



MyPi Industrial IoT Edge Gateway

User Guide

Issue : 1.7
Dated : June 2023
Prepared By : Andrew O'Connell

FEATURES

- Supports Raspberry Pi Compute Module 1/3/3+/4S providing a low power or high performance system
- 10/100 Ethernet
- Isolated CAN Interface with on-board termination resistor
- Isolated 2-Wire RS485 Interface with transparent hardware flow control and on-board bus pull and termination resistors
- 2 x mPCIe Interface supporting 3G/4G Modems (1 x on-board SIM slot) and USB WiFi N/AC cards as well as Bluetooth/LoRa/Zigbee/TCM310 RF IO modules
- SD Card secondary storage in addition to eMMC flash on Raspberry Pi Compute Module
- Integrated Infineon SLB9670 TPM 2.0
- Optional IO module with accelerometer & secondary RTC providing 'shake to wake' and 'sleep/wake timer' functionality.
- Software enabled 1.6s hardware watchdog
- 256Byte ID EEPROM
- Switched power input with 'Power Good' input line allowing for controlled power up and safe shutdown operation – ideal for automotive environments
- Wide 7-28V DC power input range, withstanding transient dips to 6V (e.g, Engine Crank)
- FCC/CE Class A approval
- Wide fan-less operational ambient temperature range -25 to +60 °C
- Small, Rugged aluminium enclosure 11 x 13 x 3cm with flexible mounting Kit

FRONT/REAR IO PORTS



Main IO Port

- | | |
|---|------------------------------------|
| 1 | CAN L |
| 2 | CAN H |
| 3 | CAN-BUS 120R Termination Enable A* |
| 4 | CAN-BUS 120R Termination Enable B* |
| 5 | RS485 – A (D+) |
| 6 | RS485 – B (D-) |
| 7 | CAN/RS485 Isolated Ground/0V |
| 8 | Not Connected |

* Connect these 2 pins together to enable the internal CAN termination resistor

DC In

- | | |
|---|-----------------------------------|
| 1 | 7-28V Vin |
| 2 | 'Power Good' signal (6-30V Input) |
| 3 | 0V |



PROG Raspberry Pi Compute Module eMMC programming connector

- 1 USB Power
- 2 USB Data -
- 3 USB Data +
- 4 NC
- 5 GND

The programming mode link is set enabled by default so the unit can be reprogrammed without opening the case.

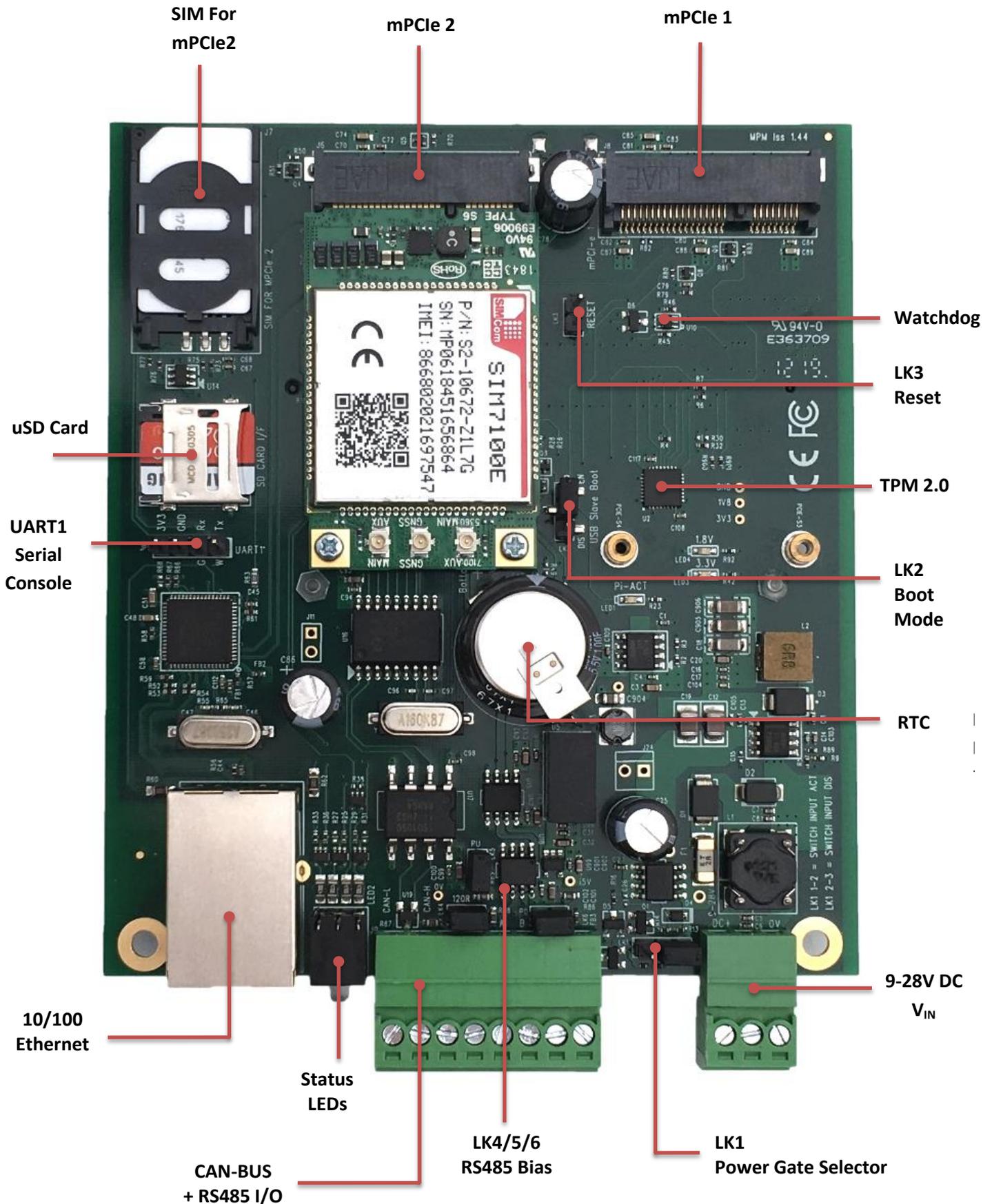
SERIAL External UART1 Connector (Serial Console)

- 1 NC
- 2 Pi Tx OUT (3V TTL Voltage Level)
- 3 Pi Rx IN (3V TTL Voltage Level)
- 4 NC
- 5 GND

Note: Development kits are shipped with a USB to TTL UART cable compatible with this connector

Antenna 1-5 functionality is documented on the configuration label affixed to the bottom of the unit

INTERNAL MAIN BOARD FEATURES



Pi eMMC Programming Port

Serial Console



Expansion Connector

CAN Interface

The EdgeGateway unit provides a fully isolated CAN 2.0b Interface using Microchip MCP2515I controller chip.

RS485 Interface

The EdgeGateway unit provides a fully isolated 2-Wire (half duplex) RS485 interface with automatic, transparent hardware flow control.

SD Card

This connects to the SD1 secondary SD card interface on the Raspberry Pi Compute module, providing easy expansion of the Pi Modules eMMC storage making it ideal for data logging.

SIM Card

This connects only to MPCIE2 (Left hand side, nearest to SIM/SD Card)

TPM 2.0

The Infineon SLB9670 TPM 2.0 Is connected via SPI1, this has kernel support from version 4.14.85 onwards

Expansion connector

Internal expansion connector for Accelerometer/RTC Sleep/Wake card

ID EEPROM

A 256Byte EEPROM for user ID storage

The lower 128Byte has read/write access for user storage, the first 4 hex bytes have been programmed with an ID code visible on the outside of the unit.

The upper 128byte is read only with the last 32bits (6 hex bytes) containing a unique ID code.

HARDWARE CONFIGURATION LINKS

LK1 Power Switch FET operation

1-2 Enable switchable power input

2-3 Disable switchable power input (Power always enabled)*



LK2 RPi Compute Module eMMC USB programming mode

1-2 Enabled*

2-3 Disabled



LK3 System Reset

1	0V
2	/RESET

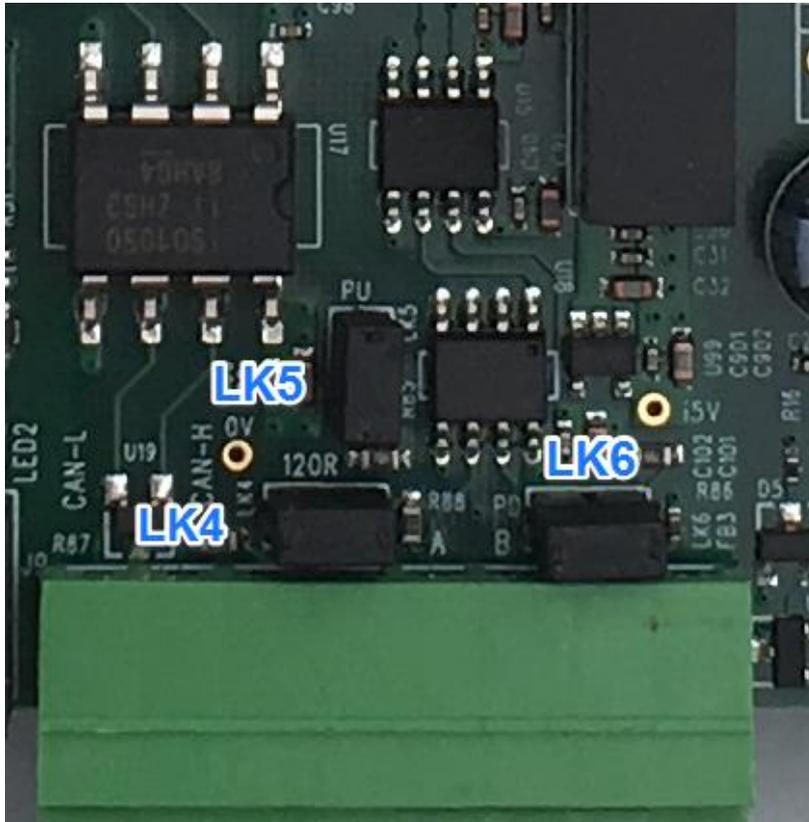


*Default fitted position as shipped.

LK4 120R RS485 Termination Resistor Enable*

LK5 640R RS485 Bus Pull Up Enable*

LK6 640R RS485 Bus Pull Down Enable*



*Default fitted position as shipped.

EXTERNAL SERIAL CONSOLE

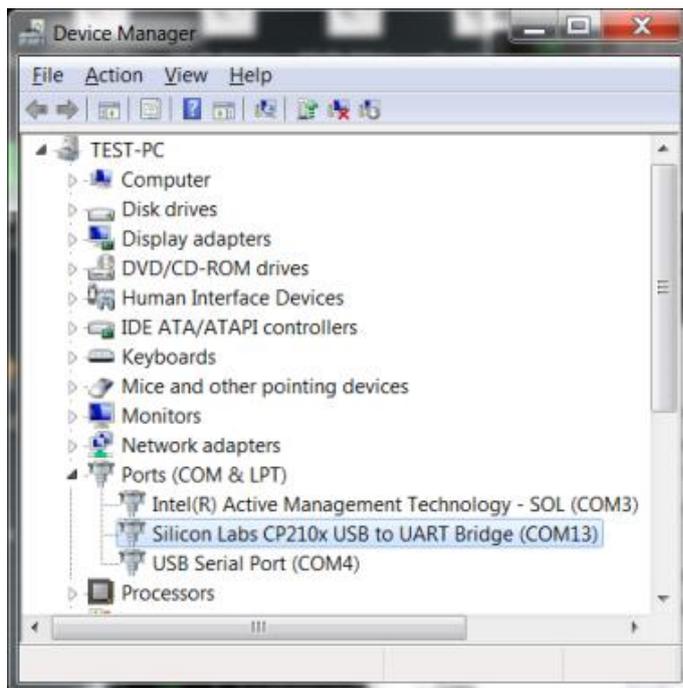
A special serial console cable with integrated USB to UART driver is included in the dev kit



In order to use this cable the vendor's drivers need to be installed, download and install the standard VCP driver for your OS here :

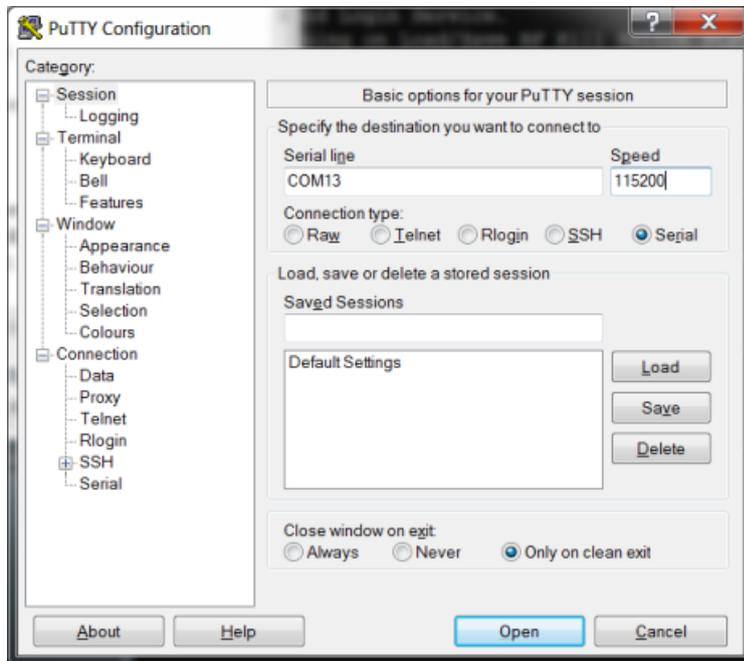
[Silicon Labs CP210x USB to UART Driver Download Link](#)

Once installed plug the cable in and make a note of the COM port the device has initialised to as this number will likely differ from the number below.

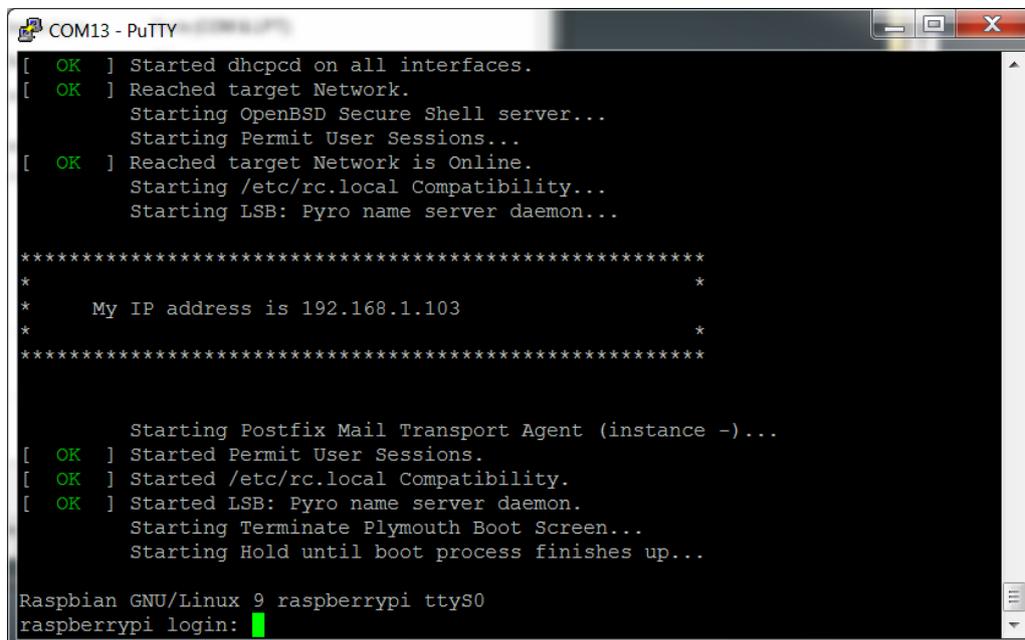


Next download and install putty for SSH/Serial console access : [Putty Download Link](#)

Configure putty for serial console access as below, using the COM number noted previously and using baud rate **115200**



Finally click “Open” and power up the EdgeGateway unit:



If the Ethernet is connected the unit will try to DHCP an address from a local server, at this point you can either log in via the serial port or establish an SSH console using Putty.

