# MyPi Industrial CM4 Integrator Board SE

# User Guide

Issue : 1.0

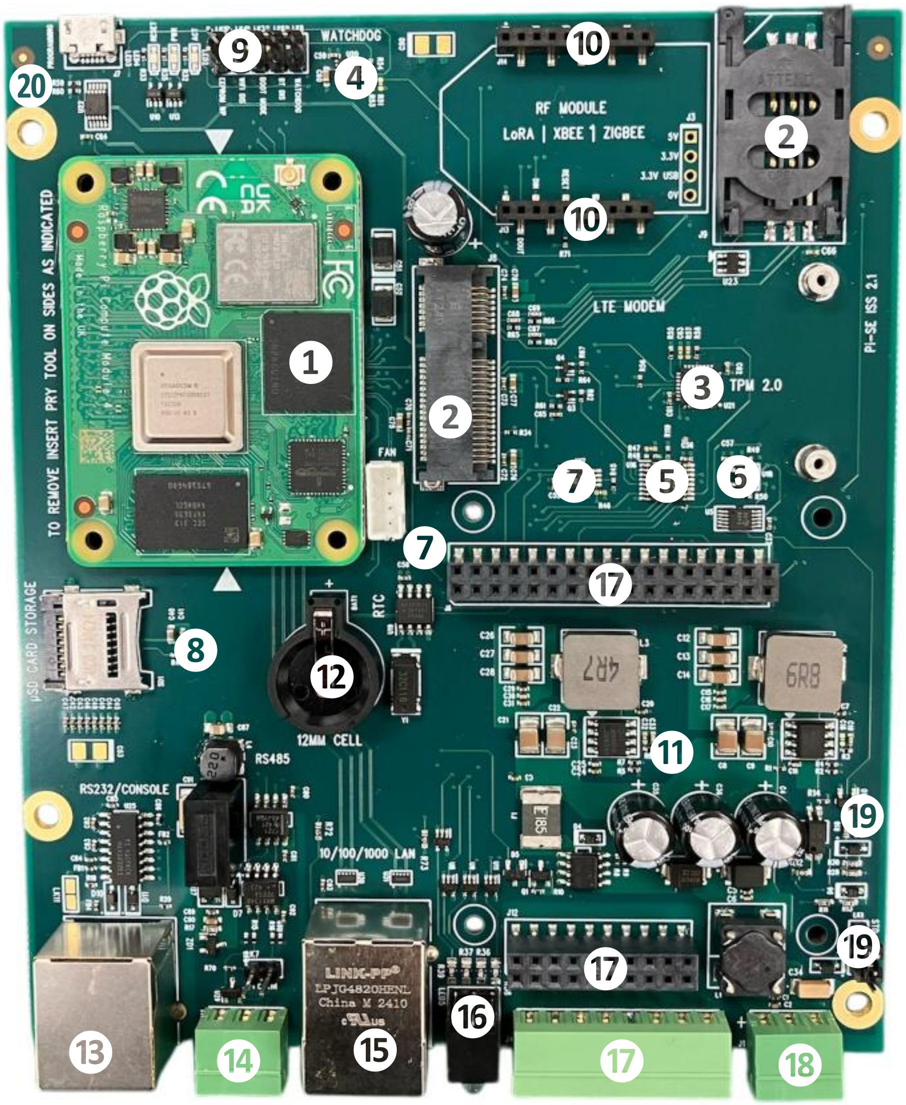
Dated : January 2025

Prepared By : Andrew O'Connell

# FEATURES

- Supports All Raspberry Pi Compute Module 4 variants and Compute Module 5
- 1 x 10/100/1000 LAN
- 1 x uSD Card Storage (SDIO Interfaced)
- 1 x mPCIe Interface (USB Interfaced) + SIM
- 1 x SPI Infineon SLB9670 TPM 2.0
- 1 x Battery Backed RTC
- 1 x Board ID EEPROM (Preprogrammed)
- 1 x Isolated RS485 Interface
- 1 x RS232 Port
- 1 x DIGI XBEE Compatible RF Slot
- 1 x Modular IO slot with 11 GPIO Pins – Compatible with ADC/CAN/485/Sensor-Kit IO boards
    - 1 x SPI
    - 4 x I2C Master
    - 2 x UART
    - 1 x I2C-Slave
- 1 x Optional 60 second watchdog (active from power up/boot)
- 1 x Temperature activated fan control (same as Pi CMIO4 board)
- 2 x Bi-colour user LEDs
- 9-28V Input with gated power switch
- Wide -20°C to +80°C Ambient operating temperature

## BOARD IO FEATURES




1. Compute Module 4 Socket
2. mPCIe Socket + Modem SIM Socket
3. SPI Interfaced SLB9670 TPM 2.0
4. 60 Second Watchdog
5. I2C GPIO Expander
6. I2C Serial EEPROM
7. I2C EMC2301 Fan Controller
8. µSD Card Socket
9. Pi Status LEDs + Mode Links
10. Digi XBEE Socket
11. Power Supply
12. I2C DS1338Z RTC + Coin Cell Backup Battery
13. RS232 Interface
14. RS485 Interface
15. Pi Gigabit 10/100/1000 Interface
16. Dual Bi-colour LED
17. GPIO IO Card interface + Front Connector
18. Power In (9-28V DC)
19. Power In Gating Selection
20. µUSB CM4 Programming port

# HARDWARE CONFIGURATION LINKS

**LED - RESET**

This LED indicates when the Pi unit is in reset condition and has asserted an external reset signal, which is routed to parts 3, 8 & 13

**LED - PWR**

This GPIO driven LED indicates ‘power’ functionality on a Raspberry Pi and can be repurposed for general usage, signal also connected the bottom red LED.

