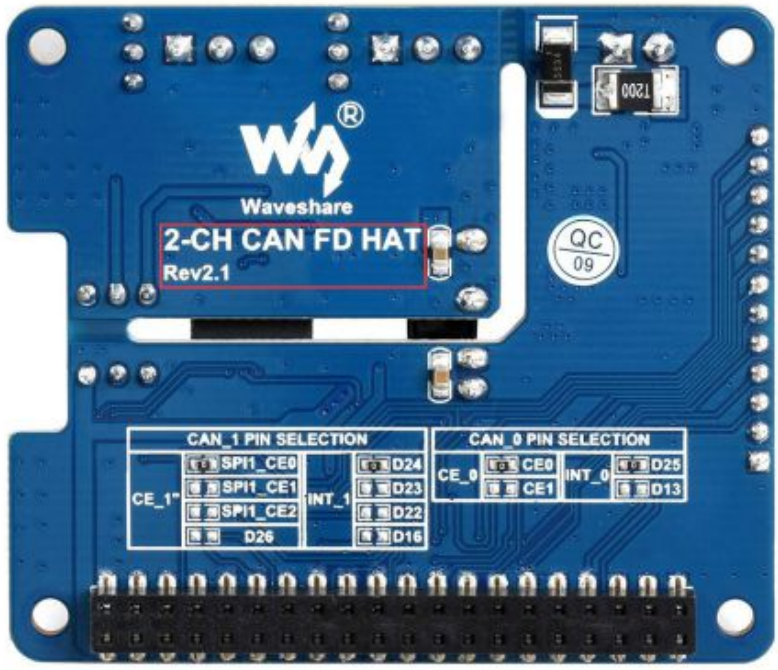
Introduction

Release Notes

2-CH CAN FD HAT, SPI interfaces

Around April 2022 we will launch the 'Rev2.1' version (this version and subsequent versions have a version number under the name of the back panel).



New version features:

1. DC circuit is optimized, the new version can supply power to the Raspberry Pi and work stably from the DC interface.
2. Add stack configuration resistors on the back, and by modifying the resistors, you can achieve up to 5-channel CAN interface expansion (three expansion boards).
3. The default configuration supports the official driver, and the driver installation operation is simpler; at the same time, the configuration resistor is reserved to support the old version driver (the old version uses a third-party driver), which is fully compatible with the old version.

Only the driver installation method is different when using the new version. For details, please refer to the 'Driver Installation' chapter.

- The new version only supports systems with kernel versions 5.4.77 and later.

Product description

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

Features

- Based on Raspberry Pi design, suitable for Raspberry Pi Zero/Zero W/Zero WH/2B/3B/3B+/4B.
- Support external wide voltage of 8~28V to supply power to Raspberry Pi and 2-CH CAN FD HAT at the same time or directly supply power to 2-CH CAN FD HAT from Raspberry Pi.
- Support CAN FD, the data baud rate breaks through the 1Mbps limit of traditional CAN2.0.
- Reserved control interface for easy control by other controllers whose logic voltage is 3.3V/5V.
- CAN bus adopts electrical isolation, the isolation voltage can reach 5 kV.
- Multiple protection devices onboard, with short-circuit protection, electrostatic protection, effective suppression of lightning and ESD, each channel provides 500W lightning surge protection.
- The interface has a 120Ω terminal resistance, which can be switched by jumper caps.
- The working mode of dual-channel CAN can be configured, that is, two channels share a set of SPI or two channels use two independent SPIs respectively.
- Provide complete supporting information manuals and demos.