The default password for root user is “root”

IMPORTANT NOTE : This should be changed this as soon as possible.

RASPBERRY PI COMPUTE MODULE PROGRAMMING

The unit as shipped is configured to allow the eMMC flash on the compute module to be re-programmed without removing the PCB from the enclosure.

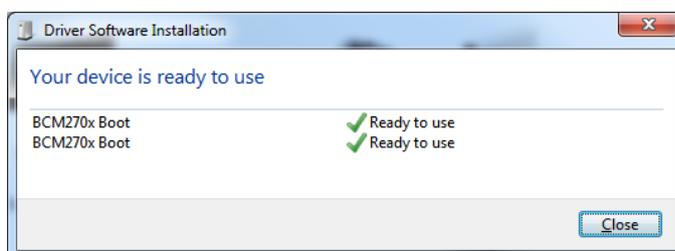
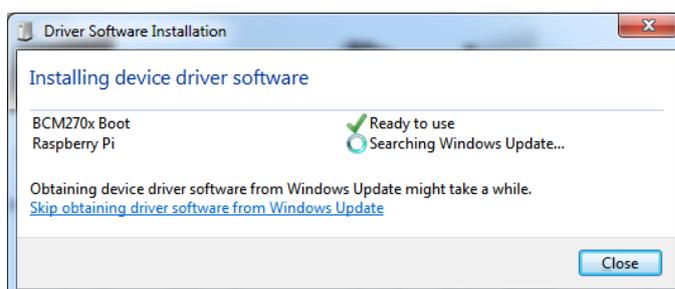
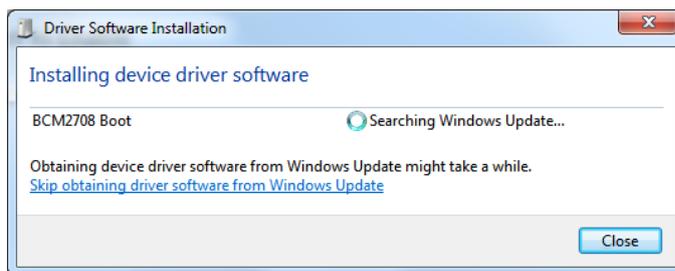
Units come pre-programmed with the demo Raspbian OS pre-installed, this section describes how to write a new disk image to the Compute Module.

First of all download the windows USB boot installer, this will install the device drivers as well as a program we'll use later called RPi-Boot (note there is a new version for CM1, CM3 & CM3+ dated November 2018) :

[Raspberry Pi RPI-BOOT Software Download Link](#)

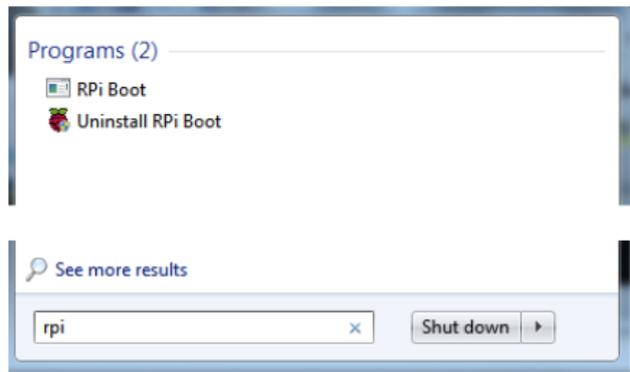
Connect the mini USB connector to the Windows PC using the supplied USB A to micro USB B data cable and then power up the unit.

Windows will then show the following stages as it configures the OS :

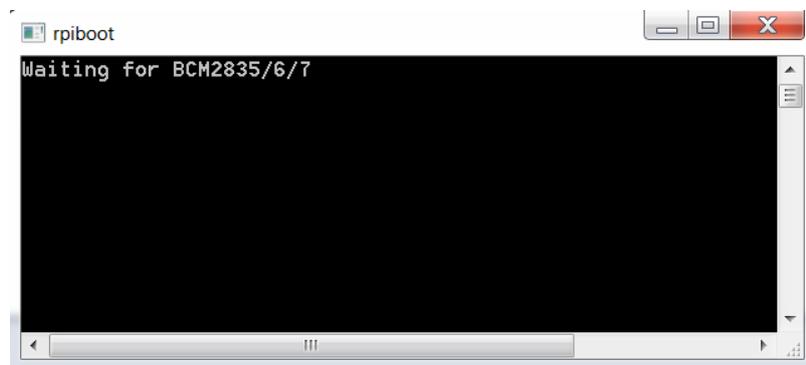


Once that sequence has finished Windows has now installed the required drivers and you can power off the unit for a moment whilst we get the PC side ready for the next step.

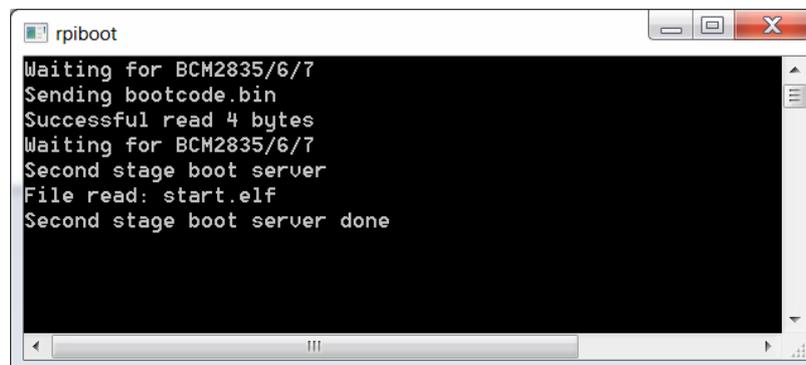
Making sure you have the unit powered off start up RPi Boot, this is easiest done via the start menu



When the RPi-Boot starts up it'll sit and wait for the attached board to boot up :



Power up the unit and RPi-Boot will configure the unit to appear as a flash drive :

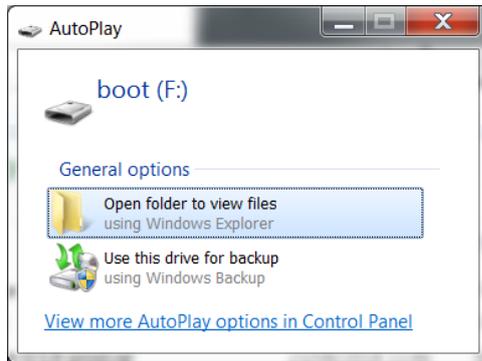


Note : Sometimes RPi-Boot doesn't correctly catch the board as it boots and as a result the re-configure sequence doesn't complete correctly. In this case the simplest thing to do is to switch the board off, close then restart RPi-Boot and then power the board back on to try again.

When done the compute module will alternate into mass storage mode (so behaving just as though it's a USB memory stick) and windows will then recognise the module as an external drive.

If the compute module eMMC already contains an OS Windows will recognise the FAT partition and assign that (at least) a drive letter, this is useful in the event that a configuration error with the boot files is made (e.g. dt-blob.bin or config.txt) and needs recovery actions to be performed.

After drive letter assignment Windows may indicate that partitions need scanning or fixing, these can be ignored/cancelled.



There are a few different ways we can load on the OS, for simplicity we'll cover using the recommended OS writing software and process from the main Raspberry Pi website

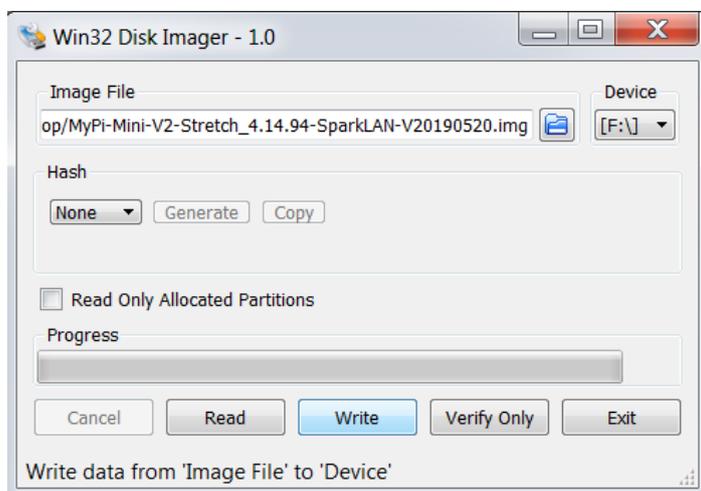
This process writes a disk image, containing the partition table as well as both FAT boot partition and Linux EXT partitions, **over the entire disk**.

The basic sequence we're following is :

1. Download the Win32DiskImager utility from this [Download Link](#)
2. Install and run the Win32DiskImager utility (You will need to run the utility as administrator, right-click on the shortcut or exe file and select **Run as administrator**)
3. Select the OS image file you wish to write
4. Select the drive letter of the compute module in the device box (in our case F:) - Again note that the disk image is a 1:1 of the entire disk (containing the partition table, FAT & EXT partitions)

Be careful to select the correct drive; if you get the wrong one you can destroy your data on the computer's hard disk!

5. Click **Write** and wait for the write to complete



RS485 Interface

The EdgeGateway unit features a fully isolated 2-wire/half-duplex RS485 interface connected to UART1 (ttyAMA0). This has automatic, transparent, baud rate independent hardware flow control built in to simplify application development.

For this example we will use two open-source projects

[MBPoll Github Page](#)

MBPoll is a command line utility useful for debugging Modbus connections, this can be compiled from source however a version has been pre-compiled and installed on the unit as shipped

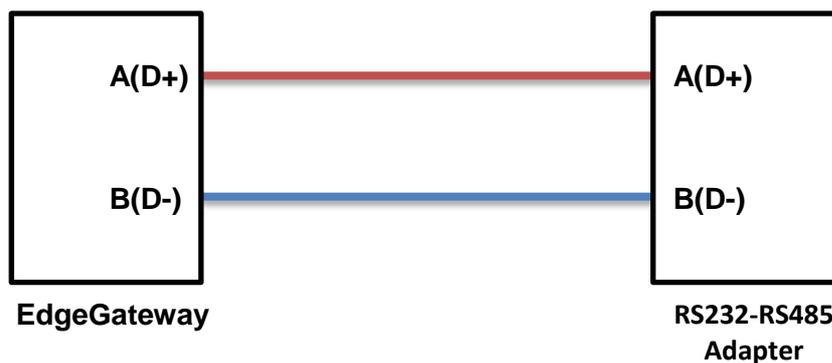
[Windows Modbus PLC Simulator](#)

This program provides a useful Modbus TCP/RTU PLC simulator, as downloaded this is a self-contained single EXE file, so does not contain an installer - take note of the requirement for a Visual C++ component if you have problems running the software.

You will need the following additional parts from the development kit and additionally some 2-Core cable to connect us the two parts

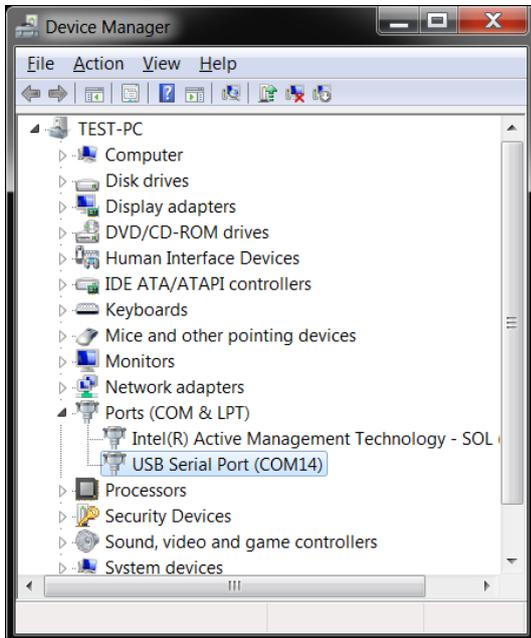
- RS485 to RS232 adapter
- USB to RS232 adapter

The two parts should be wired as below



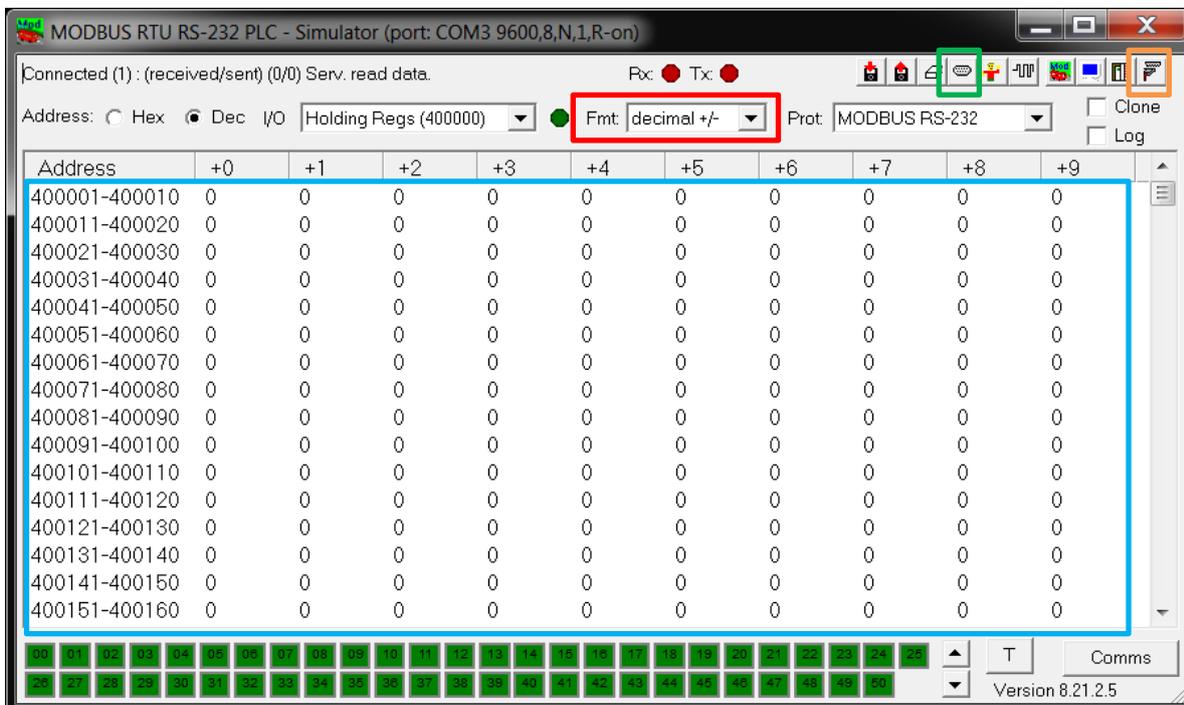
Note that in most cases the RS485 adapter will usually self-power from the USB-RS232 adapter so should not require an additional power feed.

With this setup plugged in the next step is to identify the COM port the USB-RS232 adapter has created. Opening Windows Control Panel take a note of the COM port Windows has initialised the FTDI USB Adapter to, in this case **COM14**



Next we should start up the PLC simulator,

Once loaded you'll be presented with a screen like the below, take note of the three highlighted areas:



- The **FMT Dialog Box** controls how the registers are shown/configured; we'll keep this at decimal +/- for the moment.
- The **COM Port selection box** controls the COM port & settings the simulator is connected to
- The **Cable & Port icon** controls whether the program is "plugged in" to the selected COM port
- The **Register Window** is where most of the action takes place here, in this area click on each of the first row and fill in to match the above.

