2-CH CAN FD HAT

From Waveshare Wiki

Contents

2-CH CAN FD HAT

2-CH CAN FD HAT, SPI interfaces

Primary Attribute

Category:

Brand: Waveshare

Website

English: Waveshare Website (https://www.waveshare.com/2-ch-can-fd-hat.htm)

Chinese: 官方中文站点 (http://www.waveshare.net/shop/2-CH-CAN-FD-HAT.htm)

Onboard Interfaces

SPI **CAN**

Introduction

This is a 2-Channel CAN bus expansion HAT designed for Raspberry Pi, supports CAN FD (CAN with Flexible Data-Rate), the speed is higher than the traditional 1Mbps of CAN2.0, features multi onboard protection circuits, high anti-interference capability, and stable operation. It suits for fields such as automotive devices or industrial automation.

More (https://www.waveshare.com/2-ch-can-fd-hat.htm)

Features

- Standard Raspberry Pi 40PIN GPIO extension header, supports Raspberry Pi series boards
- Supports both traditional CAN2.0 and CAN FD protocols, up to 8Mbps data rate
- Breakout SPI control pins, for connecting with host control boards like STM32/Arduino

- Onboard electrical isolation, up to 5KV isolated voltage, high anti-interference capability, stable operation
- Onboard lighting proof, ESD, short circuit protection, more safe communication
- Onboard voltage translator, select 3.3V/5V operating voltage by jumper
- Onboard 120Ω terminal resistor, configured by jumper
- Comes with development resources and manual (examples for Raspberry Pi/Arduino)

Specifications

- CAN controller: MCP2517FD
- CAN transceiver: MCP2562FD
- Communication interface: SPI (two channels independent SPI by default, swith to one channel SPI by soldering)
- Power supply: external power input terminal, or Raspberry Pi
- Power input terminal voltage: DC 8~26V
- Operating voltage: 5V
- Logic level: 3.3V/5V
- Dimensions: 65.0 x 56.5 mm

