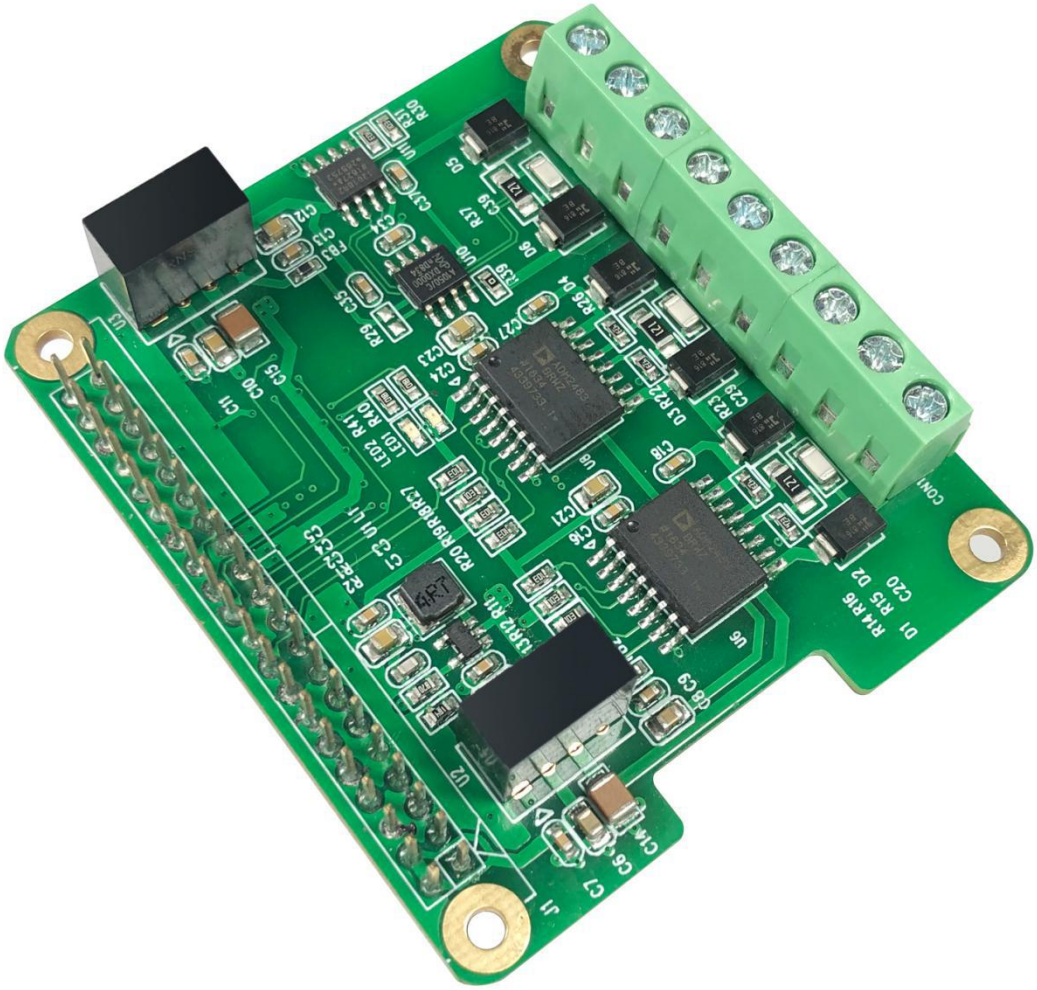
Raspberry Pi RS485&CAN Module User Manual



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+. 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.