**LED - ACT**

This LED indicates ‘Activity’ functionality on the Pi unit, by default this indicates eMMC flash access on the module, but can be reassigned to indicate other status signals.

**LK5 - EEPROM DIS**

Fitted — Indicates to CM4 module to disable EEPROM
*(Facility not enabled in firmware by default)*

Open — Default

**LK4 - WIFI DIS**

Fitted — Forces CM4 module to disable WiFi RF Output

Open — Default

**LK3 - Boot Mode**

Fitted — Forces CM4 module into eMMC programming or EEPROM Firmware update mode

Open — Default, boots as normal according to EEPROM settings

**LK5 - BT DIS**

Fitted — Forces CM4 module to disable Bluetooth RF Output

Open — Default

**LK8 - WATCHDOG RESET OUT**

Fitted — Connect External Watchdog Reset Out to CM4 RUN/RESET Line

Open — Default

**LK1 - POWER GATING OPERATION SELECTION 1**

**LK2 - POWER GATING OPERATION SELECTION 2**

See Power Good Input on Page 39 for detailed operation.

# 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.

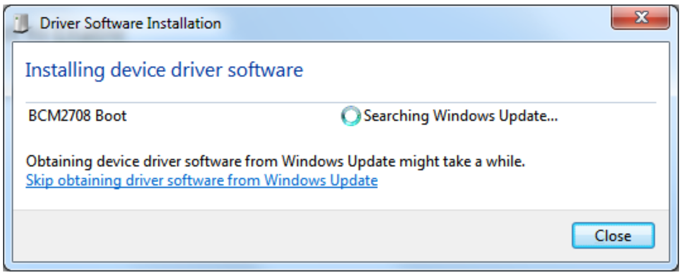
Units come pre-programmed with the demo Raspbian OS pre-installed, this section describes how to write a new disk image to the Compute Module.

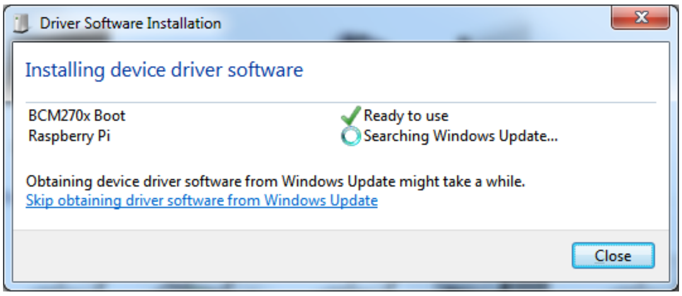
First of all download the windows USB boot installer, this will install the device drivers as well as a program we'll use later called RPi-Boot

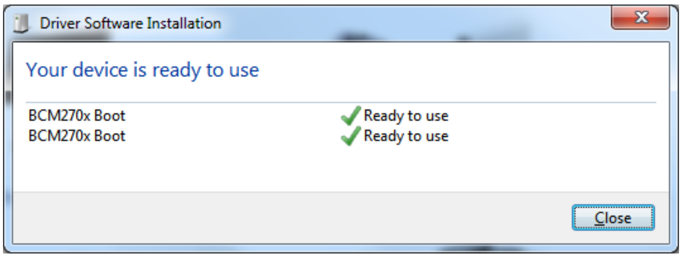
[Raspberry Pi RPI-BOOT Software Download Link]

Connect the mini USB connector to the Windows PC using the supplied USB A to micro USB B data cable, fit the boot mode jumper link (LK1) and then power up the unit.

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

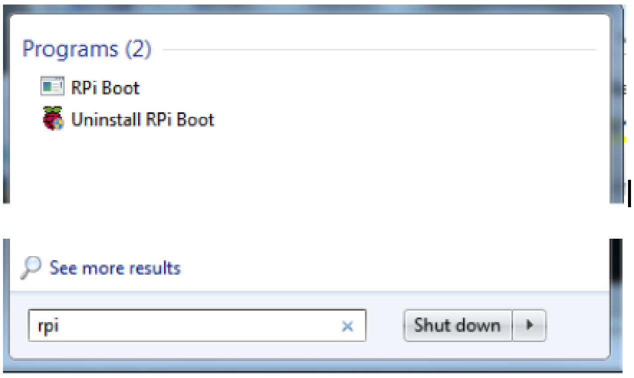


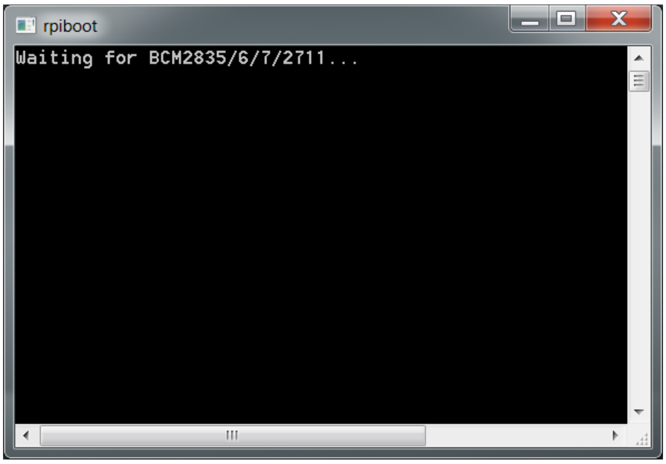




Once that sequence has finished Windows has now installed the required drivers and you can power off the unit for a moment whilst we get the PC side ready for the next step.

