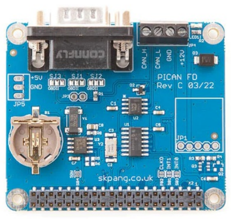
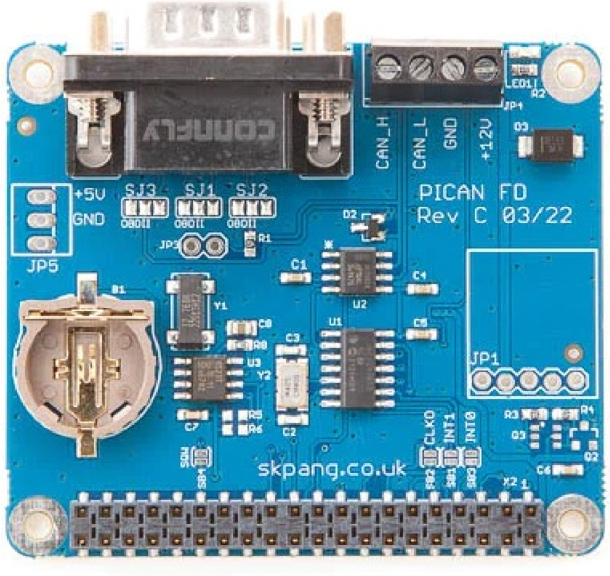


SK Pang electronics PiCAN FD with RTC User Guide

Contents [hide

SK Pang

SK Pang electronics PiCAN FD with RTC



Product name PICAN FD CAN-Bus Board for Raspberry Pi 3/4
Model number RSP-PICAN FD
Manufacturer SK Pang Electronics Ltd

Introduction

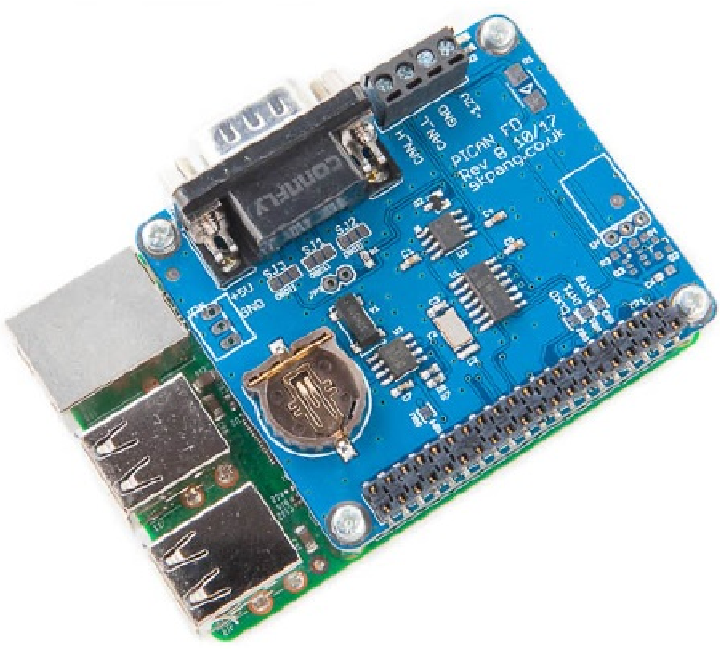
This PiCAN FD board provide CAN-Bus FD capability for the Raspberry Pi 3. It uses the Microchip MCP2517FD CAN FD controller with MCP2562FD CAN transceiver. Connections are made via DB9 or 4 way screw terminal. This board is also available with a 5v 1A SMPS that can power the Pi is well via the screw terminal or DB9 connector. A real time clock with battery back up (battery not included) is also on the board.
Easy to install SocketCAN driver. Programming can be done in C or Python.

