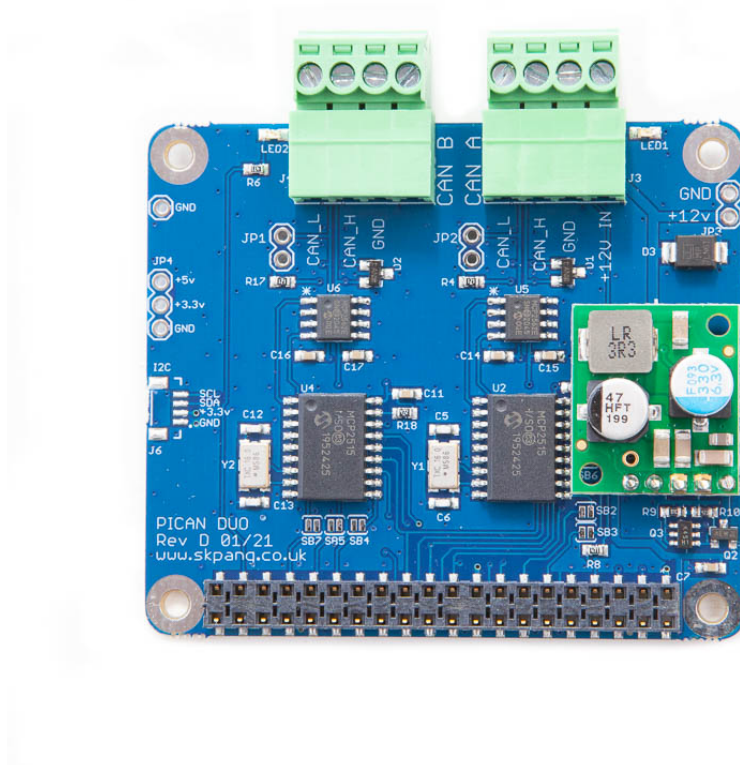




PiCAN2 DUO SMPS USER GUIDE

V1.4

Product name	PiCAN2 DUO 3A SMPS CAN-Bus Board for Raspberry Pi
Model number	RSP-PiCAN2DUOSMPS
Manufacturer	SK Pang Electronics Ltd

Contents

Table of Contents

1. Introduction	3
1.1. Features	3
1.2. Hardware Installation	3
1.3. Screw Terminal	4
1.4. 120Ω Terminator	4
1.5. LED.....	4
1.6. Not Fitted Items	4
2. Software Installation	4
1.7. Bring Up the Interface	6
1.8. Installing CAN Utils.....	6
3. Writing Your Own Software	7
1.9. Application in Python	7
1.10. Application in C	8

1. Introduction

This PiCAN2DUO board provide two independent CAN-Bus channels for the Raspberry Pi 4. It uses the Microchip MCP2515 CAN controller with MCP2551 CAN transceiver. Connections are made via plug in 4 way screw terminal. This board has a 5v 3A SMPS that can power the Pi is well via the screw terminal.

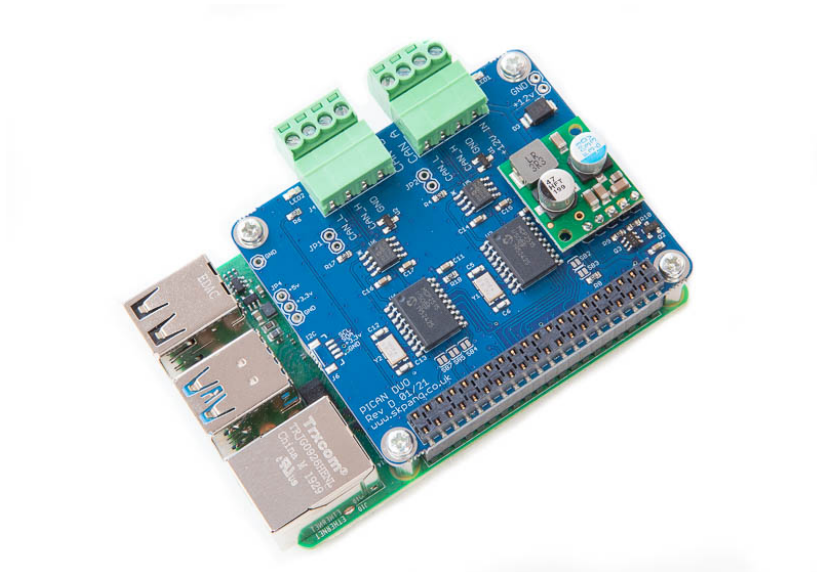
Easy to install SocketCAN driver. Programming can be done in C or Python.