Interfaces

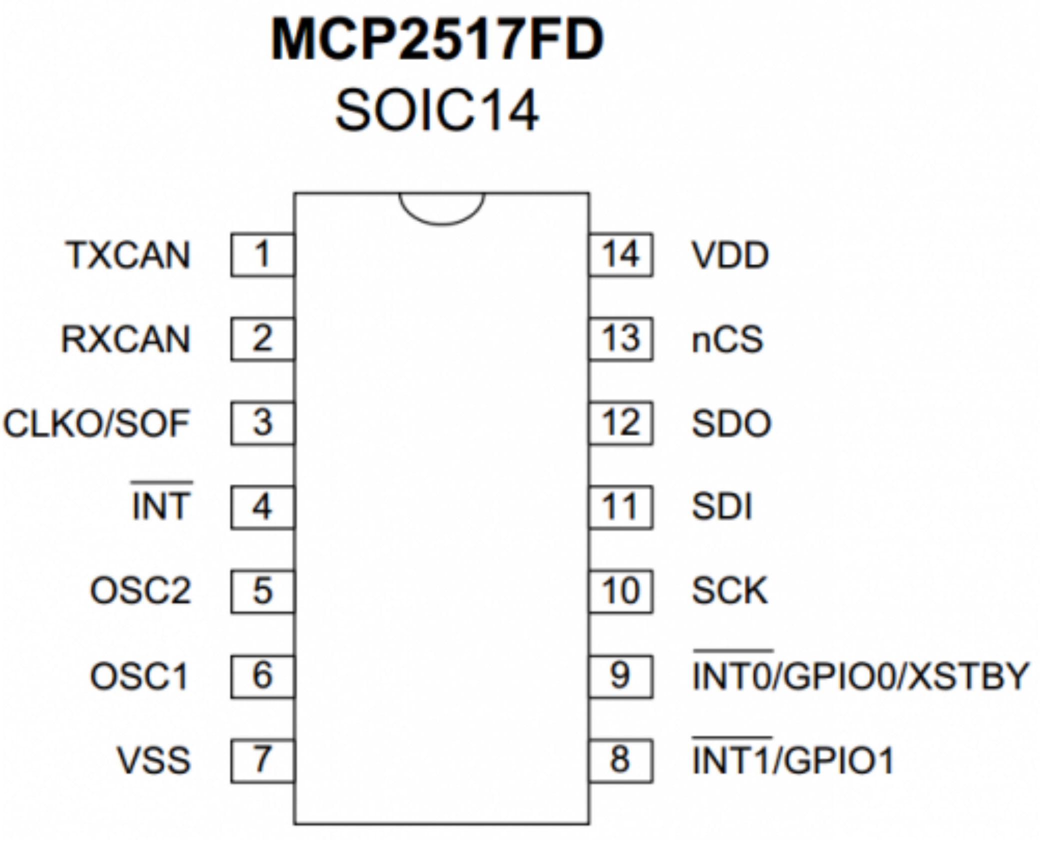
CAN interface

PIN	Raspberry Pi (BCM2835)	Raspberry Pi (WPI)	Description
5V	5V	5V	5V Power input
GND	GND	GND	Ground
MISO_0	9 (MISO)	13 (MISO)	SPI_0 Data output
MOSI_0	10 (MOSI)	12(MOSI)	SPI_0 Data input
SCK_0	11 (SCK)	14 (SCK)	SPI_0 Clock input
CS_0	8 (CE0)	10 (CE0)	CAN_0 Chip select
INT_0	25	6	CAN_0 Interrupt Pin
MISO_1	9/19 (MISO)	13/24 (MISO)	SPI_1 Data output
MOSI_1	10/20 (MOSI)	12/28 (MOSI)	SPI_1 Data input
SCK_1	11/21(SCLK)	14/29 (SCLK)	SPI_1 Clock input
CS_1	7(CE1)/26	11(CE1)/25	CAN_1 Chip select
INT_1	16	27	CAN_1 Interrupt Pin

Hardware description

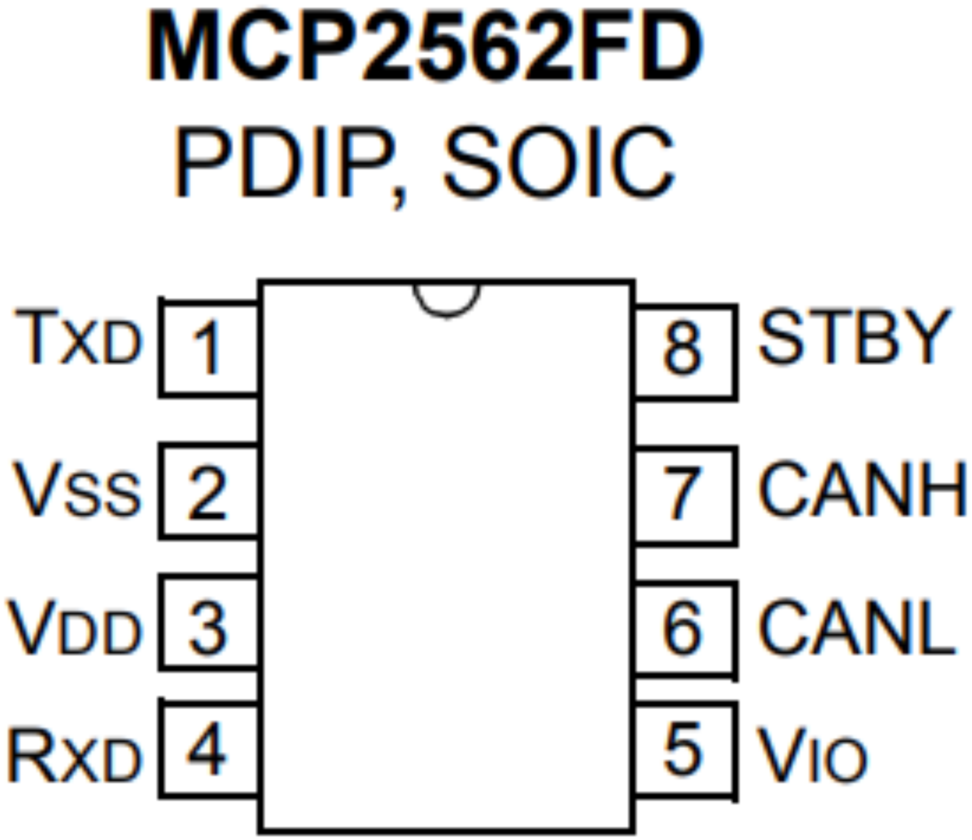
CAN

Raspberry Pi doesn't support CAN communication. If you want to do CAN communicating with Raspberry Pi, you require an expansion board which extends the CAN function. Here is the 2-CH CAN FD HAT.


MCP2517FD
SOIC14
TXCAN 1
RXCAN 2
CLKO/SOF 3
$\overline{\text{INT}}$ 4
OSC2 5
OSC1 6
VSS 7
14 VDD
13 nCS
12 SDO
11 SDI
10 SCK
9 $\overline{\text{INT0}}$/GPIO0/XSTBY
8 $\overline{\text{INT1}}$/GPIO1

The MCP2517FD is a cost-effective and small-footprint CAN FD controller that can be easily added to a microcontroller with an available SPi interface. The MCP2517FD supports both, CAN frames in the Classical format (CAN2.0B) and CAn Flexible Data Rate (CAN FD) as specified in ISO 11898-1:2015. Its arbitration bit rate is up to 1Mbps, data bit rate up to 8Mbps. It supports up to 29 MHz SPi clock speed. All the standard frame, extended frame, and remote frames are receivable and transmittable. 32 intel flexible filter and mask object allows the CP2517FD to filter usable packets to reduce consumption of MCU.

You can connect this module to your board like Raspberry Pu by SPi interface.