Press F1 at any time for help/information on how the program works

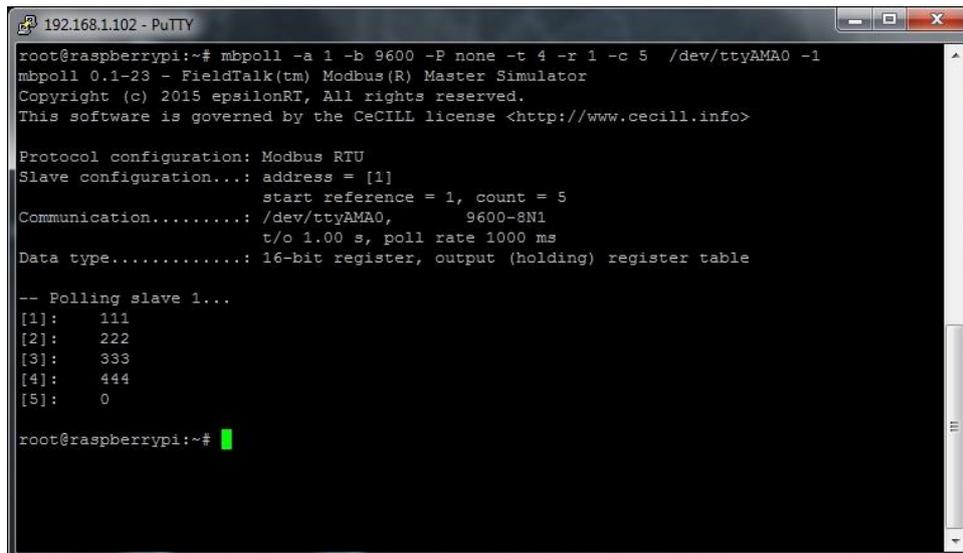
Configure the COM port settings on the PLC simulator to the USB COM port and keep the Baud rate settings the default as 9600 8N1.

Having logged into the EdgeGateway, on the command line run the following :

```
mbpoll -a 1 -b 9600 -P none -t 4 -r 1 -c 5 /dev/ttyAMA0 -1
```

Broken down, this will read from the device at address 1 (-a 1) using baud rate 9600 8N1 (-b 9600 -P none) the program will request 5 registers (-c 5) starting at register 1 (-r 1) using function code 4 to read the holding registers (-t 4). The -1 on the end means 'just make 1 poll and stop'

This is the output from a successful read:



```

192.168.1.102 - PuTTY
root@raspberrypi:~# mbpoll -a 1 -b 9600 -P none -t 4 -r 1 -c 5 /dev/ttyAMA0 -1
mbpoll 0.1-23 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2015 epsilonRT, All rights reserved.
This software is governed by the CeCILL license <http://www.cecill.info>

Protocol configuration: Modbus RTU
Slave configuration...: address = [1]
                        start reference = 1, count = 5
Communication.....: /dev/ttyAMA0, 9600-8N1
                        t/o 1.00 s, poll rate 1000 ms
Data type.....: 16-bit register, output (holding) register table

-- Polling slave 1...
[1]: 111
[2]: 222
[3]: 333
[4]: 444
[5]: 0

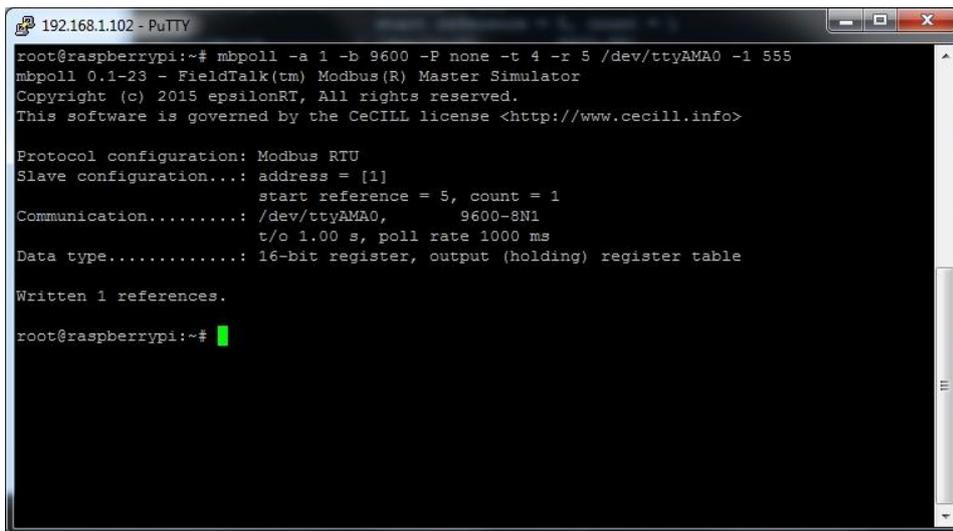
root@raspberrypi:~# █

```

Now we'll use the same program to write a register, the below will write 1 register value (555) starting at register 5 (-r 5) in address range 4 (-t 4)

```
mbpoll -a 1 -b 9600 -P none -t 4 -r 5 /dev/ttyAMA0 -1 555
```

This is the output from a successful write:



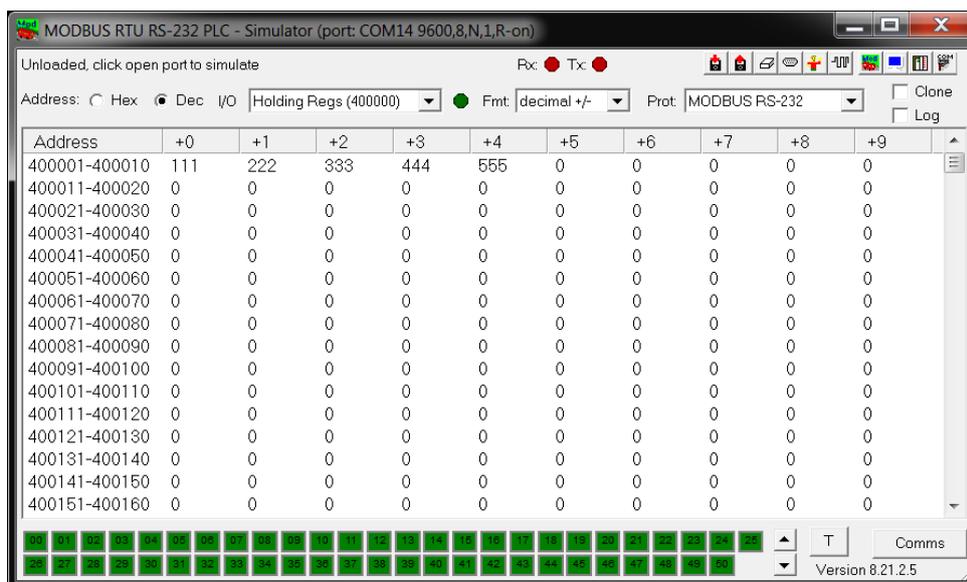
```
192.168.1.102 - PuTTY
root@raspberrypi:~# mbpoll -a 1 -b 9600 -P none -t 4 -r 5 /dev/ttyAMA0 -1 555
mbpoll 0.1-23 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2015 epsilonRT, All rights reserved.
This software is governed by the CeCILL license <http://www.cecill.info>

Protocol configuration: Modbus RTU
Slave configuration...: address = [1]
                      start reference = 5, count = 1
Communication.....: /dev/ttyAMA0, 9600-8N1
                    t/c 1.00 s, poll rate 1000 ms
Data type.....: 16-bit register, output (holding) register table

Written 1 references.

root@raspberrypi:~#
```

And checking the PLC Simulator program we can see the write have been done correctly



Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
400001-400010	111	222	333	444	555	0	0	0	0	0
400011-400020	0	0	0	0	0	0	0	0	0	0
400021-400030	0	0	0	0	0	0	0	0	0	0
400031-400040	0	0	0	0	0	0	0	0	0	0
400041-400050	0	0	0	0	0	0	0	0	0	0
400051-400060	0	0	0	0	0	0	0	0	0	0
400061-400070	0	0	0	0	0	0	0	0	0	0
400071-400080	0	0	0	0	0	0	0	0	0	0
400081-400090	0	0	0	0	0	0	0	0	0	0
400091-400100	0	0	0	0	0	0	0	0	0	0
400101-400110	0	0	0	0	0	0	0	0	0	0
400111-400120	0	0	0	0	0	0	0	0	0	0
400121-400130	0	0	0	0	0	0	0	0	0	0
400131-400140	0	0	0	0	0	0	0	0	0	0
400141-400150	0	0	0	0	0	0	0	0	0	0
400151-400160	0	0	0	0	0	0	0	0	0	0

This is a useful website to help with understanding the Modbus protocol :

<http://www.simplymodbus.ca/>

The mbpoll program uses an open source C library called libmodbus, which has a very permissive licence making it ideal for inclusion in any end application and short cutting development time.

<http://libmodbus.org/>

We have created a short C based Modbus logging application as a demonstration:

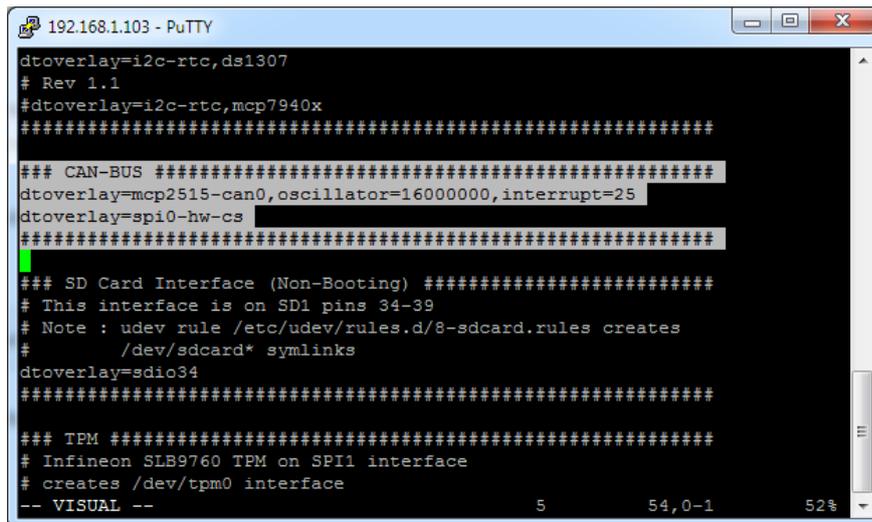
[LibModbus Demo Application](#)

CAN Interface

The EdgeGateway unit provides a fully isolated CAN 2.0b Interface using the popular Microchip MCP2515I controller chip, this is connected to the Raspberry Pi Compute module via the SPI-0 Bus.

The CAN interface is configured by the below files, these have been setup/installed as part of the shipped configuration

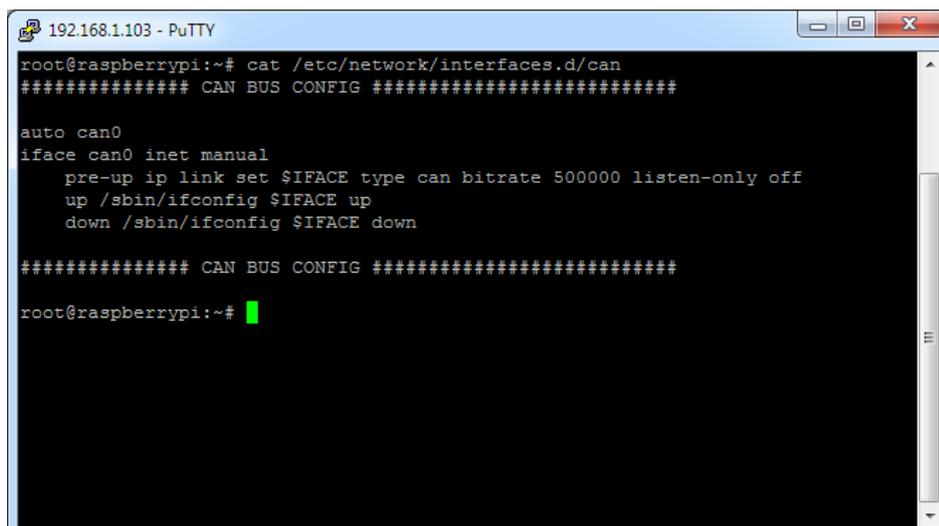
/boot/config.txt



```
192.168.1.103 - PuTTY
dtoverlay=i2c-rtc,ds1307
# Rev 1.1
#dtoverlay=i2c-rtc,mcp7940x
#####
### CAN-BUS #####
dtoverlay=mcp2515-can0,oscillator=1600000,interrupt=25
dtoverlay=spi0-hw-cs
#####
### SD Card Interface (Non-Booting) #####
# This interface is on SD1 pins 34-39
# Note : udev rule /etc/udev/rules.d/8-sdcard.rules creates
# /dev/sdcard* symlinks
dtoverlay=sdio34
#####
### TPM #####
# Infineon SLB9760 TPM on SPI1 interface
# creates /dev/tpm0 interface
-- VISUAL --          5          54,0-1          52
```

These settings should not need editing, they configure the interface at a basic hardware level and enable the required kernel modules

/etc/network/interfaces.d/can



```
192.168.1.103 - PuTTY
root@raspberrypi:~# cat /etc/network/interfaces.d/can
##### CAN BUS CONFIG #####

auto can0
iface can0 inet manual
    pre-up ip link set $IFACE type can bitrate 500000 listen-only off
    up /sbin/ifconfig $IFACE up
    down /sbin/ifconfig $IFACE down

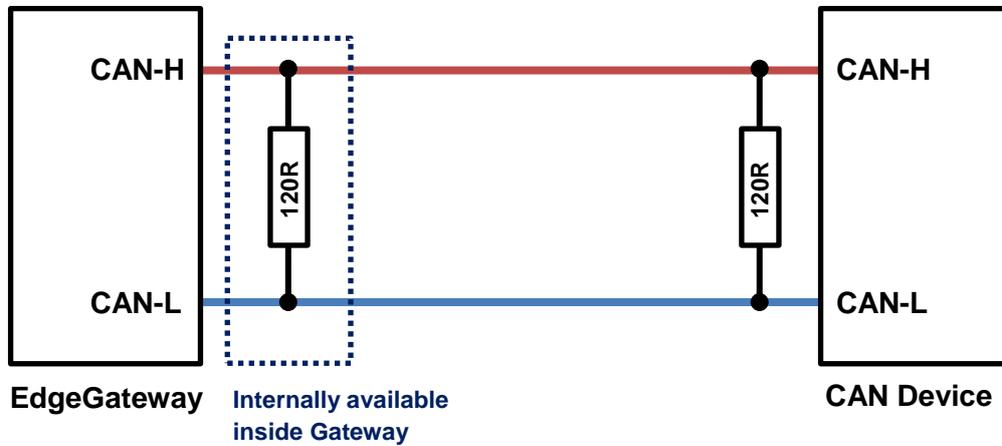
##### CAN BUS CONFIG #####

root@raspberrypi:~#
```

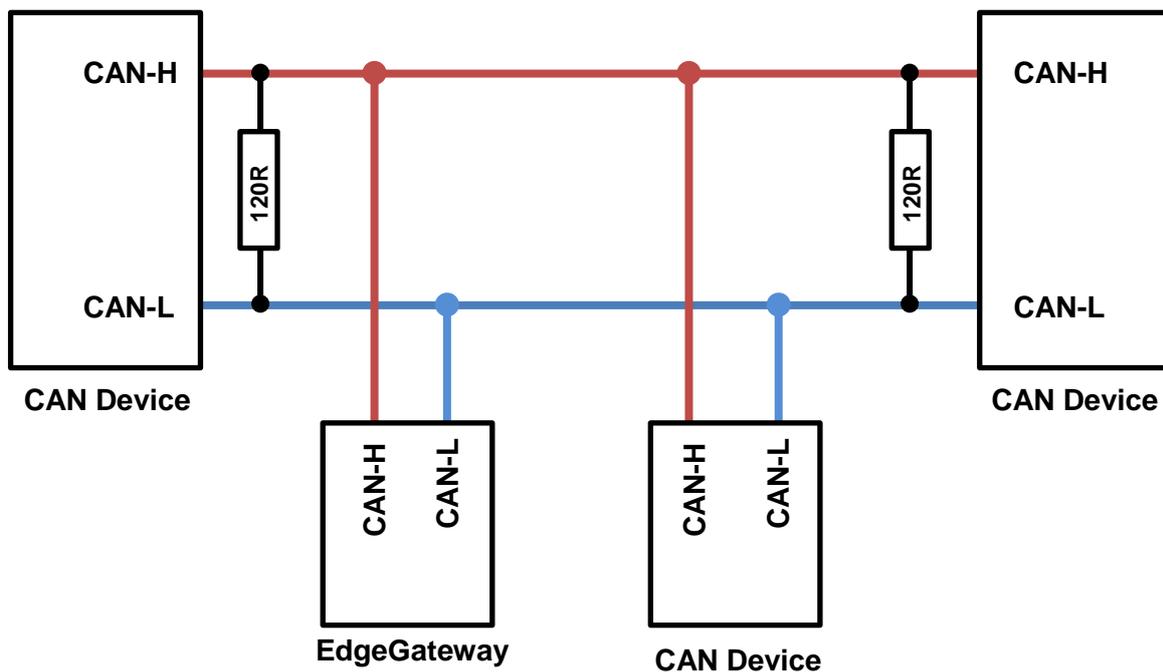
This file configures the bit rate along with whether the interface is read only or read/write and automatically brings up the interface at boot. (note that "listen-only off" setting means the system can read **and** write to the bus) :

Connecting into the CAN network, this diagram shows two example networks and where the EdgeGateway fits.