Features

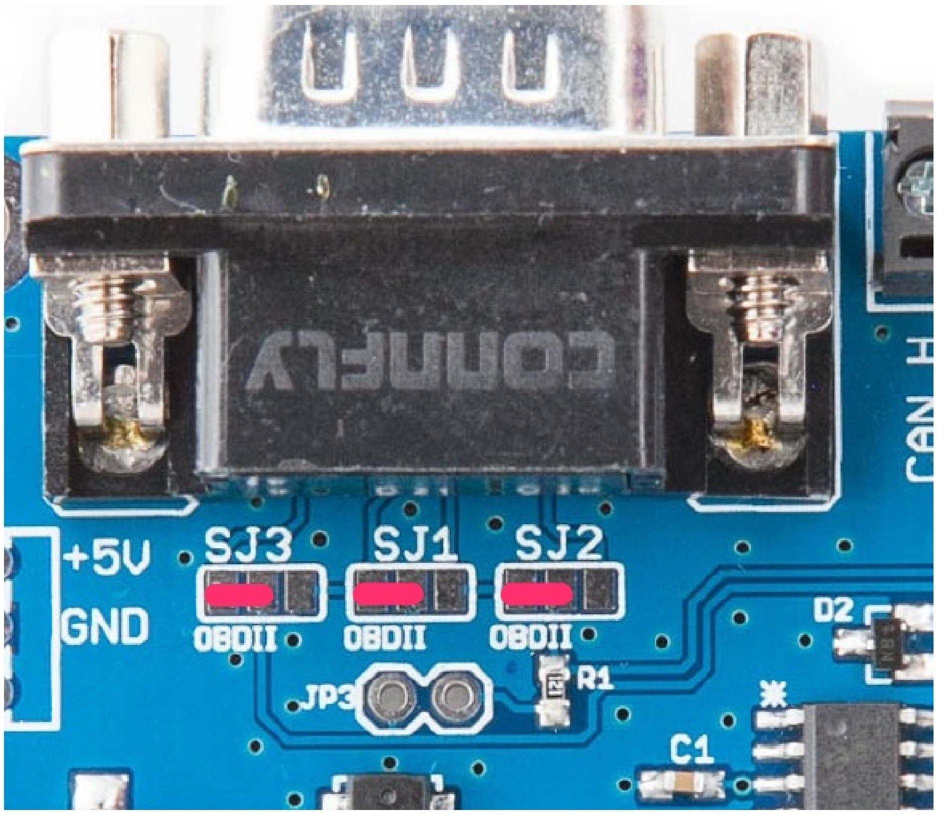
- Arbitration Bit Rate upto 1Mbps
- Data Bit Rate up to 8Mbps
- CAN FD Controller modes
- Mixed CAN2.0B and CANFD mode
- CAN2.0B mode
- Conforms to ISO11898-1:2015
- High speed SPI Interface
- CAN connection via standard 9-way sub-D connector or screw terminal
- Compatible with OBDII cable
- Solder bridge to set different configuration for DB9 connector
- 120Ω terminator ready
- Serial LCD ready
- LED indicator
- Four fixing holes, comply with Pi Hat standard
- SocketCAN driver, appears as can0 to application
- Interrupt RX on GPIO25
- RTC with battery backup (battery not included)

Hardware Installation

1. Before installing the board make sure the Raspberry is switched off. Carefully align the 40way connector on top of the Pi. Use spacer and screw (optional items) to secure the board.



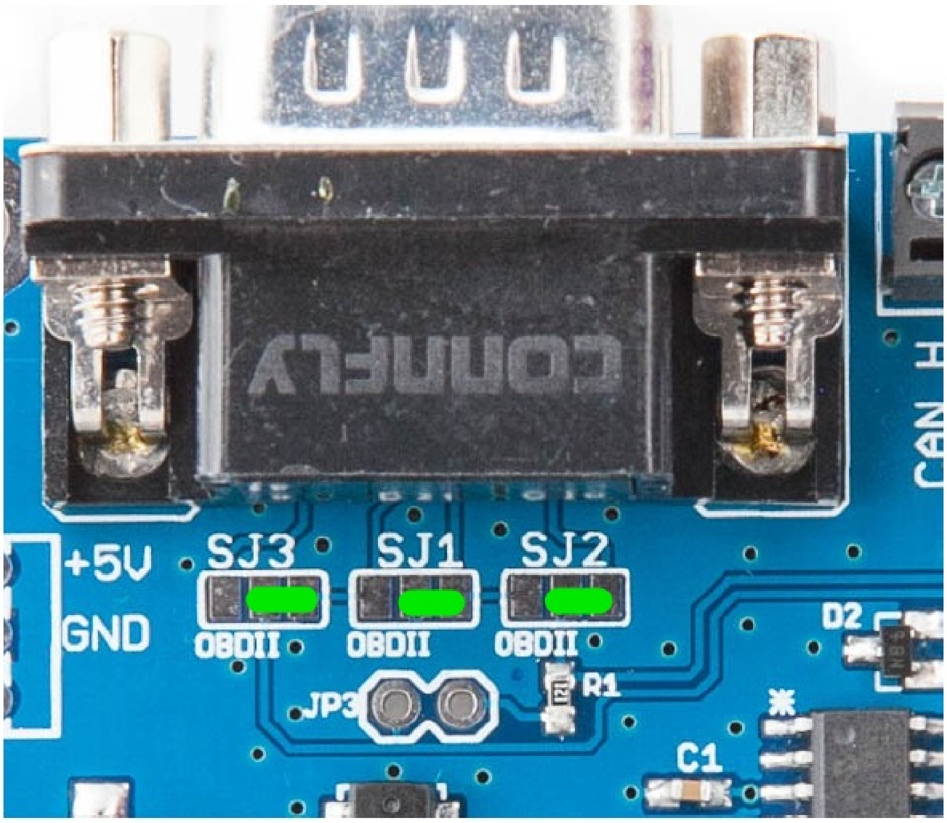
2. Configuring DB9 Connector
   The CAN connection can be made via the DB9 connector. The connector be configured for different pinout. Depend if you are using an OBDII cable or a CAN cable.
3. OBDII Cable
   Close the solder bridges on the lefthand side on SJ1, SJ2 and SJ3 as shown with a red line.