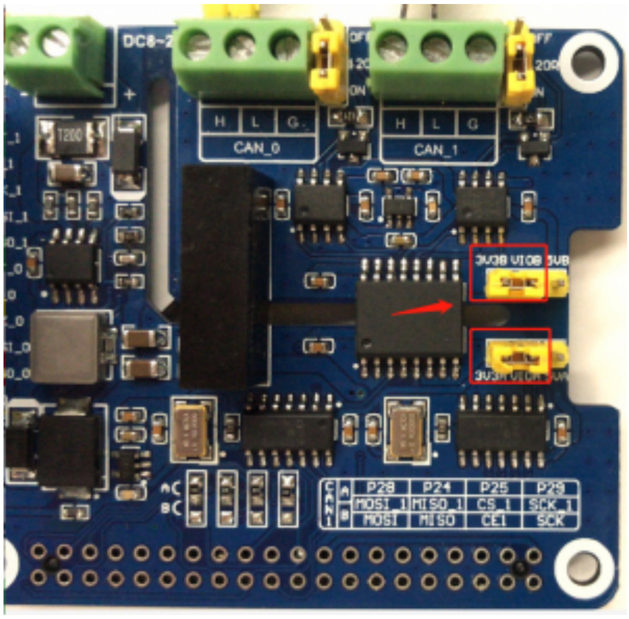


The MCP2561/2FD is a high-speed CAN device,fault-tolerant device that serves as the interface between a CAN protocol controller and the physical bus. The MCP2561/2FD device provides differential transmit and receive capability for the CAN protocol controller, and is fully compatible with the ISO-11898-2 and ISO-11898-5 standards.

The Loop Delay Symmetry is guaranteed to support data rates that are up to 5 Mbps for CAN FD (Flexible Data rate). The maximum propagation delay was improved to support longer bus length. Typically, each node in a CAN system must have a device to convert the digital signals generated by a CAN controller to signals suitable for transmission over the bus cabling (differential output).

Working with Raspberry Pi

The working voltage level of Raspberry Pi is 3.3V, therefore we need to set the VIO of 2-CH CAN FD HAT to 3.3V as below:



Install libraries

bcm2835

Open terminal and run commands below to install bcm2835 library

```
wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.60.tar.gz
tar zxvf bcm2835-1.60.tar.gz
cd bcm2835-1.60/
sudo ./configure
sudo make
sudo make check
sudo make install
```

Install wiringPi library

```
sudo apt-get install wiringpi
cd /tmp
wget https://project-downloads.drogon.net/wiringpi-latest.deb
sudo dpkg -i wiringpi-latest.deb
gpio -v
```

Install python library

python2

```
sudo apt-get update
sudo apt-get install python-pip
sudo apt-get install python-pil
sudo apt-get install python-numpy
sudo pip install RPi.GPIO
sudo pip install spidev
sudo pip2 install python-can
```

python3

```
sudo apt-get update
sudo apt-get install python3-pip
sudo apt-get install python3-pil
sudo apt-get install python3-numpy
sudo pip3 install RPi.GPIO
sudo pip3 install spidev
sudo pip3 install python-can
```

Download demo codes

The demo codes can be found and downloaded on #Resource part.

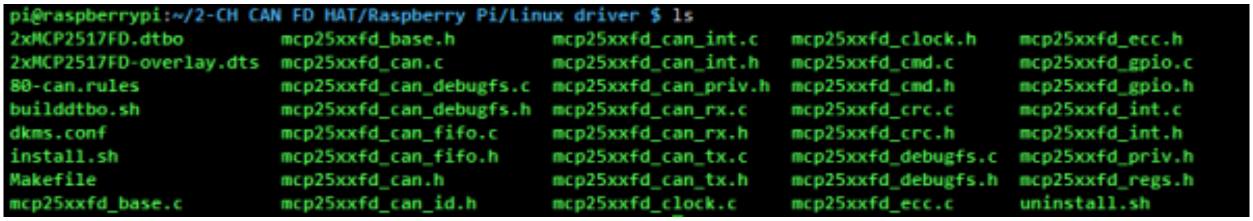
Download and unzip it.

Copy the demo codes to Raspberry Pi.