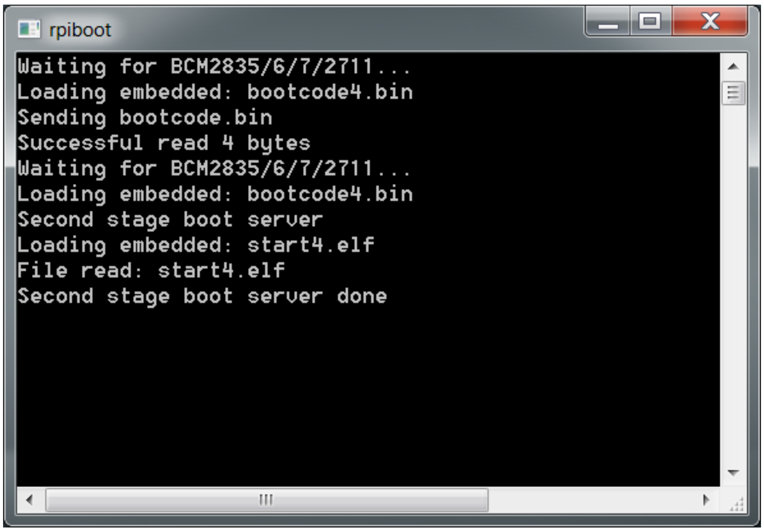
Making sure you have the unit powered off start up RPi Boot, this is easiest done via the start menu, we have found this needs to be run as 'Administrator' privilege mode for correct operation




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



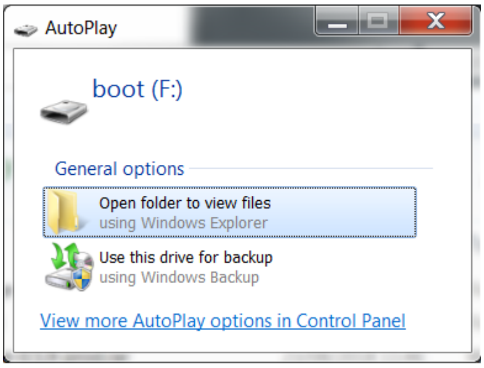

Power up the unit and RPi-Boot will configure the unit to appear as a flash drive :



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

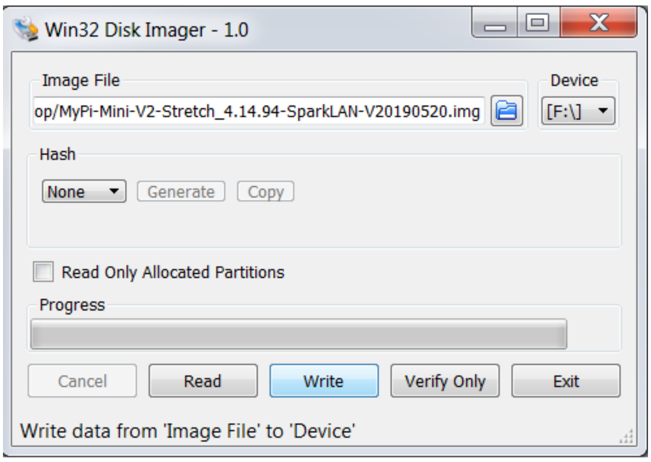
This process writes a disk image, containing the partition table as well as both FAT boot partition and Linux EXT partitions, ***over the entire disk.***

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

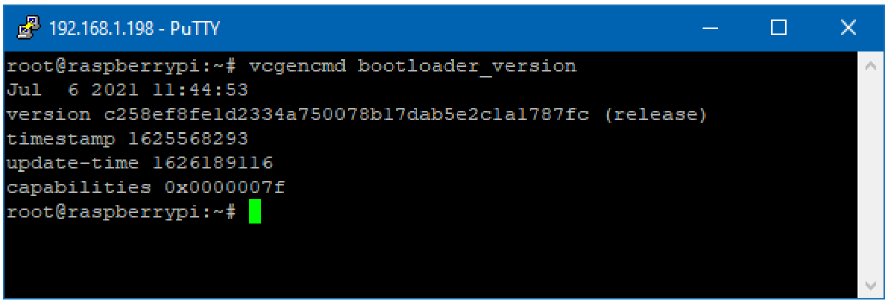


# CM4 BOOT EEPROM FIRMWARE UPDATE

On the CM4 it is not possible to update the boot firmware EEPROM from the command line

