

PiCAN FD Board with 10Base-T1S for Raspberry Pi USER GUIDE V1.0 July 2024

Product name	PiCAN FD Board with 10Base-T1S for Raspberry Pi
Model number	RSP-PICANFD-T1S
Manufacturer	SK Pang Electronics Ltd

Contents

Table of Contents

1. Introduction	3
1.1. CAN FD Features	3
1.2. 10Base-T1S Features	3
2. Hardware Installation	4
1.3. CAN Bus connection	5
1.4. 10Base-T1S Connection	5
1.5. CAN Bus 120Ω Terminator	5
1.6. 10Base-T1S Terminator	5
1.7. LED Indicators	5
1.8. Optional 3.2A SMPS	5
3. Software Installation	6
1.9. Installing CAN Utils	6
1.10. Bring Up the Interface	6
4. Python Installation and Use	7
5. 10Base-T1S Driver Install	8

1. Introduction

This board has a 10Base-T1S Single Pair Ethernet (SPE) port and a CAN FD port.

The 10Base-T1S is provided by the Microchip LAN8651 chip. The CAN FD is provided by the Microchip MCP2518FD CAN controller. Connection is via 4 way pluggable terminal.

The improved CAN FD extends the length of the data section to up to 64 bytes per frame and a data rate of up to 8 Mbps.

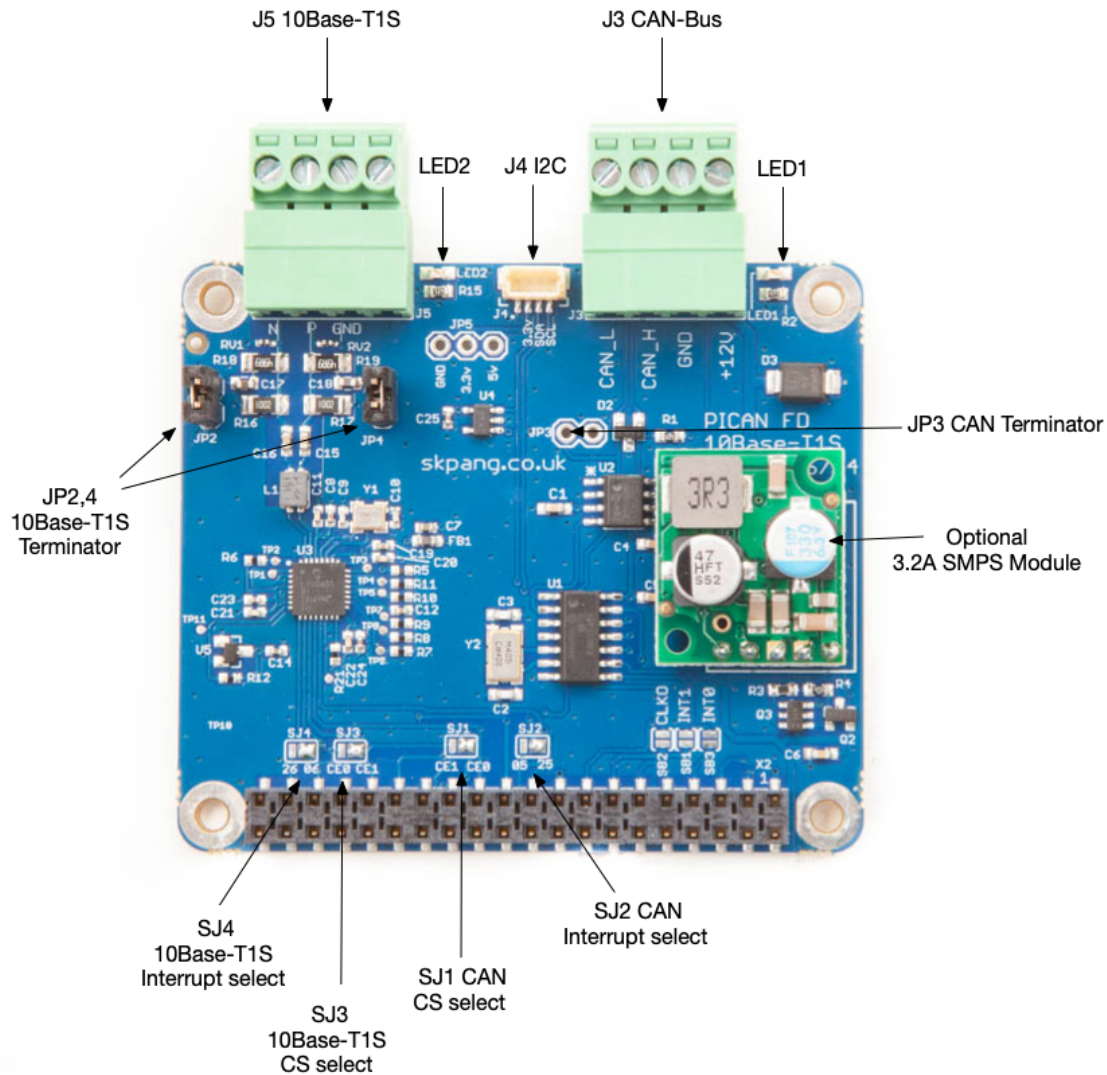
Easy to install SocketCAN driver for CAN applications.

1.1. CAN FD 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 4 way pluggable terminal
- 120Ω terminator ready
- LED indicator
- Four fixing holes, comply with Pi Hat standard
- SocketCAN driver, appears as can0 to application
- Interrupt RX on GPIO25

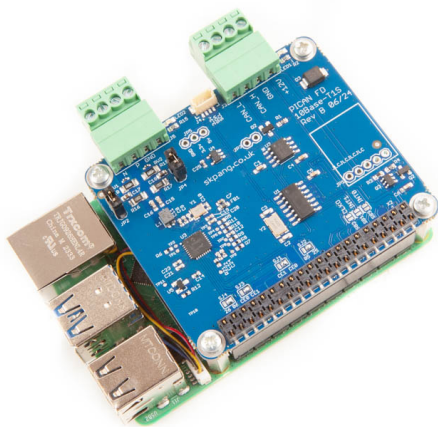
1.2. 10Base-T1S Features

- Microchip LAN8651 MAC/PHY controller
- PHY designed to IEEE Std. 802.3cgTM-2019
- 10 Mbit/s over a single balanced pair
 - Half-duplex multidrop mixing segments up to at least 25m with up to at least 8 PHYs
 - Half-duplex point-to-point link segments upto at least 15m
- Physical Layer Collision Avoidance (PLCA)
 - Burst mode for transmission of multiple packets • for latency-sensitive applications
 - Minimize latency for time-sensitive applications by assigning multiple PLCA IDs per node
- Carrier Sense Multiple Access / Collision Detection (CSMA/CD) media access control
- Application Controlled Media Access (ACMA) for implementation of collision-free Time-Division Multiple Access (TDMA) methods
- Industry standard Serial Peripheral Interface (SPI), designed to the OPEN Alliance 10BASE-T1x MAC-PHY Serial Interface specification, V1.1
- Unique 48-bit MAC address provided by 24AA02E48 IC



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.



If the install is on the Raspberry Pi 5 with a fan fitted then this height extender is required:

<https://www.skpang.co.uk/products/raspberry-5-header-extender-kit>

1.3. CAN Bus connection

The CAN bus connection is on connector J3.

J3 pin no.	Function
1	CAN_L
2	CAN_H
3	GND
4	+12v

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

1.4. 10Base-T1S Connection

The 10Base-T1S connection is on connector J5.

J5 pin no.	Function
1	N
2	P
3	GND
4	

1.5. CAN Bus 120Ω Terminator

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

1.6. 10Base-T1S Terminator

There are two 10Base-T1S terminators fitted on the board. JP2 and JP4. Remove the jumpers if terminator is not required.

1.7. LED Indicators

There are two red LEDs fitted to the board. These are user programmable via GPIOs.

LED1 connected to GPIO22.

LED2 connected to GPIO19.

1.8. Optional 3.2A SMPS

Switch mode power supply module, this is an optional 5v module that can power the Pi. It has an input voltage range of 8v to 26v. The total power drawn by the Pi and any other connected peripherals must be less than 3.2A

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/firmware/config.txt
```

Add these lines to the end of file:

```
dtparam=spi=on
```

```
dtparam=i2c_arm=on
```

```
dtoverlay=lan865x
```

```
dtoverlay=mcp251xfd,spi0-0,interrupt=25
```

Reboot Pi:

```
$ sudo reboot
```

1.9. Installing CAN Utils

Install the CAN utils by:

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

1.10. 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 500kbps / 2Mbps. Use copy and paste to a terminal.

```
$ sudo /sbin/ip link set can0 up type can bitrate 500000 dbitrate 2000000 fd on  
sample-point .8 dsample-point .9
```

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##1555555555555555
```

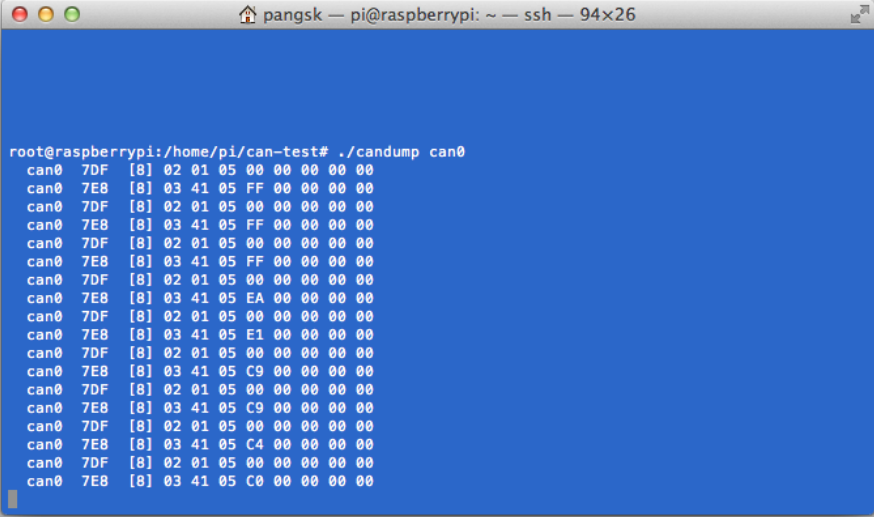
To send a CAN FD message with no BRS use :

```
$ cansend can0 7df##055555555555555555
```

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 terminal shows the command './candump can0' being executed. The output displays a series of CAN bus messages, each with a timestamp, ID, flags, and data bytes. The IDs alternate between 7DF and 7E8. The data bytes include sequences of zeros, FF, EA, E1, C9, C4, and C0.

```
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
```

4. 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', fd = True)
```

```
msg = can.Message(arbitration_id=0x7de,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 examples in github:

```
$ https://github.com/skpang/PiCAN-FD-Python-examples
```

5. 10Base-T1S Driver Install

On the Raspberry Pi, at the prompt type in:

```
$ git clone https://github.com/skpang/10Base-T1S_tools.git
$ cd 10Base-T1S_tools/
$ sudo cp lan865x.dtbo /boot/overlays/
$ chmod +x ethtool
```



```

$ unzip lan865x-linux-driver-0v4.zip

$ cd lan865x-linux-driver-0v4/

$ sudo apt-get --assume-yes install build-essential cmake subversion
libncurses5-dev bc bison flex libssl-dev python3

$ sudo wget https://raw.githubusercontent.com/RPi-Distro/rpi-
source/master/rpi-source -O /usr/local/bin/rpi-source && sudo chmod +x
/usr/local/bin/rpi-source && /usr/local/bin/rpi-source -q --tag-update

$ make

$ cp microchip_tls.ko ../

$ cp lan865x_tls.ko ../

$ sudo reboot

$ cd 10Base-T1S_tools

```

For Raspberry Pi 5, type in:

```
$ python3 eth1_up_pi5.py
```

For Raspberry Pi 4, type in:

```
$ python3 eth1_up_pi4.py
```

```
$ ifconfig
```

You should see something like this:

```

pi@raspberrypi:~/10Base-T1S_tools $ python3 eth1_up_pi5.py
#####
Raspberry Pi 10Base-T1S skpang.co.uk 06/24
MAC address 44:b7:d0:d6:34:f9
sudo ip link set dev eth1 address 44:b7:d0:d6:34:f9
PLCA settings for eth1:
  Enabled: Yes
  Local node ID: 0 (coordinator)
  Node count: 8
  TO timer: 32 BT
  Burst count: 0 (disabled)
  Burst timer: 128 BT
pi@raspberrypi:~/10Base-T1S_tools $ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.22 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fd28:411c:7344:1bc45:c02e:86c8:8009:79e prefixlen 64 scopeid 0x0<global>
    inet6 fe80::d332:4676:6d69:8696 prefixlen 64 scopeid 0x20<link>
    ether d8:3a:dd:bd:ca:f8 txqueuelen 1000 (Ethernet)
    RX packets 541 bytes 56655 (55.3 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 161 bytes 23609 (23.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 106

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.5.190 netmask 255.255.255.0 broadcast 0.0.0.0
    inet6 fe80::46b7:d0ff:fe00::9 prefixlen 64 scopeid 0x20<link>
    ether 44:b7:d0:d6:34:f9 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 32 bytes 4150 (4.1 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 187

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local loopback)
    RX packets 17 bytes 2134 (2.0 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 17 bytes 2134 (2.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether d8:3a:dd:bd:ca:f9 txqueuelen 1000 (Ethernet)
    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

pi@raspberrypi:~/10Base-T1S_tools $ ping 192.168.5.1
PING 192.168.5.1 (192.168.5.1) 56(84) bytes of data:
64 bytes from 192.168.5.1: icmp_seq=1 ttl=128 time=15.1 ms
64 bytes from 192.168.5.1: icmp_seq=2 ttl=128 time=1.32 ms
64 bytes from 192.168.5.1: icmp_seq=3 ttl=128 time=1.33 ms
64 bytes from 192.168.5.1: icmp_seq=4 ttl=128 time=1.36 ms

```

This shows eth1 is up.

Using a twisted pair wire, connect to J5 and the other end to a Microchip EVB-LAN8670-USB. Install the Windows driver and Wireshark software.

On the Raspberry Pi, at the prompt type in:

```
$ python3 10Base-T1S_UDP_demo.py
```

On the Windows machine start Wireshark software and select the EVB-LAN8670-USB interface. You should see something like this:

The screenshot shows the Wireshark interface with a live capture on Ethernet 2. The packet list pane displays 45 packets, all of which are UDP packets from 192.168.5.100 to 192.168.5.1. The packet details pane shows the structure of a selected packet (No. 22):

- Frame 1: 137 bytes on wire (1096 bits), 137 bytes captured (1096 b) on interface 0
- Ethernet II, Src: MicrochipTec_d6:34:f9 (44:b7:d0:d6:34:f9), Dst: 02:00:0c:00:00:00 (02:00:0c:00:00:00)
- Internet Protocol Version 4, Src: 192.168.5.100, Dst: 192.168.5.1
- User Datagram Protocol, Src Port: 58027, Dst Port: 5005
- Data (95 bytes)

The raw data pane shows the hexadecimal and ASCII representation of the packet payload:

```
0000 00 1e c0 d1 b0 6d 44 b7 d0 d6 34 f9 08 00 45 00  ....mD...4...E-
0010 00 7b 8f b7 40 00 40 11 1f 05 c0 a8 05 64 c0 a8  {...@...d...
0020 05 01 e2 ab 13 8d 00 67 c9 99 48 65 6c 6c 6f 2c  ....g...Hello,
0030 20 57 6f 72 6c 64 21 31 37 33 39 38 37 34 30 31  World! 73987401
0040 39 32 37 33 34 30 39 31 37 32 33 34 31 36 32 38  92734091 72341628
0050 37 33 36 34 31 38 37 32 33 36 34 31 36 32 33 39  73641872 36416239
0060 34 36 31 32 38 33 34 36 31 38 32 36 33 34 38 37  46128346 18263487
0070 31 36 32 38 33 34 36 31 38 32 33 36 34 38 31 36  16283461 82364816
0080 32 33 38 34 36 33 32 38 37 23846328 7
```