Install driver

1. Enter the directory of driver

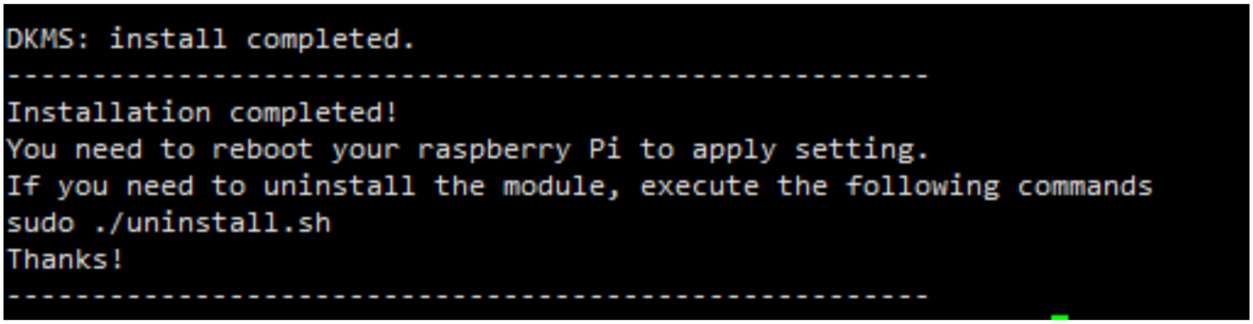
```
cd 2-CH-CAN-FD-HAT-Demo/Raspberry\ Pi/Linux\ driver/
sudo chmod -R 777 A\ mode/
cd A\ mode/
ls
```



```
pi@raspberrypi:~/2-CH CAN FD HAT/Raspberry Pi/Linux driver $ ls
2xMCP2517FD.dtbo          mcp25xxfd_base.h          mcp25xxfd_can_int.c    mcp25xxfd_clock.h      mcp25xxfd_ecc.h
2xMCP2517FD-overlay.dts   mcp25xxfd_can.c           mcp25xxfd_can_int.h    mcp25xxfd_cmd.c        mcp25xxfd_gpio.c
80-can.rules              mcp25xxfd_can_debugfs.c   mcp25xxfd_can_priv.h   mcp25xxfd_cmd.h        mcp25xxfd_gpio.h
builddtbo.sh              mcp25xxfd_can_debugfs.h   mcp25xxfd_can_rx.c     mcp25xxfd_crc.c        mcp25xxfd_int.c
dkms.conf                 mcp25xxfd_can_fifo.c      mcp25xxfd_can_rx.h     mcp25xxfd_crc.h        mcp25xxfd_int.h
install.sh                mcp25xxfd_can_fifo.h      mcp25xxfd_can_tx.c     mcp25xxfd_debugfs.c    mcp25xxfd_priv.h
Makefile                  mcp25xxfd_can.h           mcp25xxfd_can_tx.h     mcp25xxfd_debugfs.h    mcp25xxfd_regs.h
mcp25xxfd_base.c          mcp25xxfd_can_id.h        mcp25xxfd_clock.c      mcp25xxfd_ecc.c        uninstall.sh
```

2.Install driver.

```
sudo ./install.sh
```



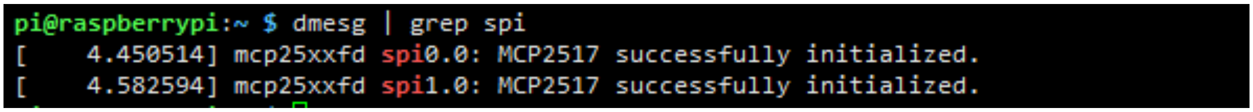
```
DKMS: install completed.
------------------------------------------------------
Installation completed!
You need to reboot your raspberry Pi to apply setting.
If you need to uninstall the module, execute the following commands
sudo ./uninstall.sh
Thanks!
------------------------------------------------------
```

Reboot and check

After installing, reboot your Raspberry Pi and check if the driver is installed properly.

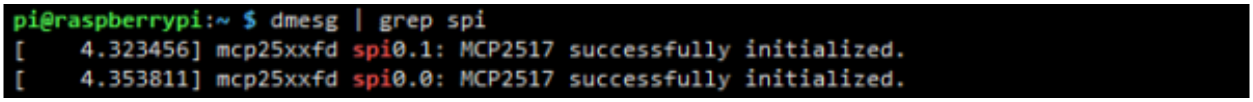
```
sudo reboot
dmesg | grep spi
```

A work mode should be



```
pi@raspberrypi:~ $ dmesg | grep spi
[    4.450514] mcp25xxfd spi0.0: MCP2517 successfully initialized.
[    4.582594] mcp25xxfd spi1.0: MCP2517 successfully initialized.
```

B work mod eshould be



```
pi@raspberrypi:~ $ dmesg | grep spi
[    4.323456] mcp25xxfd spi0.1: MCP2517 successfully initialized.
[    4.353811] mcp25xxfd spi0.0: MCP2517 successfully initialized.
```

【Note】You may get rename prompt when it works in B mode， you can just ignore it.

Set buad rate

You can set the baud rate, working mode, and buffer size.

```
sudo ip link set can1 up type can bitrate 1000000 dbitrate 8000000 restart-ms 1000 berr-reporting on fd on
sudo ifconfig can0 txqueuelen 65536
sudo ifconfig can1 txqueuelen 65536
```

The available modes:

- [loopback { on | off }]
- [listen-only { on | off }]
- [triple-sampling { on | off }]
- [one-shot { on | off }]
- [berr-reporting { on | off }]

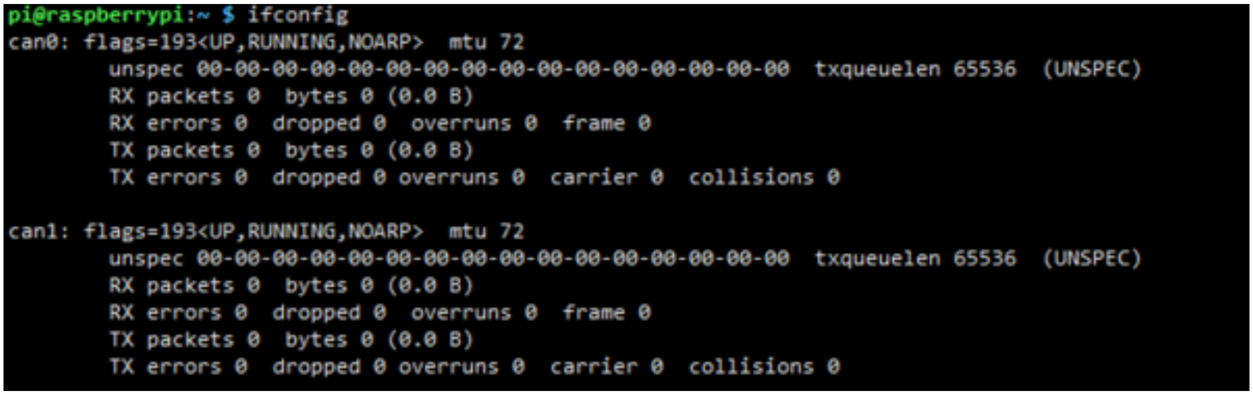
The FD type:

- [fd { on | off }]
- [fd-non-iso { on | off }]

For more details of CAN commands, please refer to :
https://www.kernel.org/doc/Documentation/networking/can.txt

Check the can bus:

```
ifconfig
```

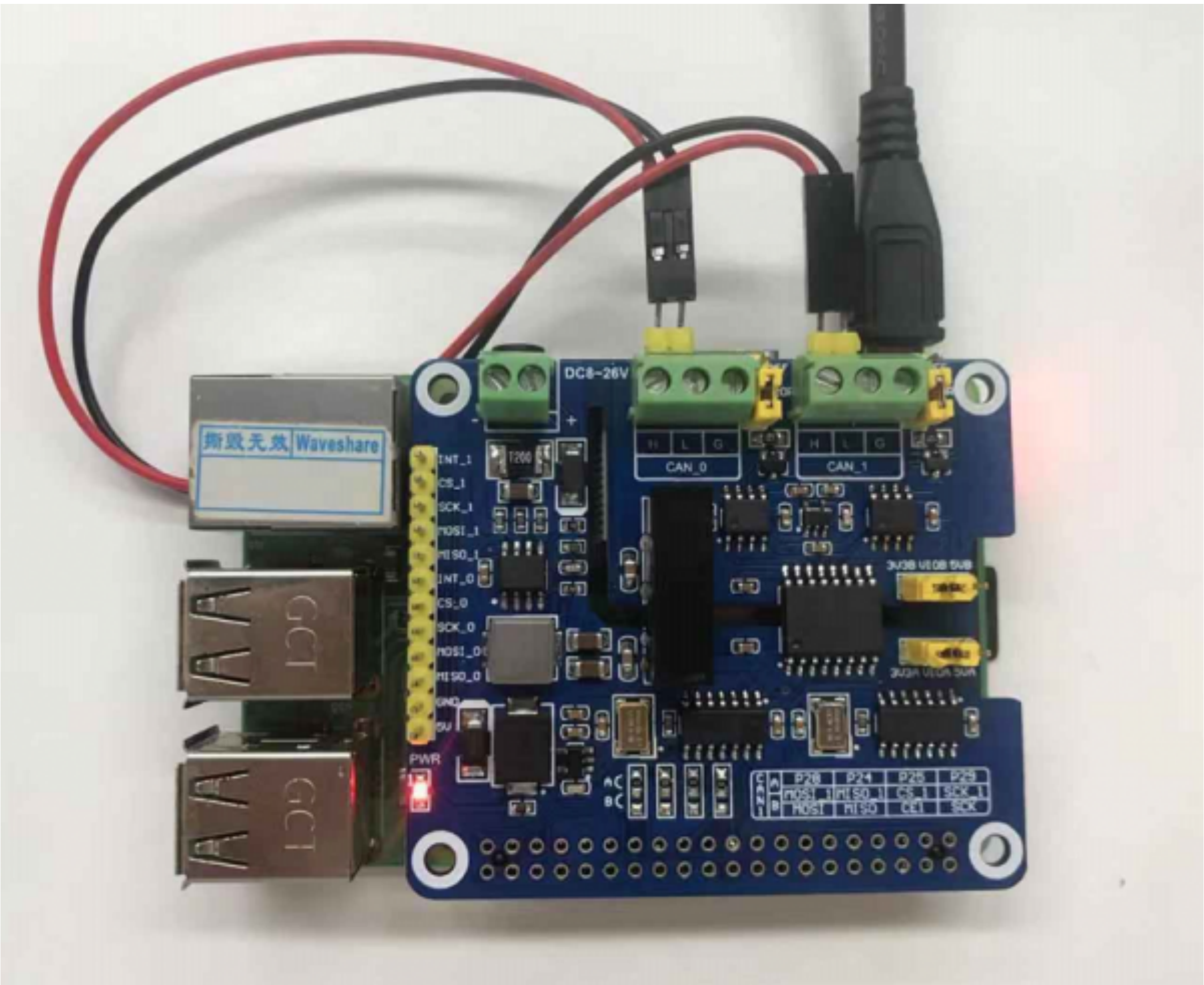


```
pi@raspberrypi:~ $ ifconfig
can0: flags=193<UP,RUNNING,NOARP>  mtu 72
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 65536  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

can1: flags=193<UP,RUNNING,NOARP>  mtu 72
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 65536  (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
```