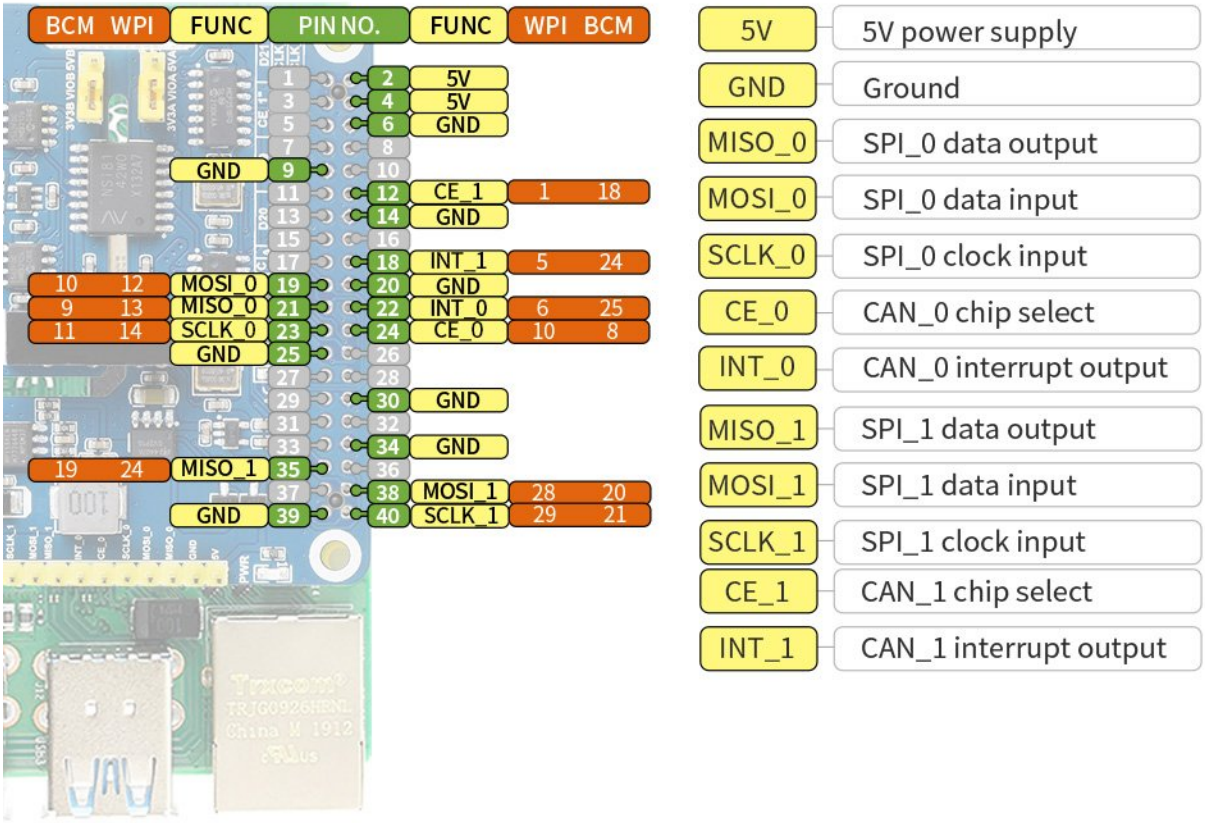
Specifications

- Power input terminal voltage: DC 8~28V
- Logic level: 3.3V/5V
- CAN controller: MCP2518FD
- Dimensions: 65.0 x 56.5 mm
- Fixed hole through diameter: 3.0mm

Interfaces

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)/7(CE1)	10(CE0)/11(CE1)	CAN_0 Chip select
INT_0	25/13	6/23	CAN_0 Interrupt Pin
MISO_1	19(MISO)/9	24(MISO)/13	SPI_1 Data output
MOSI_1	20(MOSI)/10	28(MOSI)/12	SPI_1 Data input
SCK_1	21(SCLK)/11	29(SCLK)/14	SPI_1 Clock input
CS_1	18(SPI1_CE0)/17/16/26	1/0/27/25	CAN_1 Chip select
INT_1	24/23/22/16	5/4/3/27	CAN_1 Interrupt Pin



Hardware description

CAN

The function of the CAN module is to handle the reception and transmission of all messages on the CAN bus. When a message is sent, the message is first loaded into the correct message buffer and control registers. A transmit operation can be initiated by setting the corresponding bit in the control register through the SPI interface or by using the transmit enable pin. Communication status and errors can be checked by reading the corresponding registers. Any message detected on the CAN bus is checked for errors and then matched against a user-defined filter to determine whether to move the message to one of two receive buffers.

Since the Raspberry Pi itself does not support the CAN bus, the CAN controller with the SPI interface is used with a transceiver to complete the CAN function.

The MCP2518FD is a CAN FD (Flexible Data Rate) controller produced by Microchip, which fully supports CAN framing in the Classic (CAN2.0) and CAN Flexible Data Rate (CAN FD) formats. The arbitration bit rate is as high as 1Mbps, the data baud rate also breaks through the 1Mbps limit of traditional CAN2.0, and the SPI clock speed is as high as 20MHz, which conforms to the ISO11898-1:2015 standard. The device can transmit and receive standard and extended data frames as well as remote frames. The MCP2518FD comes with 32 flexible filters and shielding objects that can filter out unwanted packets, thus reducing the overhead of the main microcontroller (MCU). The MCU is connected to the device through the SPI interface, that is, the Raspberry Pi is connected to the chip through the SPI interface. For the Raspberry Pi to use the chip, the device can be driven through the prepared device tree file. For more details, please refer to the data sheet.

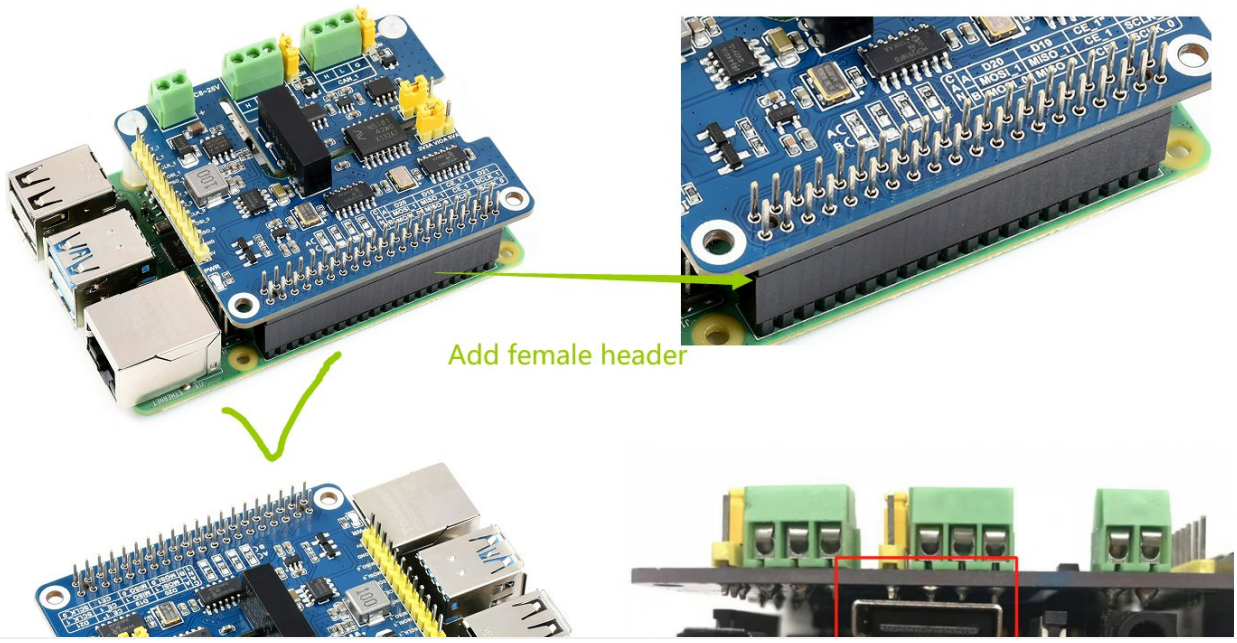
2-CH CAN FD HAT Manual-1.png

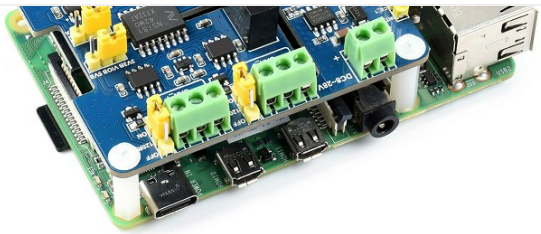
Working with Raspberry Pi

Hardware Connection

When connecting to the Raspberry Pi, a female header must be added, and then the pins pass through the bottom plate. The effect is as follows:

> Note: The pinheader on the expansion board is soldered using lead-free surface mount technology. When plugging or unplugging the pinheader, it is recommended to use tools to pry them open. Make sure to apply even and gentle force on both sides to avoid damaging the solder joints. If you attempt to pry them open directly with your hands, the numerous connections between the header and pins make them generally secure when fully plugged in. However, this method can easily lead to the detachment of the PCBA and pinheader due to insufficient soldering.





