

## **PiCAN3 CAN-Bus with 3A SMPS + RTC**

## **USER GUIDE V1.0**

Product name	PiCAN3 CAN-Bus Board for Raspberry Pi 4 with 3A SMPS + RTC
Model number	RSP-PICAN3
Manufacturer	SK Pang Electronics Ltd

# Contents

## Table of Contents

<b>1. Introduction .....</b>	<b>3</b>
1.1. Features .....	3
<b>2. Hardware Installation.....</b>	<b>3</b>
1.2. Configuring DB9 Connector .....	4
1.3. OBDII Cable .....	4
1.4. CAN Cable.....	4
1.5. Screw Terminal.....	5
1.6. 120Ω Terminator .....	5
1.7. LED.....	5
<b>3. Software Installation .....</b>	<b>5</b>
1.8. Installing CAN Utils .....	6
1.9. Bring Up the Interface .....	6
<b>4. Real Time Clock (RTC) Software Installation.....</b>	<b>7</b>
<b>5. Python Installation and Use.....</b>	<b>9</b>

## 1. Introduction

This PiCAN2 board provide CAN-Bus capability for the Raspberry Pi 4. It uses the Microchip MCP2515 CAN controller with MCP2551 CAN transceiver. Connection are made via DB9 or 3 way screw terminal. This board includes a 3A switch mode power suppler that powers the Pi is well.

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 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
- Foot print for two mini push buttons
- Four fixing holes, comply with Pi Hat standard
- SocketCAN driver, appears as can0 to application
- Interrupt RX on GPIO25
- 5v 3A SMPS to power Raspberry Pi and accessories from DB9 or screw terminal
  - Reverse polarity protection
  - High efficiency switch mode design
  - 6v to 20v input range
- RTC with battery backup (battery not included, requires CR1225 cell)

## 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.

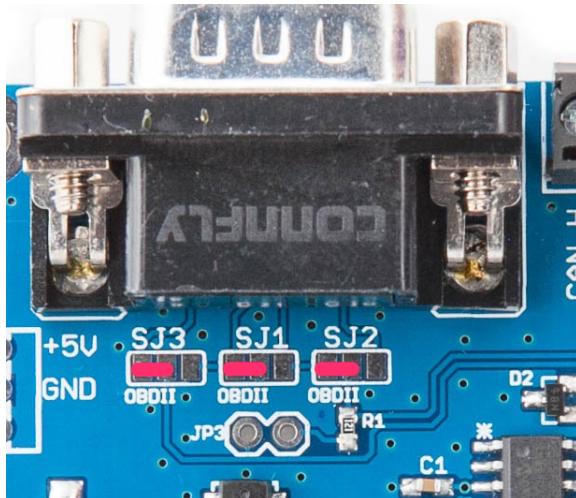


## 1.2. Configuring DB9 Connector

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.

## 1.3. OBDII Cable

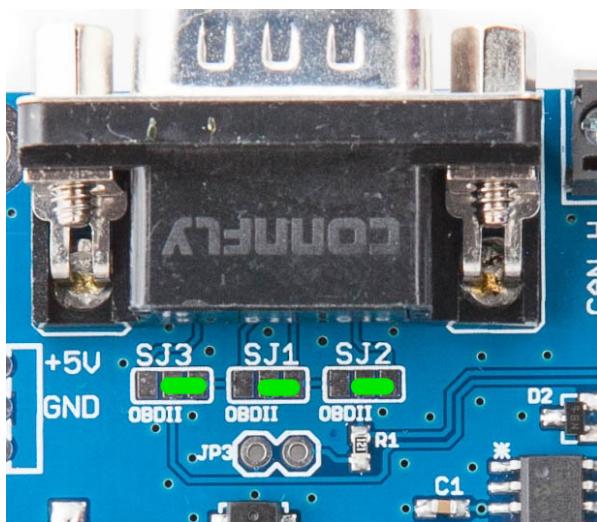
Close the solder bridges on the lefthand side on SJ1, SJ2 and SJ3 as shown with a red line.



DB9 Pin number	Function
2	GND
3	CAN_H
5	CAN_L

## 1.4. CAN Cable

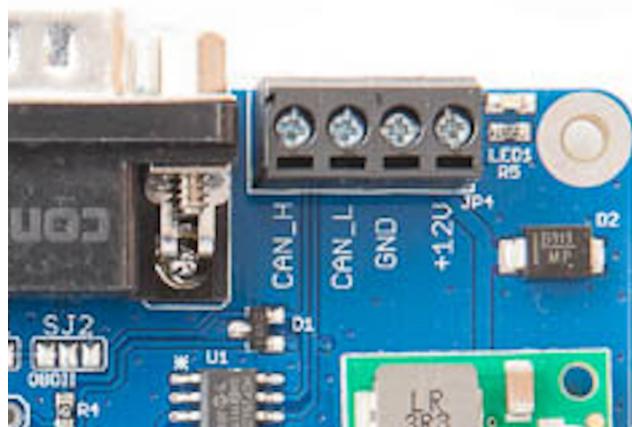
Close the solder bridges on the righthand side on SJ1, SJ2 and SJ3 as shown with a green line.



DB9 Pin number	Function
3	GND
7	CAN_H
2	CAN_L

### 1.5. Screw Terminal

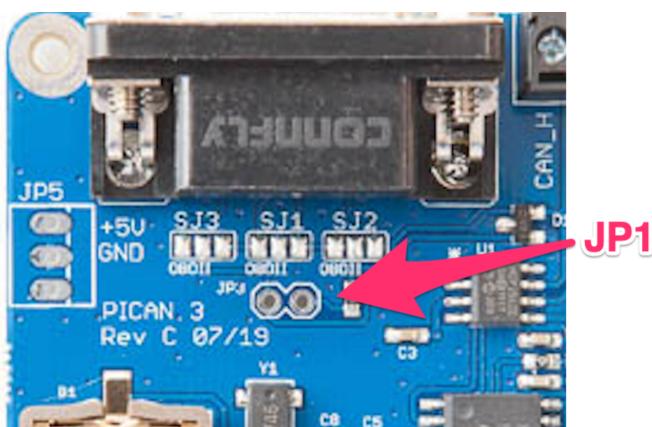
The CAN connection can also be made via the 4 way screw terminal.



Pin number	Function
1	CAN_H
2	CAN_L
3	GND
4	+12v In

### 1.6. 120Ω Terminator

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



### 1.7. LED

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

## 3. 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 3 lines to the end of file:

```
dtparam=spi=on  
dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25  
dtoverlay=spi-bcm2835-overlay
```

```
#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  
  
dtparam=spi=on  
dtoverlay=mcp2515-can0,oscillator=16000000,interrupt=25  
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
```

## 1.8. Installing CAN Utils

Install the CAN utils by:

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

## 1.9. Bring Up the Interface

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

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

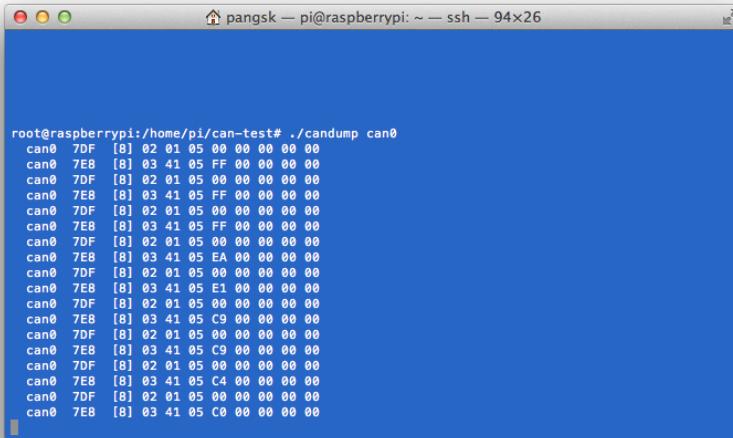
To send a CAN 2.0 message use :

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

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

```
candump can0
```

You should see something like this:



A screenshot of a terminal window titled "pangsk — pi@raspberrypi: ~ — ssh — 94x26". The window displays the output of the command "candump can0". The output shows several CAN frames transmitted on the bus. Each frame is represented by a header (can0, 7DF or 7EB) followed by a timestamp [8], and then the data bytes in hex format. The data bytes for most frames are 02 01 05 00 00 00 00 00.

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

## 4. Real Time Clock (RTC) Software Installation

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

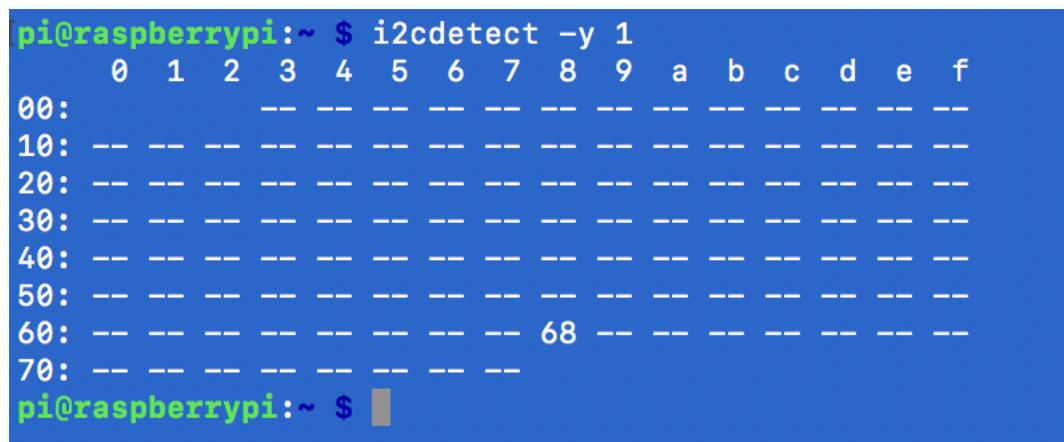
Install the i2c-tools by:

```
sudo apt-get install i2c-tools
```

Then check the RTC:

```
sudo i2cdetect -y 1
```

You should see 68 or UU on address 0x68:



A screenshot of a terminal window titled "pi@raspberrypi: ~ \$". The window displays the output of the command "i2cdetect -y 1". The output shows a map of I2C addresses from 00 to 7F. Address 68 is marked with a value of 68, indicating the presence of a device at that address. The other addresses are marked with dashes, indicating they are free or not detected.

	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: ~ $
```

Add the overlays by:

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

Add these lines to the end of file:

```
dtparam=i2c_arm=on  
dtoverlay=i2c-rtc,pcf8523
```

Reboot Pi:

```
sudo reboot
```

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  
sudo systemctl disable fake-hwclock
```

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
```

and comment out these two lines:

```
#/sbin/hwclock --rtc=$dev --systz --badyear  
#/sbin/hwclock --rtc=$dev --systz
```

```

#!/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
#    /sbin/hwclock --rtc=$dev --hctosys --badyear
else
#    /sbin/hwclock --rtc=$dev --systz
#    /sbin/hwclock --rtc=$dev --hctosys
fi

# Note 'touch' may not be available in initramfs
> /run/udev/hwclock-set

```

Comment out  
these 3 lines

Comment out  
these two lines

**^G Get Help    ^O Write Out    ^W Where Is    ^K Cut Text    ^J Justify    ^C Cur Pos    M-U Undo**  
**^X Exit    ^R Read File    ^\ Replace    ^U Uncut Text    ^T To Spell    ^\_ Go To Line    M-E Redo**

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`

## 5. Python Installation and Use

Ensure the driver for PiCAN 3 board 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
```

Now start python3 and try the transmit with CAN frame.

```
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)
```

```
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 in:

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

```
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
```