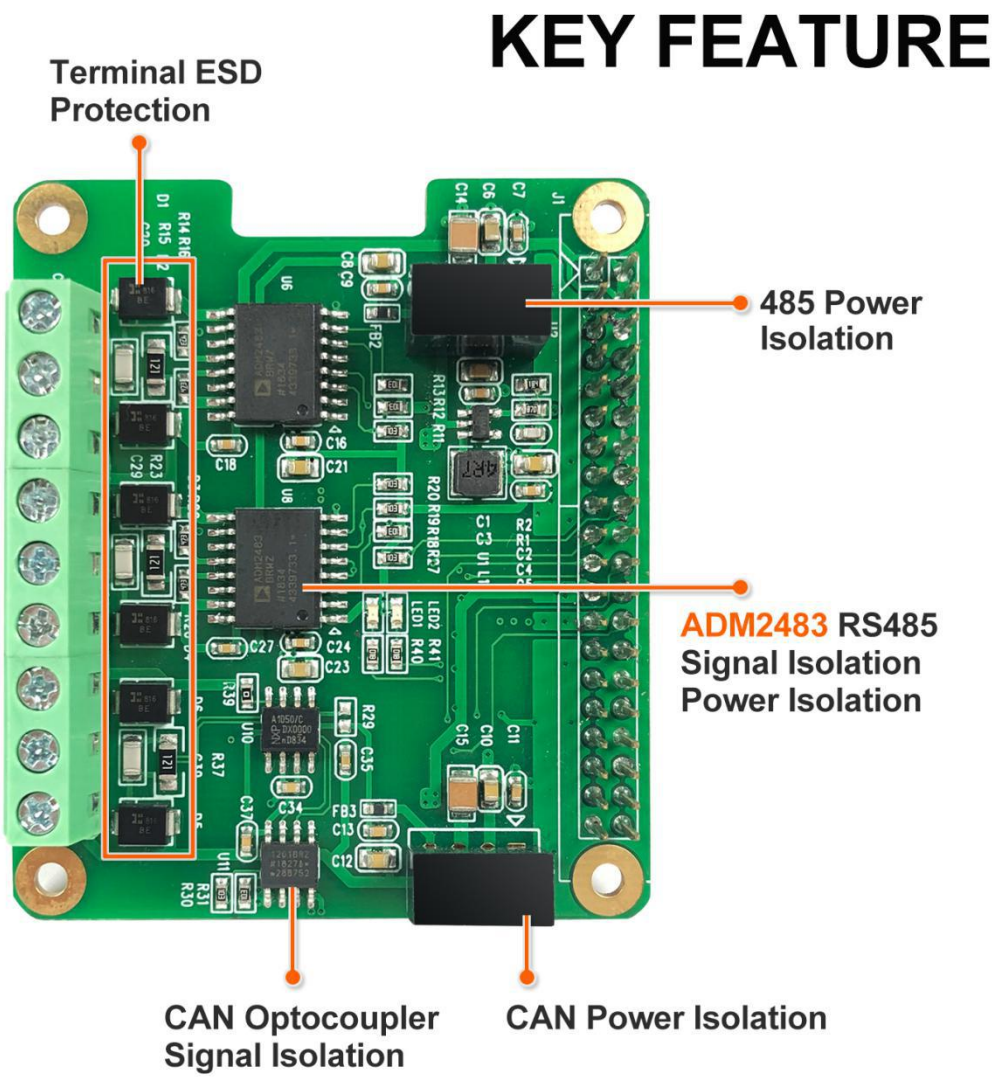
2.3 CAN function: on-board CAN controller MCP2515 via SPI interface, high speed CAN transceiver, digital isolation ADUM1201BRZ, and communication Rates 20Kbps-1Mpbs can be programmed arbitrarily.

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.

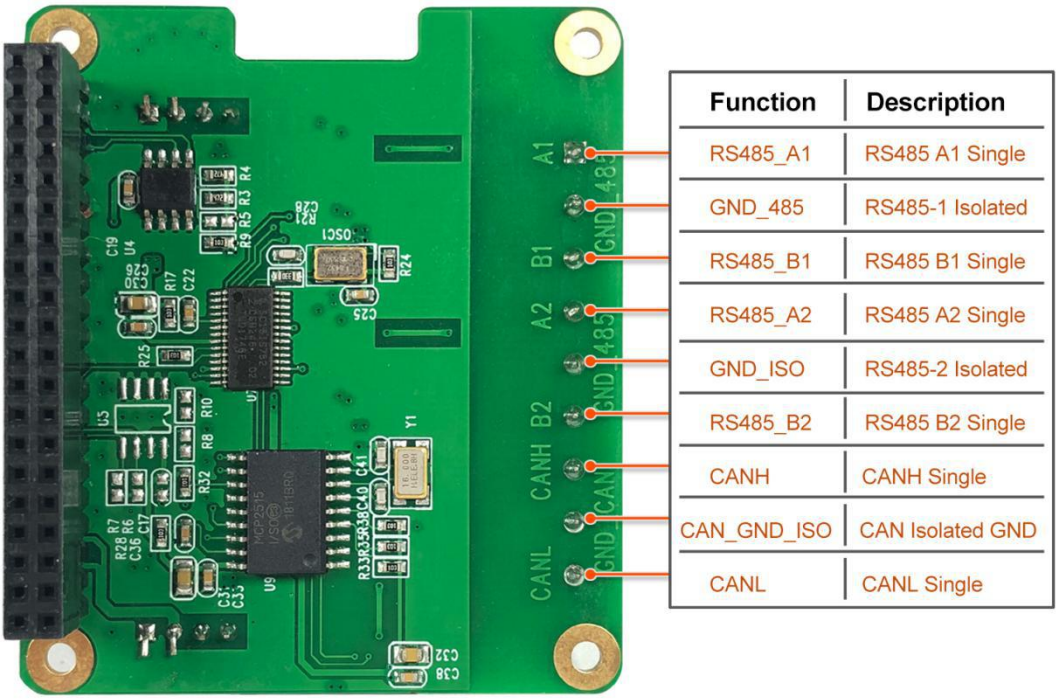
2.5 On-board individual 120 Ohm terminal resistance, Impedance matching and guarantee the ability to drive. On-board User ATC24C32 EEPROM,