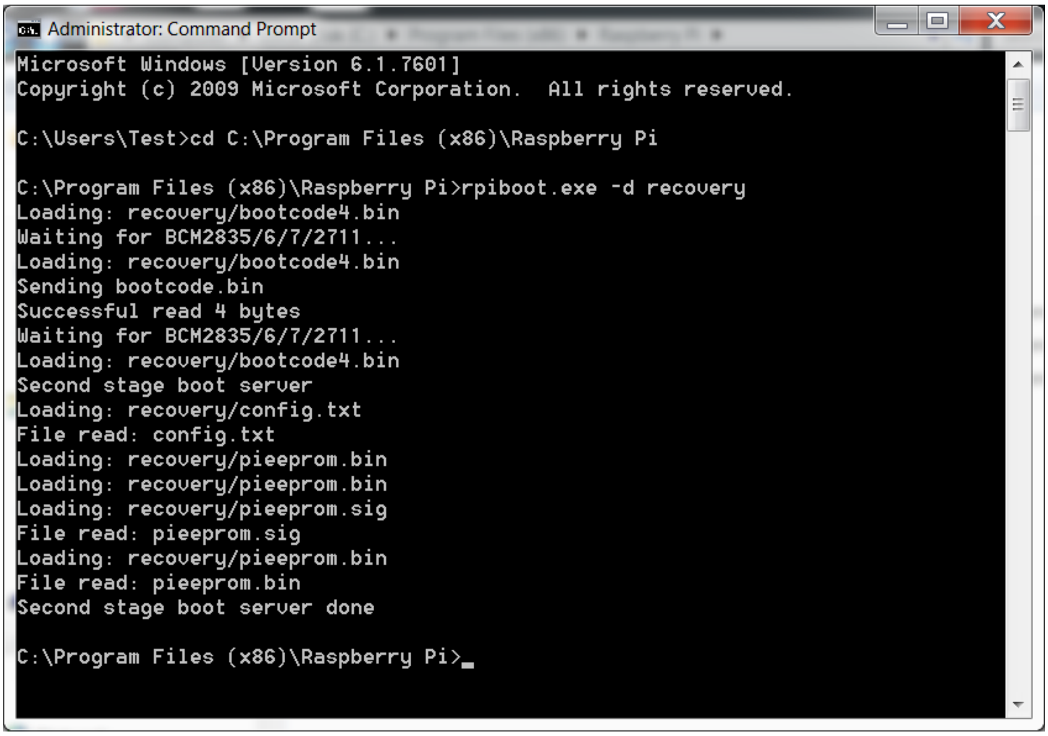
To find the current boot loader version run **vcgencmd bootloader_version**

**For best USB and Camera support we recommend installing version July 6th 2021 or later**




To update the firmware on a CM4 device the same **rpiboot** program is used, but with a different syntax to usual. This directs the system to push a different set of files to the CM4 device containing the firmware update files.

Fit the boot mode link and connect up the microUSB programming cable and run rpiboot as shown in the screenshot below. This process takes a short amount of time and once completed the HDMI output will indicate a green screen with rapidly blinking status LED to show success.




**To change/update the firmware version files of your install replace the 'recovery' folder on your local machine with that of the 'recovery' folder from the main github repo below**

[https://github.com/raspberrypi/usbboot](https://github.com/raspberrypi/usbboot)

# CM4 SECURE BOOT

Secure boot facilities on the CM4 are currently in beta release, see below website link for examples and notes on how to create a signed boot image

**https://github.com/raspberrypi/usbboot#secure-boot---image-creation**

**See also TPM Hardware Security Module notes following**

# SYSTEM GPIO

In order to minimise CM4 GPIO line usage an I2C interfaced TCA9554 8-bit GPIO expander has been included to deal with board IO

These present as **gpio496-503** on the OS and are configured via the system device tree overlays :

```
# I2C Bus (Note : Also need to add i2c-dev to /etc/modules)
dtparam=i2c_arm=on
dtoverlay=pca953x,addr=0x41,pca9536
```

Bash scripts **/usr/local/bin/setup-gpio.sh** (called from **/etc/udev/rules.d/40-i2c-gpio.rules**) and **/usr/local/bin/setup-adc.sh** (called from **/etc/udev/rules.d/50-i2c-adc.rules**) create sysfs GPIO exports and shortcut links.

These are run at boot-up which create the below symbolic links for quick access in **/dev**

```
mpcie-wdisble       -> /sys/class/gpio/gpio496/value

mpcie-reset         -> /sys/class/gpio/gpio497/value

power-fail          -> /sys/class/gpio/gpio499/value

led1-red            -> /sys/class/gpio/gpio500/value

tpm-reset           -> /sys/class/gpio/gpio501/value

led2-green          -> /sys/class/gpio/gpio502/value

led2-red            -> /sys/class/gpio/gpio503/value
```

Additional symlinks have been configured for :

```
wd-enable     -> /sys/class/gpio/gpio6/value

wd-input      -> /sys/class/gpio/gpio17/value

adc1          -> /sys/bus/i2c/devices/1-0049/iio:device0

adc_ch0       -> /sys/bus/i2c/devices/1-0049/iio:device0/in_voltage0_raw

led-power     -> /sys/class/leds/led1/brightness

ttyR232       -> ttyAMA0

ttyRS485      -> ttyAMA2

ttyXBEE       -> ttyAMA1

rtc_nvram     -> /sys/class/rtc/rtc0/device/nvram
```

Example usage :

```
$ echo 1 >/dev/led1-red
```

```
$ echo 0 >/dev/led1-red
$ echo 1 >/dev/led2-green
$ echo 0 >/dev/led2-green


$ echo 1 >/dev/mpcie-reset
$ echo 0 >/dev/mpcie-reset

# echo 'battery backed up ram' > /dev/rtc_nvram
cat /dev/rtc_nvram
battery backed up ram

$ readdcv.sh
DCINVOLTS=24.11
```

## Board OS Configuration

The sample OS image provided has been produced by overlaying a series of files over a standard Raspberry Pi Lite OS Image. The configuration files can be downloaded using the tar file linked to below