1 - Single CAN device - No existing network



2 - Multiple CAN Devices - Existing network



Remember to only connect the **CAN-H** and **CAN-L** lines to the bus, don't inter-connect the ground wire or swap the **H** and **L** lines, and only wire in the internal 120R termination resistor if required.

When connecting CAN controllers together then it is vital that the bus is terminated correctly or the network will not operate correctly.

If 120R termination resistors are used to terminate the network these should be positioned between the **H** and **L** lines at the furthest ends of the bus to correctly terminate the network, the network will not operate correctly without them.

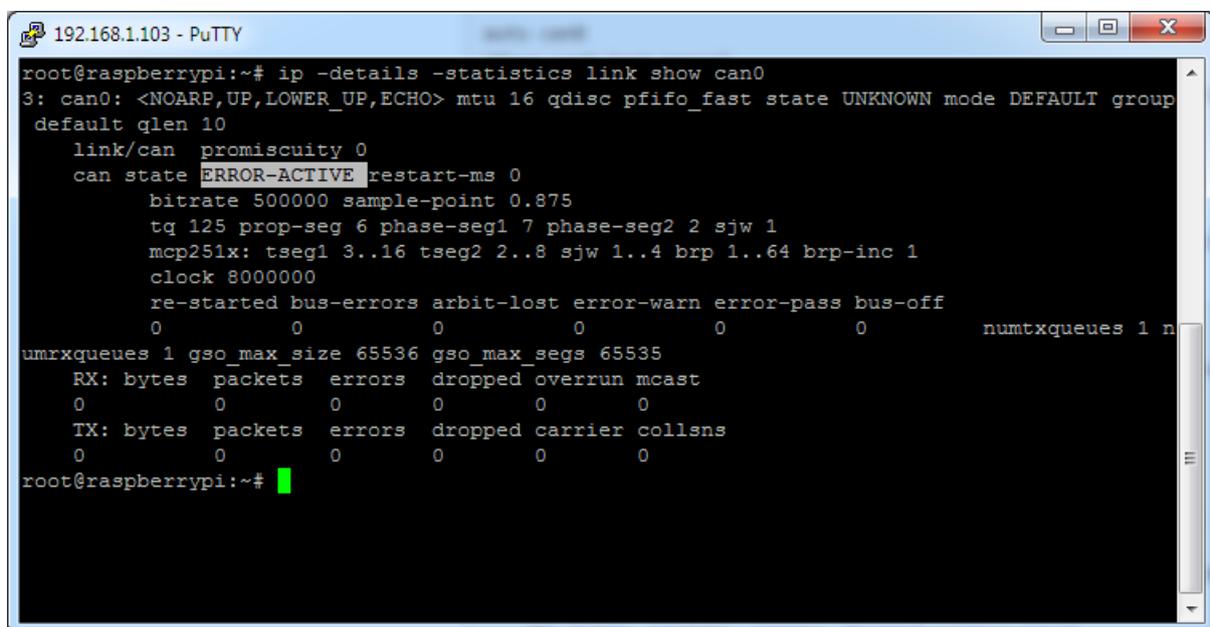
A quick resistance check done with the equipment powered off using a multi-meter across the **H** & **L** lines should give approximately 60 Ohms if all is wired up correctly.

When connecting a CAN-BUS network we strongly recommend using appropriate cabling e.g. Belden 9481 120 Ohm impedance twisted-pair wire (or equivalent)

Once the network is wired up correctly then we can proceed to checking the gateway setup

The command below will show the current CAN setup and give line statistics

```
ip -details -statistics link show can0
```



```
192.168.1.103 - PuTTY
root@raspberrypi:~# ip -details -statistics link show can0
3: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state UNKNOWN mode DEFAULT group
default qlen 10
  link/can  promiscuity 0
  can state  ERROR-ACTIVE restart-ms 0
             bitrate 500000 sample-point 0.875
             tq 125 prop-seg 6 phase-seg1 7 phase-seg2 2 sjw 1
             mcp251x: tseg1 3..16 tseg2 2..8 sjw 1..4 brp 1..64 brp-inc 1
             clock 8000000
             re-started bus-errors arbit-lost error-warn error-pass bus-off
             0 0 0 0 0 0 numtxqueues 1 n
umrxqueues 1 gso_max_size 65536 gso_max_segs 65535
  RX: bytes  packets  errors  dropped overrun mcast
     0 0 0 0 0 0
  TX: bytes  packets  errors  dropped carrier collsns
     0 0 0 0 0 0
root@raspberrypi:~#
```

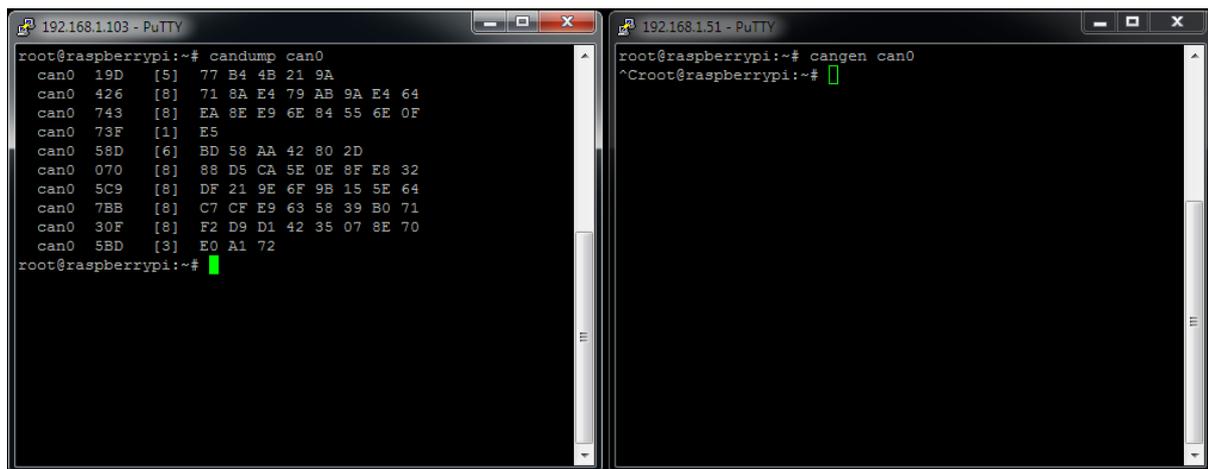
“can state” is the actual MCP2515 CAN controller state, so **ERROR-ACTIVE** is correct operation, **ERROR-PASSIVE** or **BUS-OFF** are signs something is wrong. For more information see the MCP2515 datasheet.

The SocketCAN can-utils package provides a number of programs for sending, receiving, monitoring and (selectively) logging data from the bus, as well as the ability to replay a log file:

```
asc2log, bcmserver, canbusload, can-calc-bit-timing, candump, canfdtest,
cangen, cangw, canlogserver, canplayer, cansend, cansniffer, isotpdump,
isotprecv, isotpperf, isotpsend, isotpsniffer, isotptun,
log2asc, log2long, slcan_attach, slcand and slcanpty.
```

Command line utilities **candump** and **cangen** can be useful for application development/debug, especially where no local CAN network exists as two Gateway units can be wired back-to-back to allow easy simulation

e.g.



```
192.168.1.103 - PuTTY
root@raspberrypi:~# candump can0
can0 19D [5] 77 B4 4B 21 9A
can0 426 [8] 71 8A E4 79 AB 9A E4 64
can0 743 [8] EA 8E E9 6E 84 55 6E 0F
can0 73F [1] E5
can0 58D [6] BD 58 AA 42 80 2D
can0 070 [8] 88 D5 CA 5E 0E 8F E8 32
can0 5C9 [8] DF 21 9E 6F 9B 15 5E 64
can0 7BB [8] C7 CF E9 63 58 39 B0 71
can0 30F [8] F2 D9 D1 42 35 07 8E 70
can0 5BD [3] E0 A1 72
root@raspberrypi:~# █

192.168.1.51 - PuTTY
root@raspberrypi:~# cangen can0
^Croot@raspberrypi:~# █
```

For more information on how the Linux Socket CAN interface works see links below :

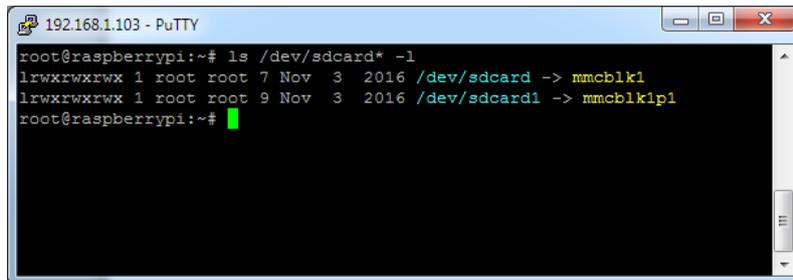
<https://www.kernel.org/doc/html/latest/networking/can.html>

<https://github.com/linux-can/can-utils/>

SD Card

The on-board micro SD Card is interfaced to the Raspberry Pi Compute Module using the secondary SDIO interface.

The configuration file `/etc/udev/rules.d/8-sdcard.rules` creates the below `/dev` shortcuts for the main SD Card and any partitions contained



```
192.168.1.103 - PuTTY
root@raspberrypi:~# ls /dev/sdcard* -l
lrwxrwxrwx 1 root root 7 Nov  3  2016 /dev/sdcard -> mmcblk1
lrwxrwxrwx 1 root root 9 Nov  3  2016 /dev/sdcard1 -> mmcblk1p1
root@raspberrypi:~#
```

This SD card cannot be booted from however can be auto mounted at boot (via `/etc/fstab`) so offers a low cost method of expanding the core eMMC filesystem

We recommend the use of industrial grade SD cards, which whist more expensive have greater operating temperature range, on-device wear-levelling and generally greater endurance than commercial grade parts.

For more information please see our knowledgebase article below

<https://embeddedpi.com/documentation/sd-card-interface/raspberry-pi-industrial-micro-sd-cards>

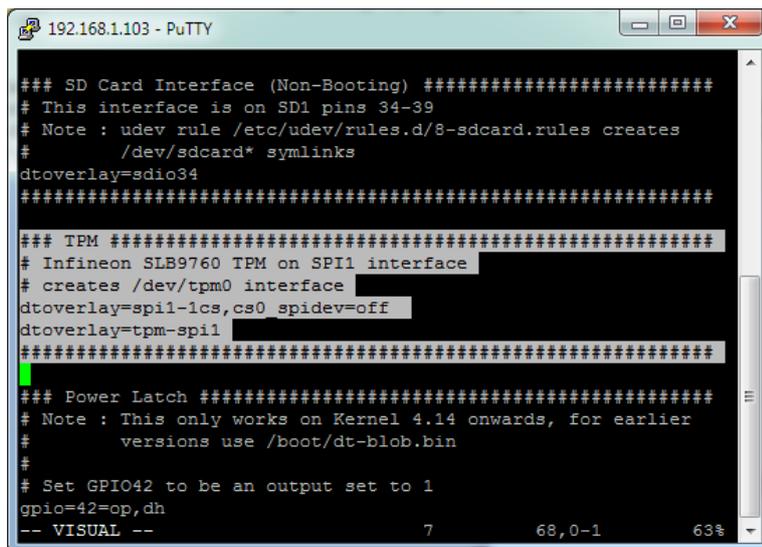
TPM 2.0 Device

Integrated on-board the EdgeGateway unit is an Infineon SLB9670 TPM 2.0 device, this is connected to the Raspberry Pi via the SPI-1 bus.

Support for this device was included in mainline Kernel 4.14.85 and the device is configured via the files below

/boot/config.txt

/boot/overlays/tpm-spi1.dtbo



```
192.168.1.103 - PuTTY
### SD Card Interface (Non-Booting) #####
# This interface is on SD1 pins 34-39
# Note : udev rule /etc/udev/rules.d/8-sdcard.rules creates
#       /dev/sdcard* symlinks
dtoverlay=sdio34
#####

### TPM #####
# Infineon SLB9670 TPM on SPI1 interface
# creates /dev/tpm0 interface
dtoverlay=spi1-1cs,cs0 spidev=off
dtoverlay=tpm-spi1
#####

### Power Latch #####
# Note : This only works on Kernel 4.14 onwards, for earlier
#       versions use /boot/dt-blob.bin
#
# Set GPIO42 to be an output set to 1
gpio=42=op,dh
-- VISUAL --                7          68,0-1          63%
```

This configures SPI-1 interface and the TPM, when the system has booted it will create a **/dev/tpm0** node.

We have pre-installed **tpmtool** and **tpmupdate** software utilities to allow for administration of the device

/usr/local/bin/tpminfo

/usr/local/bin/tpmtool

/root/tpm-toolkit/

For more information see the github repository below

<https://github.com/Infineon/eltt2>

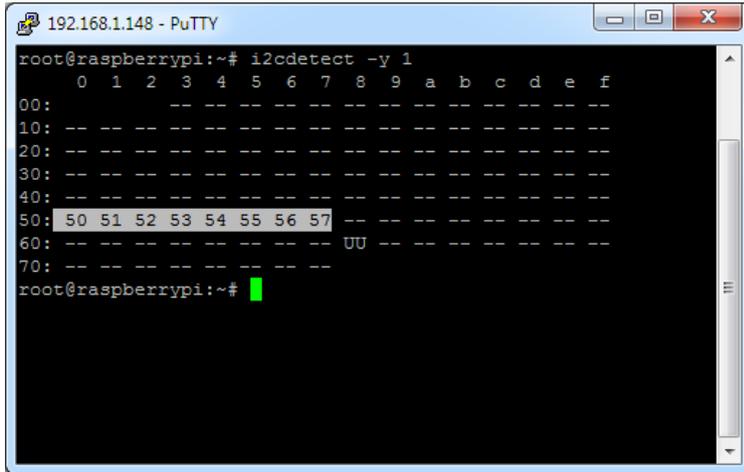
USER EEPROM

A 256Byte EEPROM for user ID storage

The lower 128Byte has read/write access for user storage, the first 4 hex bytes have been programmed with an ID code visible on the outside of the unit.

