RS485 CAN HAT
用户手册

产品概述

RS485 CAN HAT 是微雪电子为树莓派开发的一款的带 RS485 和 CAN 通信功能的扩展板，具备 RS485、CAN 通信功能。

特点

- 基于树莓派接口设计，适用于 Raspberry Pi Zero/Zero W/Zero WH/2B/3B/3B+
- 具备 CAN 功能，使用 SPI 接口 CAN 控制器 MCP2515，搭配收发器 SN65HVD230
- 具备 RS485 功能，使用 UART 控制，半双工通讯，收发器为 SP3485
- 预留控制接口，方便其他控制器控制
- 提供完善的配套资料手册(提供 wiringPi 与 python 例程)

产品参数

工作电压：	3.3V
CAN 控制芯片：	MCP2515
CAN 收发器：	SN65HVD230
485 收发器：	SP3485
产品尺寸：	65mmx30mm
固定孔通经：	3.0mm

版本：V1.0.0，日期：2018 年 05 月 29 日

接口说明

CAN 总线接口

功能引脚	树莓派接口（BCM）	描述
3V3	3V3	3.3V 电源正
GND	GND	电源地
SCK	SCK	SPI 时钟输入
MOSI	MOSI	SPI 数据输入
MISO	MISO	SPI 数据输出
CS	CE0	数据/命令选择
INT	25	中断输出

RS485 总线接口

功能引脚	树莓派接口（BCM）	描述
3V3	3V3	3.3V 电源正
GND	GND	电源地
RXD	RXD	串口接收
TXD	TXD	串口发送
RSE	18	设置收发

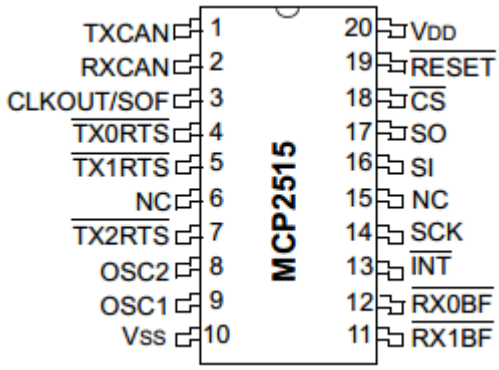
硬件说明

1.CAN 总线

CAN 模块的功能是处理所有 CAN 总线上的报文接收和发送。报文发送时，首先将报文装载到正确的报文缓冲器和控制寄存器中。通过 SPI 接口设置控制寄存器中的相应位或使用发送使能引脚均可启动发送操作。通过读取相应的寄存器可以检查通讯状态和错误。 会对在 CAN 总线上检测到的任何报文进行错误检查，然后与用户定义的滤波器进行匹配，以确定是否将报文移到两个接收缓冲器中的一个。

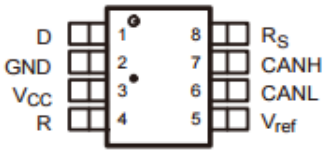
由于树莓派本身并不支持 CAN 总线，因此使用 SPI 接口的 CAN 控制器，搭配一个收发器完成 CAN 功能。

Microchip 的 MCP2515 是一款 CAN 协议控制器，完全支持 CAN V2.0B 技术规范。该器件能发送和接收标准和扩展数据帧以及远程帧。 MCP2515 自带的两个验收屏蔽寄存器和六个验收滤波寄存器可以过滤掉不想要的报文，因此减少了主单片机（MCU）的开销。MCU 通过 SPI 接口与该器件连接,即树莓派通过 SPI 接口连接芯片，对于树莓派使用该芯片不需要编写驱动，只需要打开设备树中的内核驱动即可使用。



更多详细请参考数据手册；

SN65HVD230 是德州仪器公司生产的 3.3V CAN 收发器，该器件适用于较高通信速率、良好抗干扰 能力和高可靠性 CAN 总线的串行通信。SN65HVD230 具有高速、斜率和等待 3 种不同的工作模式。 其工作模式控制可通过 Rs 控制引脚来实现。CAN 控制器的输出引脚 Tx 接到 SN65HVD230 的数据 输入端 D，可将此 CAN 节点发送的数据传送到 CAN 网络中；而 CAN 控制器的接收引脚 Rx 和 SN65HVD230 的数据输出端 R 相连，用于接收数据。