DB9 Pin number	Function
2	GND
3	CAN_H
5	CAN_L

4. CAN Cable

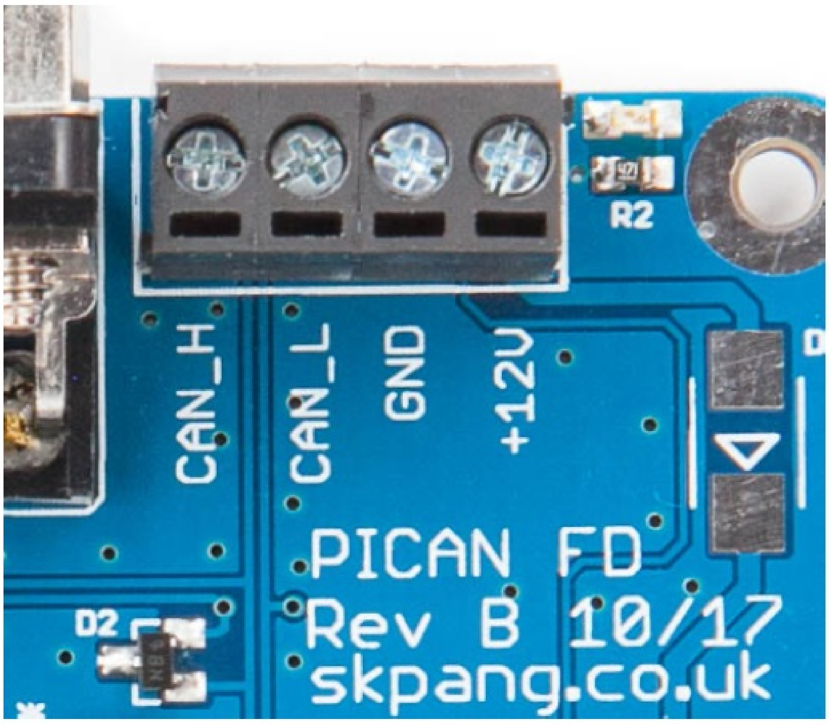
   Close the solder bridges on the righthand side on SJ1, SJ2 and SJ3 as shown with a green line.



DB9 Pin number	Function
3	GND
7	CAN_H
2	CAN_L

5. Screw Terminal

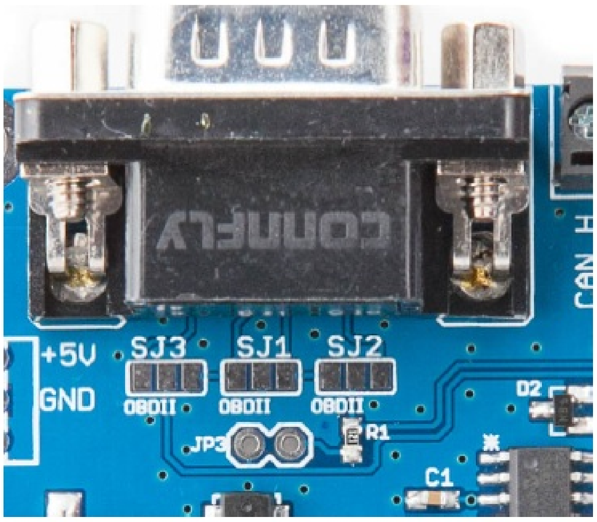
   The CAN connection can also be made via the 4 way screw terminal.



Note : The +12v In is only used on the PiCAN2 FD board with SMPS option fitted.

6. 120W Terminator

   There is a 120W fitted to the board. To use the terminator solder a 2way header pin to JP3 then insert a jumper.



7. LED

   There is a red LED fitted to the board. This is connected to GPIO22.

8. Not Fitted Items

   JP5 can be use to power a serial LCD with data on TXD line from the Pi. There is also 5v supply on JP5.