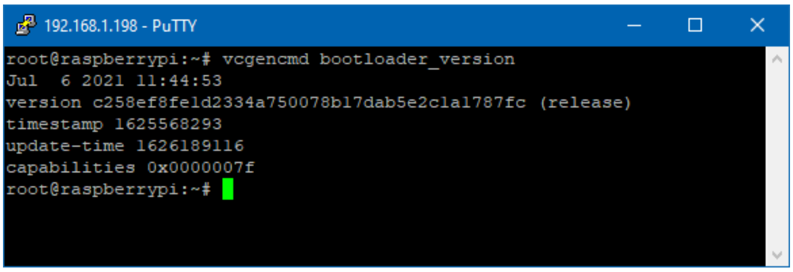
https://drive.google.com/file/d/16S1MHzzPl6UmExy6FnVkbVqq5qveyyQZ/view?usp=sharing

# USB INTERFACE

There are two critical settings that determine the USB controller active on the CM4, without these steps the system will either disable the USB port or enable the low bandwidth port controller.

## 1. Firmware version

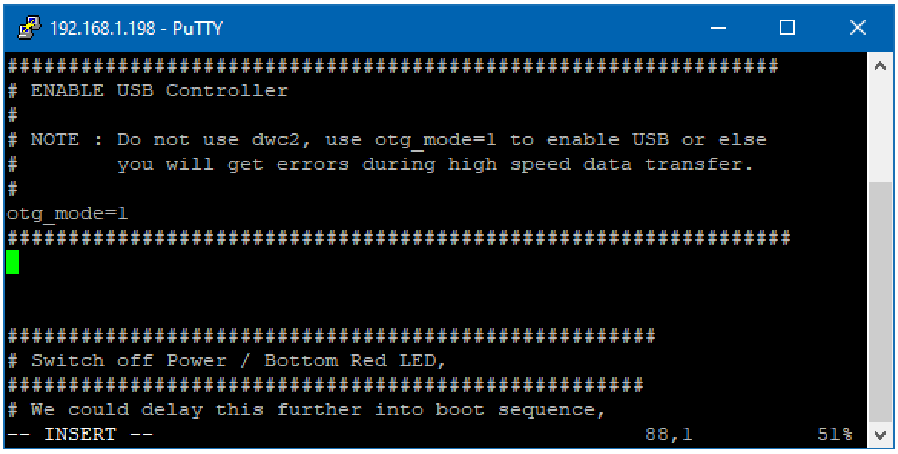
This should be version dated **July 6th 2021** or later, needed to enable the internal controller



```
192.168.1.198 - PuTTY
root@raspberrypi:~# vcgencmd bootloader_version
Jul  6 2021 11:44:53
version c258ef8fe1d2334a750078b17dab5e2c1a1787fc (release)
timestamp 1625568293
update-time 1626189116
capabilities 0x0000007f
root@raspberrypi:~#
```

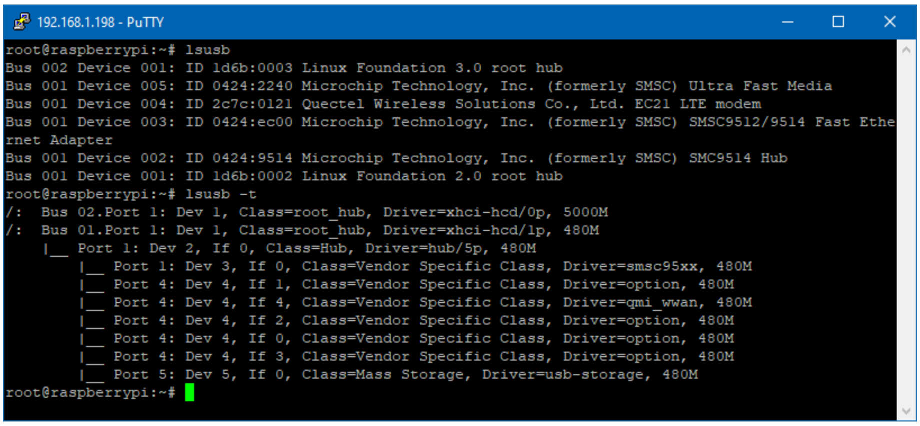
## 2. Config.txt setting

**otg_mode=1** should be included in the configuration file to select the **xhci-hcd** controller



```
192.168.1.198 - PuTTY
################################################################
# ENABLE USB Controller
#
# NOTE : Do not use dwc2, use otg_mode=1 to enable USB or else
#        you will get errors during high speed data transfer.
#
otg_mode=1
#################################################################



######################################################
# Switch off Power / Bottom Red LED,
#####################################################
# We could delay this further into boot sequence,
-- INSERT --                                            88,1          51%
```

