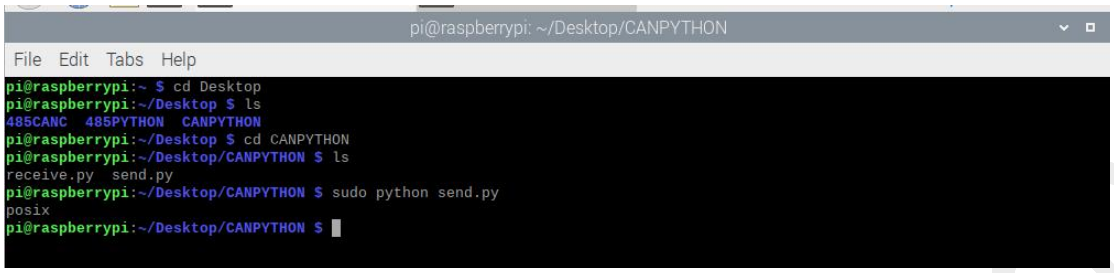
sudo python receive.py



```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ls
485CANC  485PYTHON  CANPYTHON
pi@raspberrypi:~/Desktop $ cd CANPYTHON
pi@raspberrypi:~/Desktop/CANPYTHON $ ls
receive.py  send.py
pi@raspberrypi:~/Desktop/CANPYTHON $ sudo python receive.py
None
No message was received
None
No message was received
None
No message was received
None
No message was received
```

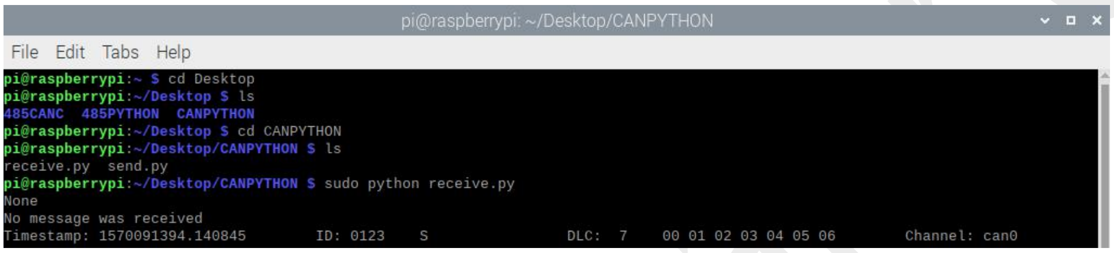
(3) Set the other Raspberry Pi as sender.

```
sudo python send.py
```



```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ls
485CANC  485PYTHON  CANPYTHON
pi@raspberrypi:~/Desktop $ cd CANPYTHON
pi@raspberrypi:~/Desktop/CANPYTHON $ ls
receive.py  send.py
pi@raspberrypi:~/Desktop/CANPYTHON $ sudo python send.py
posix
pi@raspberrypi:~/Desktop/CANPYTHON $ 
```

(5) View received result.



```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ls
485CANC  485PYTHON  CANPYTHON
pi@raspberrypi:~/Desktop $ cd CANPYTHON
pi@raspberrypi:~/Desktop/CANPYTHON $ ls
receive.py  send.py
pi@raspberrypi:~/Desktop/CANPYTHON $ sudo python receive.py
None
No message was received
Timestamp: 1570091394.140845        ID: 0123    S                DLC:  7    00 01 02 03 04 05 06        Channel: can0
```

(6) If you're use our USB2CAN module for testing, please change the CAN device 'can0' to 'can1' in either 'send.py' or 'receive.py' file.

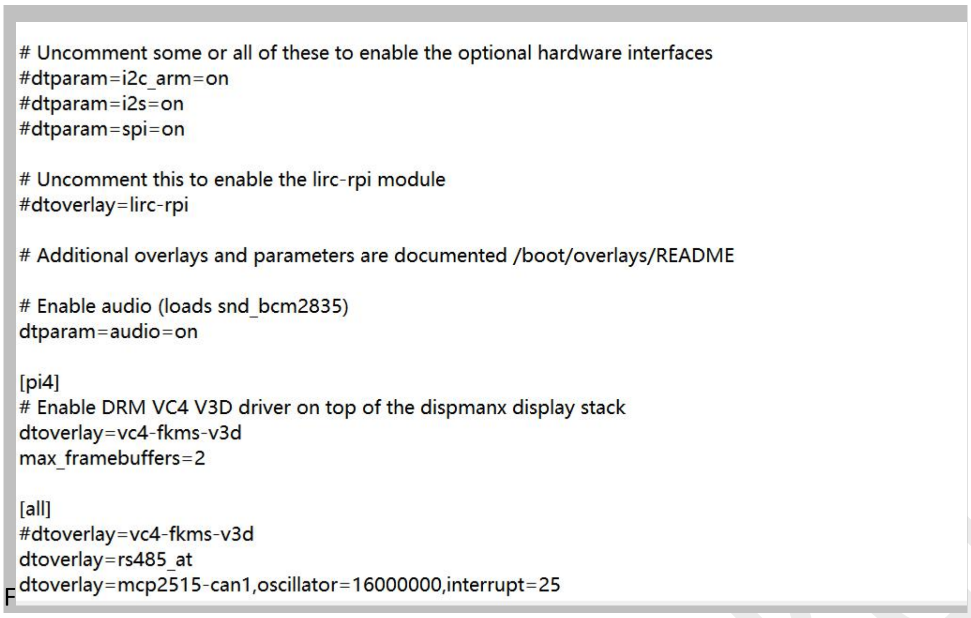
4.6 How to use 485&CAN module in a new Raspbian System

(1) Insert a TF cart into your computer, load your image of Raspbian to it

(2) Download the file named 'rs485_at.dtbo' from our wiki and copy it to /boot/overlays of the TF card.

(3) Open 'config.txt' and add below two lines in the end.

```
dtoverlay=rs485_at
dtoverlay=mcp2515-can1,oscillator=16000000,interrupt=25
```



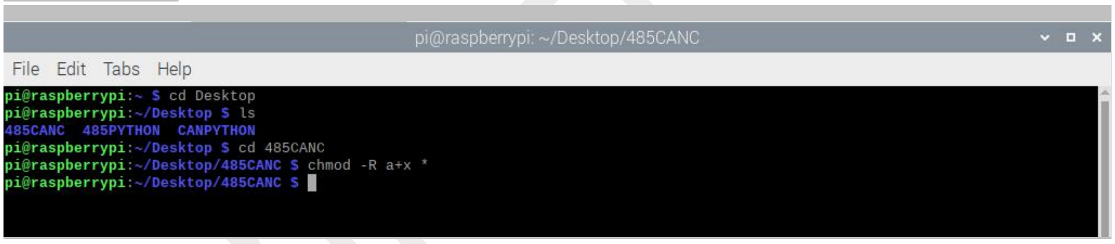
```
# Uncomment some or all of these to enable the optional hardware interfaces
#dtparam=i2c_arm=on
#dtparam=i2s=on
#dtparam=spi=on

# Uncomment this to enable the lirc-rpi module
#dtoverlay=lirc-rpi

# Additional overlays and parameters are documented /boot/overlays/README

# Enable audio (loads snd_bcm2835)
dtparam=audio=on

[pi4]
# Enable DRM VC4 V3D driver on top of the dispmanx display stack
dtoverlay=vc4-fkms-v3d
max_framebuffers=2

[all]
#dtoverlay=vc4-fkms-v3d
dtoverlay=rs485_at
dtoverlay=mcp2515-can1,oscillator=16000000,interrupt=25
```

(4) Insert the TF card into the Raspberry Pi and power on.

(5) Download test codes and copy the folder to the Raspberry Pi by USB flash Dsik or remote login.

(6) Go into the folders and change the permission. Otherwise you can't run the Demo.

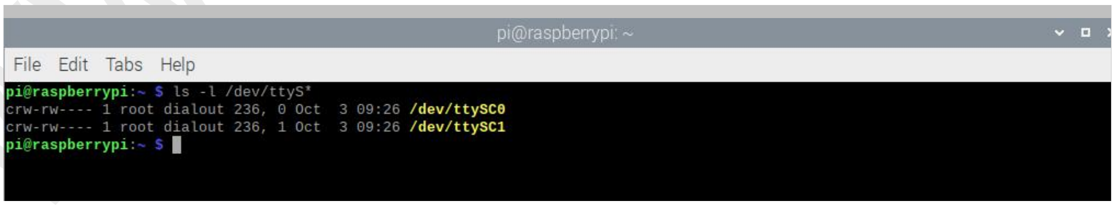
chmod -R a+x *



```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ ls
485CANC  485PYTHON  CANPYTHON
pi@raspberrypi:~/Desktop $ cd 485CANC
pi@raspberrypi:~/Desktop/485CANC $ chmod -R a+x *
pi@raspberrypi:~/Desktop/485CANC $ 
```

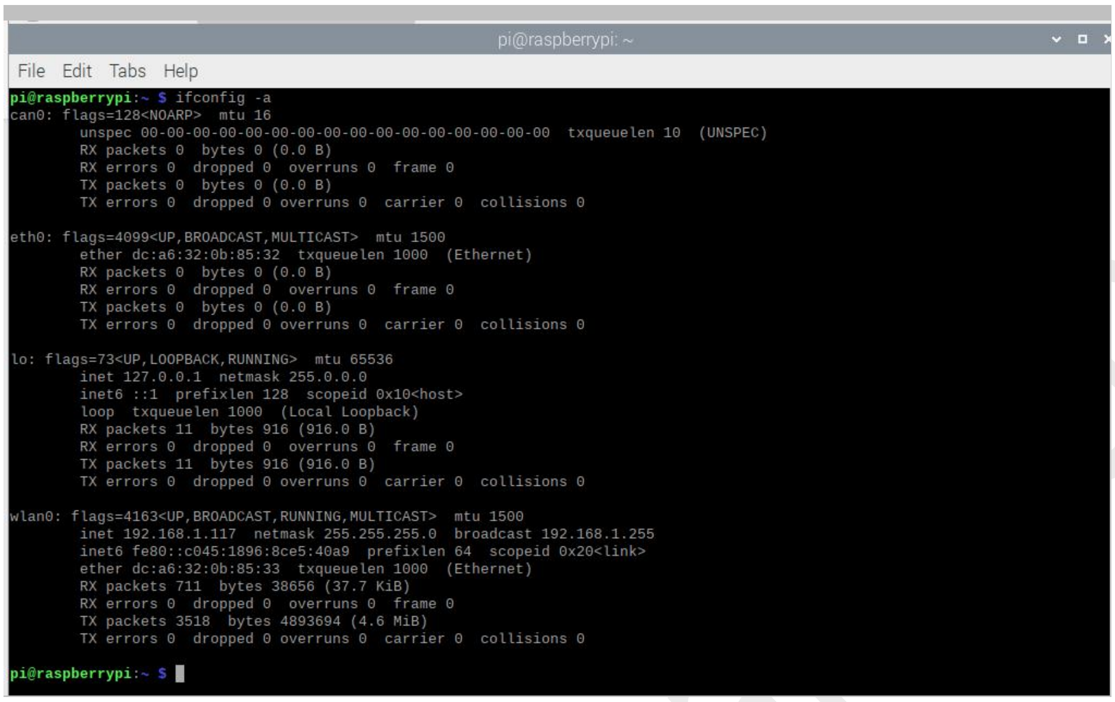
(7) Check the kernel log to see if RS485 was initialized successfully.

ls -l /dev/ttyS*



```
pi@raspberrypi:~ $ ls -l /dev/ttyS*
crw-rw---- 1 root dialout 236, 0 Oct  3 09:26 /dev/ttySC0
crw-rw---- 1 root dialout 236, 1 Oct  3 09:26 /dev/ttySC1
pi@raspberrypi:~ $ 
```

(8) Check the kernel log to see if CAN was initialized successfully.

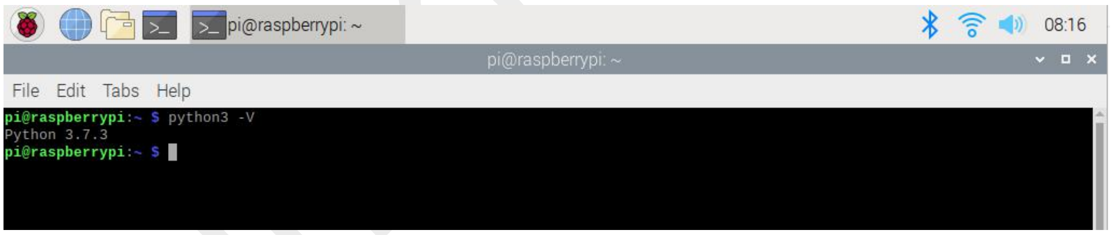


```
pi@raspberrypi:~ $ ifconfig -a
can0: flags=128<NOARP>  mtu 16
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 10  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether dc:a6:32:0b:85:32  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 11  bytes 916 (916.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11  bytes 916 (916.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.117  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::c045:1896:8ce5:40a9  prefixlen 64  scopeid 0x20<link>
        ether dc:a6:32:0b:85:33  txqueuelen 1000  (Ethernet)
        RX packets 711  bytes 38656 (37.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3518  bytes 4893694 (4.6 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

pi@raspberrypi:~ $
```

-----------------If you don't want to use Python, you needn't do below these steps.---------------------

(9)Check the Python version of your Raspbian. Python 3.7.3 default in 2019-07-10-Raspbian.img. Our Demo can run on any Python3 version.

python3 -V



```
pi@raspberrypi:~ $ python3 -V
Python 3.7.3
pi@raspberrypi:~ $
```

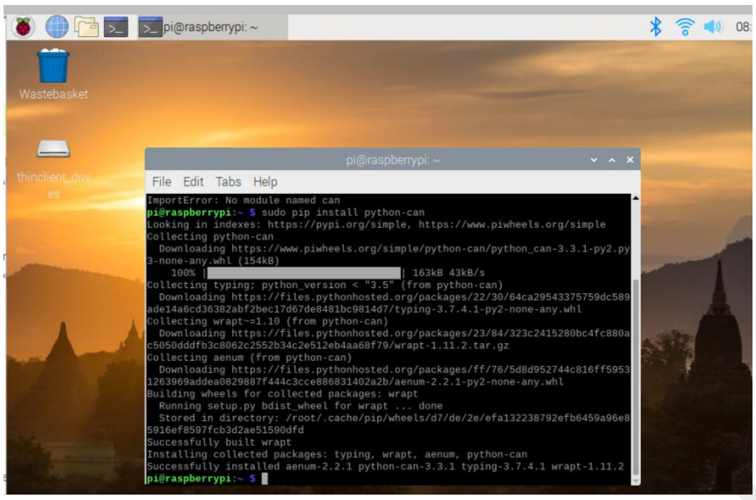
(10)If you can't find the Python3 in system. Install the Python3

sudo apt-get install python-pip

sudo apt-get install python3 idle3 nano

(11)Install Python CAN library.

sudo pip install python-can



```
ImportError: No module named can
pi@raspberrypi:~ $ sudo pip install python-can
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting python-can
  Downloading https://www.piwheels.org/simple/python-can/python_can-3.3.1-py2.py
3-none-any.whl (154kB)
    100% |████████████████████████████████| 163kB 43kB/s
Collecting typing; python_version < "3.5" (from python-can)
  Downloading https://files.pythonhosted.org/packages/22/30/64ca29543375759dc589
ade14a6cd36382abf2bec17d67de8481bc9814d7/typing-3.7.4.1-py2-none-any.whl
Collecting wrapt~=1.10 (from python-can)
  Downloading https://files.pythonhosted.org/packages/23/84/323c2415280bc4fc880a
c5050dddfb3c8062c2552b34c2e512eb4aa68f79/wrapt-1.11.2.tar.gz
Collecting aenum (from python-can)
  Downloading https://files.pythonhosted.org/packages/ff/76/5d8d952744c816ff5953
1263969addea0829887f444c3cce886831402a2b/aenum-2.2.1-py2-none-any.whl
Building wheels for collected packages: wrapt
  Running setup.py bdist_wheel for wrapt ... done
  Stored in directory: /root/.cache/pip/wheels/d7/de/2e/efa132238792efb6459a96e8
5916ef8597fcb3d2ae51590dfd
Successfully built wrapt
Installing collected packages: typing, wrapt, aenum, python-can
Successfully installed aenum-2.2.1 python-can-3.3.1 typing-3.7.4.1 wrapt-1.11.2
pi@raspberrypi:~ $
```

5.Software Description

5.1 Linux C Programing

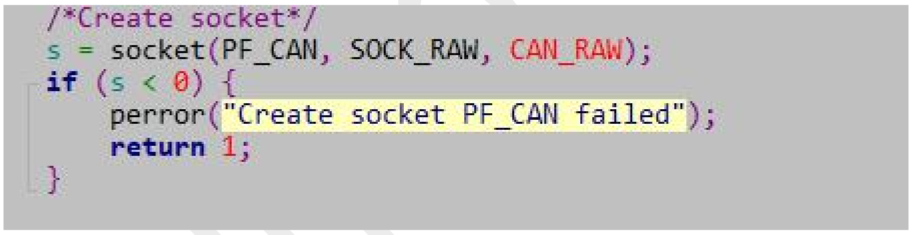
Now with previous demo's code to show you how to program socket can in linux. The socket can is an implementation of CAN protocols(Controller Area Network) for Linux. CAN is a networking technology which has widespread use in automation, embedded devices, and automotive fields. While there have been 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.

5.1.1 C For CAN

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

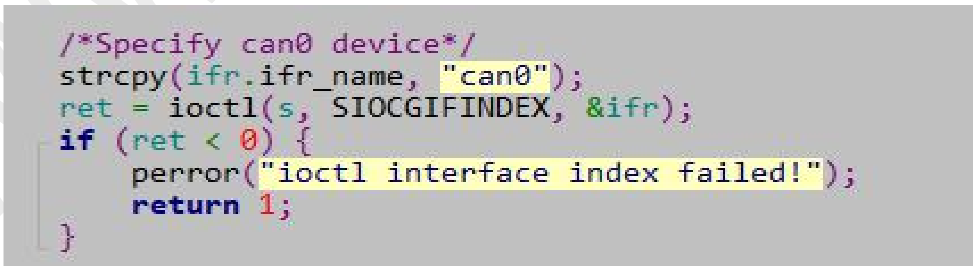
(1) For Sender's code:

a: 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;
}
```

b: 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;
}
```

c: 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;
}
```

d: 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);
```

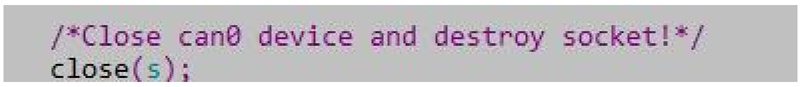
e: Assembly data to send.

```
/*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");
```

f: 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");
}
```

g: Close can0 device and disable socket.

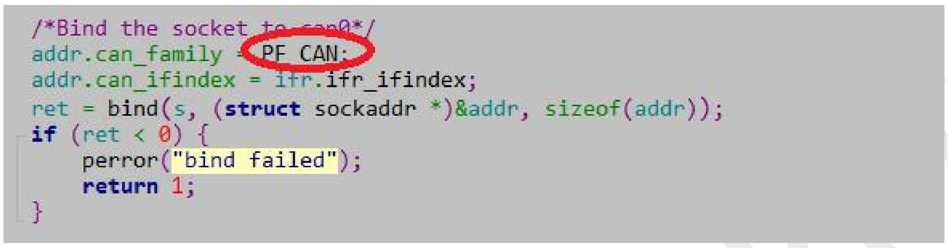


```
/*Close can0 device and destroy socket!*/
close(s);
```

(2) For Receiver'code:

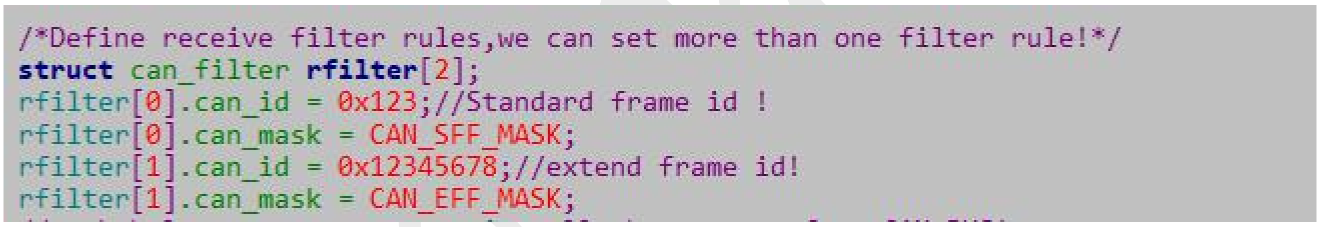
Step a and step b is same as Sender's code.

c: It's different from Sender's



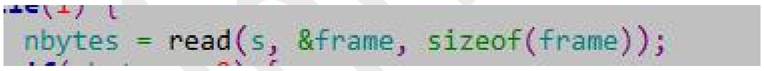
```
/*Bind the socket to can0*/
addr.can_family = PF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
ret = bind(s, (struct sockaddr *)&addr, sizeof(addr));
if (ret < 0) {
    perror("bind failed");
    return 1;
}
```

d： 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;
```

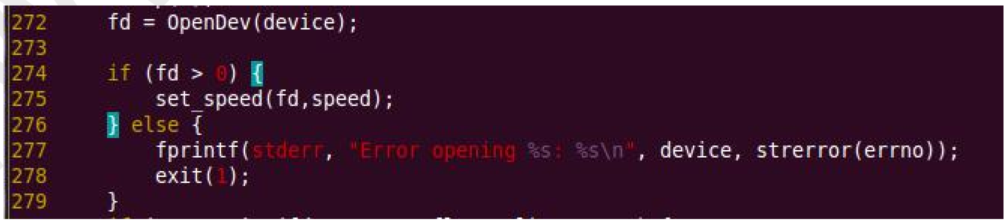
e: Read data back from can bus.



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

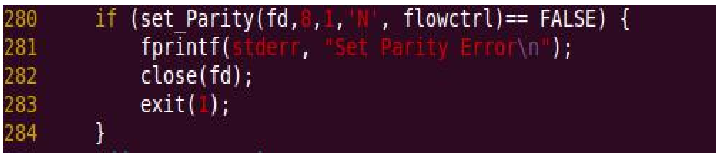
5.2.1 C For RS485

a: Open RS485 Device and set baud rate.



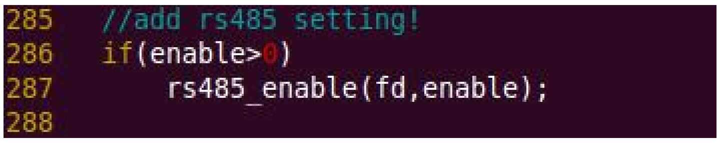
```
    fd = OpenDev(device);

    if (fd > 0) {
        set_speed(fd,speed);
    } else {
        fprintf(stderr, "Error opening %s: %s\n", device, strerror(errno));
        exit(1);
    }
```

b: Set RS485 parity mode and flow control.



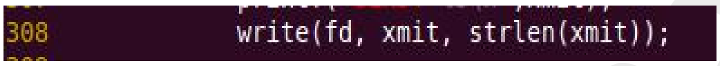
```
    if (set_Parity(fd,8,1,'N', flowctrl)== FALSE) {
        fprintf(stderr, "Set Parity Error\n");
        close(fd);
        exit(1);
    }
```

c: Set RS485 work mode.