PS: It is recommended to use "apt" instead of "apt-get" when using the Bullseye branch system; and the Bullseye branch system only supports Python3.

When using the Raspberry Pi to demonstrate this routine, it should be noted that since the Raspberry Pi belongs to a 3.3V logic system, it is necessary to change the jumper cap of the logic voltage pin of the 2-CH CAN FD HAT to 3.3V, as shown in the figure below.

2-CH CAN FD HAT Manual-3.png

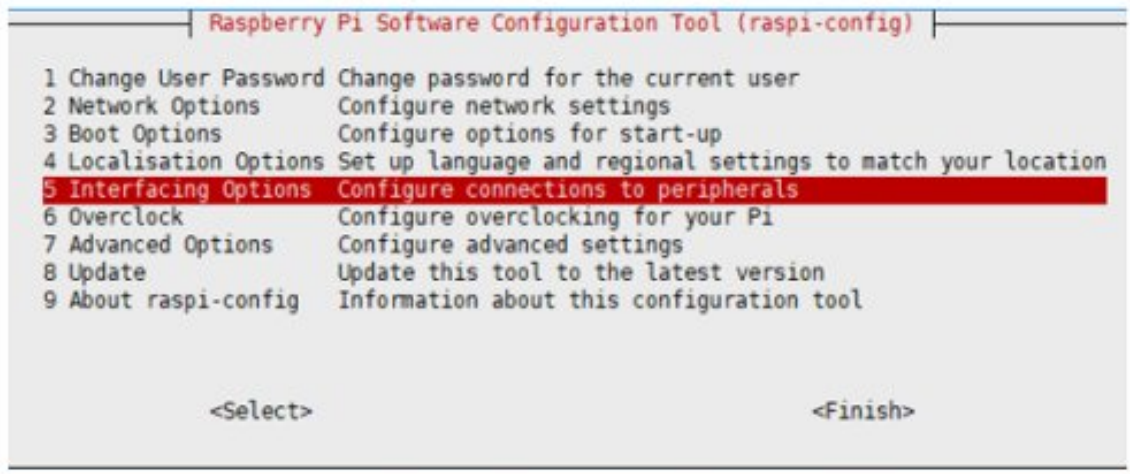
Enable I2C Interface

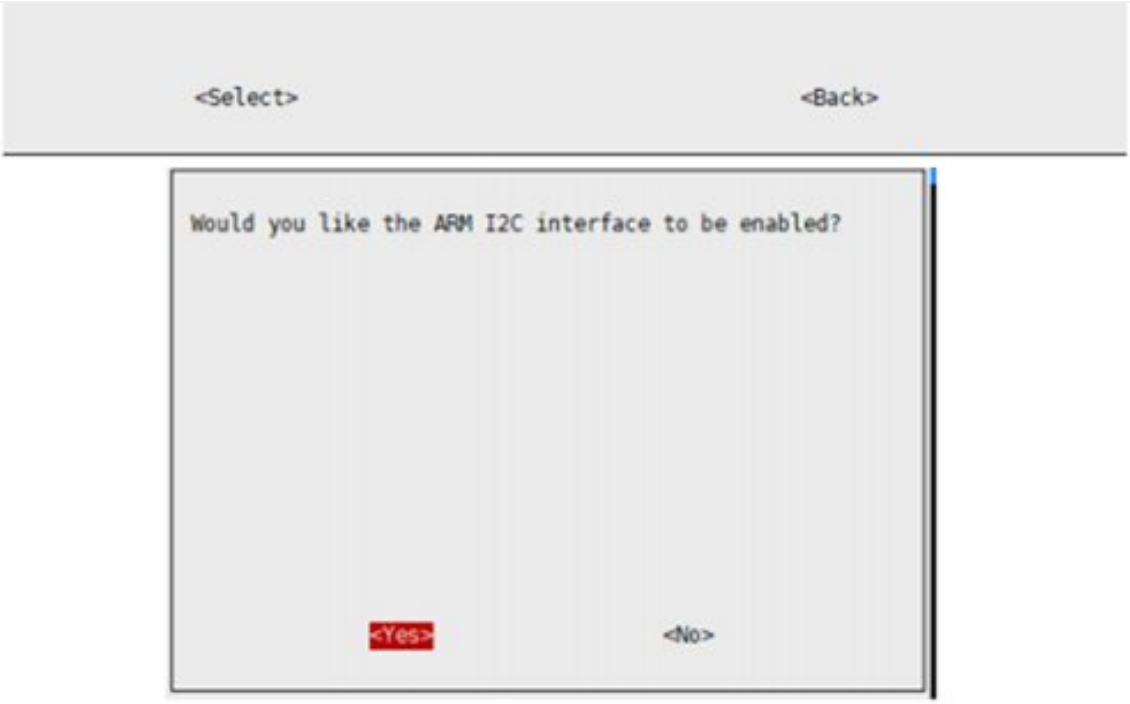
Open a terminal and run the following commands:

```
sudo raspi-config
Choose Interfacing Options -> I2C -> Yes.
```

Reboot Raspberry Pi:

```
sudo reboot
```





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
# For more information, please refer to the official website: http://www.airspayce.c
om/mikem/bcm2835/
```

Install wiringPi library

```
sudo apt-get install wiringpi
#For Raspberry Pi 4B may need to be upgraded:
wget https://project-downloads.drogon.net/wiringpi-latest.deb
sudo dpkg -i wiringpi-latest.deb
gpio -v
# Run gpio -v and version 2.52 will appear. If it does not appear, the installation
is wrong
```

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
```

Driver installation (new version)

Note: Around April 2022 we have released the 'Rev2.1' version (with the version number under the name on the back panel), this chapter is only applicable to 'Rev2.1' and For future versions, if you are using an older version, please read the 'Driver Installation (Legacy)' chapter.