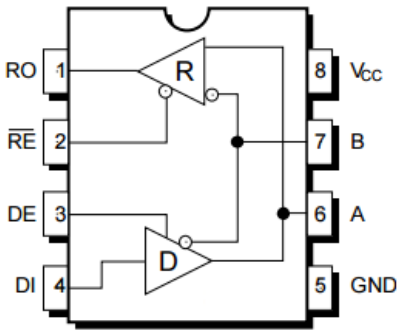


SN65HVD230 引脚
（顶视图）

引脚	名称	说明
1	D	驱动输入 Driver input
2	GND	电源地线 Ground
3	V_{CC}	电源线 Supply voltage
4	R	接收输出 Receiver output
5	V_{ref}	参考输出 Reference output
6	CANL	低总线输出 Low bus output
7	CANH	高总线输出 High bus output
8	RS	工作模式控制端 Standby/slope control

2. RS485 总线

SP3485 接口芯片是一种 RS－485 驱动芯片。用于 RS-485 通信的低功耗收发器。 采用单一电源+3.3 V 工作，采用半双工通讯方式。RO 和 DI 端分别为接收器的输出和驱动器的输入端；$\overline{RE}$和 DE 端分别为接收和发送的使能端，当$\overline{RE}$为逻辑 0 时，器件处于接收状态；当 DE 为逻辑 1 时，器件处于发送状态；A 端和 B 端分别为接收和发送的差分信号端，当 A-B>+0.2V 时，RO 输出逻辑 1；当 A-B<-0.2V 时，RO 输出逻辑 0。A 和 B 端之间加匹配电阻，一般可选 100Ω的电阻。



Top View
SP485/MAX485/SP3485/MAX3485
引脚（顶视图）

引脚	名称	说明
1	RO	接收器输出 Receiver Output
2	$\overline{RE}$	接收输出使能 Receiver Output Enable 低电平有效 Active LOW
3	DE	发送输出使能 Driver Output Enable 高电平有效 Active HIGH
4	DI	输出驱动器输入 Driver Input
5	GND	地 Ground Connection
6	A	差分信号正向端 Driver Output/Receiver Input. Non-inverting
7	B	差分信号反向端 Driver Output/Receiver Input. Inverting
8	V_{CC}	

SP485 / MAX485 是 5V 的 RS485 收发器
SP3485 / MAX3485 是 3.3V 的 RS485 收发器

树莓派使用

安装必要的函数库

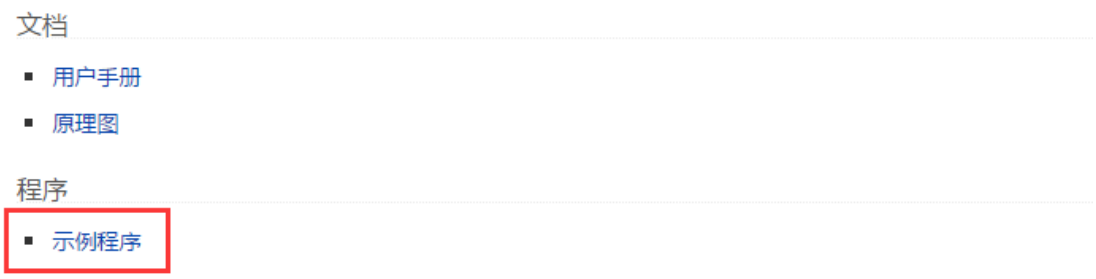
需要安装必要的函数库（wiringPi、bcm2835、python 库），否则以下的示例程序可能无法正常工作。安装方法详见：

http://www.waveshare.net/wiki/Pioneer600_Datasheets

如若使用 python 库，装完上述基本的库之后，还有如下库需要安装

sudo apt-get install python-pip
sudo pip install python-can

在官网上找到对应产品，在产品资料打开下载路径，在 wiki 中下载示例程序：



得到解压包，解压并将程序复制到树莓派中。

CAN 使用

本演示程序使用了两个树莓派以及两个 RS485 CAN HAT 模块

提供 python 与 c 语言程序

前置工作

将模块插在树莓派上，然后修改开机脚本 config.txt
sudo vi /boot/config.txt
在最后一行加入如下：
dtparam=spi=on

dtoverlay=mcp2515-can0,oscillator=8000000,interrupt=25,spimaxfrequency=1000000

保存退出后，重启树莓派：

sudo reboot

重启后，运行命令查看是否初始化成功：

dmesg | grep -i '\(can\|spi\)'

将会有如下的打印信息：

```
pi@raspberrypi:~ $ dmesg | grep -i '\(can\|spi\)'
[   16.369968] systemd[1]: Cannot add dependency job for unit regenerate_ssh_host_keys.service, ignoring: Unit regener
ate_ssh_host_keys.service failed to load: No such file or directory.
[   16.568756] systemd[1]: Cannot add dependency job for unit display-manager.service, ignoring: Unit display-manager.
service failed to load: No such file or directory.
[   20.892310] CAN device driver interface
[   20.915484] mcp251x spi0.0 can0: MCP2515 successfully initialized.
```