   Switch mode power supply, this is a 5v module that can power the Pi. It has an input voltage range of 6v to 30v.

Software Installation

It is best to start with a brand new Raspbian image. Download the latest from:
https://www.raspberrypi.org/downloads/raspbian/

After first time boot up, do an update and upgrade first.

- sudo apt-get update
- sudo apt-get upgrade
- sudo reboot

Add the overlays by:

- sudo nano /boot/config.txt
- Add these lines to the end of file:
- dtparam=spi=on
- dtoverlay=i2c-rtc,pcf8523
- dtoverlay=mcp251xfd,spi0-0,interrupt=25

Reboot Pi:

- sudo reboot

Installing CAN Utils
Install the CAN utils by:
sudo apt-get install can-utils

Bring Up the Interface

You can now bring the CAN interface up with CAN 2.0B at 500kbps:

- sudo /sbin/ip link set can0 up type can bitrate 500000 or CAN FD at 500kpbs / 2Mbps. Use copy and paste to a terminal.
- sudo /sbin/ip link set can0 up type can bitrate 500000 dbitrate 2000000 fd on
- Connect the PiCAN2 to your CAN network via screw terminal or DB9.

To send a CAN 2.0 message use :

- cansend can0 7DF#0201050000000000
- This will send a CAN ID of 7DF. Data 02 01 05 – coolant temperature request.

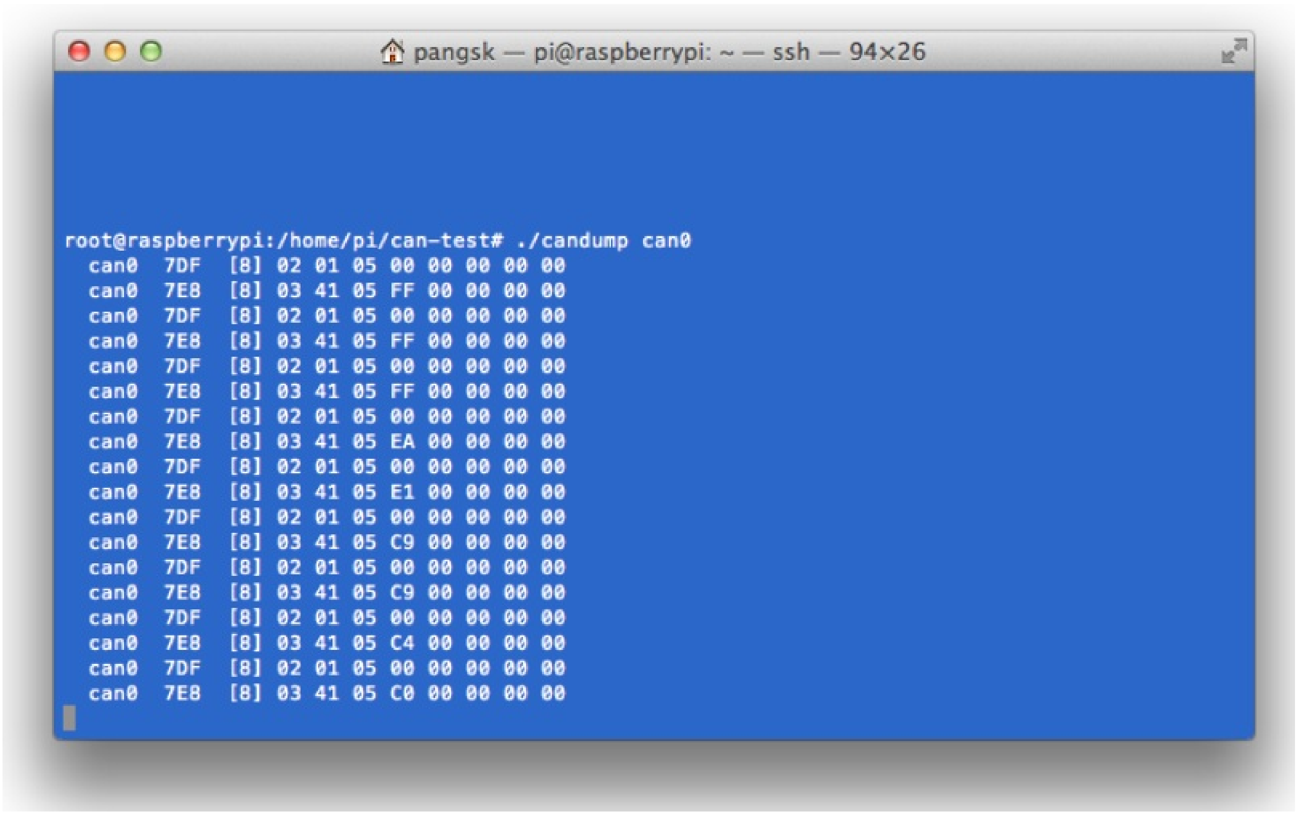
To send a CAN FD message with BRS use :