You can flexibly adjust the SPI and interrupt pins used by modifying the 0R resistor, **select any of the following modes.**

Dual SPI Mode

'A' mode (default mode), use SPI0-0 and SPI1-0 to control two CAN channels, **hardware does not need any modification**, namely:
CAN_0 uses SPI0-0, interrupt pin is 25, CAN_1 uses SPI1-0, interrupt pin is 24

Insert the module to Raspberry Pi, and then modify config.txt file:

```
sudo nano /boot/config.txt
```

⚠ The above configuration is based on the **Raspbian OS** system. If it is used in the **Ubuntu** system, the config.txt path is generally "/boot/firmware/config.txt "

Add the following commands at the last line:

```
dtparam=spi=on
dtoverlay=spi1-3cs
dtoverlay=mcp251xfd,spi0-0,interrupt=25
dtoverlay=mcp251xfd,spi1-0,interrupt=24
```

After the addition is complete, restart and enter the 'Configure CAN' chapter.

Single SPI Mode

'B' mode, use SPI0-0 and SPI0-1 to control two CAN outputs, **just move the four 0R resistors on the front of the module to the 'B' mode position**, namely:
CAN_0 uses SPI0-0, interrupt pin is 25, CAN_1 uses SPI0-1, interrupt pin is 24, please add in /boot/config.txt:

```
dtoverlay=spi1-3cs
dtoverlay=mcp251xfd,spi0-0,interrupt=25
dtoverlay=mcp251xfd,spi0-1,interrupt=24
```

After the addition is complete, restart and enter the 'Configure CAN' chapter.

Stack Mode

- Principle: Modify the 0R resistor in the CAN_x PIN SELECTION (x is 0 or 1) area on the back to change the CE and INT pins used by the CAN controller.
- **Note: When stacking is required, the hardware configuration needs to be kept in dual SPI mode ('A' mode, default setting).**
- CAN_0 has two groups to choose from:

CE_0	INT_0	config settings	description
CE0	D25	dtoverlay=mcp251xfd,spi0-0,interrupt=25	CAN_1 uses SPI0-0, the interrupt pin is 25
CE1	D13	dtoverlay=mcp251xfd,spi0-1,interrupt=13	CAN_1 uses SPI0-1, interrupt pin 13

- CAN_1 has three groups to choose from:

CE_1"	INT_1	config settings	description
SPI1_CE0	D24	dtoverlay=mcp251xfd,spi1-0,interrupt=24	CAN_1 uses SPI1-0, the interrupt pin is 24
SPI1_CE1	D23	dtoverlay=mcp251xfd,spi1-1,interrupt=23	CAN_1 uses SPI1-1, the interrupt pin is 23
SPI1_CE2	D22	dtoverlay=mcp251xfd,spi1-2,interrupt=22	CAN_1 uses SPI1-2, the interrupt pin is 22

A careful friend may see that there is still a set of configurations that have not been written. This is reserved for compatibility with old versions, and new users can ignore it.

Example of use

- If you have **two boards, how to configure them when using 4-way CAN?**

Answer: The first board is not modified, the CAN_0 configuration area on the back of

the second board selects CE1 and D13, and the CAN_1 configuration area selects SPI1_CE1 and D23 and then adds at the end of /boot/config.txt:

```
dtoverlay=spi1-3cs
dtoverlay=mcp251xfd,spi0-0,interrupt=25
dtoverlay=mcp251xfd,spi0-1,interrupt=13
dtoverlay=mcp251xfd,spi1-0,interrupt=24
dtoverlay=mcp251xfd,spi1-1,interrupt=23
```

After the addition is complete, restart and enter the 'Configure CAN' chapter.

Compatibility Mode

- The new version has adjusted some default settings in order to be compatible with the standard SPI, so some modifications need to be made to be compatible with the old version
    - Compatible with A mode: CE_1" resistor select to D26; INT_1 resistor select to D16
    - Compatible with B mode: CE_1" can not be modified, it has no effect; INT_1 resistor is selected to D16
- For driver installation, please refer to "Driver Installation (Old Version) Chapter"

Driver Installation (Legacy)

'A' mode (default mode), use SPI0-0 and SPI1-0 to control two CAN channels, **hardware does not need any modification**, namely:

Insert the module to Raspberry Pi, and then modify config.txt file:

```
sudo nano /boot/config.txt
```

⚠ The above configuration is based on the **Raspbian OS** system. If it is used in the **Ubuntu** system, the config.txt path is generally "/boot/firmware/config.txt "

```
dtparam=spi=on
dtoverlay=waveshare-can-fd-hat-mode-a
```

After the addition is complete, restart and enter the 'Configure CAN' chapter.