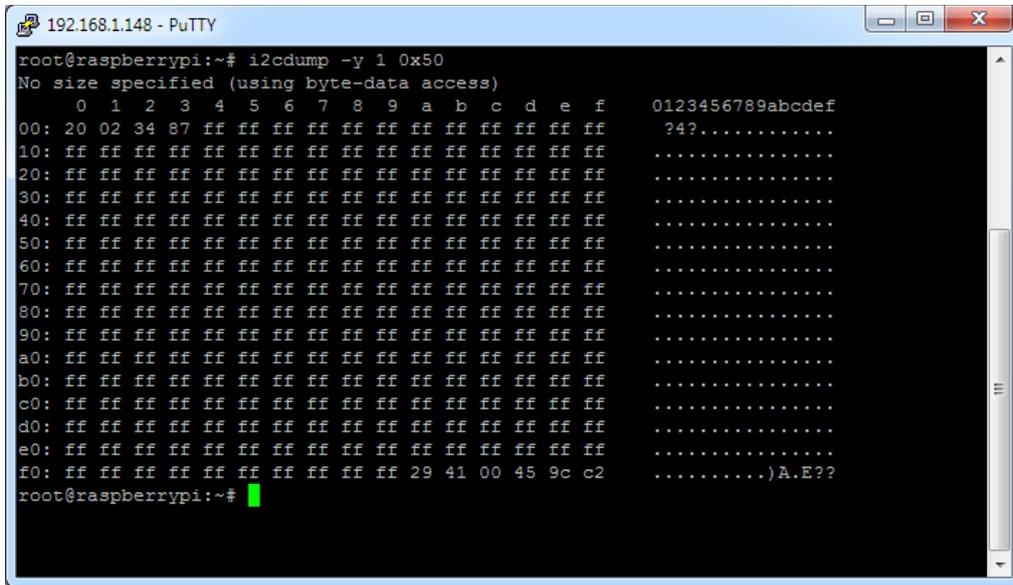
The upper 128byte is read only with the last 32bits (6 hex bytes) containing a unique ID code.

The EEPROM's id is 0x50 with shadow addresses at 0x51-0x57



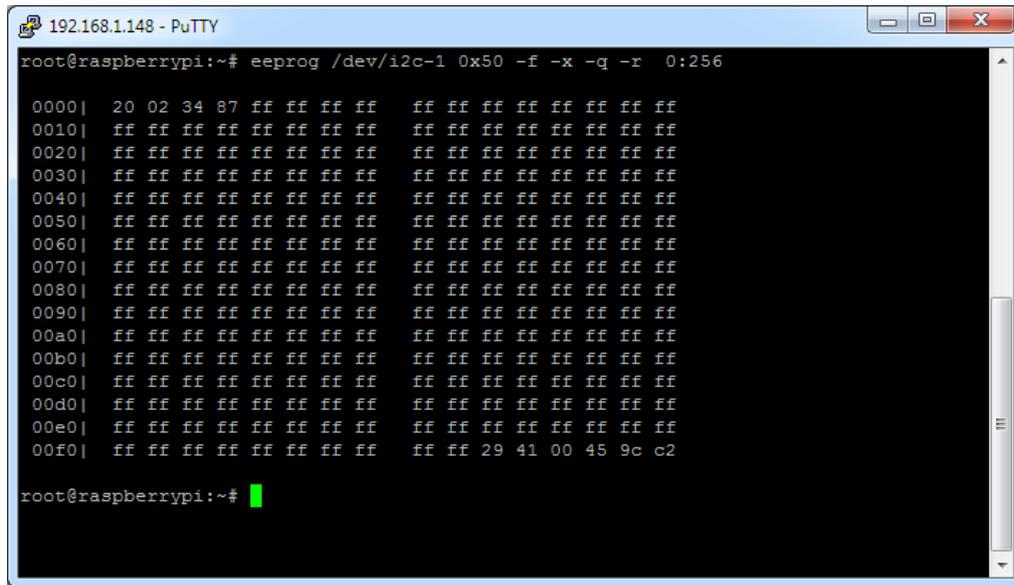
```
192.168.1.148 - PuTTY
root@raspberrypi:~# i2cdetect -y 1
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: 50 51 52 53 54 55 56 57 -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- UU -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- --
root@raspberrypi:~#
```

The EEPROM can be accessed for read/write operations using **i2c-tools** utilities, such as **i2cdump**



```
192.168.1.148 - PuTTY
root@raspberrypi:~# i2cdump -y 1 0x50
No size specified (using byte-data access)
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  0123456789abcdef
00: 20 02 34 87 ff  ?4?.....
10: ff  .....
20: ff  .....
30: ff  .....
40: ff  .....
50: ff  .....
60: ff  .....
70: ff  .....
80: ff  .....
90: ff  .....
a0: ff  .....
b0: ff  .....
c0: ff  .....
d0: ff  .....
e0: ff  .....
f0: ff 29 41 00 45 9c c2  .....)A.E??
root@raspberrypi:~#
```

Also included on the factory Raspbian OS image is the **eeprog** command line utility that can also be used to read/write the EEPROM (source code in **/root/eeprom**)



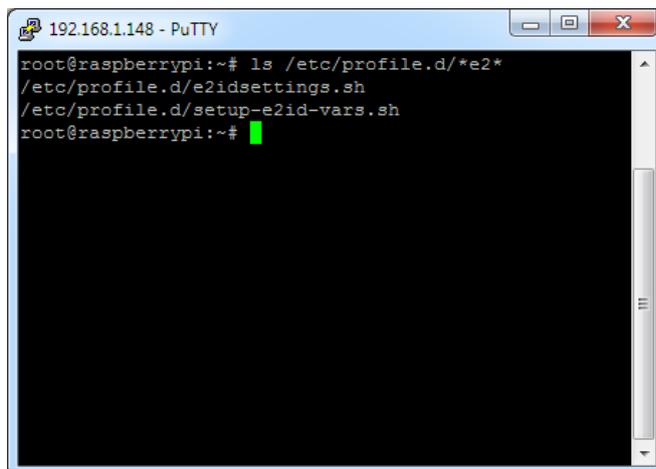
```
root@raspberrypi:~# eeprog /dev/i2c-1 0x50 -f -x -q -r 0:256

0000| 20 02 34 87 ff ff
0010| ff ff
0020| ff ff
0030| ff ff
0040| ff ff
0050| ff ff
0060| ff ff
0070| ff ff
0080| ff ff
0090| ff ff
00a0| ff ff
00b0| ff ff
00c0| ff ff
00d0| ff ff
00e0| ff ff
00f0| ff 29 41 00 45 9c c2

root@raspberrypi:~#
```

For convenience a script to create 2 Bash environment variables has been created in **/etc/profile.d**

setup-e2id-vars.sh creates **e2idsettings.sh** on first run

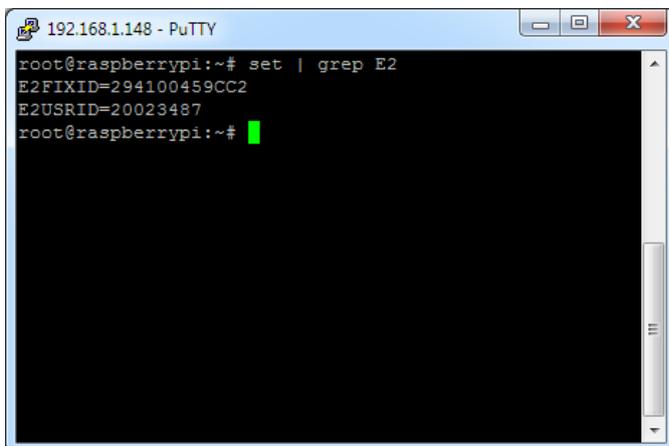


```
root@raspberrypi:~# ls /etc/profile.d/*e2*
/etc/profile.d/e2idsettings.sh
/etc/profile.d/setup-e2id-vars.sh
root@raspberrypi:~#
```

These environment variables can be used in scripting by any user



```
192.168.1.148 - PuTTY
root@raspberrypi:~# echo $E2USRID
20023487
root@raspberrypi:~# echo $E2FIXID
294100459CC2
root@raspberrypi:~#
```



```
192.168.1.148 - PuTTY
root@raspberrypi:~# set | grep E2
E2FIXID=294100459CC2
E2USRID=20023487
root@raspberrypi:~#
```

GPIO SYSTEM USAGE

The following table illustrates the system usage of the Pi Module GPIO Lines

All IO line signals pulled to inactive setting as default, in some cases the board is fitted with a matching resistor pull in the same direction as the default internal 50k pull.

GPIO	Pi PU/PD	Signal
2		I2C-SDA
3		I2C-SCL
4	PU	LED1-RED
5	PU	/CAN RESET
7		SPIO-CE1
8		SPIO-CE0
9		SPIO-MISO
10		SPIO-MOSI
11		SPIO-SCLK
14		UART-0 (ttyAMA0) TX – RS485
15		UART-0 (ttyAMA0) RX – RS485
16	PU	/TPM-RESET
17	PU	/TPM-PIRQ
18		SPI1-CE0
19		SPI1-MISO
20		SPI1-MOSI
21		SP1-CLK
22	PD	MPCIE 2 RESET
23	PD	MPCIE 2 WDIS
24	PD	SIERRA RESET**
25		CANIRQ
26	PU	/WD ENABLE
27	PU	/WD INPUT
32		UART-1 (ttyS0) TX – Console
33		UART-1 (ttyS0) RX – Console
34	PD	SD1-CLK
35	PU	SD1-CMD
36	PU	SD1-D0
37	PU	SD1-D1
38	PU	SD1-D2
39	PU	SD1-D3
40	PD	MPCIE 1 WDIS
41	PD	MPCIE 1 RESET
42	PD	POWER LATCH
43	PU	/POWER-ON
44	(PU)*	/LAN_RESET

* This line has an external pull up resistor only.

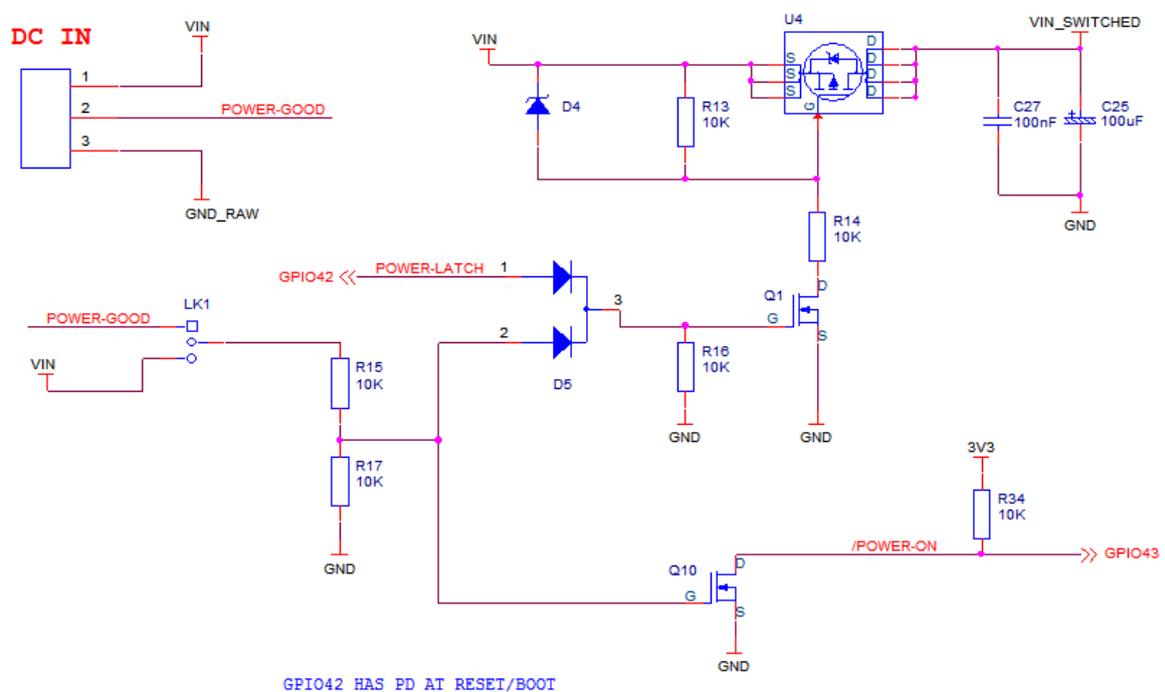
** Some Sierra Wireless models use an alternate reset pin to standard

POWER GATING OPERATION

The middle 'Power Good' input line on the main power in connector provides a mechanism to allow the system to control power up and graceful power down, this line can accept a wide voltage range from 6-30V DC.

This feature is intended to allow the unit to be used in an automotive environment where the unit can run from the raw battery power but be switched on by the usage of the accessory power line which is usually switched from the main ignition key.

The operation of this feature is as below, with reference to the below schematic extract:



With link LK1 set in position 1-2

The 'Power-Good' input from the main DC power in plug is fed, via the diode wired-OR gate created by D5, through to Q1 which controls whether the main power feed FET (U4) is ON or OFF.

A DC Input of 6-30V on 'Power-Good' input will enable the main power feed FET and the board will power up.

Built into the Raspberry Pi power up boot sequence is the ability to condition the state of GPIO lines very early on, usually within 3-5 seconds from power on, and well before the Linux kernel is loaded.

Using this facility we can effectively latch the main power feed FET on by driving GPIO42 to a logic high shortly after power up, this will keep the power ON even if the 'Power Good' input line is subsequently disconnected.

This can be achieved either by using a custom device tree file (`/boot/dt-blob.bin`), or using the GPIO overlay command in `/boot/config.txt` (`gpio=42=op,dh`) if using a recent Raspberry Pi kernel (4.14 onwards)

Additionally 'Power Good' line and so can also be monitored via GPIO43, this line is normally pulled high by R34 if GPIO43 is reading a 0 (logic low) then it can be determined that the 'Power Good' input voltage is present (due Q10 is pulling R34 Low)

If GPIO43 reads 1 (logic high) then it can be determined that the 'Power Good' input voltage has been disconnected or switched off.

So by a combination of using the 'Power Latch' (GPIO42) line to latch the power supply switch FET at start-up and monitoring 'Power Good' via (GPIO43) we can check for a power down/switch off event (e.g. Ignition key removed from car) and the system can be put into a controlled shutdown sequence (e.g. Linux command 'halt').

As the default state for the Power Latch (GPIO42) line is logic low the board will power off at the end of the standard Linux shutdown sequence in the same manner as a Desktop PC with an ATX power supply (assuming that Power-Good line remains low for the latter part of the shutdown sequence)

With LK1 set to positions 2-3

The 'Power Good' voltage input line is ignored and Q1 is instead fed from the main DC input power, as a result of this the system will only switch off once the main system power feed is removed.

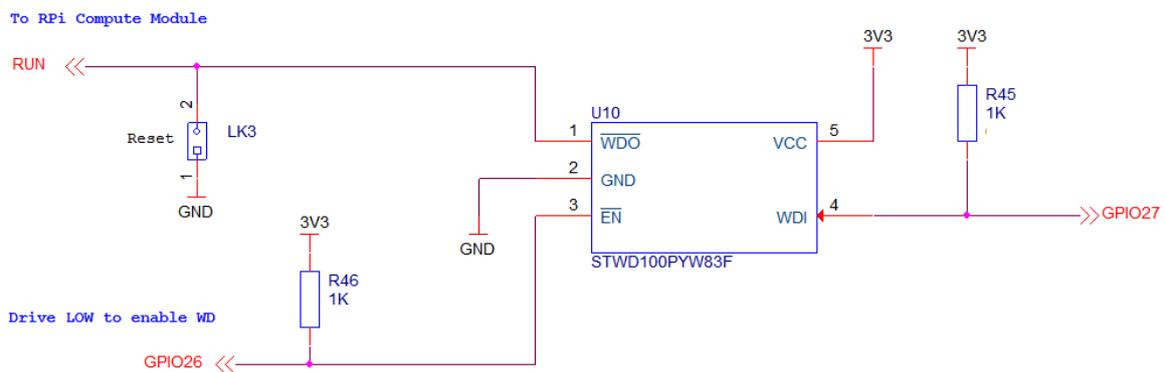
If the Linux command halt is used in this case the system will hang at the end of the shutdown sequence as it cannot cut off its own power feed due to 'Power-Good' being held at >3V.

WATCHDOG

The on-board external 1.6 second watchdog is a single chip part provided by ST STWD100PYW83F, the reset output of this part is connected to the Raspberry Pi Compute module.

This is provided to give an extra layer of resilience over a system lockup in the event that the user considers the RPi on-chip watchdog is unsuitable for their application.

The external watchdog device is driven by GPIO26 (/WD Enable) and GPIO27 (/WD Input), by default the watchdog is disabled.



Once the watchdog is enabled the WD Input pin on the device must be toggled H-L-H at least once per watchdog time-out period (1.6 seconds) and the low level pulse period must be >1 μ s long for the transition to be valid.