如果不接上模块可能提示如下：

```
pi@raspberrypi:~ $ dmesg | grep -i '\(can\|spi\)'
[   16.300731] systemd[1]: Cannot add dependency job for unit regenerate_ssh_host_keys.service, ignoring: Unit regener
ate_ssh_host_keys.service failed to load: No such file or directory.
[   16.499602] systemd[1]: Cannot add dependency job for unit display-manager.service, ignoring: Unit display-manager.
service failed to load: No such file or directory.
[   20.661718] CAN device driver interface
[   20.680261] mcp251x spi0.0: Cannot initialize MCP2515. Wrong wiring?
[   20.680293] mcp251x spi0.0: Probe failed, err=19
```

请检查是否连接上模块。是否开启 SPI 并开启 MCP2515 内核驱动。是否进行重启。

把两个模块的 H 与 L 对应连接

C 语言例程

浏览目录：

```
pi@raspberrypi:~ $ ls RS485_CAN_HAT_code/can/c/
receive  send
```

接收：

进入目录：cd /RS485_CAN_HAT_code/can/c/receive

编译：make

执行：sudo ./can_receive

```
pi@raspberrypi:~/RS485_CAN_HAT_code/can/c/receive $ sudo ./can_receive
this is a can receive demo
```

此时，接收程序是阻塞的，直到读取到数据就结束。

发送：

进入目录：cd /RS485_CAN_HAT_code/can/c/send

编译：make

执行：sudo ./can_ send

```
pi@raspberrypi:~/RS485_CAN_HAT_code/can/c/send $ sudo ./can_send
this is a can send demo
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
```

此时接收接收到对于的 id 的报文：

```
pi@raspberrypi:~/RS485_CAN_HAT_code/can/c/receive $ sudo ./can_receive
RTNETLINK answers: Device or resource busy
this is a can receive demo
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
```

Python 例程

浏览目录：

```
pi@raspberrypi:~/RS485_CAN_HAT_code/can/c $ ls
receive  send
pi@raspberrypi:~/RS485_CAN_HAT_code/can/c $ cd ../python/
pi@raspberrypi:~/RS485_CAN_HAT_code/can/python $ ls
README.txt  receive.py  send.py
```

先运行接收：

sudo python can_reveive.py

发送端：

sudo python can_send.py

RS485 使用

本演示程序使用了两个树莓派以及两个 RS485 CAN HAT 模块

提供 python 与基于 wiringPi 库开发程序

前置工作

由于树莓派默认的串口是用于调试信息，需要释放串口：
执行如下命令进入树莓派配置
sudo raspi-config

选择 Advanced Options ->Serial ->no 关闭串口调试功能
打开/boot/config.txt 文件，找到如下配置语句使能串口，如果没有，可添加在文件最后面。
Enable_uart=1

对于树莓派 3B 用户，串口用于蓝牙，需要注释掉：
#dtoverlay=pi3-miniuart-bt

然后重启树莓派：
sudo reboot

把两个模块的 A,B 对应连接

WiringPi 程序

查看目录：

```
pi@raspberrypi:~/RS485_CAN_HAT_code/485/WiringPi $ ls
receive  send
```

接收：

进入目录：cd /RS485_CAN_HAT_code/can/c/receive

编译：make

执行：sudo ./can_receive

```
pi@raspberrypi:~/RS485_CAN_HAT_code/485/WiringPi/receive $ sudo ./485_receive
set wiringPi lib success  !!!
```

此时，接收程序是阻塞的，直到读取到数据就结束。

发送：

进入目录：cd /RS485_CAN_HAT_code/can/c/send

编译：make

执行：sudo ./can_ send

```
pi@raspberrypi:~/RS485_CAN_HAT_code/485/WiringPi/send $ sudo ./485_send
set wiringPi lib success  !!!
send data 123456789
```

此时接收端接收到数据：

```
pi@raspberrypi:~/RS485_CAN_HAT_code/485/WiringPi/receive $ sudo ./485_receive
set wiringPi lib success  !!!
1
2
3
4
5
5
6
7
8
9
```

Python 例程

浏览目录：

```
pi@raspberrypi:~/RS485_CAN_HAT_code/485/python $ ls
receive.py  send.py
```

先运行接收：

sudo python reveive.py

发送端：

sudo python send.py

代码分析

CAN

分别提供了两种例程，一个 C 语言，一种 python；

C 语言使用了 socket-can 编程，python 也采用了类似的的库。

C 语言

对于 c 语言，使用的是类似 linux 中网络编程的技巧，使用套接字来进行控制。如果你了解 linux 网络编程，那么这里是类似的：Socketcan 套接字是 Linux 下 CAN 协议的实现方法。

在发送之前要先创建一个 can 设备，因为前面只是启用 MCP2515 内核：

```
system("sudo ip link set can0 type can bitrate 100000");
system("sudo ifconfig can0 up");
```