- Reboot and check

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

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

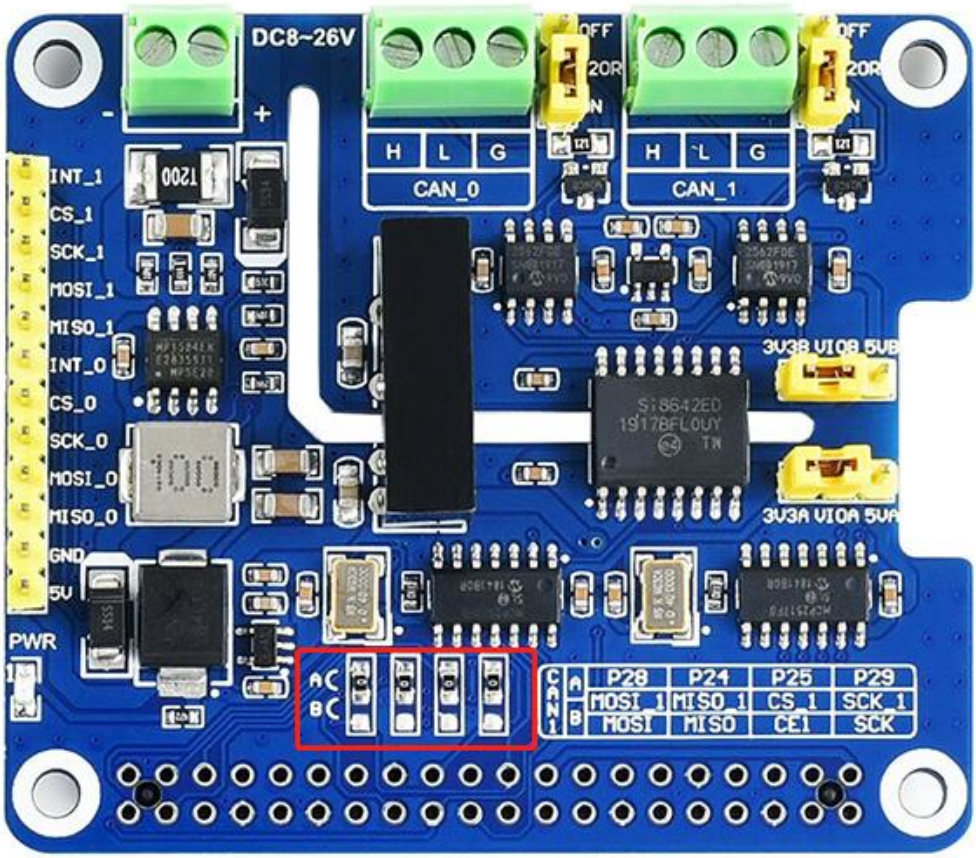
A work mode should be

```
pi@raspberrypi:~ $ dmesg | grep spi
[    4.722088] mcp25xxfd spi0.0 can0: MCP2518FD rev0.0 (-RX_INT -MAB_NO_WARN +CRC_REG +CRC_RX +CRC_TX +ECC -HD m:20.00MHz r:18.50MHz
 e:0.00MHz) successfully initialized.
[    4.765608] mcp25xxfd spi0.1 (unnamed net_device) (uninitialized): Failed to detect MCP25xxFD (osc=0x00000000).
[    4.858634] mcp25xxfd spi1.0 (unnamed net_device) (uninitialized): Detected MCP2518FD, but firmware specifies a MCP2517FD. Fixing
 up.
[    4.859335] mcp25xxfd spi1.0 can1: MCP2518FD rev0.0 (-RX_INT -MAB_NO_WARN +CRC_REG +CRC_RX +CRC_TX +ECC -HD m:20.00MHz r:18.50MHz
 e:0.00MHz) successfully initialized.
```

B work mod should be

2-CH CAN FD HAT Manual-7.png

【Note1】Mode A is the factory default, and mode B CAN be realized by changing the resistance. In mode A, two CAN channel use two sets of independent SPI respectively, while in mode B, two CAN channel share one set of SPI. See the figure below.



【Note2】Because the compatible detection method is adopted, there will be additional information during the initialization, which does not affect the normal use and can be ignored.

Note: If after installing the driver according to the above method, it is prompted that the kernel reports an error, you need to download the latest version of the image from the official Raspberry Pi, or directly update the system kernel:

```
sudo apt update
```

```
sudo apt upgrade
uname -a
```

Configure CAN

Set the baud rate, operating mode, whether to enable FD, and configure the transmission buffer size:

- Baud rate setting:

```
sudo ip link set can0 up type can bitrate 1000000 dbitrate 8000000 restart-ms 1000 b
err-reporting on fd on
sudo ip link set can1 up type can bitrate 1000000 dbitrate 8000000 restart-ms 1000 b
err-reporting on fd on
```

bitrate xxxxxx (bps)：arbitration bitrate
dbitrate xxxxxx (bps)：data bitrate
The following is an example of setting the bit rate

```
Example configuring 500 kbit/s arbitration bitrate and 4 Mbit/s data bitrate:

  $ ip link set can0 up type can bitrate 500000 sample-point 0.75 \
                                 dbitrate 4000000 dsample-point 0.8 fd on
  $ ip -details link show can0
  5: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 72 qdisc pfifo_fast state UNKNOWN \
           mode DEFAULT group default qlen 10
  link/can  promiscuity 0
  can <FD> state ERROR-ACTIVE (berr-counter tx 0 rx 0) restart-ms 0
        bitrate 500000 sample-point 0.750
        tq 50 prop-seg 14 phase-seg1 15 phase-seg2 10 sjw 1
        pcan_usb_pro_fd: tseg1 1..64 tseg2 1..16 sjw 1..16 brp 1..1024 \
        brp-inc 1
        dbitrate 4000000 dsample-point 0.800
        dtq 12 dprop-seg 7 dphase-seg1 8 dphase-seg2 4 dsjw 1
        pcan_usb_pro_fd: dtseg1 1..16 dtseg2 1..8 dsjw 1..4 dbrp 1..1024 \
        dbrp-inc 1
        clock 80000000
```

If the CAN FD frame data sent by other devices cannot be received, you can try to add sample-point .8 dsample-point .8 at the end of the command. Note: the sampling point setting (.8) needs to be the same as that of the sending terminal.