If the device sees a valid low-to-high transition on the input pin the internal 1.6 second countdown timer is reset and restarted. If the device does not see a valid input pulse within the watchdog time out period it will pull the RPi CPU module reset line low, which will also cause GPIO26 (/WD Enable) to be pulled high (as the Pi CPU resets) and so disable the watchdog allowing the system to boot.

With this in mind if the external watchdog is not used a hard reset of the Pi module can be effected by setting WD enable line high and then not toggling the watchdog input line.

The Reset lines for all other devices (including mPCIe) are available via separate, independent GPIO lines.

When the system hard resets in this manner all GPIO lines will revert back to their default state, which will have implications if the gated power input facility is in use.

External Watchdog OS Integration

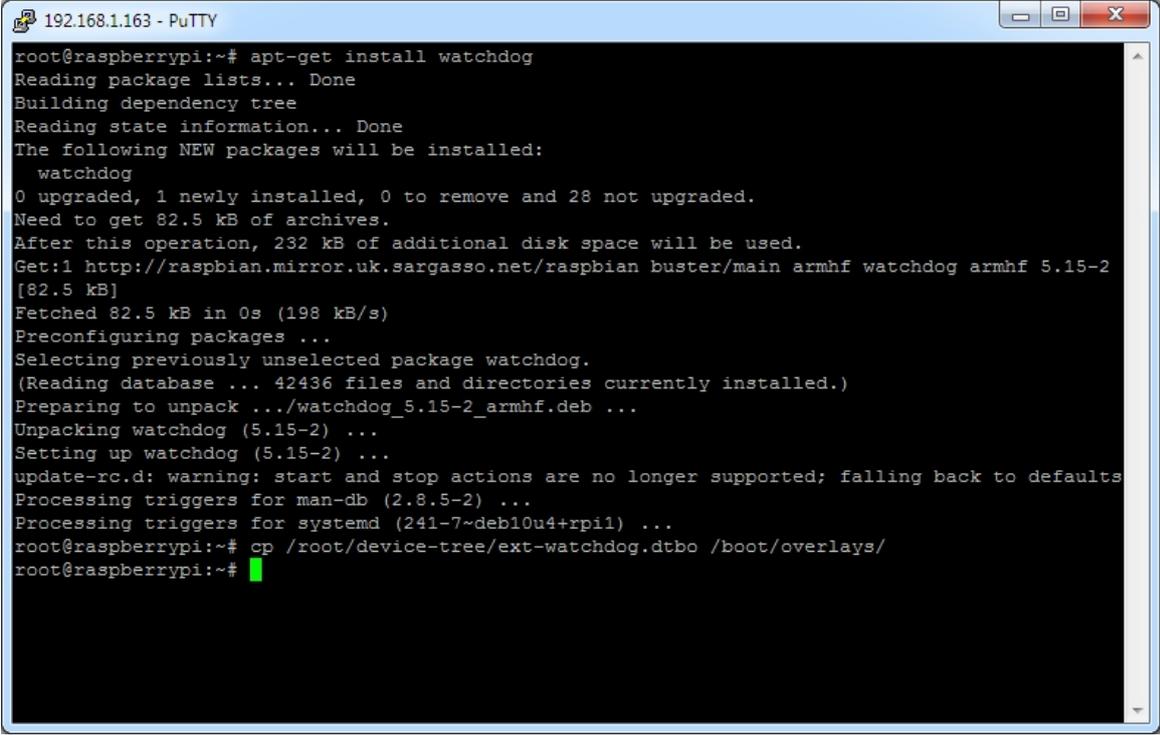
Integrated into the Raspbian kernel and OS there are pre-built utilities for configuring and managing watchdogs, in this example we will show how to configure the OS such that a file's last update timestamp will trigger a watchdog time out.

In this configuration if the target file is not updated the system will attempt an "orderly" reset as it performs some basic "clean-up" tasks prior to finally stopping the watchdog input line toggling, and so causing the Raspberry Pi Compute Module's reset line (aka RUN pin) to be momentarily pulled low by the watchdog device resulting in a hard reset.

The watchdog system is configured by altering 4 main files

- A device tree configuration file to enable the GPIO Watchdog timer **/dev/watchdog1**
- A systemd service file **/lib/systemd/system/watchdog.service**
- The conditional check options specified in **/etc/watchdog.conf**
- Edit and remove the watchdog section from **/etc/init.d/mypi.sh**

Start by installing the requisite files and configuring them



```
192.168.1.163 - PuTTY
root@raspberrypi:~# apt-get install watchdog
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  watchdog
0 upgraded, 1 newly installed, 0 to remove and 28 not upgraded.
Need to get 82.5 kB of archives.
After this operation, 232 kB of additional disk space will be used.
Get:1 http://raspbian.mirror.uk.sargasso.net/raspbian buster/main armhf watchdog armhf 5.15-2
[82.5 kB]
Fetched 82.5 kB in 0s (198 kB/s)
Preconfiguring packages ...
Selecting previously unselected package watchdog.
(Reading database ... 42436 files and directories currently installed.)
Preparing to unpack ../watchdog_5.15-2_armhf.deb ...
Unpacking watchdog (5.15-2) ...
Setting up watchdog (5.15-2) ...
update-rc.d: warning: start and stop actions are no longer supported; falling back to defaults
Processing triggers for man-db (2.8.5-2) ...
Processing triggers for systemd (241-7~deb10u4+rpil) ...
root@raspberrypi:~# cp /root/device-tree/ext-watchdog.dtbo /boot/overlays/
root@raspberrypi:~# █
```

Add the below line to the end of **/boot/config.txt**

```
dtoverlay=ext-watchdog
```

```
192.168.1.163
root@raspberrypi:~# cd /root/config_backups/ext-watchdog/
root@raspberrypi:~/config_backups/ext-watchdog# cp watchdog.conf /etc/
root@raspberrypi:~/config_backups/ext-watchdog# cp watchdog.service /lib/systemd/system/
cp: overwrite '/lib/systemd/system/watchdog.service'? y
root@raspberrypi:~/config_backups/ext-watchdog#
```

The watchdog service file has been altered as shown below to start the watchdog process, then enable the watchdog and during boot – **Ensure the same section has been edited out of /etc/init.d/myipi.sh to avoid clashing**

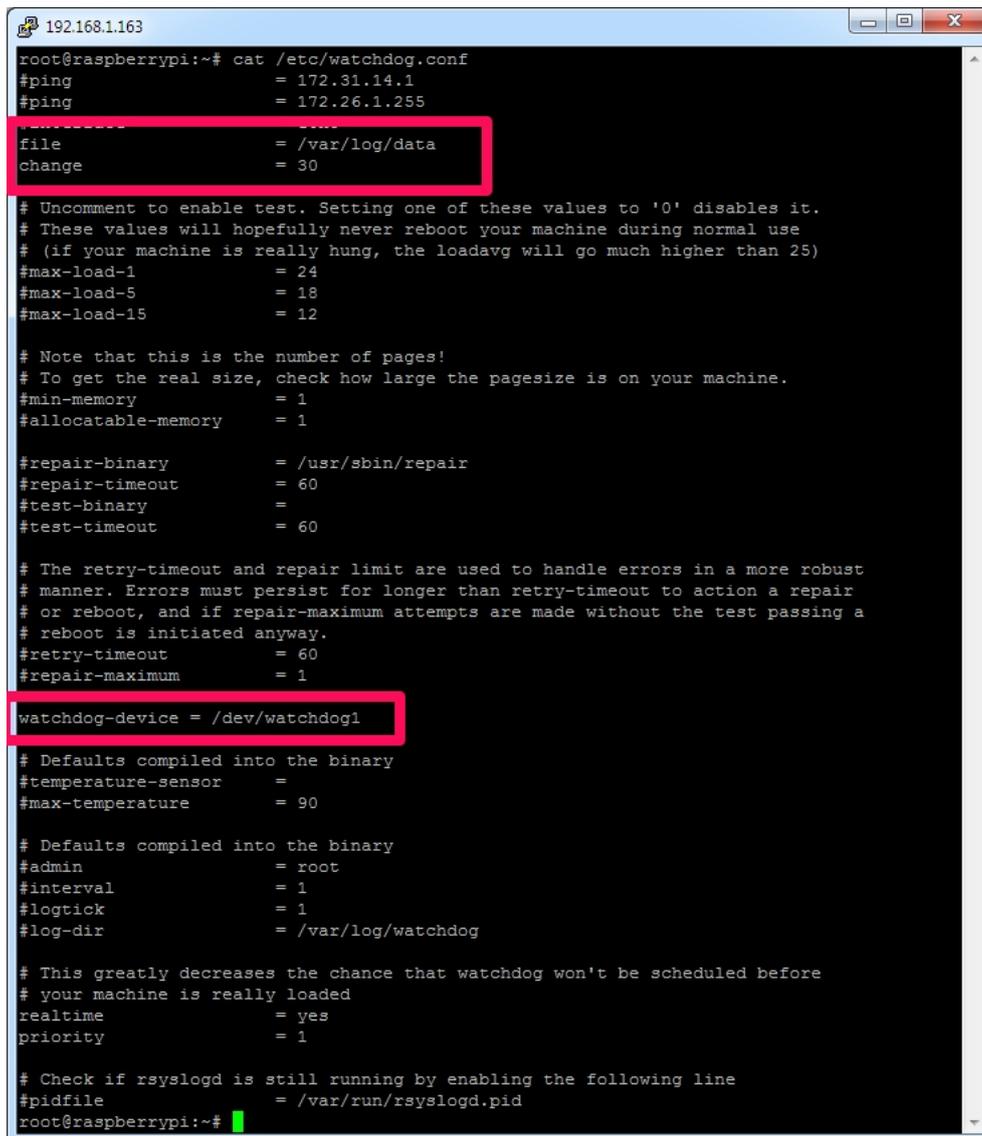
```
192.168.1.163
root@raspberrypi:~# cat /lib/systemd/system/watchdog.service
[Unit]
Description=watchdog daemon
Conflicts=wd_keepalive.service
After=multi-user.target
OnFailure=wd_keepalive.service

[Service]
Type=forking
EnvironmentFile=/etc/default/watchdog
ExecStartPre=/bin/sh -c 'touch /var/log/data'
ExecStart=/bin/sh -c '[ $run_watchdog != 1 ] || exec /usr/sbin/watchdog $watchdog_options'
ExecStartPost=/bin/sh -c 'echo 26 >/sys/class/gpio/export'
ExecStartPost=/bin/sh -c 'echo 1 >/sys/class/gpio/gpio26/active_low'
ExecStartPost=/bin/sh -c 'echo high >/sys/class/gpio/gpio26/direction'
ExecStartPost=/bin/sh -c 'echo 1 >/dev/wdog-enable'
ExecStopPost=/bin/sh -c '[ $run_wd_keepalive != 1 ] || false'
ExecStopPost=/bin/sh -c 'echo 0 >/dev/wdog-enable'

[Install]
WantedBy=default.target

root@raspberrypi:~#
```

The configuration we're using to determine both the watchdog device the system should be using and the test for system time out is setup in `/etc/watchdog.conf`



```
192.168.1.163
root@raspberrypi:~# cat /etc/watchdog.conf
#ping = 172.31.14.1
#ping = 172.26.1.255
file = /var/log/data
change = 30

# Uncomment to enable test. Setting one of these values to '0' disables it.
# These values will hopefully never reboot your machine during normal use
# (if your machine is really hung, the loadavg will go much higher than 25)
#max-load-1 = 24
#max-load-5 = 18
#max-load-15 = 12

# Note that this is the number of pages!
# To get the real size, check how large the pagesize is on your machine.
#min-memory = 1
#allocatable-memory = 1

#repair-binary = /usr/sbin/repair
#repair-timeout = 60
#test-binary =
#test-timeout = 60

# The retry-timeout and repair limit are used to handle errors in a more robust
# manner. Errors must persist for longer than retry-timeout to action a repair
# or reboot, and if repair-maximum attempts are made without the test passing a
# reboot is initiated anyway.
#retry-timeout = 60
#repair-maximum = 1

watchdog-device = /dev/watchdog1

# Defaults compiled into the binary
#temperature-sensor =
#max-temperature = 90

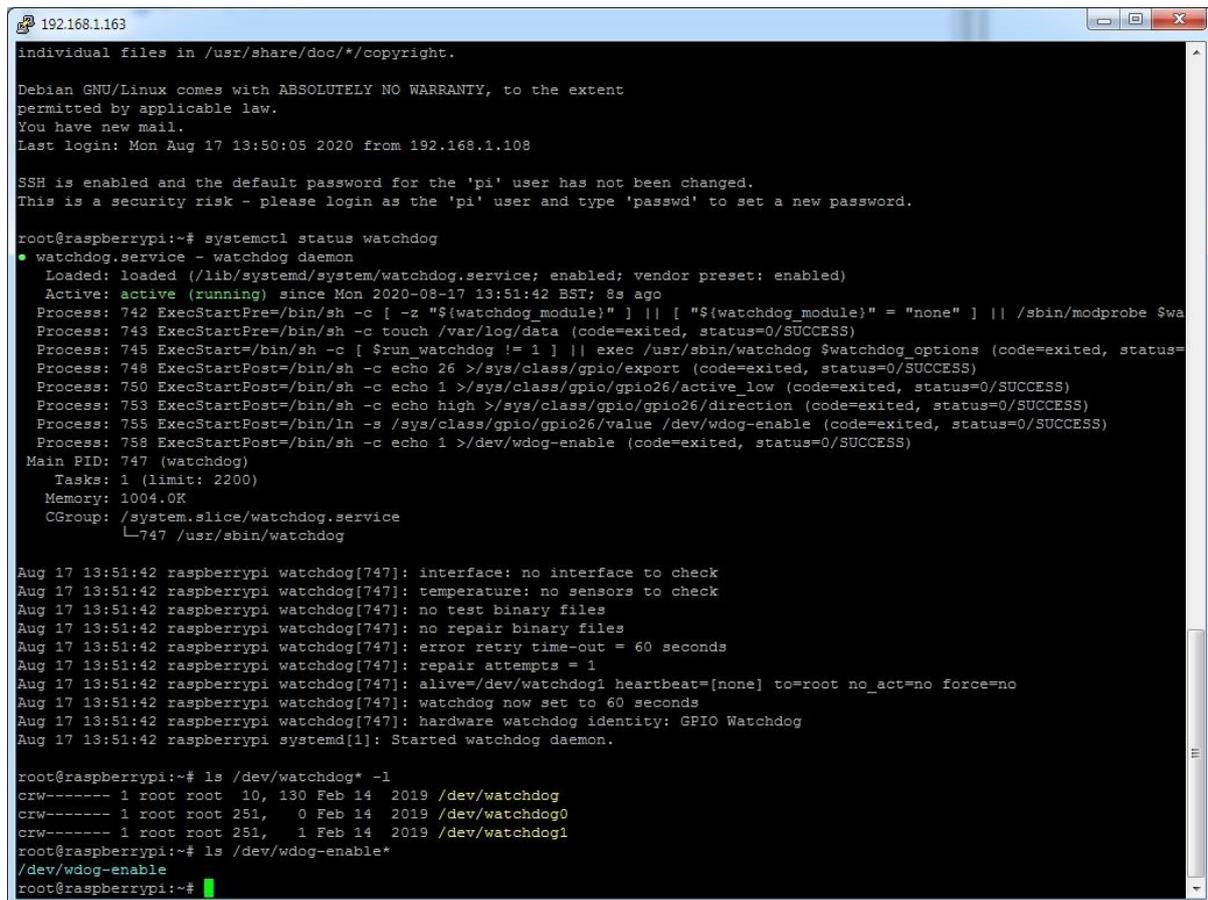
# Defaults compiled into the binary
#admin = root
#interval = 1
#logtick = 1
#log-dir = /var/log/watchdog

# This greatly decreases the chance that watchdog won't be scheduled before
# your machine is really loaded
realtime = yes
priority = 1

# Check if rsyslogd is still running by enabling the following line
#pidfile = /var/run/rsyslogd.pid
root@raspberrypi:~#
```