第一步：打开套接字

```
s = socket(PF_CAN, SOCK_RAW, CAN_RAW);
```

如果失败了会返回-1

第二步：指定设备 can0

```
strcpy(ifr.ifr_name, "can0");
ret = ioctl(s, SIOCGIFINDEX, &ifr);
```

第三步：将套接字绑定在 CAN 接口

```
addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
ret = bind(s, (struct sockaddr *)&addr, sizeof(addr));
```

第四步：设置规则，不接收数据包，只发送

```
setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, NULL, 0);
```

第五步：设置发送的数据

```
struct can_frame frame;
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;
```

第六步：发送数据

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

通过 write()函数将发送的数据写入套接字中，如果失败了会返回-1,成功返回写入的字节数
因此可以通过这个来判断是否发送成：

```
if(nbytes != sizeof(frame)) {
    printf("Send Error frame[0]!\r\n");
    system("sudo ifconfig can0 down");
}
```

第七步：关闭套接字和 CAN 设备

```
close(s);
system("sudo ifconfig can0 down");
```

如果不关闭 CAN 设备，那么下次发送会提示总线忙。

对于接收而言：
在绑定套接字的时候不一样

```
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;
}
```

可以定义接收规则：
只接受 ID 为 0X123 的报文：

```
struct can_filter rfilter[1];
rfilter[0].can_id = 0x123;
rfilter[0].can_mask = CAN_SFF_MASK;
setsockopt(s, SOL_CAN_RAW, CAN_RAW_FILTER, &rfilter, sizeof(rfilter));
```

读取使用 read():

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

将会返回读取到的字节数

更多 socket-can 的编程请参考：https://www.kernel.org/doc/Documentation/networking/can.txt

Python

使用 python 例程，确保以及安装了 python-can 库

与 c 语言类似的都是要先创建 can 设备:

```
os.system('sudo ip link set can0 type can bitrate 100000')
os.system('sudo ifconfig can0 up')
```

第一步：连接到 CAN 总线

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

第二步：创建信息

```
msg = can.Message(arbitration_id=0x123, data=[0, 1, 2, 3, 4, 5, 6, 7], extended_id=False)
```

第三步：发送信息

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

最后同样要关闭 can 设备

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

接收数据：

```
msg = can0.recv(10.0)
```

recv()中定义超时接收时间。

更多请参考：https://python-can.readthedocs.io/en/stable/interfaces/socketcan.html

RS485

分别提供了两种例程，一中基于树莓派 wiringPi 库，一种 python；

wirngPi

第一步：设置接收与发送：

从硬件说明得知：SP3485 芯片的 RE 与 DE 管脚是设置接收与发送；

```
#define EN_485 18

    if(wiringPiSetupGpio() < 0) { //use BCM2835 Pin number table

        printf("set wiringPi lib failed      !!! \r\n");

        return -1;

    } else {
```

```
        printf("set wiringPi lib success    !!! \r\n");

    }

    pinMode(EN_485, OUTPUT);

    digitalWrite(EN_485,HIGH);
```

这里配置成发送，这里的 Pin18 管脚是 BCM 编码，对于 wiringPi 而言可以使用 bcm 管脚编码也可使用 wiringpi 编码的管脚，wiringPiSetupGpio()是设置采用 BCM 编码，wiringPiSetup()是设置成 wiringpi 编码。

第二步：创建文件描述符，打开串口文件/dev/ttyS0，并设置波特率：

```
    if((fd = serialOpen ("/dev/ttyS0",9600)) < 0) {
        printf("serial err\n");
        return -1;
    }
```

第三步：发送数据

```
    serialFlush(fd);
    serialPrintf(fd,"\r");
    serialPuts(fd, "12345");
```

serialFlush()是清除串口中全部数据，等待发送给定设备

serialPrintf()类似 printf 函数，可以将需要发送的数据绑定到文件描述符上；

serialPuts()将以 nul 结尾的字符串发送到由给定文件描述符标识的串行设备；

由于接收端的 serialGetchar(fd)函数返回串行设备上可用的下一个字符，是一个比较坑的函数，因此发送端主动发送一个\r 字符来抵消掉这个现象。（如果您有更好的解决方法请联系我们）。

更多操作函数，请参考 http://wiringpi.com/reference/serial-library/

Python

对于 python 控制 RS485 就比较简单了，即直接操作串口：

同样的打开串口文件并设置波特率：

```
t = serial.Serial("/dev/ttyS0",115200)
```

```
strInput = raw_input('enter some words:')
```

可以输入你想要发送的数据

把数据写入串口文件中，将返回写入的字节数:

```
n = t.write(strInput)
```

读取：

```
    str = ser.readall()
```