Testing

If you have only one 2-CH CAN FD HAT, you can connect CAN0_H to CAN1_H, and CAN0_L to CAN0_L to CAN1_L



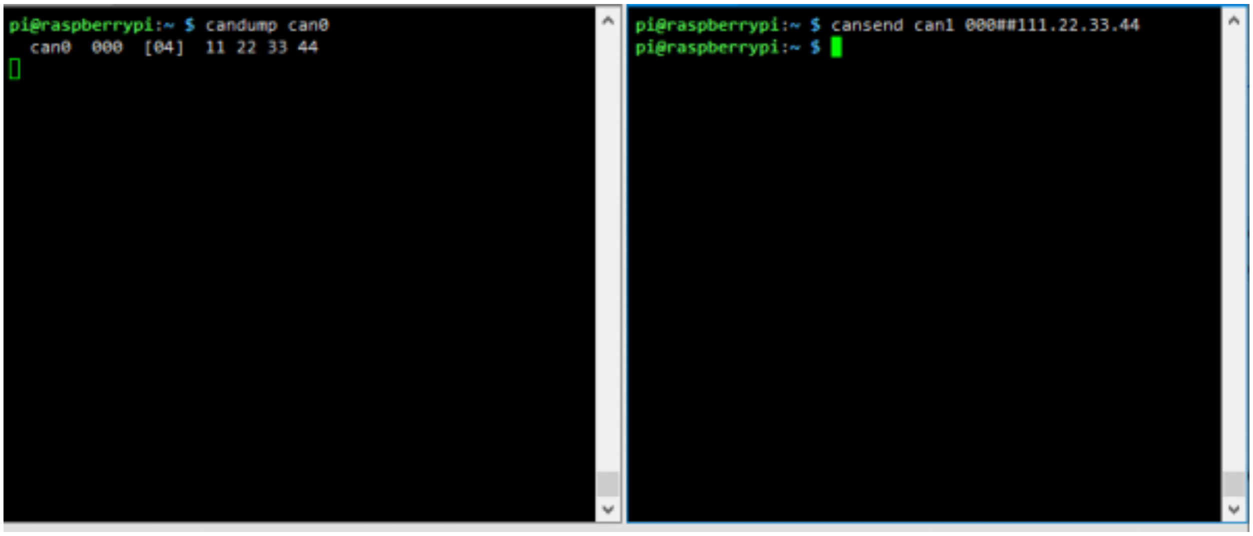
Open two terminals, one works are sender and another is receiver.

receiver:

```
candump can0
```

sender:

```
cansend can1 000##11.22.33.44
```



```
pi@raspberrypi:~ $ candump can0
  can0  000   [04]  11 22 33 44
```

```
pi@raspberrypi:~ $ cansend can1 000##111.22.33.44
pi@raspberrypi:~ $
```

Python example

1. Enter the direcory of python example

```
cd 2-CH CAN FD HAT/Raspberry Pi/python
```

2. Run the receiver.py script in receiver terminal

```
sudo python receive.py
```

3. Run the send.py script in sender terminal

```
sudo python send.py
```

【Note】The CAN1 is used as sender, and CAN0 is used as receiver.

Codes analysis

The demo codes provided is based on python, please check that if you installed the python-can library.

1. Before you send data, you should create a CAN device firstly,

```
os.system('sudo ip link set can0 up type can bitrate 1000000 dbitrate 8000000 restart-ms 1000 berr-reporting on fd on')
```

This code is used to initialize CAN0 as receiver/sender. If you want to change it to CNA1, you can use this one:

```
os.system('sudo ip link set can1 up type can bitrate 1000000 dbitrate 8000000 restart-ms 1000 berr-reporting on fd on')
```

2. Connect to CAN BUS

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

or

```
can1 = can.interface.Bus(channel = 'can1', bustype = 'socketcan_ctypes')# socketcan_native
```

3. Create message

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

4. Send message

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

or

```
can1.send(msg)
```

5. Finally, close CAN device

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

or

```
os.system('sudo ifconfig can1 down')
```

Receive message

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

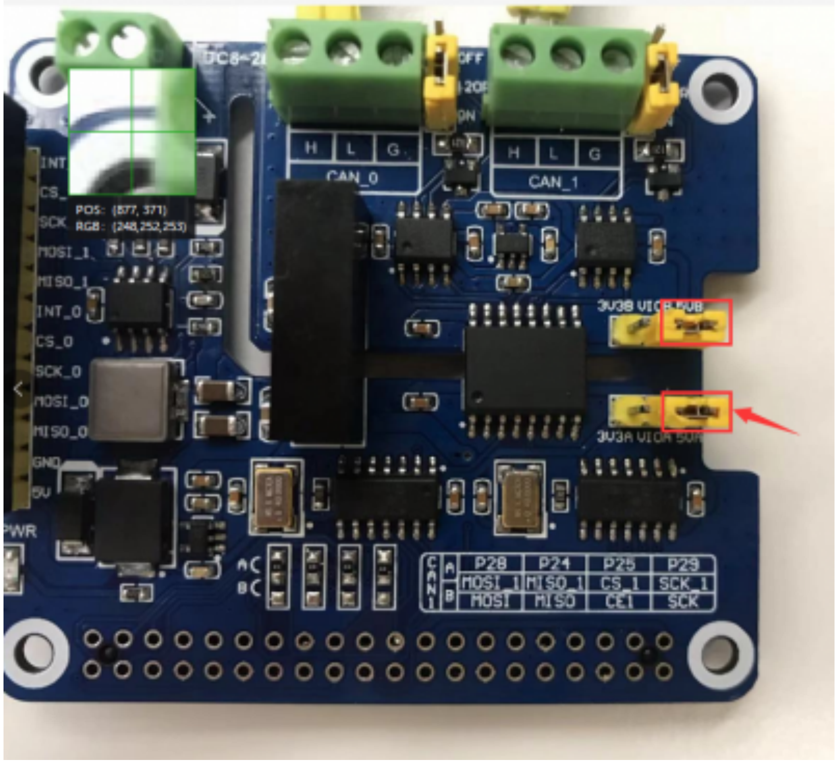
The variables of recv() function is the timeout of receving.