1.1. Features

- CAN v2.0B at 1 Mb/s
- High speed SPI Interface (10 MHz)
- Standard and extended data and remote frames
- CAN connection via screw terminal
- 120Ω terminator ready
- Serial LCD ready
- LED indicator
- Four fixing holes, comply with Pi Hat standard
- SocketCAN driver, appears as can0 and can1 to application
- Interrupt RX on GPIO25 and GPIO24
- 5v 3A SMPS to power Raspberry Pi and accessories from screw terminal
 - Reverse polarity protection
 - High efficiency switch mode design
 - 7v to 24v input range

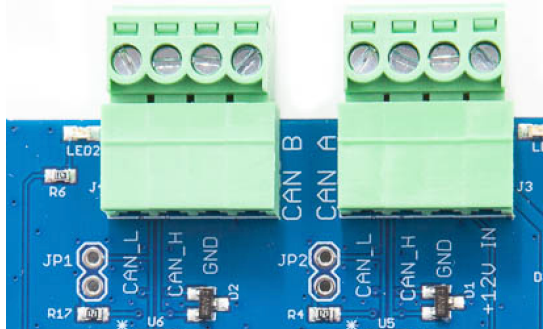
1.2. Hardware Installation

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.



1.3. Screw Terminal

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



CAN B (J4)	
Pin number	Function
1	CAN_L
2	CAN_H
3	GND
4	n/c

CAN A (J3)	
Pin number	Function
1	CAN_L
2	CAN_H
3	GND
4	+12v In

Connector J3 pin 4 is the input for the SMPS, it has an input voltage range of 6v to 20v that is used to power the Raspberry Pi.

1.4. 120Ω Terminator

There is a 120Ω fitted to the board. To use the terminator solder a 2way header pin to JP1 and JP2 then insert a jumper.

1.5. LED

There are two red LEDs fitted to the board. This is connected to GPIO04 and GPIO26.

1.6. 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.

2. 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
```

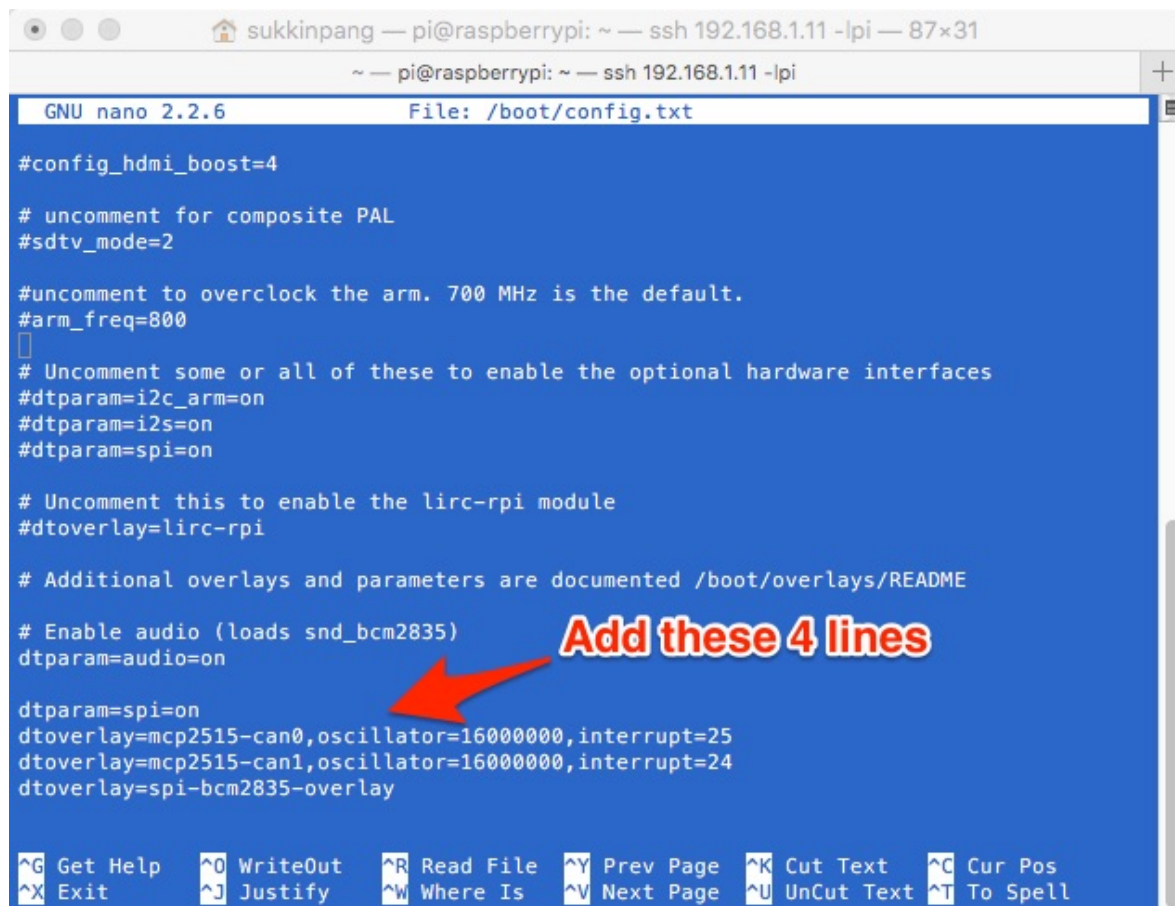
Add these 4 lines to the end of file:

```
dtoverlay=spi=on
```

```
dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25
```

```
dtoverlay=mcp2515-can1,oscillator=16000000,interrupt=24
```

```
dtoverlay=spi-bcm2835-overlay
```



```
sukkinpang — pi@raspberrypi: ~ — ssh 192.168.1.11 -lpi — 87x31
~ — pi@raspberrypi: ~ — ssh 192.168.1.11 -lpi
GNU nano 2.2.6 File: /boot/config.txt

#config_hdmi_boost=4

# uncomment for composite PAL
#sdtv_mode=2

#uncomment to overclock the arm. 700 MHz is the default.
#arm_freq=800
[]
# Uncomment some or all of these to enable the optional hardware interfaces
#dtparam=i2c_arm=on
#dtparam=i2s=on
#dtparam=spi=on

# Uncomment this to enable the lirc-rpi module
#dtoverlay=lirc-rpi

# Additional overlays and parameters are documented /boot/overlays/README

# Enable audio (loads snd_bcm2835)
dtparam=audio=on

dtoverlay=spi=on
dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25
dtoverlay=mcp2515-can1,oscillator=16000000,interrupt=24
dtoverlay=spi-bcm2835-overlay

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Reboot Pi:

```
sudo reboot
```

1.7. Bring Up the Interface

You can now bring the CAN interfaces up:

```
sudo /sbin/ip link set can0 up type can bitrate 500000
sudo /sbin/ip link set can1 up type can bitrate 500000
```

1.8. Installing CAN Utils

Install the CAN utils by:

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

Connect the PiCAN2 Duo to your CAN network via screw terminal .

To send a CAN message on can0 (CAN B J4) use :

```
cansend can0 7DF#0201050000000000
```

This will send a CAN ID of 7DF. Data 02 01 05 – coolant temperature request.

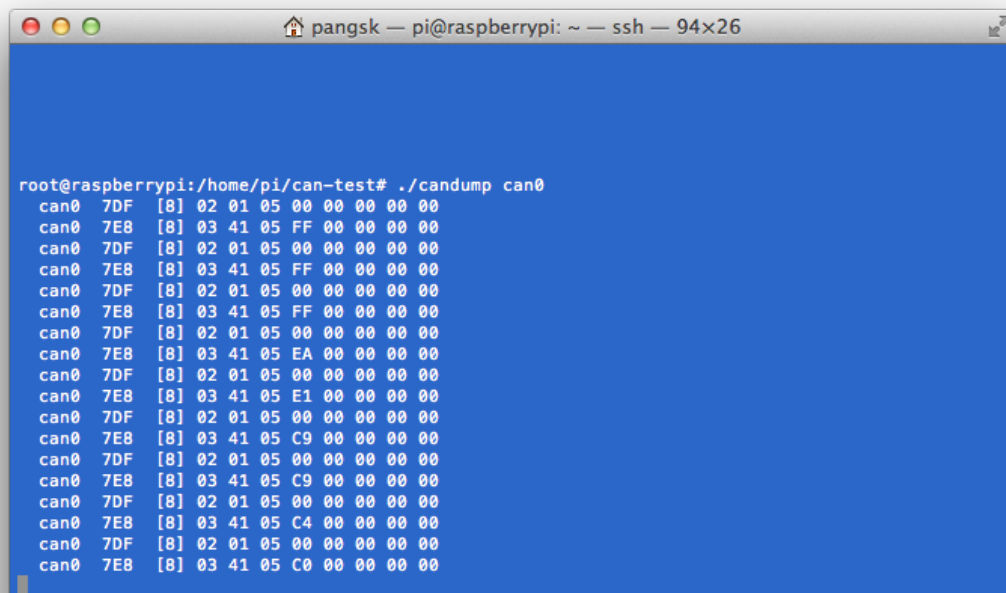
To send a CAN message on can1 (CAN A J3) use :

```
cansend can1 7DF#0201050000000000
```