- cansend can0 7df##15555555555555555

To send a CAN FD message with no BRS use :
cansend can0 7df##05555555555555555

Connect the PiCAN to a CAN-bus network and monitor traffic by using command:

candump can0
You should see something like this:



```
root@raspberrypi:/home/pi/can-test# ./candump can0
  can0  7DF  [8] 02 01 05 00 00 00 00 00
  can0  7E8  [8] 03 41 05 FF 00 00 00 00
  can0  7DF  [8] 02 01 05 00 00 00 00 00
  can0  7E8  [8] 03 41 05 FF 00 00 00 00
  can0  7DF  [8] 02 01 05 00 00 00 00 00
  can0  7E8  [8] 03 41 05 FF 00 00 00 00
  can0  7DF  [8] 02 01 05 00 00 00 00 00
  can0  7E8  [8] 03 41 05 EA 00 00 00 00
  can0  7DF  [8] 02 01 05 00 00 00 00 00
  can0  7E8  [8] 03 41 05 E1 00 00 00 00
  can0  7DF  [8] 02 01 05 00 00 00 00 00
  can0  7E8  [8] 03 41 05 C9 00 00 00 00
  can0  7DF  [8] 02 01 05 00 00 00 00 00
  can0  7E8  [8] 03 41 05 C9 00 00 00 00
  can0  7DF  [8] 02 01 05 00 00 00 00 00
  can0  7E8  [8] 03 41 05 C4 00 00 00 00
  can0  7DF  [8] 02 01 05 00 00 00 00 00
  can0  7E8  [8] 03 41 05 C0 00 00 00 00
```

Real Time Clock (RTC) Software Installation

Insert a CR1220 battery (not supplied) into battery holder. Ensure the “+” is facing upward.

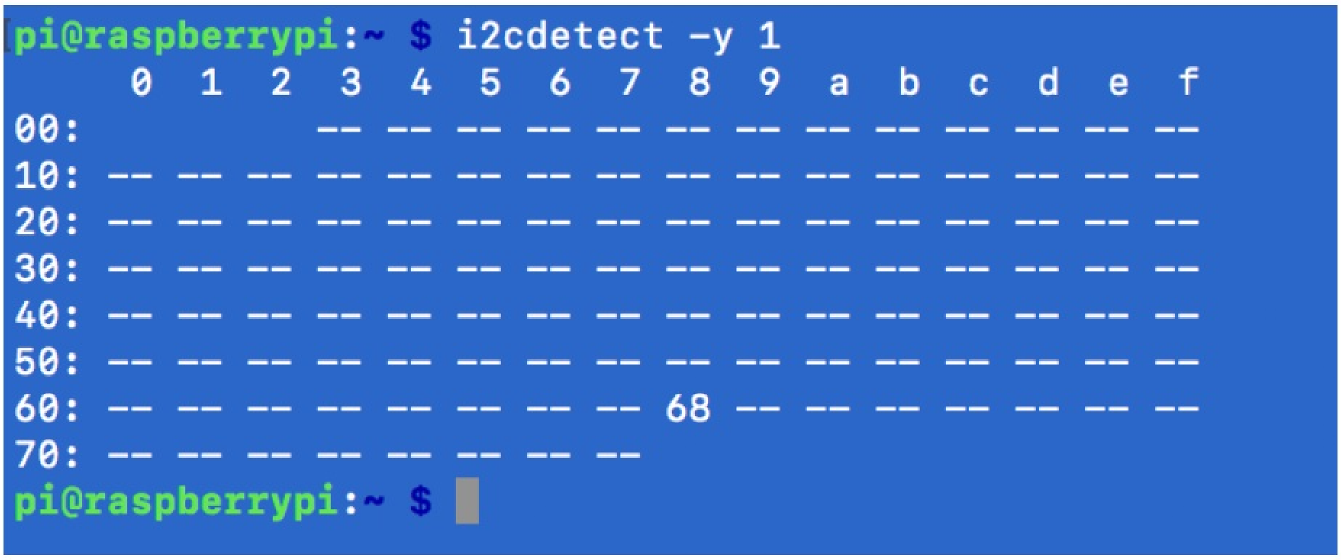
Install the i2c-tools by:

- sudo apt-get install i2c-tools

Then check the RTC:

- sudo i2cdetect -y 1

You should see 68 or UU on address 0x68:



```
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
```

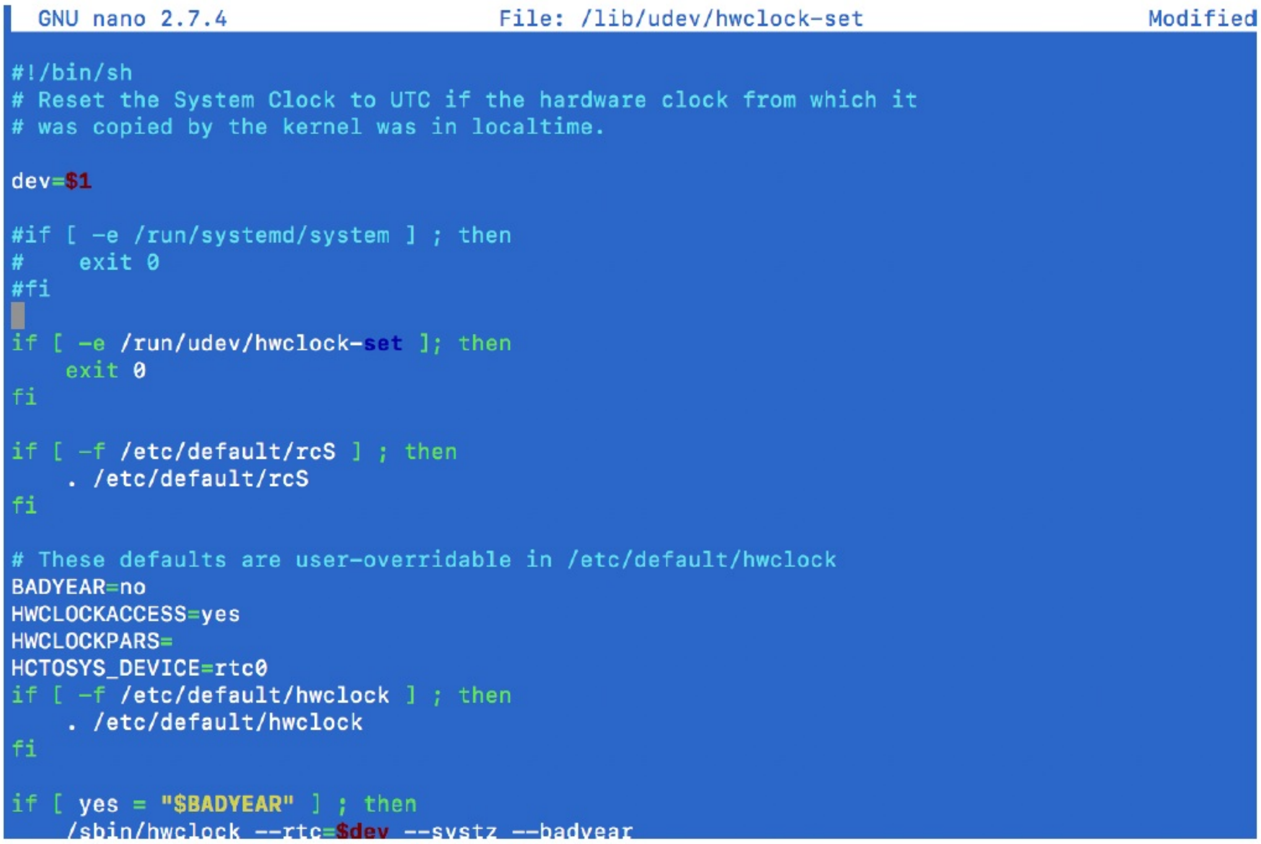
Now you need to disable the “fake hwclock” which interferes with the ‘real’ hwclock
sudo apt-get -y remove fake-hwclock
sudo update-rc.d -f fake-hwclock remove

Start the original hw clock script by:

- sudo nano /lib/udev/hwclock-set

and comment out these three lines:

- #if [-e /run/systemd/system] ; then
- # exit 0
- #fi



```
  GNU nano 2.7.4                  File: /lib/udev/hwclock-set                   Modified

#!/bin/sh
# Reset the System Clock to UTC if the hardware clock from which it
# was copied by the kernel was in localtime.

dev=$1

#if [ -e /run/systemd/system ] ; then
#    exit 0
#fi

if [ -e /run/udev/hwclock-set ]; then
    exit 0
fi

if [ -f /etc/default/rcS ] ; then
    . /etc/default/rcS
fi

# These defaults are user-overridable in /etc/default/hwclock
BADYEAR=no
HWCLOCKACCESS=yes
HWCLOCKPARS=
HCTOSYS_DEVICE=rtc0
if [ -f /etc/default/hwclock ] ; then
    . /etc/default/hwclock
fi

if [ yes = "$BADYEAR" ] ; then
    /sbin/hwclock --rtc=$dev --systz --badyear
```

Reboot the Pi.
Ensure the Ethernet cable or Wifi is on. This will get the time from the network.

Set the clock by:

- sudo hwclock -w

To read the clock:

- sudo hwclock -r

Python Installation and Use

Ensure the driver for PiCAN FD is installed and working correctly first.

Clone the pythonCan repository by:

- git clone https://github.com/hardbyte/python-can
- cd python-can
- sudo python3 setup.py install

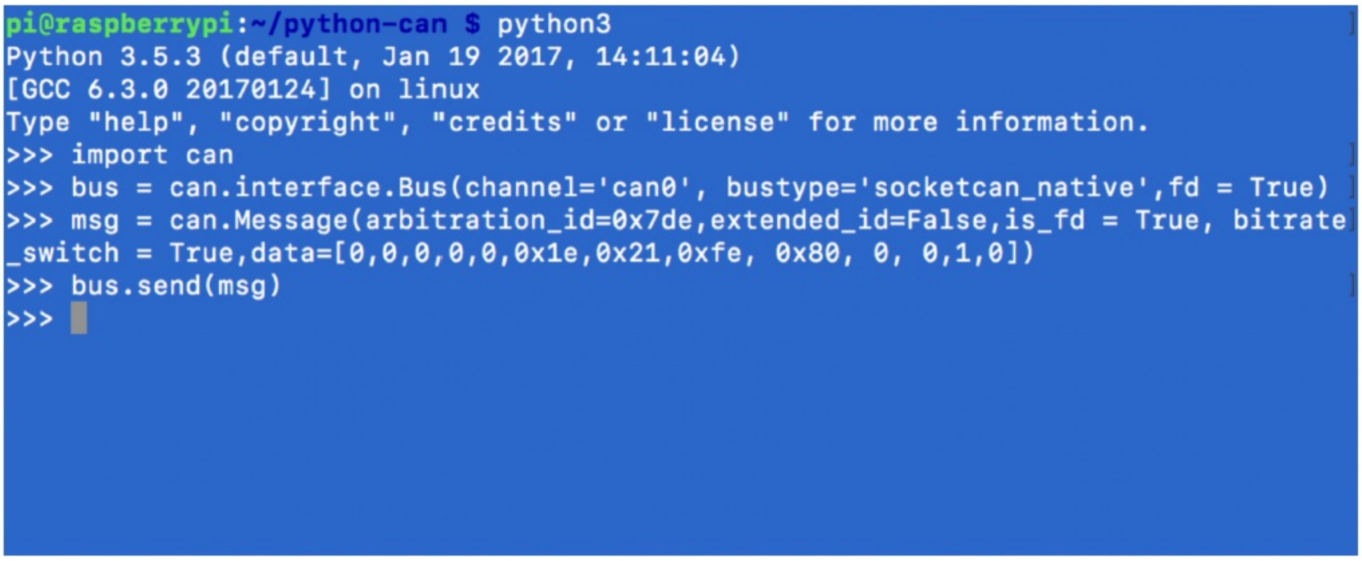
Check there is no error been displayed.

Bring up the can0 interface:

- sudo /sbin/ip link set can0 up type can bitrate 500000 dbitrate 2000000 fd on sample-point .8 dsample-point .8

Now start python3 and try the transmit with CAN FD and BRS set.

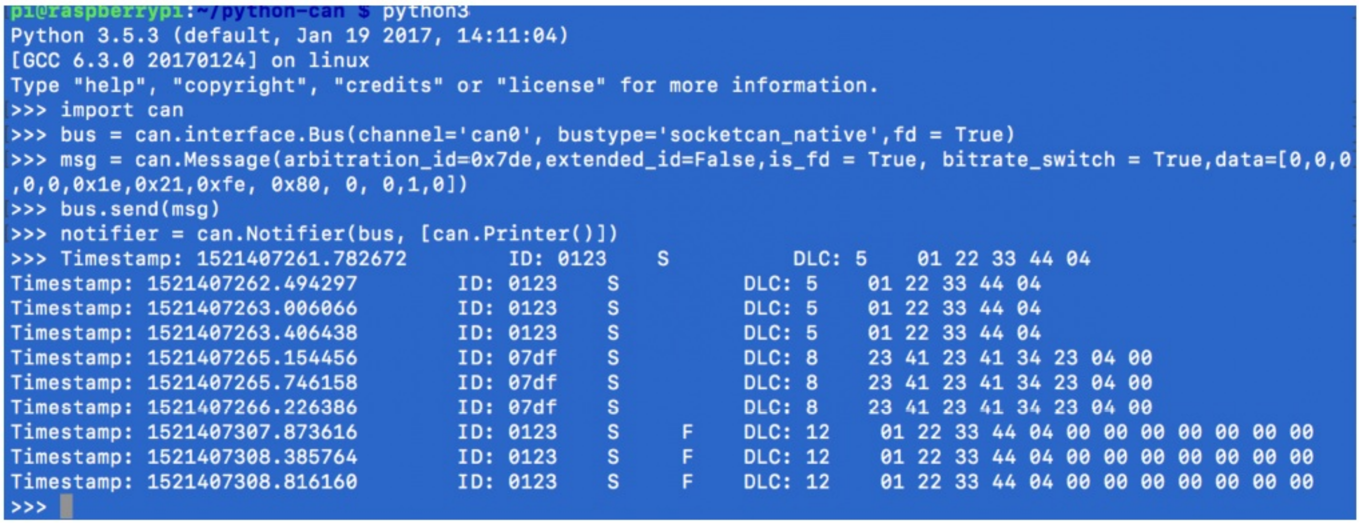
- python3
- import can
- bus = can.interface.Bus(channel='can0', bustype='socketcan_native',fd =True)
- msg = can.Message(arbitration_id=0x7de,extended_id=False,is_fd = True,
- bitrate_switch = True,data=[0,0,0,0,0,0x1e,0x21,0xfe, 0x80, 0, 0,1,0])
- bus.send(msg)



```
pi@raspberrypi:~/python-can $ python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import can
>>> bus = can.interface.Bus(channel='can0', bustype='socketcan_native',fd = True)
>>> msg = can.Message(arbitration_id=0x7de,extended_id=False,is_fd = True, bitrate
_switch = True,data=[0,0,0,0,0,0x1e,0x21,0xfe, 0x80, 0, 0,1,0])
>>> bus.send(msg)
>>>
```

To received messages and display on screen type in:

- notifier = can.Notifier(bus, [can.Printer()])



```
pi@raspberrypi:~/python-can $ python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import can
>>> bus = can.interface.Bus(channel='can0', bustype='socketcan_native',fd = True)
>>> msg = can.Message(arbitration_id=0x7de,extended_id=False,is_fd = True, bitrate_switch = True,data=[0,0,0
,0,0,0x1e,0x21,0xfe, 0x80, 0, 0,1,0])
>>> bus.send(msg)
>>> notifier = can.Notifier(bus, [can.Printer()])
>>> Timestamp: 1521407261.782672        ID: 0123    S          DLC: 5    01 22 33 44 04
Timestamp: 1521407262.494297        ID: 0123    S          DLC: 5    01 22 33 44 04
Timestamp: 1521407263.006066        ID: 0123    S          DLC: 5    01 22 33 44 04
Timestamp: 1521407263.406438        ID: 0123    S          DLC: 5    01 22 33 44 04
Timestamp: 1521407265.154456        ID: 07df    S          DLC: 8    23 41 23 41 34 23 04 00
Timestamp: 1521407265.746158        ID: 07df    S          DLC: 8    23 41 23 41 34 23 04 00
Timestamp: 1521407266.226386        ID: 07df    S          DLC: 8    23 41 23 41 34 23 04 00
Timestamp: 1521407307.873616        ID: 0123    S     F    DLC: 12    01 22 33 44 04 00 00 00 00 00 00 00
Timestamp: 1521407308.385764        ID: 0123    S     F    DLC: 12    01 22 33 44 04 00 00 00 00 00 00 00
Timestamp: 1521407308.816160        ID: 0123    S     F    DLC: 12    01 22 33 44 04 00 00 00 00 00 00 00
>>> 
```

Documentation for python-can can be found at :
https://python-can.readthedocs.io/en/stable/index.html

More expamles in github:
https://github.com/skpang/PiCAN-FD-Python-examples

Documents / Resources

	SK Pang electronics PiCAN FD with RTC [pdf] User Guide PiCAN FD with RTC, PiCAN FD, PiCAN RTC, PiCAN

References

- SK Pang Electronics Ltd - Electronic supply for engineer and hobbyist
- GitHub - hardbyte/python-can: The can package provides controller area network support for Python developers
- GitHub - skpang/PiCAN-FD-Python-examples
- python-can 4.1.0 documentation
- Raspberry Pi OS – Raspberry Pi