For more information, please refer to https://python-can.readthedocs.io/en/stable/interfaces/socketcan.html

Arduino example

The example of Arduino is based on the MCP2517FD project of Pierre Molinaro (Thanks to Pierre Molinaro).

To run the example, you should prepare two Arduino boards and two 2-CH CAN FD HAT. Note that the working level of most of Arduino board is 5V, therefore we should set the VIO to 5V as below:



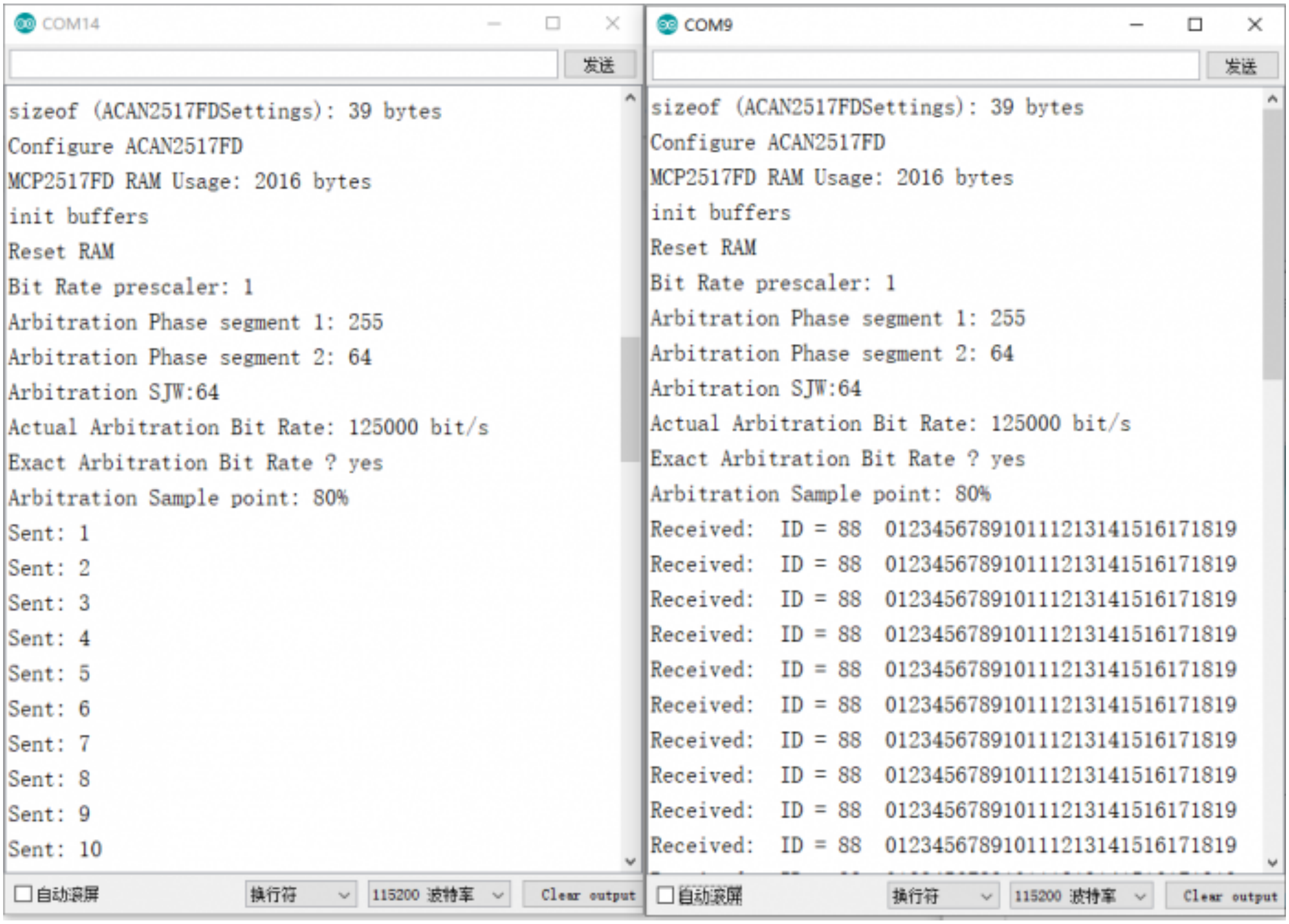
Connection

Connect 2-CH CAN FD HAT to Arduino

PIN	Arduino UNO
5V	5V
GND	GND
MISO_O	D12(MISO)
MOSI_0	D11(MOSI)
SCK_0	D13(SCK)
CS_0	D10

INT_0	D2

Connect CAN0_H and CAN0-L of one HAT to another's. Set the baudrate of Serial monitor to 115200 and check the data:



```
sizeof (ACAN2517FDSettings): 39 bytes
Configure ACAN2517FD
MCP2517FD RAM Usage: 2016 bytes
init buffers
Reset RAM
Bit Rate prescaler: 1
Arbitration Phase segment 1: 255
Arbitration Phase segment 2: 64
Arbitration SJW:64
Actual Arbitration Bit Rate: 125000 bit/s
Exact Arbitration Bit Rate ? yes
Arbitration Sample point: 80%
Sent: 1
Sent: 2
Sent: 3
Sent: 4
Sent: 5
Sent: 6
Sent: 7
Sent: 8
Sent: 9
Sent: 10
```

```
sizeof (ACAN2517FDSettings): 39 bytes
Configure ACAN2517FD
MCP2517FD RAM Usage: 2016 bytes
init buffers
Reset RAM
Bit Rate prescaler: 1
Arbitration Phase segment 1: 255
Arbitration Phase segment 2: 64
Arbitration SJW:64
Actual Arbitration Bit Rate: 125000 bit/s
Exact Arbitration Bit Rate ? yes
Arbitration Sample point: 80%
Received:  ID = 88  012345678910111213141516171819
Received:  ID = 88  012345678910111213141516171819
Received:  ID = 88  012345678910111213141516171819
Received:  ID = 88  012345678910111213141516171819
Received:  ID = 88  012345678910111213141516171819
Received:  ID = 88  012345678910111213141516171819
Received:  ID = 88  012345678910111213141516171819
Received:  ID = 88  012345678910111213141516171819
Received:  ID = 88  012345678910111213141516171819
Received:  ID = 88  012345678910111213141516171819
```

FAQ

Question:

There are two demo codes named as A mode and B mode, which one should I use?

Answer: [Collapse]

The CAN HAT support two modes, in A mode, the two CAN channels use different SPI interfaces, and the two CAN channels use the same SPI in B mode. The hardware is set to A mode by default. If you are the first time receive it and use, please choose the A mode codes.

Question:

Why does CAN initialize failed?

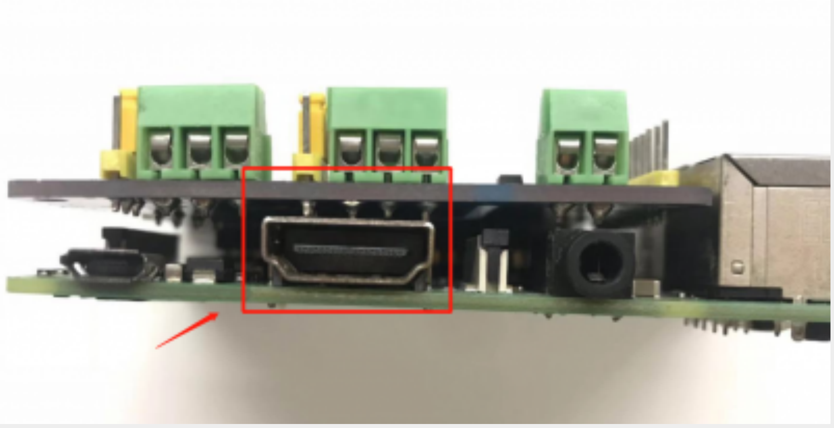
Answer: [Collapse]
Please check if you connect the hardware correctly and restart devices to test again.

Question:
The speed of CAN communication is less than the MAX value expected?

Answer: [Collapse]
The maximum bps is experimental data, the actual speed may be influenced by the quality of cable, software and communicating distance, etc. Therefore, the actual Max speed may less than it.

Question:
The CAN bus is failed to works?

Answer: [Collapse]
If the devices cannot communicate successfully even you initialize devices successfully and set the proper speed. Please check if you connect CANx_H to CANx_H, and CANx_L to CANx_L. Then please check if you set the correct CAN bus according to codes. Sometimes it is caused by short problem (HDMI and CAN interfaces). In this case, you need to higher the CAN HAT by the pin-header provided.



Resources

- Demo code

- Schematic

Datasheet

- MCP2517FD
- MCP2562FD

FAQ

Support

Contact your seller (fast response and most recommended)

or send emails to **service@waveshare.com** (not fast enough but please be patient) for help.

Our working time: 09:00-18:00 (**UTC+8** Monday to Saturday)

Retrieved from "https://www.waveshare.com/w/index.php?title=2-CH_CAN_FD_HAT&oldid=17909"

- This page was last modified on 17 December 2019, at 11:31.
- This page has been accessed 5,163 times.