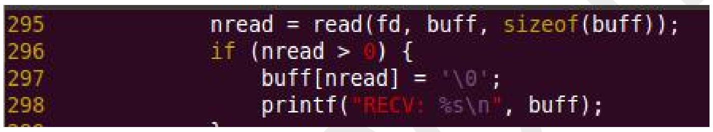
```
//add rs485 setting!
if(enable>0)
    rs485_enable(fd,enable);
```

d: Send data through RS485 bus.



```
            write(fd, xmit, strlen(xmit));
```

e: Get data from RS485 bus.



```
            nread = read(fd, buff, sizeof(buff));
            if (nread > 0) {
                buff[nread] = '\0';
                printf("RECV: %s\n", buff);
```

5.2 Linux Python Programing

5.2.1 Python For CAN

a: 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

b: 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')
```

5.2.1 Python For RS485

a:Enable RS485

```
# RS485 ioctls
TIOCGRS485 = 0x542E
TIOCSRS485 = 0x542F
SER_RS485_ENABLED = 0b00000001
SER_RS485_RTS_ON_SEND = 0b00000010
SER_RS485_RTS_AFTER_SEND = 0b00000100
SER_RS485_RX_DURING_TX = 0b00010000


buf = array.array('i', [0] * 8) # flags, delaytx, delayrx, padding
ser = serial.Serial("/dev/ttySC0",115200)
fcntl.ioctl(ser, TIOCGRS485, buf)

buf[0] |=   SER_RS485_ENABLED|SER_RS485_RTS_AFTER_SEND
buf[1]   = 0
buf[2]   = 0
fcntl.ioctl(ser, TIOCSRS485, buf)
```

b: Send data packet

```
command = ['a','b','c','d']
n = ser.write(command)
```

c: Receive data packet

```
str = ser.read(ser.inWaiting())
```

6.Modbus Description

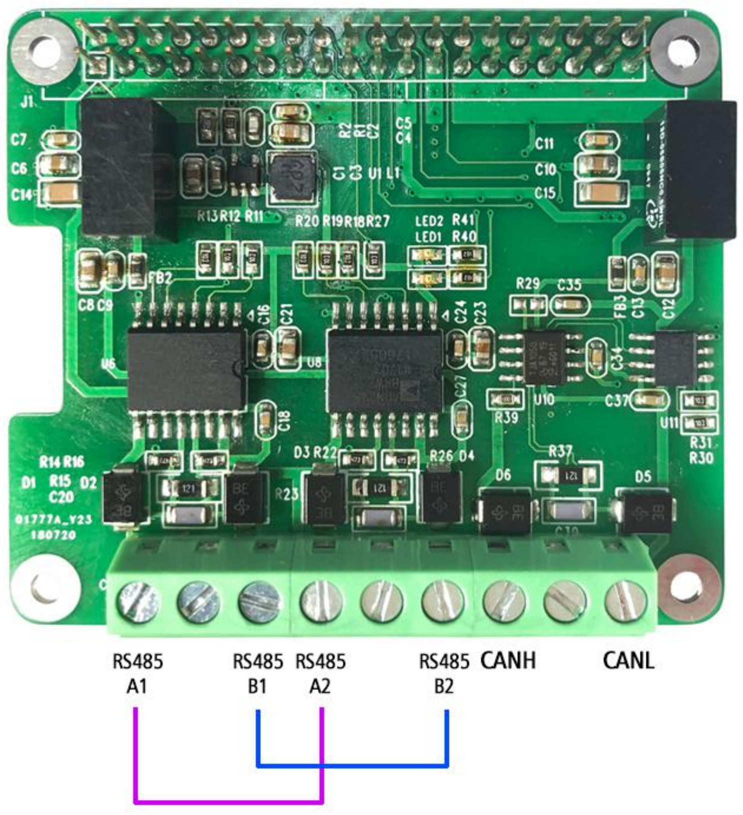
Modbus is a serial communication protocol developed by Modicon published by Modicon® in 1979 for use with its programmable logic controllers (PLCs). In simple terms, it is a method used for transmitting information over serial lines between electronic devices. The device requesting the information is called the Modbus Master and the devices supplying information are Modbus Slaves. In a standard Modbus network, there is one Master and up to 247 Slaves, each with a unique Slave Address from 1 to 247. The Master can also write information to the Slaves.

The official Modbus specification can be found at **http://www.modbus.org/**

Now we use two RS485 ports on-board to make a modbus demo with Python3. set one port as modbus master and the other port as modbus slave in same shield.

6.1 Hardware Connection

Only need to connected the two RS485 ports.



6.2 Run Demo

Download the demo files named ‘modbus’ from our wiki:

wiki.inno-maker.com/display/RS485CANMODULE

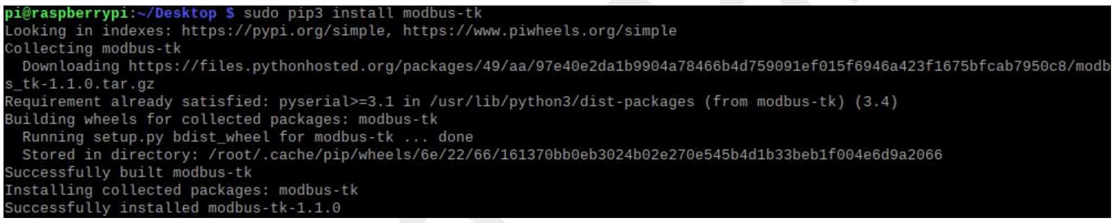
or https://www.jianguoyun.com/p/DTxr8-EQpdSrBxi0irsB

There are two files in the folder, ’rtumaster.py’ and ‘rtuslave.py’. I just make a simple demo to show you how to use ‘READ_HOLDING_REGISTERS’ command to read data with modbus viaRS485.

Demo is base on the one release by luc.jean (**https://github.com/ljean/modbus-tk**). May I suggest that you’d better read his codes and documents carefully, because it can help you have a better understanding of Modbus with coding in Python.

(1) To install dependencies:

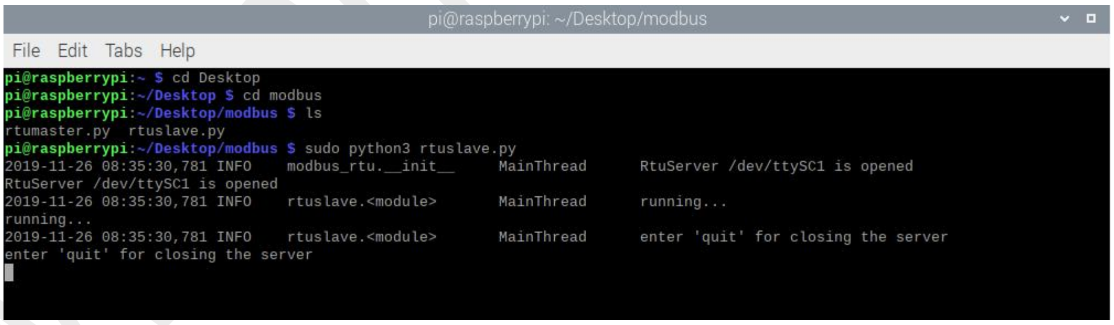
```
sudo pip3 install modbus-tk
```



```
pi@raspberrypi:~/Desktop $ sudo pip3 install modbus-tk
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting modbus-tk
  Downloading https://files.pythonhosted.org/packages/49/aa/97e40e2da1b9904a78466b4d759091ef015f6946a423f1675bfcab7950c8/modb
s_tk-1.1.0.tar.gz
Requirement already satisfied: pyserial>=3.1 in /usr/lib/python3/dist-packages (from modbus-tk) (3.4)
Building wheels for collected packages: modbus-tk
  Running setup.py bdist_wheel for modbus-tk ... done
  Stored in directory: /root/.cache/pip/wheels/6e/22/66/161370bb0eb3024b02e270e545b4d1b33beb1f004e6d9a2066
Successfully built modbus-tk
Installing collected packages: modbus-tk
Successfully installed modbus-tk-1.1.0
```

(2) Set slave:

```
sudo python3 rtuslave.py
```

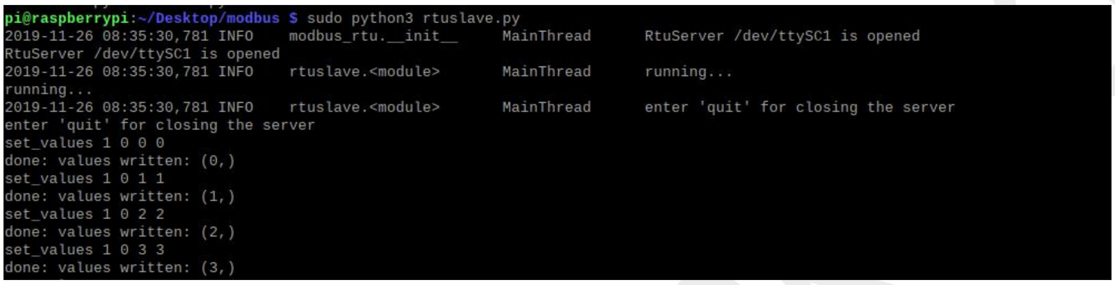


```
pi@raspberrypi:~ $ cd Desktop
pi@raspberrypi:~/Desktop $ cd modbus
pi@raspberrypi:~/Desktop/modbus $ ls
rtumaster.py  rtuslave.py
pi@raspberrypi:~/Desktop/modbus $ sudo python3 rtuslave.py
2019-11-26 08:35:30,781 INFO    modbus_rtu.__init__     MainThread      RtuServer /dev/ttySC1 is opened
RtuServer /dev/ttySC1 is opened
2019-11-26 08:35:30,781 INFO    rtuslave.<module>       MainThread      running...
running...
2019-11-26 08:35:30,781 INFO    rtuslave.<module>       MainThread      enter 'quit' for closing the server
enter 'quit' for closing the server
```

(3) Set the values to salve_1 for test.

'set_values' command format: **slave_id** + **name** + **address** + **values**

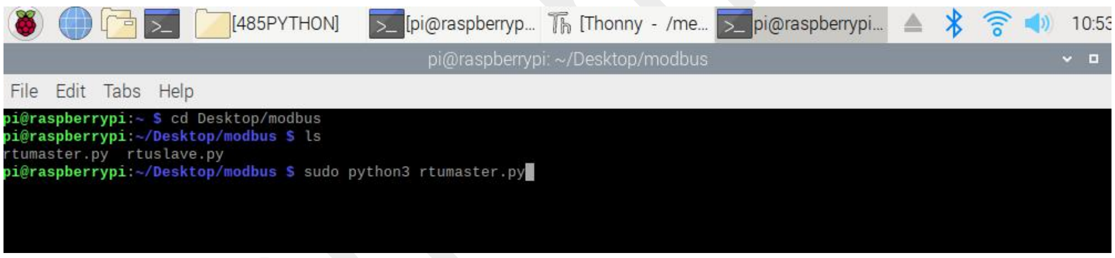
```
set_values 1 0 0 0
set_values 1 0 1 1
set_values 1 0 3 3
```



```
pi@raspberrypi:~/Desktop/modbus $ sudo python3 rtuslave.py
2019-11-26 08:35:30,781 INFO    modbus_rtu.__init__     MainThread      RtuServer /dev/ttySC1 is opened
RtuServer /dev/ttySC1 is opened
2019-11-26 08:35:30,781 INFO    rtuslave.<module>       MainThread      running...
running...
2019-11-26 08:35:30,781 INFO    rtuslave.<module>       MainThread      enter 'quit' for closing the server
enter 'quit' for closing the server
set_values 1 0 0 0
done: values written: (0,)
set_values 1 0 1 1
done: values written: (1,)
set_values 1 0 2 2
done: values written: (2,)
set_values 1 0 3 3
done: values written: (3,)
```

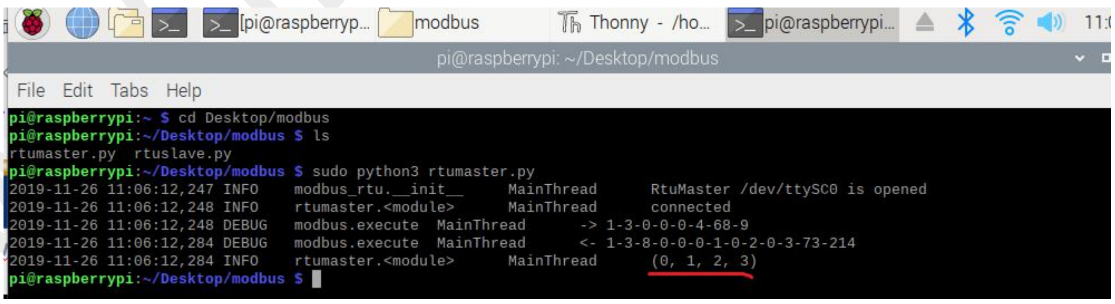
(4) Open a new terminal to set the master

```
sudo python3 rtumaster.py
```



```
pi@raspberrypi:~ $ cd Desktop/modbus
pi@raspberrypi:~/Desktop/modbus $ ls
rtumaster.py  rtuslave.py
pi@raspberrypi:~/Desktop/modbus $ sudo python3 rtumaster.py
```

(5) You can see the result read from master. If you not set values follow step 3, The result will be 0,0,0,0 default.



```
pi@raspberrypi:~ $ cd Desktop/modbus
pi@raspberrypi:~/Desktop/modbus $ ls
rtumaster.py  rtuslave.py
pi@raspberrypi:~/Desktop/modbus $ sudo python3 rtumaster.py
2019-11-26 11:06:12,247 INFO    modbus_rtu.__init__     MainThread      RtuMaster /dev/ttySC0 is opened
2019-11-26 11:06:12,248 INFO    rtumaster.<module>      MainThread      connected
2019-11-26 11:06:12,248 DEBUG   modbus.execute  MainThread      -> 1-3-0-0-0-4-68-9
2019-11-26 11:06:12,284 DEBUG   modbus.execute  MainThread      <- 1-3-8-0-0-0-1-0-2-0-3-73-214
2019-11-26 11:06:12,284 INFO    rtumaster.<module>      MainThread      (0, 1, 2, 3)
pi@raspberrypi:~/Desktop/modbus $
```

Version Descriptions

Version	Description	Author	Date	E-mail
V1.0.0.0	First edition	Calvin	2019.04.30	calvin@inno-maker.com
V1.0.0.1	Added Python, Added chapter 4.6	Calvin	2019.10.07	calvin@inno-maker.com
V1.0.0.2	Added description of 120 ohm	Calvin	2019.10.09	calvin@inno-maker.com
V1.0.0.3	Added description of Modbus	Calvin	2019.11.26	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.