With these files in place reboot the unit so the changes take effect

On reboot you should be able to issue the commands shown below to check the services have started correctly.



```
192.168.1.163
Individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have new mail.
Last login: Mon Aug 17 13:50:05 2020 from 192.168.1.108

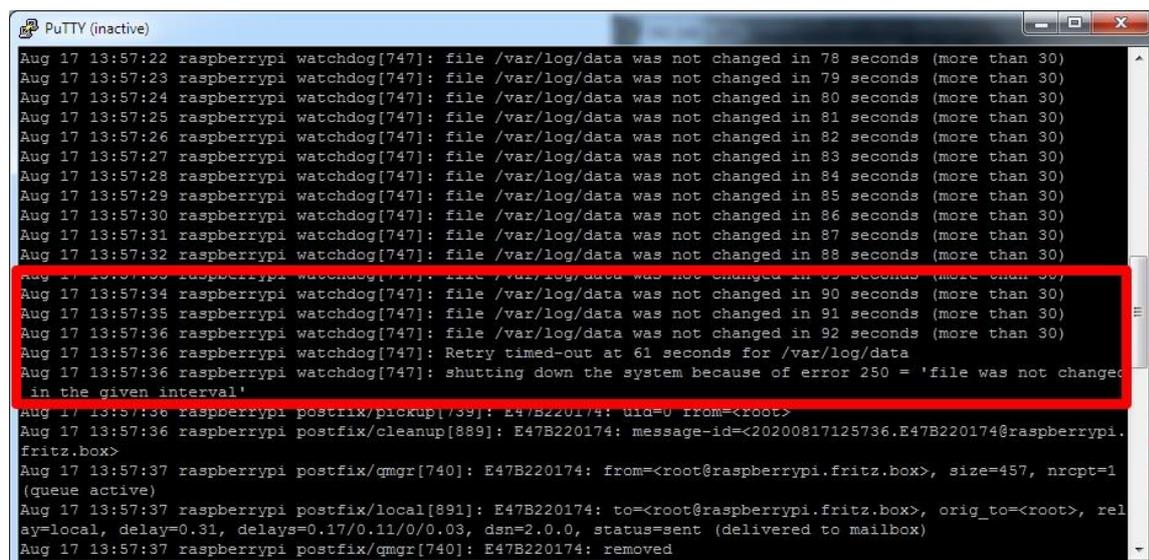
SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.

root@raspberrypi:~# systemctl status watchdog
● watchdog.service - watchdog daemon
   Loaded: loaded (/lib/systemd/system/watchdog.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2020-08-17 13:51:42 BST; 8s ago
     Process: 742 ExecStartPre=/bin/sh -c [ -z "${watchdog_module}" ] || [ "${watchdog_module}" = "none" ] || /sbin/modprobe $wa
     Process: 743 ExecStartPre=/bin/sh -c touch /var/log/data (code=exited, status=0/SUCCESS)
     Process: 745 ExecStart=/bin/sh -c [ $run_watchdog != 1 ] || exec /usr/sbin/watchdog $watchdog_options (code=exited, status=
     Process: 748 ExecStartPost=/bin/sh -c echo 26 >/sys/class/gpio/gpio26/export (code=exited, status=0/SUCCESS)
     Process: 750 ExecStartPost=/bin/sh -c echo 1 >/sys/class/gpio/gpio26/active_low (code=exited, status=0/SUCCESS)
     Process: 753 ExecStartPost=/bin/sh -c echo high >/sys/class/gpio/gpio26/direction (code=exited, status=0/SUCCESS)
     Process: 755 ExecStartPost=/bin/sh -s /sys/class/gpio/gpio26/value /dev/wdog-enable (code=exited, status=0/SUCCESS)
     Process: 758 ExecStartPost=/bin/sh -c echo 1 >/dev/wdog-enable (code=exited, status=0/SUCCESS)
   Main PID: 747 (watchdog)
     Tasks: 1 (limit: 2200)
     Memory: 1004.0K
     CGroup: /system.slice/watchdog.service
           └─747 /usr/sbin/watchdog

Aug 17 13:51:42 raspberrypi watchdog[747]: interface: no interface to check
Aug 17 13:51:42 raspberrypi watchdog[747]: temperature: no sensors to check
Aug 17 13:51:42 raspberrypi watchdog[747]: no test binary files
Aug 17 13:51:42 raspberrypi watchdog[747]: no repair binary files
Aug 17 13:51:42 raspberrypi watchdog[747]: error retry time-out = 60 seconds
Aug 17 13:51:42 raspberrypi watchdog[747]: repair attempts = 1
Aug 17 13:51:42 raspberrypi watchdog[747]: alive=/dev/watchdog1 heartbeat=[none] to=root no_act=no force=no
Aug 17 13:51:42 raspberrypi watchdog[747]: watchdog now set to 60 seconds
Aug 17 13:51:42 raspberrypi watchdog[747]: hardware watchdog identity: GPIO Watchdog
Aug 17 13:51:42 raspberrypi systemd[1]: Started watchdog daemon.

root@raspberrypi:~# ls /dev/watchdog* -l
crw----- 1 root root 10, 130 Feb 14 2019 /dev/watchdog
crw----- 1 root root 251, 0 Feb 14 2019 /dev/watchdog0
crw----- 1 root root 251, 1 Feb 14 2019 /dev/watchdog1
root@raspberrypi:~# ls /dev/wdog-enable*
/dev/wdog-enable
root@raspberrypi:~#
```

If the file we have configured as the test for watchdog time out is not written to for a period of 3 x the change value (in seconds) then the system will attempt a managed restart, by shutting as many services down as possible etc and then stopping the watchdog timer, causing a hard reset



```
Aug 17 13:57:22 raspberrypi watchdog[747]: file /var/log/data was not changed in 78 seconds (more than 30)
Aug 17 13:57:23 raspberrypi watchdog[747]: file /var/log/data was not changed in 79 seconds (more than 30)
Aug 17 13:57:24 raspberrypi watchdog[747]: file /var/log/data was not changed in 80 seconds (more than 30)
Aug 17 13:57:25 raspberrypi watchdog[747]: file /var/log/data was not changed in 81 seconds (more than 30)
Aug 17 13:57:26 raspberrypi watchdog[747]: file /var/log/data was not changed in 82 seconds (more than 30)
Aug 17 13:57:27 raspberrypi watchdog[747]: file /var/log/data was not changed in 83 seconds (more than 30)
Aug 17 13:57:28 raspberrypi watchdog[747]: file /var/log/data was not changed in 84 seconds (more than 30)
Aug 17 13:57:29 raspberrypi watchdog[747]: file /var/log/data was not changed in 85 seconds (more than 30)
Aug 17 13:57:30 raspberrypi watchdog[747]: file /var/log/data was not changed in 86 seconds (more than 30)
Aug 17 13:57:31 raspberrypi watchdog[747]: file /var/log/data was not changed in 87 seconds (more than 30)
Aug 17 13:57:32 raspberrypi watchdog[747]: file /var/log/data was not changed in 88 seconds (more than 30)
Aug 17 13:57:33 raspberrypi watchdog[747]: file /var/log/data was not changed in 89 seconds (more than 30)
Aug 17 13:57:34 raspberrypi watchdog[747]: file /var/log/data was not changed in 90 seconds (more than 30)
Aug 17 13:57:35 raspberrypi watchdog[747]: file /var/log/data was not changed in 91 seconds (more than 30)
Aug 17 13:57:36 raspberrypi watchdog[747]: file /var/log/data was not changed in 92 seconds (more than 30)
Aug 17 13:57:36 raspberrypi watchdog[747]: Retry timed-out at 61 seconds for /var/log/data
Aug 17 13:57:36 raspberrypi watchdog[747]: shutting down the system because of error 250 = 'file was not changed
in the given interval'
Aug 17 13:57:36 raspberrypi postfix/pickup[739]: E47B220174: uid=0 from=<root>
Aug 17 13:57:36 raspberrypi postfix/cleanup[889]: E47B220174: message-id=<20200817125736.E47B220174@raspberrypi.
fritz.box>
Aug 17 13:57:37 raspberrypi postfix/qmgr[740]: E47B220174: from=<root@raspberrypi.fritz.box>, size=457, nrcpt=1
(queue active)
Aug 17 13:57:37 raspberrypi postfix/local[891]: E47B220174: to=<root@raspberrypi.fritz.box>, orig_to=<root>, rel
ay=local, delay=0.31, delays=0.17/0.11/0/0.03, dsn=2.0.0, status=sent (delivered to mailbox)
Aug 17 13:57:37 raspberrypi postfix/qmgr[740]: E47B220174: removed
```

At any point up to this final time out writing/touching the file will reset the counter.

To test the system operation in the event of a kernel fault run the below to provoke a kernel panic

```
# echo c > /proc/sysrq-trigger
```

Alternately a recursive "fork bomb" which causes all CPU resources to be used can be provoked using the command below

```
# :(){ :|:& };:
```

mPCIe Compatibility

The mPCIe sockets installed on the base board are wired to the below standard :

Pin	Signal	Pin	Signal
1	-	2	3.3V
3	-	4	GND
5	-	6	1.5V
7	-	8	SIM_VCC
9	GND	10	SIM_I/O
11	-	12	SIM_CLK
13	-	14	SIM_RST
15	GND	16	SIM_VPP
Mechanical Key			
17	-	18	GND
19	-	20	WDIS# (GPIO23)
21	GND	22	PERST# (GPIO39)
23	-	24	3.3V
25	-	26	GND
27	GND	28	-
29	GND	30	-
31	-	32	-
33	-	34	GND
35	GND	36	USB_D+
37	-	38	USB_D-
39	3.3V	40	GND
41	3.3V	42	LED_WWAN#
43	GND	44	LED_WLAN#
45	-	46	-
47	-	48	-
49	-	50	GND
51	-	52	3.3V

For mPCIe-2 (Left hand side nearest SIM Socket) pins 42/43 are connected to the bottom Green LED driver.

For mPCIe-1 (Right hand side) pins 8/10/12/14/16 (SIM) are not connected

Pins 20/22 on both sockets are connected to GPIOs to provide software controllable access to Reset and Wireless Disable lines (see GPIO Table earlier)

For mPCIe-2 Pin 33 is connected to GPIO24 for Sierra Wireless Modems with non-standard reset lines.

Modem Compatibility/Operation

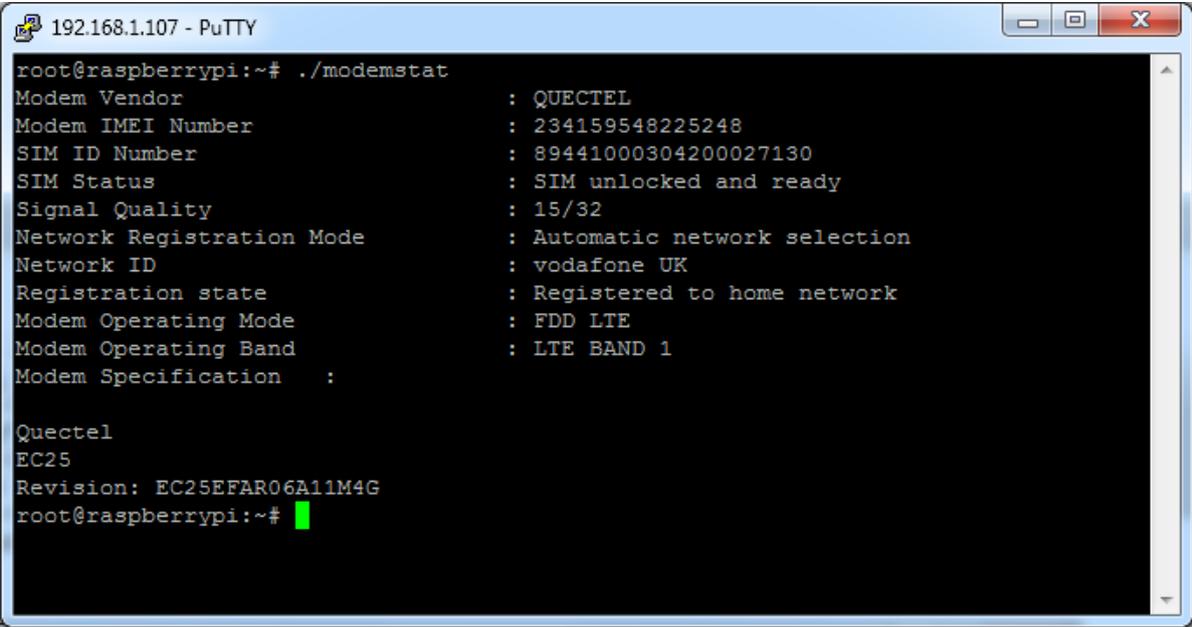
See the below link to pages from the main modem documentation section for details on how to operate modems :

<http://www.embeddedpi.com/documentation/3g-4g-modems>

The system has been pre-installed with modem helper status script **modemstat** which supports Sierra Wireless, Quectel and Simcom

See web page below for more details

<https://embeddedpi.com/documentation/3g-4g-modems/mypi-industrial-raspberry-pi-3g-4g-modem-status>



```
192.168.1.107 - PuTTY
root@raspberrypi:~# ./modemstat
Modem Vendor           : QUECTEL
Modem IMEI Number     : 234159548225248
SIM ID Number         : 89441000304200027130
SIM Status             : SIM unlocked and ready
Signal Quality         : 15/32
Network Registration Mode : Automatic network selection
Network ID             : vodafone UK
Registration state     : Registered to home network
Modem Operating Mode   : FDD LTE
Modem Operating Band   : LTE BAND 1
Modem Specification    :

Quectel
EC25
Revision: EC25EFAR06A11M4G
root@raspberrypi:~# █
```

A number of udev rules have been added to provide consistent shortcut symbolic links for easy identification of the various ttyUSB interfaces created by the modem. These udev rule files are contained in the **/etc/udev/rules.d/modem-rules** folder.

Note that increasingly modems are requiring **raw ip** connection method to be implemented, to this end we have added **qmi-network-raw** in **/usr/local/bin** which makes this connection type easier along with **udhcp** which supports raw ip mode for obtaining an IP address once connection has been made.