Correctly setup the system will report the root hub as being an **xhci-hcd** device as shown below



```
192.168.1.198 - PuTTY
root@raspberrypi:~# lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 005: ID 0424:2240 Microchip Technology, Inc. (formerly SMSC) Ultra Fast Media
Bus 001 Device 004: ID 2c7c:0121 Quectel Wireless Solutions Co., Ltd. EC21 LTE modem
Bus 001 Device 003: ID 0424:ec00 Microchip Technology, Inc. (formerly SMSC) SMSC9512/9514 Fast Ethe
rnet Adapter
Bus 001 Device 002: ID 0424:9514 Microchip Technology, Inc. (formerly SMSC) SMC9514 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
root@raspberrypi:~# lsusb -t
/:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci-hcd/0p, 5000M
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci-hcd/1p, 480M
    |__ Port 1: Dev 2, If 0, Class=Hub, Driver=hub/5p, 480M
        |__ Port 1: Dev 3, If 0, Class=Vendor Specific Class, Driver=smsc95xx, 480M
        |__ Port 4: Dev 4, If 1, Class=Vendor Specific Class, Driver=option, 480M
        |__ Port 4: Dev 4, If 4, Class=Vendor Specific Class, Driver=qmi_wwan, 480M
        |__ Port 4: Dev 4, If 2, Class=Vendor Specific Class, Driver=option, 480M
        |__ Port 4: Dev 4, If 0, Class=Vendor Specific Class, Driver=option, 480M
        |__ Port 4: Dev 4, If 3, Class=Vendor Specific Class, Driver=option, 480M
        |__ Port 5: Dev 5, If 0, Class=Mass Storage, Driver=usb-storage, 480M
root@raspberrypi:~#
```

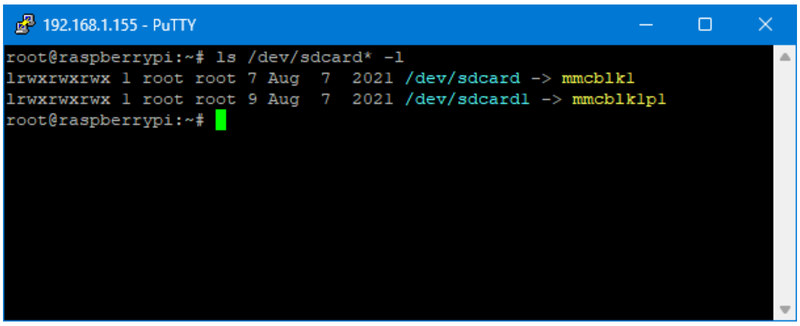
# SDIO SD CARD INTERFACE

The on-board micro SD Card is interfaced to the Raspberry Pi Compute Module using the CM4 SDIO interface controller, this provides fast access to secondary storage for datalogging.  Choice is between using the older interface which is slower, but allows use of the CM4s WiFi controller, or the newer native SDIO interface that usually drives the WiFi controller chip.

This is enabled via the **config.txt** file:

```
################################################################
# SD Card Interface
################################################################
# Older GPIO Based interface, slower but allows use of WiFi
#dtoverlay=sdio,poll_once,sdio_overclock=25

# Super Native SDIO interface, usage mutually exclusive with WiFi
dtoverlay=disable-wifi
dtoverlay=sdhost-custom
```

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




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](https://embeddedpi.com/documentation/sd-card-interface/raspberry-pi-industrial-micro-sd-cards)

# USB MINI-PCIE INTERFACE

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

| Pin | Signal | Pin | Signal |
|---|---|---|---|
| 1 | WAKE# | 2 | 3.3V |
| 3 | Reserved | 4 | GND |
| 5 | Reserved | 6 | 1.5V |
| 7 | CLKREQ# | 8 | SIM_VCC |
| 9 | GND | 10 | SIM_I/O |
| 11 | REFCLK- | 12 | SIM_CLK |
| 13 | REFCLK+ | 14 | SIM_RST |
| 15 | N/C or GND | 16 | SIM_VPP |
| **Mechanical Key** | | | |
| 17 | Reserved | 18 | GND |
| 19 | Reserved | 20 | W_DIS# |
| 21 | GND | 22 | PERST# |
| 23 | PERn0 | 24 | +3.3Vaux |
| 25 | PERp0 | 26 | GND |
| 27 | GND | 28 | +1.5V |
| 29 | GND | 30 | SMB_CLK |
| 31 | PETn0 | 32 | SMB_DATA |
| 33 | PETp0 | 34 | GND |
| 35 | GND | 36 | USB_D- |
| 37 | Reserved | 38 | USB_D+ |
| 39 | +3.3V | 40 | GND |
| 41 | +3.3V | 42 | LED_WWAN# |
| 43 | GND | 44 | LED_WLAN# |
| 45 | Reserved | 46 | LED_WPAN# |
| 47 | Reserved | 48 | +1.5V |
| 49 | Reserved | 50 | GND |
| 51 | Reserved | 52 | +3.3V |

**Signals in RED are not available**

The below GPIO connections are made to the connector

```
mpcie-reset      -> /sys/class/gpio/gpio500/value

mpcie-wdisble    -> /sys/class/gpio/gpio501/value
```

The WWAN LED is connected to the front top green bi-colour LED to indicate modem network registration/data transmission status.