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

```
candump can0
```

You should see something like this:



The screenshot shows a terminal window titled 'pangsk — pi@raspberrypi: ~ — ssh — 94x26'. The user is at the prompt 'root@raspberrypi:/home/pi/can-test#'. They have entered the command './candump can0'. The output shows a series of CAN bus messages on can0. Each line displays the CAN ID, the data length in bytes, and the data bytes in hexadecimal. The messages alternate between ID 7DF and 7E8, with data bytes 02 01 05 and 03 41 05 respectively. The data bytes are followed by four zeros, indicating a 4-byte data field.

```
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 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
```

3. Writing Your Own Software

You can write your own application software in either C or Python.

1.9. Application in Python

Download the Python-CAN files from:

<https://github.com/hardbyte/python-can/releases/tag/3.2.0>

Unzip and copy over to the Pi.

```
sudo python3 setup.py install
```

Bring the CAN interface up if it is not already done:

```
sudo /sbin/ip link set can0 up type can bitrate 500000
```

Now start python3

```
python3
```

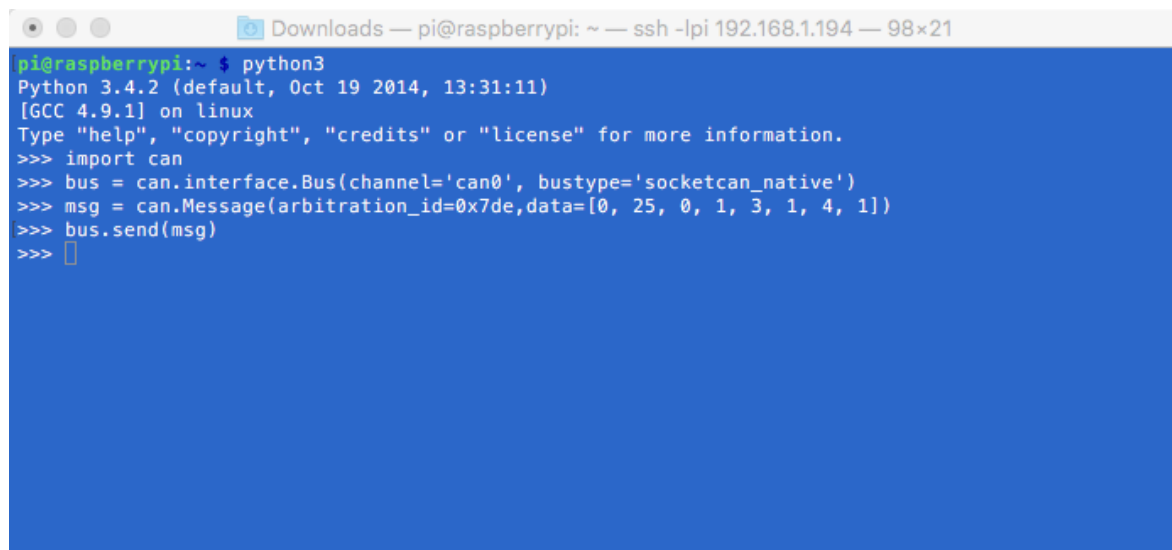
To sent a message out type the following lines:

```
import can

bus = can.interface.Bus(channel='can0', bustype='socketcan_native')

msg = can.Message(arbitration_id=0x7de,
                  data=[0, 25, 0, 1, 3, 1, 4, 1],
                  extended_id=False)

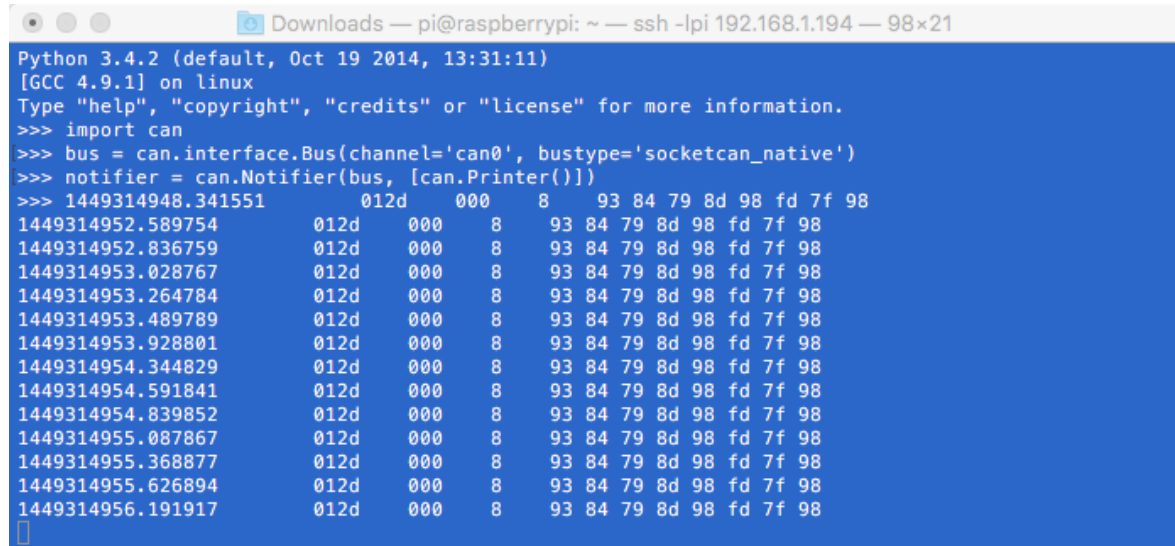
bus.send(msg)
```

A screenshot of a terminal window titled "Downloads — pi@raspberrypi: ~ — ssh -lpi 192.168.1.194 — 98x21". The terminal shows the execution of the Python code from the previous block. The output includes the Python version (3.4.2), GCC version (4.9.1), and the successful execution of the code to send a CAN message. The prompt is now ready for the next command.

```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import can
>>> bus = can.interface.Bus(channel='can0', bustype='socketcan_native')
>>> msg = can.Message(arbitration_id=0x7de,data=[0, 25, 0, 1, 3, 1, 4, 1])
>>> bus.send(msg)
>>> 
```

To received messages and display on screen type:

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



The screenshot shows a terminal window titled 'Downloads — pi@raspberrypi: ~ — ssh -lpi 192.168.1.194 — 98x21'. The terminal output shows the following:

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import can
>>> bus = can.interface.Bus(channel='can0', bustype='socketcan_native')
>>> notifier = can.Notifier(bus, [can.Printer()])
>>> 1449314948.341551      012d      000      8      93 84 79 8d 98 fd 7f 98
1449314952.589754      012d      000      8      93 84 79 8d 98 fd 7f 98
1449314952.836759      012d      000      8      93 84 79 8d 98 fd 7f 98
1449314953.028767      012d      000      8      93 84 79 8d 98 fd 7f 98
1449314953.264784      012d      000      8      93 84 79 8d 98 fd 7f 98
1449314953.489789      012d      000      8      93 84 79 8d 98 fd 7f 98
1449314953.928801      012d      000      8      93 84 79 8d 98 fd 7f 98
1449314954.344829      012d      000      8      93 84 79 8d 98 fd 7f 98
1449314954.591841      012d      000      8      93 84 79 8d 98 fd 7f 98
1449314954.839852      012d      000      8      93 84 79 8d 98 fd 7f 98
1449314955.087867      012d      000      8      93 84 79 8d 98 fd 7f 98
1449314955.368877      012d      000      8      93 84 79 8d 98 fd 7f 98
1449314955.626894      012d      000      8      93 84 79 8d 98 fd 7f 98
1449314956.191917      012d      000      8      93 84 79 8d 98 fd 7f 98
```

1.10. Application in C

Bring the CAN interface up if it is not already done:

```
sudo /sbin/ip link set can0 up type can bitrate 500000
```

Download the source code and example files by typing the following in the command prompt:

```
wget http://skpang.co.uk/dl/cantest.tar
```

Unpack the tar file and change into directory by:

```
tar xf cantest.tar
cd linux-can-utils
```

The example file is called cantest.c to edit this file, type the following in the command prompt:

```
nano cantest.c
```

Line 77 is the CAN message to be sent out.

```
unsigned char buff[] = "7DF#0201050000000000";
```

7DF is the message ID and 0201050000000000 is the data. Change the data to suit. Press CTRL-X to exit.

To compile the program type:

```
make
```

Check there are no errors. To run the program type:

```
./cantest
```