3.Hardware Description

3.1 Overview



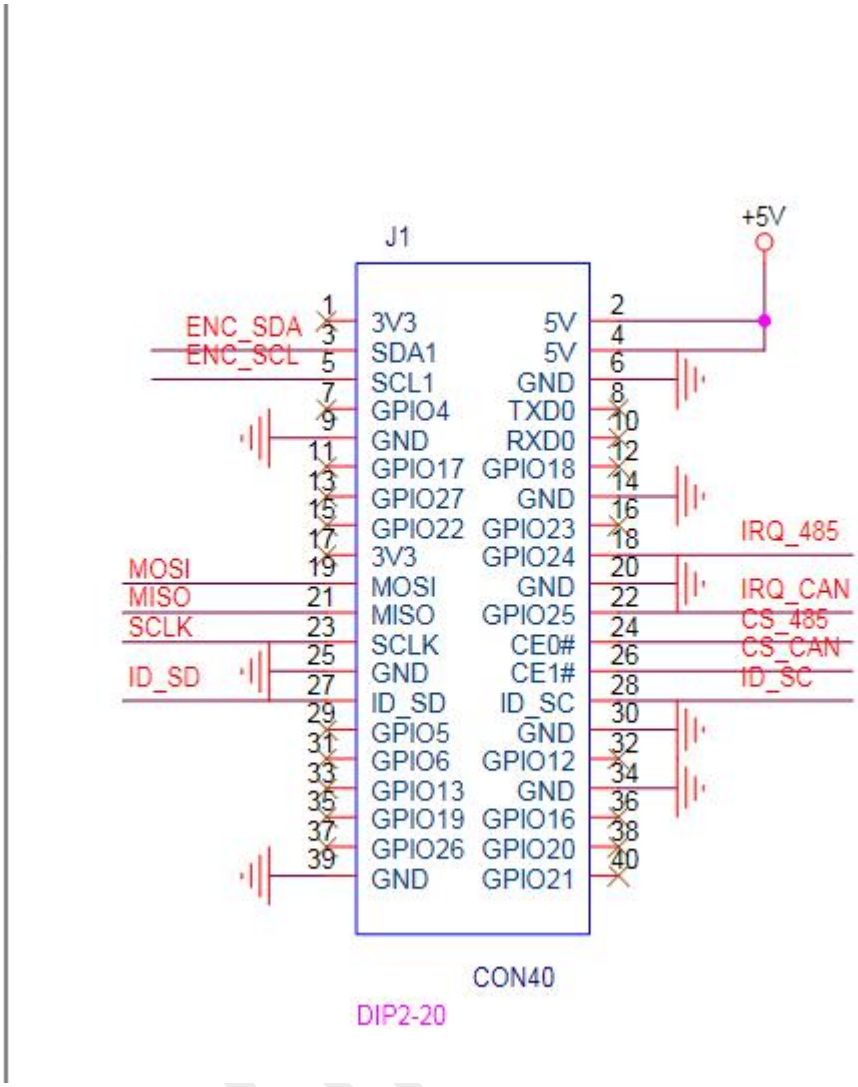
3.2 Output Innterface Desceiption



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 Isolated GND
3	RS485_B1	RS485-1 B Signal
4	RS485_A2	RS485-2 A Signal
5	GND_ISO	RS485-2 Isolated 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

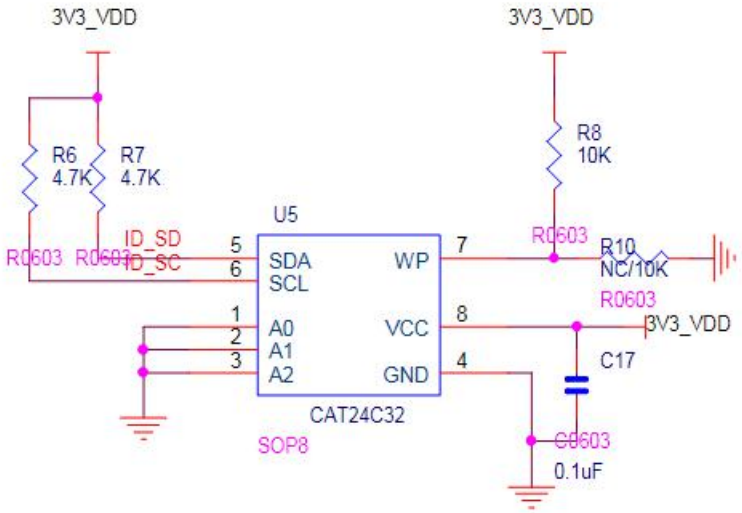


PIN	Symbol	Description
2, 4	+5V	+5V Supply Pin, connected to the main 5V supply of the Raspberry Pi
3, 5	IIC_SDA and IIC_SCL	IIC Used for EEPROM(U4 AT24C32)
19	GPIO_10	SPI0_MOSI
21	GPIO_9	SPI0_MISO
23	GPIO_11	SPI0 _SCLK
24	GPIO_8	RS485 Transfer Chip Selection
26	GPIO_7	CAN Transfer Chip

		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.		

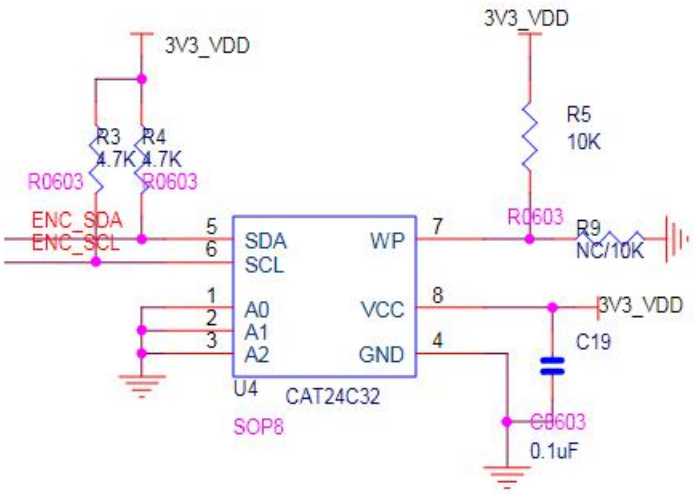
3.4 Extended Function

(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)



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.Software Description

4.1 Download Image on to TF Card

(1) Download the demo image from INNO wiki page:

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

or **http://www.inno-maker.com/wiki/**

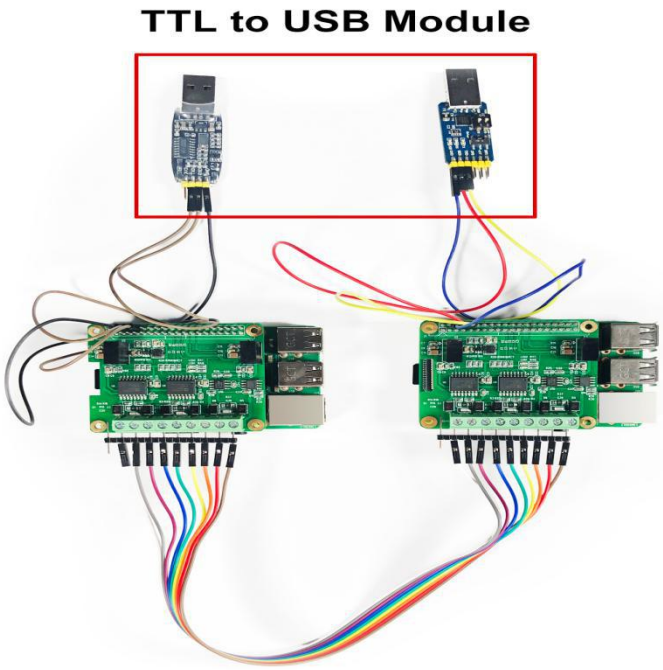
Note :You must use a TF card at least 16GB, otherwise it will write failed.

(2) 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

4.2 CAN Demo Code Testing

(1) Prepare two RS485&CAN modules, two TF cards and two Raspberry Pi boards. Insert the TF card with image into the Raspberry Pi separately. Connect the two connectors in proper order. Connect Raspberry Pi 's Debug UART with your computer (or you can use a display instead).



(2) Turn on the power of both Raspberry Pi and login (login: **pi**, password: **raspberry**),you can see below information. Execute commands to access the test demo.

```
Raspbian GNU/Linux 9 raspberrypi ttyS0
raspberrypi login: pi
Password:
Last login: Tue Jan  8 20:17:09 GMT 2019 on tty1
Linux raspberrypi 4.14.50-v7+ #1122 SMP Tue Jun 19 12:26:26 BST 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
pi@raspberrypi:~$ ls
Desktop    Downloads  MagPi  Pictures  python_games    Templates  veye
Documents  gps        Music  Public    qt application  test       Videos
pi@raspberrypi:~$ cd test
pi@raspberrypi:~/test$
```

(3) Execute following commands in serial terminal. You should see the “can 0” socket can device.

ifconfig –a

```
pi@raspberrypi:~/test$ ifconfig -a
can0: flags=193<UP,RUNNING,NOARP>  mtu 16
```

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

./can_receive

```
pi@raspberrypi:~/test$ ./can_receive
This is a can receive demo,can0 with 1Mbps!
```

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

./can_send

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
pi@raspberrypi:~$ cd test
pi@raspberrypi:~/test$ ./can_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
```

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

```
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
```

4.3 RS485 Demo Code Testing

(1) As in the previous step 1 and step 2.

(2) Execute following commands in both serial terminal to start RS485-1. You will see the data bidirection communication through RS485.

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

```
pi@raspberrypi:~/test$ ./rs485_test -d /dev/ttySC0 -s abcdefghigklmnopqrstuvwxyz
 -e &
[1] 829
pi@raspberrypi:~/test$ Port currently RS485 mode is Not set
RS485 mode will be set
Confirm RS485 mode is set
SEND: abcdefghigklmnopqrstuvwxyz
SEND: abcdefghigklmnopqrstuvwxyz
RECV: abcdefghigklmnopqrstuvwxyz
SEND: abcdefghigklmnopqrstuvwxyz
RECV: abcdefghigklmnopqrstuvwxyz
SEND: abcdefghigklmnopqrstuvwxyz
RECV: abcdefghigklmnopqrstuvwxyz
SEND: abcdefghigklmnopqrstuvwxyz
RECV: abcdefghigklmnopqrstuvwxyz
SEND: abcdefghigklmnopqrstuvwxyz
RECV: abcdefghigklmnopqrstuvwxyz
SEND: abcdefghigklmnopqrstuvwxyz
RECV: abcdefghigklmnopqrstuvwxyz
```

(3) Execute following commands in both serial terminal to start RS485-2. You will see the data bidirection communication through RS485.

```
pi@raspberrypi:~/test$ ./rs485_test -d /dev/ttySC1 -s 123456789abcdef -e
Port currently RS485 mode is Not set
RS485 mode will be set
Confirm RS485 mode is set
SEND:  123456789abcdef
SEND:  123456789abcdef
RECV:  123456789abcdef
SEND:  123456789abcdef
RECV:  123456789abcdef
SEND:  123456789abcdef
RECV:  123456789abcdef
SEND:  123456789abcdef
RECV:  123456789abcdef
```

4.4 Linux C Programing For CAN

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.

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));
```

4.5 Linux C Programing For RS485

a: Open RS485 Device and set baud rate.

```
272     fd = OpenDev(device);
273
274     if (fd > 0) {
275         set_speed(fd,speed);
276     } else {
277         fprintf(stderr, "Error opening %s: %s\n", device, strerror(errno));
278         exit(1);
279     }
```

b: Set RS485 parity mode and flow control.

```
280     if (set_Parity(fd,8,1,'N', flowctrl)== FALSE) {
281         fprintf(stderr, "Set Parity Error\n");
282         close(fd);
283         exit(1);
284     }
```

c: Set RS485 work mode.

```
285   //add rs485 setting!
286   if(enable>0)
287       rs485_enable(fd,enable);
288
```

d: Send data through RS485 bus.

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

e: Get data from RS485 bus.

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

Version Descriptions

Version	Description	Author	Date	E-mail
V1.0.0.0	First edition	Calvin	2019.04.30	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.