- Can be configured into the following modes:

  [loopback { on | off }]

  [listen-only { on | off }]

  [triple-sampling { on | off }]

  [one-shot { on | off }]

  [berr-reporting { on | off }]

  FD is the opening command:

  [fd { on | off }]

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

More related CAN kernel commands can be viewed:
https://www.kernel.org/doc/Documentation/networking/can.txt

- Configuration buffer

```
sudo ifconfig can0 txqueuelen 65536
sudo ifconfig can1 txqueuelen 65536
```

- View ifconfig:

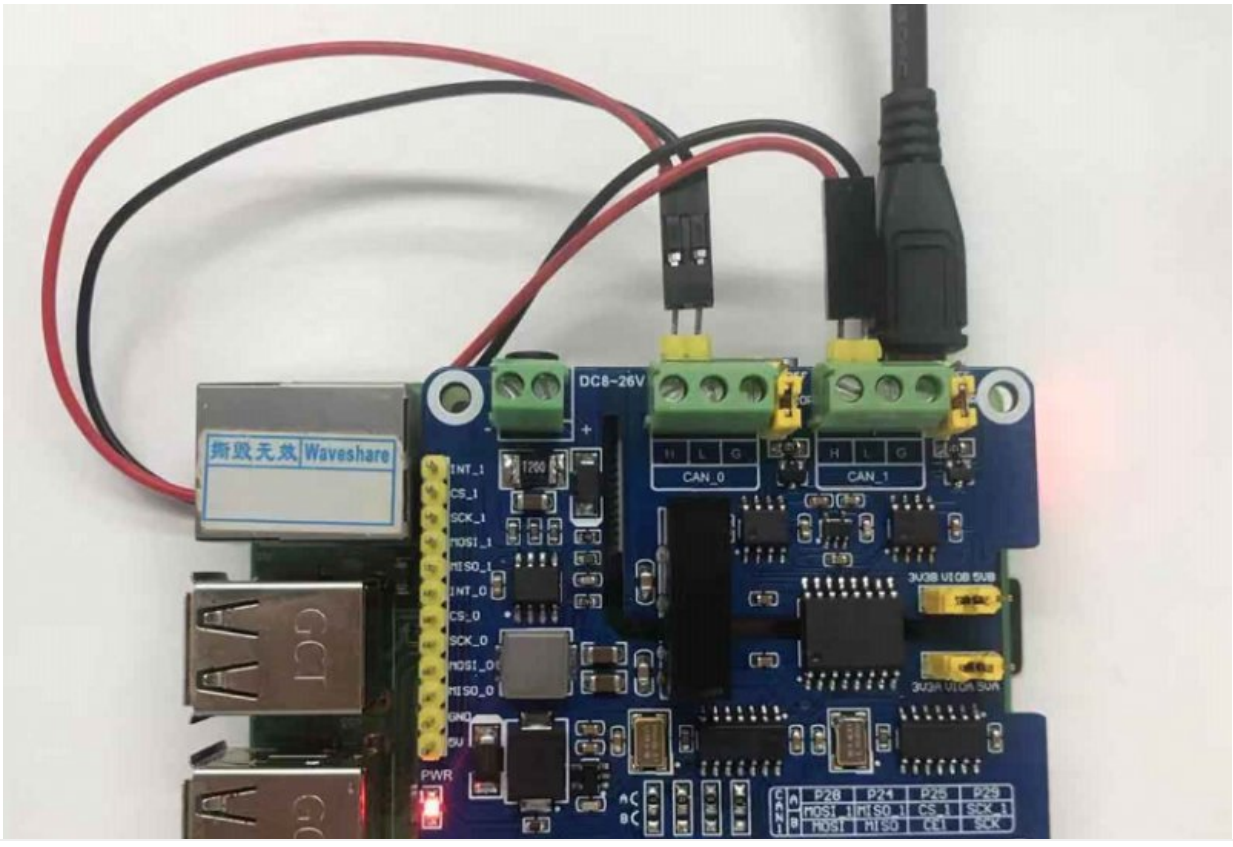
```
ifconfig
```

```
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
```

- start testing:

If you only have one 2-CH CAN FD HAT, you can connect CAN0_H to CAN1_H, CAN0_L to CAN1_L of the module, as shown in the figure below:



- Install can-utils:

```
sudo apt-get install can-utils
```

- Open two terminal windows:

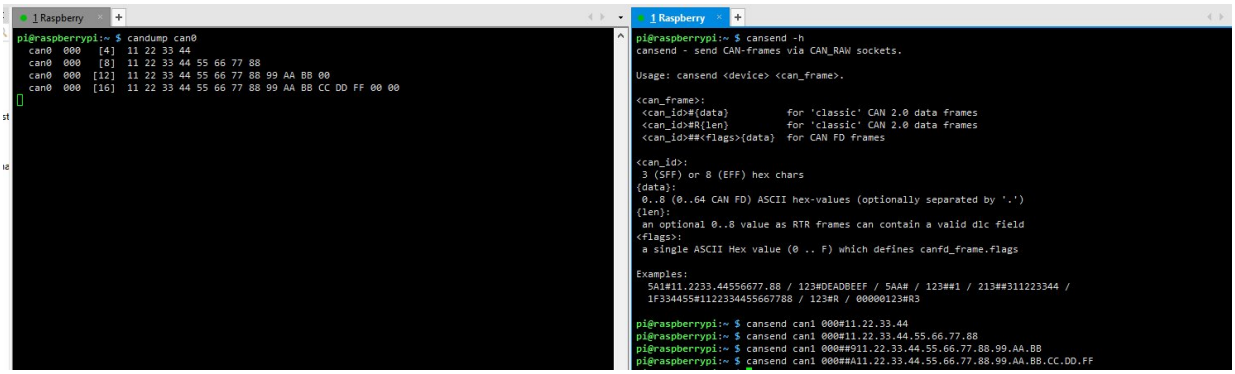
One of the terminal inputs receives the CAN0 data command:

```
candump can0
```

Another terminal input sends CAN1 data command:

```
cansend can1 000#11.22.33.44
```

- The demonstration effect is as follows: (the left is receiving, the right is sending)



If you have two 2-CH CAN FD HATs, you can directly connect CAN_H and CAN_L two by two. The effect is the same as above, but pay attention to matching the communication rate, identifying the ID, and outputting the serial number of the interface.

Fixed Device Number

If it is found that the system device number of the CAN interface and the name of the board interface do not match, you can use the following methods to fix the device number:

- Create a rules file in the /etc/udev/rules.d directory:

```
sudo nano /etc/udev/rules.d/80-can.rules
```

- Add the following to the file:

```
ACTION=="add", SUBSYSTEM=="net", DEVPATH=="/devices/platform/soc/*/spi0.0/net/can?",
NAME="can0"
ACTION=="add", SUBSYSTEM=="net", DEVPATH=="/devices/platform/soc/*/spi0.1/net/can?",
NAME="can1"
ACTION=="add", SUBSYSTEM=="net", DEVPATH=="/devices/platform/soc/*/spi1.0/net/can?",
NAME="can1"
```

- If it is used in stacking, you can add corresponding commands according to your own needs, you only need to modify DEVPATH and NAME, for example, name spi1-1 can3:

```
ACTION=="add", SUBSYSTEM=="net", DEVPATH=="/devices/platform/soc/*/spi1.1/net/can?",
NAME="can3"
```

- More information please refer to: https://wiki.archlinux.org/title/Udev#top-page

Python examples

- Download and decompress the demos and drivers (downloaded can be skipped):

```
cd ~
wget https://files.waveshare.com/upload/4/46/2-CH-CAN-FD-HAT-Demo.7z
7z x 2-CH-CAN-FD-HAT-Demo.7z -o./2-CH-CAN-FD-HAT-Demo
```

- Enter the program directory:

```
cd 2-CH-CAN-FD-HAT-Demo/Raspberry_Pi/Python/
```

- The receiver runs receive0.py:

```
sudo python receive0.py
```

- The sender runs send1.py:

```
sudo python send1.py
```

It should be noted that the sender here is sent by CAN1, and the receiver is CAN0. For specific modifications, please refer to the code analysis below.

Code Analysis

This demo is based on the python platform, make sure that the python-can library is installed and a can device must be created before sending because the MCP2518FD

kernel is only enabled in the front:

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

The above one is initialized and enabled through the configuration of CAN0, and CAN0 is designated as the transmit/receive interface. If you need to change to CAN1, the code is as follows:

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

- First step: connect to the CAN bus

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

As the python-can version is upgraded to 4.0.0, the corresponding code changes are as follows, otherwise an error will be reported:

```
can1 = can.interface.Bus(channel = 'can0', bustype = 'socketcan')
```

- If it needs to be changed to CAN1, the code is as follows:

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

- Step 2: Create a message

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

As the python-can version is upgraded to 4.0.0, the corresponding code changes are as follows, otherwise an error will be reported

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

- Step 3: Send the message

```
can0.send(msg)<br />
```

- If it needs to be changed to CAN1, the code is as follows:

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

- Finally, also close the can device

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

- To be changed to CAN1, the code is as follows:

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

- Receive data:

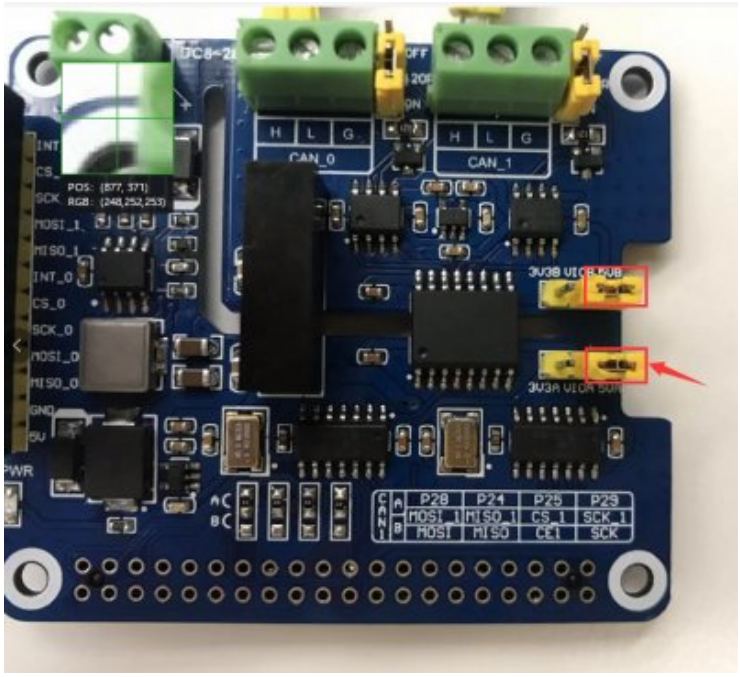
```
msg = can0.recv(10.0)
```

The timeout reception time is defined in recv().
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 MCP2518FD 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:

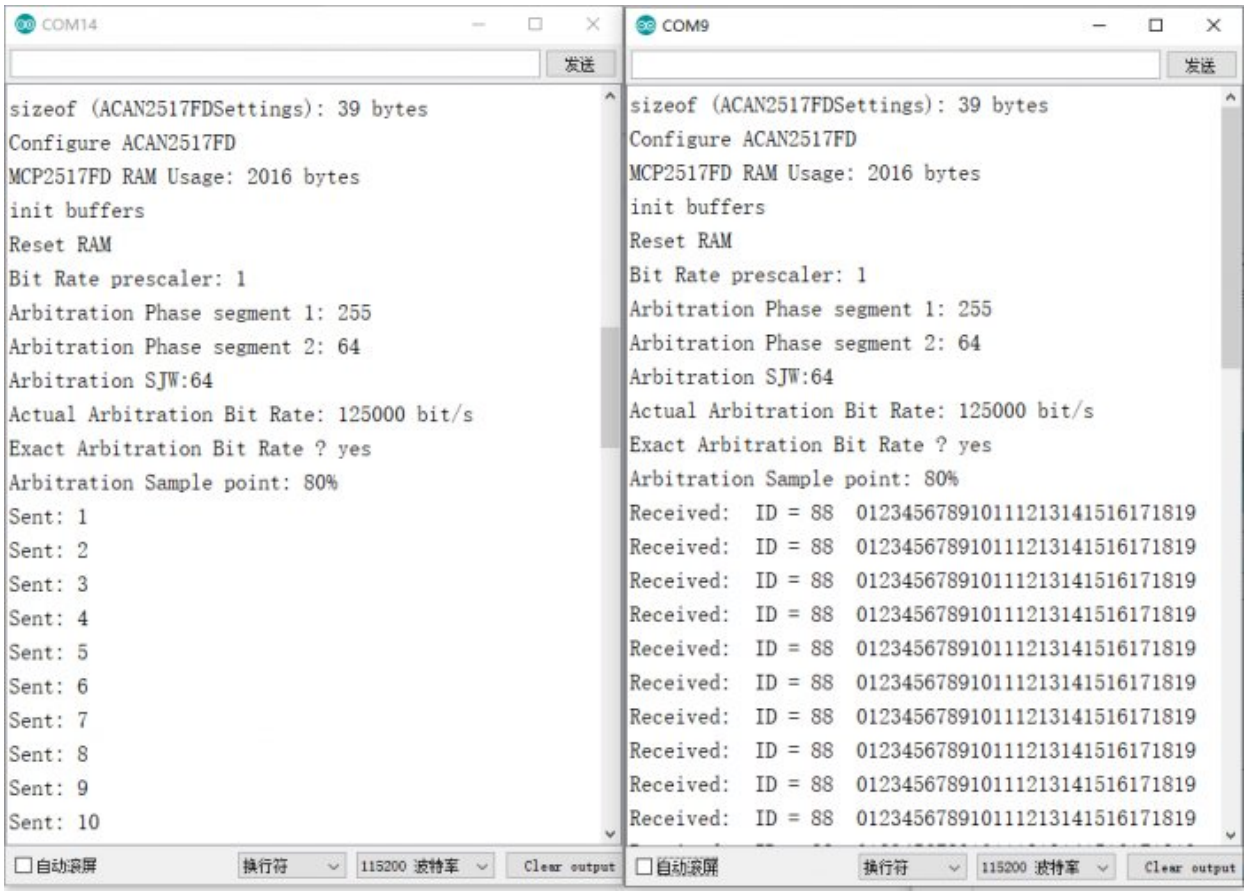


Connection

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 the H and L of CAN_0 of the two 2-CH CAN FD HATs, set the baud rate to 115200, open the two serial monitors, and you can see the following results (the number of transmissions on the left, and the received data on the right) :



FAQ

Question: Can I attach the 2-CH CAN FD HAT with 2-CH CAN HAT?

Answer:
Attaching with 2-CH CAN HAT is not supported, but 2x 2-CH CAN FD can be attached.

Question: Every time I turn it on, I find that the order of CAN0 and CAN1 is random, how to fix it?

Answer:
You can add the 80-can.rules file under /etc/udev/rules.d/ and add the code:

```
ACTION=="add", SUBSYSTEM=="net", DEVPATH=="/devices/platform/soc/*/spi0.0/net/ca
n?", NAME="can0"
ACTION=="add", SUBSYSTEM=="net", DEVPATH=="/devices/platform/soc/*/spi0.1/net/ca
n?", NAME="can1"
ACTION=="add", SUBSYSTEM=="net", DEVPATH=="/devices/platform/soc/*/spi1.0/net/ca
n?", NAME="can1"
```

Question: What is the maximum baudrate of CAN FD?

Answer:
The theoretical baud rate limit of CAN FD is 8Mbps, but in the actual application, the baud rate will be affected by the interference of the communication environment and the communication distance, and other factors, the actual environment is subject to the actual measurement.

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

Answer:
The CAN HAT support two modes, in A mode, the two CAN channel use different SPI interfaces, and the two CAN channel 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 it, please choose the A mode codes.

Question: Is the G pin of the CAN interface connected?

Answer:
G is the signal ground, also known as the isolation ground. In the industrial environment, CAN is subject to a changeable environment. To ensure stability, the G of the two communication modules needs to be connected.

Question: Why does CAN initialize failed?

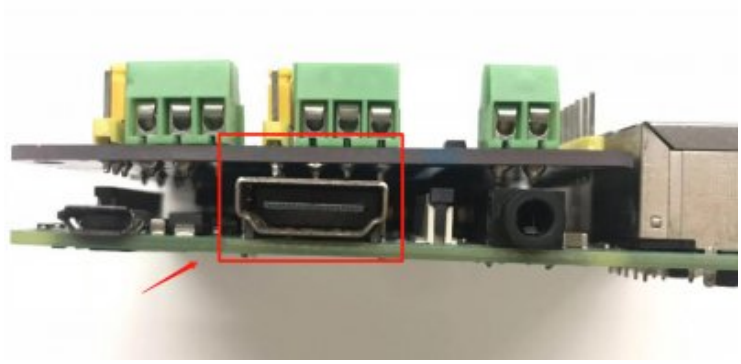
Answer:
Please check if you connect the hardware correctly and restart the devices to test again.

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

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

Question: The CAN bus failed to work?

Answer:
If the initialization is successful and the speed is set properly, it still cannot communicate normally. Please check whether the CAN_H of the next two 2-CH CAN FD HATs is connected to another CAN_H and whether CAN_L is connected to another CAN_L. It should be noted that these two lines cannot be reversed. H corresponds to H, and L corresponds to L. If there is no problem, please check whether the CAN port configured by the program is correct. For example, CAN0 is required, but the program code configures CAN1. At the same time, it is also necessary to check whether the hardware is connected correctly and whether there is a short circuit in the CAN interface. As shown in the figure below, the CAN interface and the HDMI interface of the Raspberry Pi are accidentally connected to cause a short circuit, which makes the CAN unable to communicate normally. At this time, it is necessary to purchase a 2-CH The accessories 2×20PIN headers delivered during CAN FD HAT are increased in height.



Question: Can multiple devices be connected to the CAN interface?

Answer:
In theory, as long as the CAN IDs are different, multiple CAN devices can be connected for communication. However, the specific number of connections is limited by various

factors such as the communication volume and electrical load of the CAN sub-devices, and needs to be tested according to the actual application. Assuming that the sub-devices communicate with a large amount of data and have high real-time requirements, it may not be possible to carry 2 to 3 devices at the same time.

Resources

- Demo code
- Schematic (Old version)
- Schematic (Rev2.1 version)

Datasheet

- MCP2518FD

Support

Technical Support

If you need technical support or have any feedback/review, please click the **Submit Now** button to submit a ticket, Our support team will check and reply to you within 1 to 2 working days. Please be patient as we make every effort to help you to resolve the issue.
Working Time: 9 AM - 6 AM GMT+8 (Monday to Friday)

Submit Now