QMI Network Connection example :

```
192.168.1.107 - PuTTY
root@raspberrypi:~# modemstat
SIM status           : SIM unlocked and ready
Signal Quality       : 15/32   (Bit error rate cannot be determined)
Network Registration : Automatic network selection
Network ID           : "vodafone UK"
Registration state    : Registered to home network
GPRS/EDGE/UMTS/HSDPA Availability : FDD LTE
GPRS/EDGE/UMTS/HSDPA Mode Status : LTE BAND 1

Quectel
EC25
Revision: EC25EFAR06A11M4G

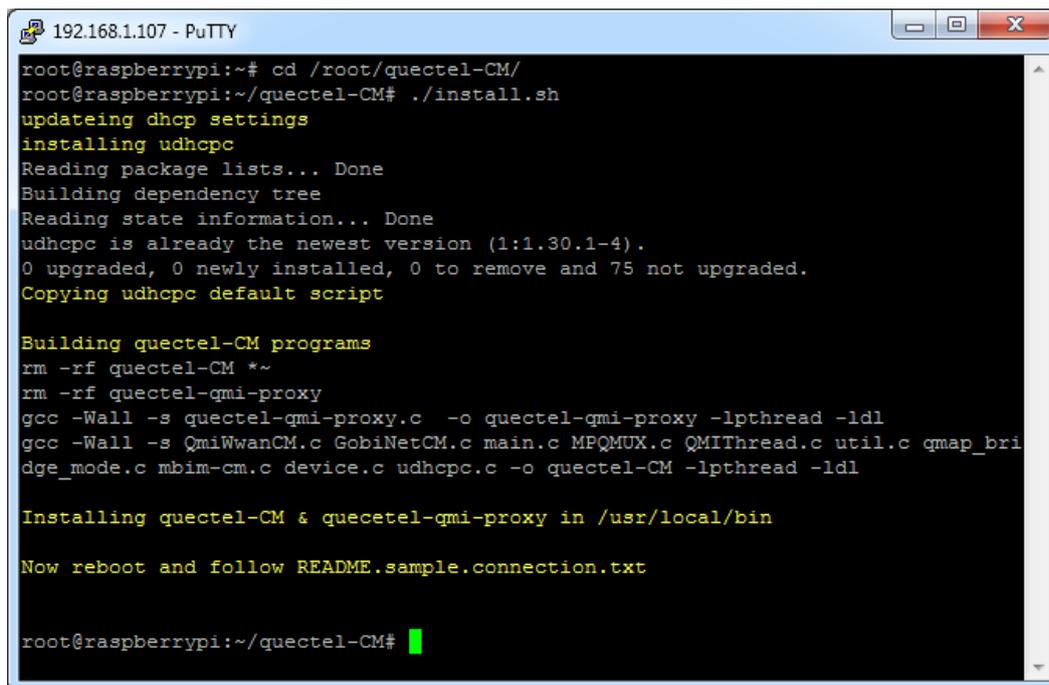
root@raspberrypi:~# ifconfig wwan0 down
root@raspberrypi:~# echo "APN=pp.vodafone.co.uk" >/etc/qmi-network.conf
root@raspberrypi:~# qmi-network-raw /dev/cdc-wdm0 start
Loading profile at /etc/qmi-network.conf...
  APN: pp.vodafone.co.uk
  APN user: unset
  APN password: unset
  qmi-proxy: no
Checking data format with 'qmicli -d /dev/cdc-wdm0 --wda-get-data-format '...
Device link layer protocol retrieved: raw-ip
Getting expected data format with 'qmicli -d /dev/cdc-wdm0 --get-expected-data-format'
...
Expected link layer protocol retrieved: raw-ip
Device and kernel link layer protocol match: raw-ip
Starting network with 'qmicli -d /dev/cdc-wdm0 --device-open-net=net-raw-ip|net-no-qos
-header --wds-start-network=apn='pp.vodafone.co.uk',ip-type=4 --client-no-release-cid
'...'
Saving state at /tmp/qmi-network-state-cdc-wdm0... (CID: 4)
Saving state at /tmp/qmi-network-state-cdc-wdm0... (PDH: 2267301456)
Network started successfully
root@raspberrypi:~# udhcpc -i wwan0
udhcpc: started, v1.30.1
No resolv.conf for interface wwan0.udhcpc
udhcpc: sending discover
udhcpc: sending select for 10.9.164.170
udhcpc: lease of 10.9.164.170 obtained, lease time 7200
Too few arguments.
Too few arguments.
root@raspberrypi:~# route -n
Kernel IP routing table
Destination      Gateway         Genmask        Flags Metric Ref    Use Iface
0.0.0.0          10.9.164.169   0.0.0.0        UG    0     0      0 wwan0
0.0.0.0          192.168.1.1    0.0.0.0        UG    202   0      0 eth0
10.9.164.168    0.0.0.0        255.255.255.252 U    0     0      0 wwan0
192.168.1.0     0.0.0.0        255.255.255.0  U    202   0      0 eth0
root@raspberrypi:~# █
```

QUECTEL-CM example

Quectel Modems have a utility provided by Quectel to manage the connection process and which will automatically configure any raw-ip settings

First install the all-in-one quectel-cm connection helper program; this will automatically configure any raw-ip settings

<https://github.com/mypiandrew/quectel-cm/releases/download/V1.6.0.12/quectel-CM.tar.gz>



```
192.168.1.107 - PuTTY
root@raspberrypi:~# cd /root/quectel-CM/
root@raspberrypi:~/quectel-CM# ./install.sh
updateing dhcp settings
installing udhcpd
Reading package lists... Done
Building dependency tree
Reading state information... Done
udhcpd is already the newest version (1:1.30.1-4).
0 upgraded, 0 newly installed, 0 to remove and 75 not upgraded.
Copying udhcpd default script

Building quectel-CM programs
rm -rf quectel-CM *~
rm -rf quectel-qmi-proxy
gcc -Wall -s quectel-qmi-proxy.c -o quectel-qmi-proxy -lpthread -ldl
gcc -Wall -s QmiWwanCM.c GobiNetCM.c main.c MPQMUX.c QMThread.c util.c qmap_bri
dge_mode.c mbim-cm.c device.c udhcpd.c -o quectel-CM -lpthread -ldl

Installing quectel-CM & quectel-qmi-proxy in /usr/local/bin

Now reboot and follow README.sample.connection.txt

root@raspberrypi:~/quectel-CM#
```

The command has the below syntax

```
quectel-CM [-s [apn [user password auth]]]
           [-p pincode] [-f logfilename] -s [apn [user password auth]]
```

Example 1: ./quectel-CM

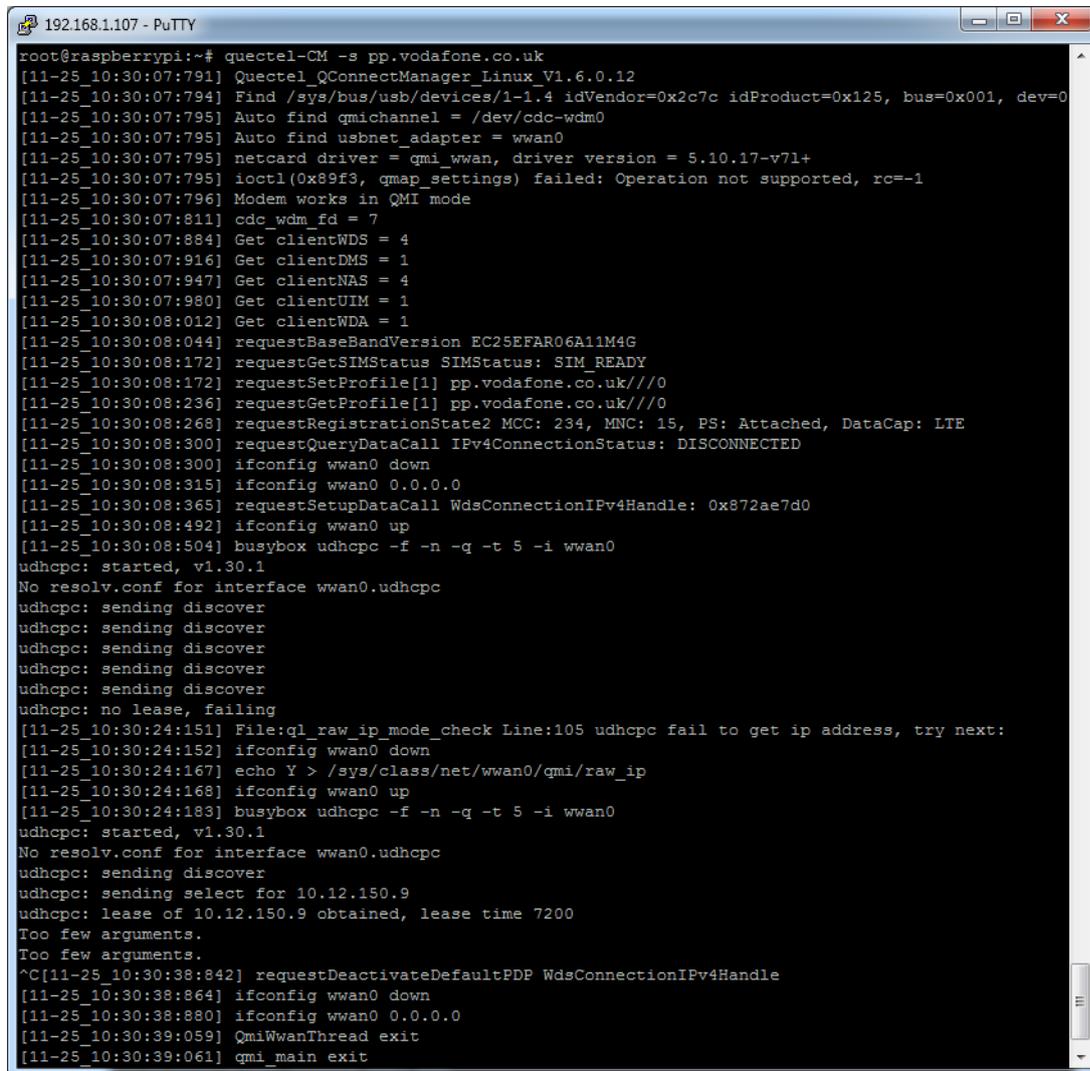
Example 2: ./quectel-CM -s pp.vodafone.co.uk

Example 3: ./quectel-CM -s internet web web 0 -p 1234 -f modemconnect.log

Note that this is a non-exiting process so will not automatically fork and run in the background

Sample Connection output, note the fall back to raw-ip is automatic.

Killing the process or issuing Ctrl-C results in the connection to be disconnected and interface disabled.



```
192.168.1.107 - PuTTY
root@raspberrypi:~# quectel-CM -s pp.vodafone.co.uk
[11-25_10:30:07:791] Quectel_QConnectManager_Linux_V1.6.0.12
[11-25_10:30:07:794] Find /sys/bus/usb/devices/1-1.4 idVendor=0x2c7c idProduct=0x125, bus=0x001, dev=0
[11-25_10:30:07:795] Auto find qmichannel = /dev/cdc-wdm0
[11-25_10:30:07:795] Auto find usbnet_adapter = wwan0
[11-25_10:30:07:795] netcard driver = qmi_wwan, driver version = 5.10.17-v71+
[11-25_10:30:07:795] ioctl(0x89f3, qmap_settings) failed: Operation not supported, rc=-1
[11-25_10:30:07:796] Modem works in QMI mode
[11-25_10:30:07:811] cdc_wdm_fd = 7
[11-25_10:30:07:884] Get clientWDS = 4
[11-25_10:30:07:916] Get clientDMS = 1
[11-25_10:30:07:947] Get clientNAS = 4
[11-25_10:30:07:980] Get clientUIM = 1
[11-25_10:30:08:012] Get clientWDA = 1
[11-25_10:30:08:044] requestBaseBandVersion EC25EFAR06A11M4G
[11-25_10:30:08:172] requestGetSIMStatus SIMStatus: SIM_READY
[11-25_10:30:08:172] requestSetProfile[1] pp.vodafone.co.uk///0
[11-25_10:30:08:236] requestGetProfile[1] pp.vodafone.co.uk///0
[11-25_10:30:08:268] requestRegistrationState2 MCC: 234, MNC: 15, PS: Attached, DataCap: LTE
[11-25_10:30:08:300] requestQueryDataCall IPv4ConnectionStatus: DISCONNECTED
[11-25_10:30:08:300] ifconfig wwan0 down
[11-25_10:30:08:315] ifconfig wwan0 0.0.0.0
[11-25_10:30:08:365] requestSetupDataCall WdsConnectionIPv4Handle: 0x872ae7d0
[11-25_10:30:08:492] ifconfig wwan0 up
[11-25_10:30:08:504] busybox udhcpd -f -n -q -t 5 -i wwan0
udhcpd: started, v1.30.1
No resolv.conf for interface wwan0.udhcpd
udhcpd: sending discover
udhcpd: no lease, failing
[11-25_10:30:24:151] File:ql_raw_ip_mode_check Line:105 udhcpd fail to get ip address, try next:
[11-25_10:30:24:152] ifconfig wwan0 down
[11-25_10:30:24:167] echo Y > /sys/class/net/wwan0/qmi/raw_ip
[11-25_10:30:24:168] ifconfig wwan0 up
[11-25_10:30:24:183] busybox udhcpd -f -n -q -t 5 -i wwan0
udhcpd: started, v1.30.1
No resolv.conf for interface wwan0.udhcpd
udhcpd: sending discover
udhcpd: sending select for 10.12.150.9
udhcpd: lease of 10.12.150.9 obtained, lease time 7200
Too few arguments.
Too few arguments.
^C[11-25_10:30:38:842] requestDeactivateDefaultPDP WdsConnectionIPv4Handle
[11-25_10:30:38:864] ifconfig wwan0 down
[11-25_10:30:38:880] ifconfig wwan0 0.0.0.0
[11-25_10:30:39:059] QmiWwanThread exit
[11-25_10:30:39:061] qmi_main exit
```

mPCIe IO Cards

Also available are our range of pre-certified RF modules :

- LoRa (Microchip RN2483/RN2903)
- Bluetooth 4.0 BLE (Silicon Labs/BlueGiga BLE112)
- Bluetooth 5 (Laird BL652)
- enOcean TCM310
- ZIGBEE/802.15.4 (Silicon Labs/Telegesis RX357 Module L.R. UFL)
- XBEE

These all feature an FTDI230X USB to UART chip and so appear automatically as a standard serial port ready to run with minimal configuration needed, so offer a fast development cycle.

In order to make the **ttyUSBx** serial port for the mPCIe cards above constantly easy to identify we use a udev rule to help us, this is called **10-ftdi-usbserial.rules** and is located **/etc/udev/rules.d/**

This udev rule creates a symlink for the FTDI ttyUSBx serial port called **/dev/ttyS1**

For more information on how each card works please see the respective documentation page on the website.

OS CONFIGURATION FILES

There are a handful of files needed to configure the MyPi-Mini from the base Raspbian install, these are available from this link:

<https://drive.google.com/file/d/1FZLFnooJoLJ8ERhm10IsipobfFo0XbKa>

Filename	File location	Description
config.txt	/boot	System Configuration File
cmdline.txt	/boot	Kernel boot command line options
dt-blob.bin	/boot	Custom Device tree File to latch GPIO42 high at startup (alternate to using config.txt on pre-4.14 kernels)
mypi.sh	/etc/init.d	Bash Script to configure Linux OS GPIO exports for command line use & create /dev shortcuts.
sdio34.dtbo	/boot/overlays	Enable secondary SD Card interface (SD1)
tpm-spi1.dtbo	/boot/overlays	Enable TPM2.0 on SPI-1 interface
(multiple)	/etc/udev/rules.d	Install udev rules to create /dev shortcuts for modem, SD Card and mPCIe IO cards
setup-e2id-vars.sh	/etc/profile.d/	Reads ID EEPROM and configures bash environment variables \$E2FIXID & \$E2USRID

config.txt controls which on-board peripherals are enabled, there should be no need to alter this unless the user wants to alter CPU speed for optimising thermal/power operation.

cmdline.txt this controls what kernel command line options are enabled at boot, again there should be no need to alter this from the default unless you wish to disable the serial console.

dt-blob.bin by placing this in /boot/ this overrides the default device tree file for the CM3/3+, the only difference being that GPIO42 is pulled high immediately at start-up (and before the OS is loaded) to latch the power switch FET on.

mypi.sh creates the below GPIO exports and shortcuts, the demo image has this enabled to run during the startup sequence

```
can-reset           -> /sys/class/gpio/gpio5/value
lan-disable         -> /sys/class/gpio/gpio44/value
mPCIe1-reset       -> /sys/class/gpio/gpio41/value
mPCIe1-wdisble     -> /sys/class/gpio/gpio40/value
mPCIe2-reset       -> /sys/class/gpio/gpio22/value
mPCIe2-sierra-reset -> /sys/class/gpio/gpio24/value
mPCIe2-wdisble     -> /sys/class/gpio/gpio23/value
power-good         -> /sys/class/gpio/gpio43/value
power-latch        -> /sys/class/gpio/gpio42/value
wd-enable          -> /sys/class/gpio/gpio26/value
wd-input           -> /sys/class/gpio/gpio27/value
rtc_nvram          -> /sys/class/rtc/rtc0/device/ds1307_nvram0/nvram
```

Example usage :

```
echo 1 >/dev/lan-disable
```

```
echo 1 >/dev/mpciel-reset
```

```
echo 0 >/dev/mpciel-reset
```

```
cat /dev/power-good
```

```
0
```

```
echo 'battery backed up ram' > /dev/rtc_nvram
```

```
cat /dev/rtc_nvram
```

```
battery backed up ram
```

For more information load the **mypi.sh** file into a text editor.

FCC Class A Statement

This equipment has been tested and complies with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case the user will be required to correct the interference at his own expense.

Properly shielded and grounded cables and connectors must be used in order to meet FCC emission limits. Embedded Micro Technology is not responsible for any radio or television interference caused by using other than recommended cables and connectors or by unauthorized changes or modifications to this equipment. Unauthorized changes or modifications could void the user's authority to operate the equipment.

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation