USB-CAN MODULE User Manual

General Description:

USB-CAN module is a “plug and play ” and bi-directional port powered USB to CAN converter which realizes long-distance communication between your Raspberry Pi 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.

Features:

1. Compatible with Raspberry Pi Zero/Zero W/2B/3B/3B+.
2. Full support for CAN bus 2.0A and 2.0B specification.
3. Elimination the need for software drivers when you run Debian9 system of Raspberry Pi.
4. Comes with demos of Socket CAN, save your development time.
5. “plug and play” (hot-pluggable) USB device ,no external power required.
6. On board high-speed STM32 microcontroller, stronger data processing ability and lower power consumption.

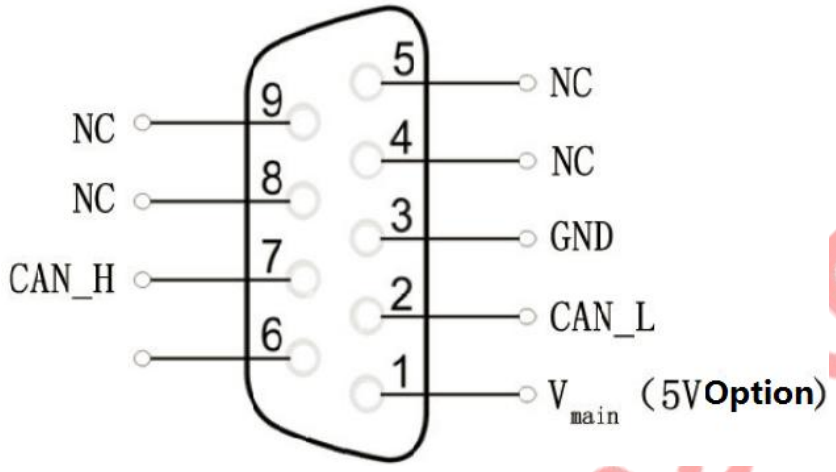
6.Wider CAN baud rate, 20Kbps----1Mpbs can be programmed arbitrarily.

7. Power supply and signal isolation ,Built-in surge and static protection.

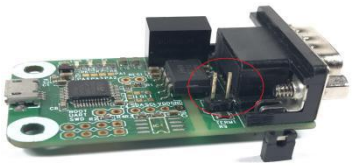
8. 120 Ohm selectable jumper feature.

Hardware Description

1. Pin Out Configuration



2. 120 Ohm resistor setting.



Disable 120 Ohm resistor.

Enable120 Ohm resistor.

Software Description

1. CAN Communication Test Demo

Run demo for two Raspberry Pi：

(1) Prepare two USB-CAN module and two Raspberry Pi board. Connect the H and L port of USB-CAN module to other's port, and then insert into the USB Host of Raspberry PI board.

(2) Down the 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 than 16Gb, Otherwise it will write failed.

(3) Execute commands to visited the test demo.

```
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$
```

(4) 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
```

(5) Set one Pi as receiver, execute following commands in serial terminal. Now this Raspberry pi is blocked.
./can0_receive

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

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

```
./can0_send
```

```
pi@raspberrypi:~/test$ ./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
```

(7) 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
```

Run demo for only one Raspberry Pi：

In this case, you also can insert two USB-CAN module into a Raspberry PI board to test. You should see two can socket “can 0” and “can 1” devices. So notice that you need to modify the code to change “can 0” to “can 1” when you use “can 1” device.

```
pi@raspberrypi:~/test$ 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 3  bytes 24 (24.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1  bytes 8 (8.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

can1: flags=193<UP,RUNNING,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 1  bytes 8 (8.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3  bytes 24 (24.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

pi@raspberrypi:~/test$ ./can0_receive &
[1] 1744
pi@raspberrypi:~/test$ This is a can receive demo,can0 with 1Mbps!

pi@raspberrypi:~/test$ ./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
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:~/test$
```

2. 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.

(1) Preparations:

Elimination the need for software drivers when you run Debian9 system of Raspberry Pi.(2018-06-27-raspbian-stretch.img is validated already)or later.

If you meet problems in other system, You need to reconfigure the kernel drivers. Enable gs_usb can 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.

The compiled firmware with LINUX -4.14.50 version is available from our wiki link as below:
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 than 16Gb, Otherwise it will write failed.

(2) For Sender's code:

Step 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;
}
```

Step 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;
}
```

Step 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;
}
```

Step 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);
```

Step 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");
```

Step 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");
}
```

Step 7: Close can0 device and disable socket.

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

(3) For Receiver’code:

step 1 and step 2 is same as Sender’s code.
step 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;
}
```

Step 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;
```

Step 5: Read data back from can bus.

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

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

Version Descriptions

Version	Description	Author	Date	E-mail
V1.0.0.0	First edition	Calvin	2019.01.015	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.