## Modem Compatibility/Operation

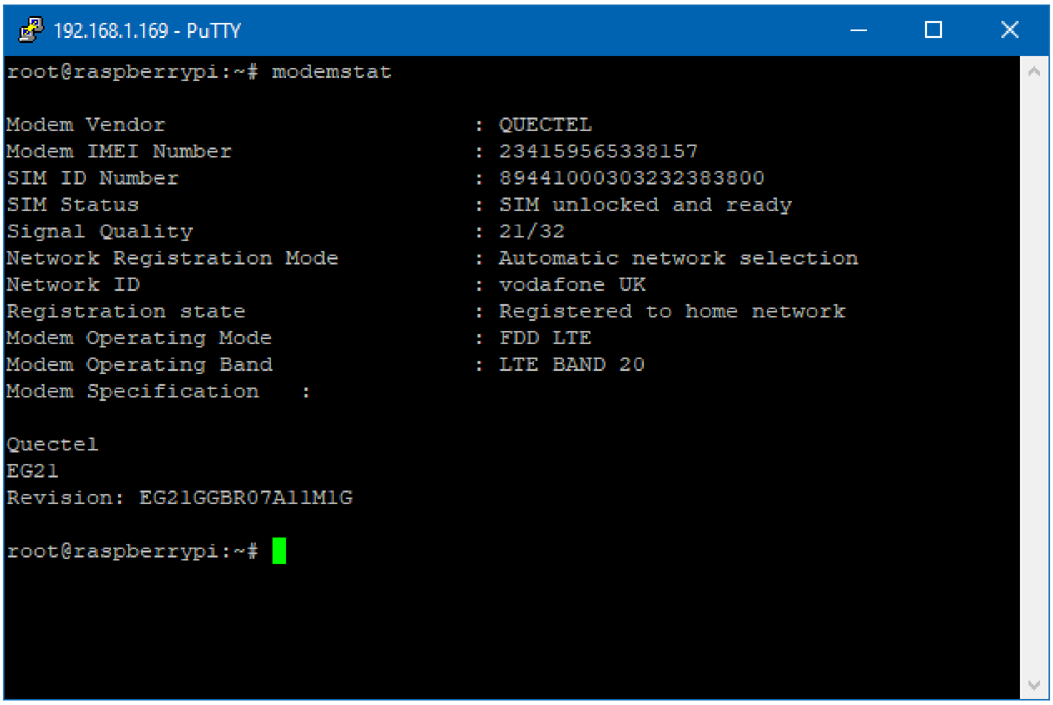
See the below link to pages from the main modem documentation section for details on how to operate modems :

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




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.

We would recommend looking at our connection management scripting as a starting point for managing connections over long periods:

https://github.com/mypiandrew/connman/

QMI Network Connection example :

192.168.1.169 - PuTTY

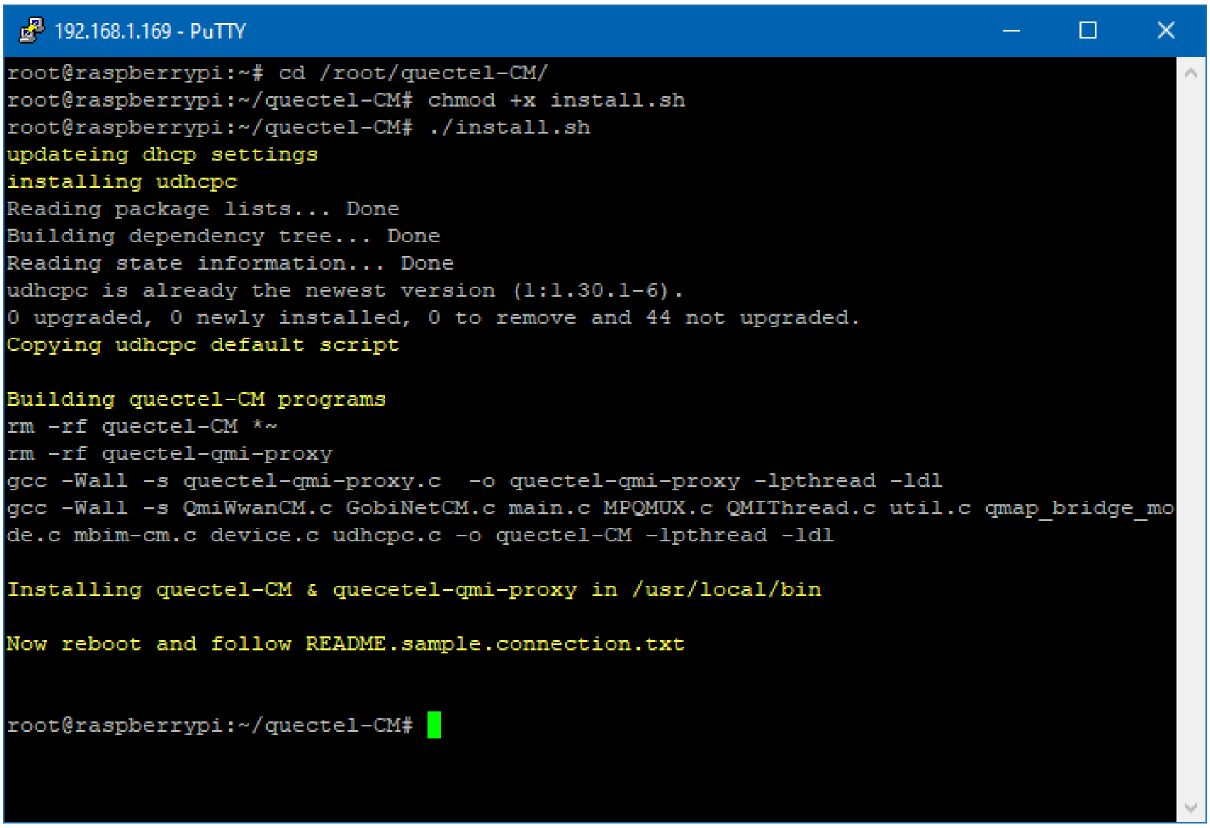
```
root@raspberrypi:~# modemstat

Modem Vendor                             : QUECTEL
Modem IMEI Number                        : 234159565338157
SIM ID Number                            : 89441000303232383800
SIM Status                               : SIM unlocked and ready
Signal Quality                           : 21/32
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 20
Modem Specification    :

Quectel
EG21
Revision: EG21GGBR07A11M1G

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: 5)
Saving state at /tmp/qmi-network-state-cdc-wdm0... (PDH: 3783131952)
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 10.32.10.179
udhcpc: lease of 10.32.10.179 obtained, lease time 7200
root@raspberrypi:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.32.10.180    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
10.32.10.176    0.0.0.0         255.255.255.248 U     0      0        0 wwan0
192.168.1.0     0.0.0.0         255.255.255.0   U     202    0        0 eth0
root@raspberrypi:~# ping -c3 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=645 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=104 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=53.4 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 53.423/267.736/645.355/267.827 ms
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](https://github.com/mypiandrew/quectel-cm/releases/download/V1.6.0.12/quectel-CM.tar.gz)




```
root@raspberrypi:~# cd /root/quectel-CM/
root@raspberrypi:~/quectel-CM# chmod +x install.sh
root@raspberrypi:~/quectel-CM# ./install.sh
updateing dhcp settings
installing udhcpc
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
udhcpc is already the newest version (1:1.30.1-6).
0 upgraded, 0 newly installed, 0 to remove and 44 not upgraded.
Copying udhcpc 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 QMIThread.c util.c qmap_bridge_mo
de.c mbim-cm.c device.c udhcpc.c -o quectel-CM -lpthread -ldl

Installing quectel-CM & quecetel-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.

192.168.1.169 - PuTTY

```
root@raspberrypi:~/quectel-CM# quectel-CM -s pp.vodafone.co.uk
[06-23_10:46:01:365] Quectel_QConnectManager_Linux_V1.6.0.12
[06-23_10:46:01:369] Find /sys/bus/usb/devices/1-1.4 idVendor=0x2c7c idProduct=0x121,
bus=0x001, dev=0x004
[06-23_10:46:01:369] Auto find qmichannel = /dev/cdc-wdm0
[06-23_10:46:01:369] Auto find usbnet_adapter = wwan0
[06-23_10:46:01:369] netcard driver = qmi_wwan, driver version = 5.10.92-v7+
[06-23_10:46:01:370] ioctl(0x89f3, qmap_settings) failed: Operation not supported, rc=
-1
[06-23_10:46:01:371] Modem works in QMI mode
[06-23_10:46:01:395] cdc_wdm_fd = 7
[06-23_10:46:01:496] Get clientWDS = 5
[06-23_10:46:01:528] Get clientDMS = 1
[06-23_10:46:01:562] Get clientNAS = 2
[06-23_10:46:01:594] Get clientUIM = 2
[06-23_10:46:01:628] Get clientWDA = 1
[06-23_10:46:01:660] requestBaseBandVersion EG21GGBR07A11M1G
[06-23_10:46:01:792] requestGetSIMStatus SIMStatus: SIM_READY
[06-23_10:46:01:792] requestSetProfile[1] pp.vodafone.co.uk///0
[06-23_10:46:01:858] requestGetProfile[1] pp.vodafone.co.uk///0
[06-23_10:46:01:892] requestRegistrationState2 MCC: 234, MNC: 15, PS: Attached, DataCa
p: LTE
[06-23_10:46:01:925] requestQueryDataCall IPv4ConnectionStatus: DISCONNECTED
[06-23_10:46:01:925] ifconfig wwan0 down
[06-23_10:46:01:942] ifconfig wwan0 0.0.0.0
[06-23_10:46:02:245] requestSetupDataCall WdsConnectionIPv4Handle: 0xe17ca260
[06-23_10:46:02:378] ifconfig wwan0 up
[06-23_10:46:02:389] busybox udhcpc -f -n -q -t 5 -i wwan0
udhcpc: started, v1.30.1
No resolv.conf for interface wwan0.udhcpc
udhcpc: sending discover
udhcpc: sending discover
udhcpc: sending discover
udhcpc: sending discover
udhcpc: sending discover
udhcpc: no lease, failing
[06-23_10:46:17:998] File:ql_raw_ip_mode_check Line:105 udhcpc fail to get ip address,
 try next:
[06-23_10:46:17:999] ifconfig wwan0 down
[06-23_10:46:18:012] echo Y > /sys/class/net/wwan0/qmi/raw_ip
[06-23_10:46:18:013] ifconfig wwan0 up
[06-23_10:46:18:025] busybox udhcpc -f -n -q -t 5 -i wwan0
udhcpc: started, v1.30.1
No resolv.conf for interface wwan0.udhcpc
udhcpc: sending discover
udhcpc: sending select for 10.32.10.179
udhcpc: lease of 10.32.10.179 obtained, lease time 7200
^C[06-23_10:46:42:799] requestDeactivateDefaultPDP WdsConnectionIPv4Handle
[06-23_10:46:42:988] ifconfig wwan0 down
[06-23_10:46:43:001] ifconfig wwan0 0.0.0.0
[06-23_10:46:43:185] QmiWwanThread exit
[06-23_10:46:43:187] qmi_main exit
root@raspberrypi:~/quectel-CM#
```

# SPI TRUSTED PLATFORM MODULE

Integrated on-board 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**

```
dtoverlay=spi1-1cs,cs0_spidev=disabled
dtoverlay=tpm-spi1
```

This configures SPI1 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.sh`

`/usr/local/bin/tpmtool`

`/root/tpm-toolkit/`

`/usr/local/bin/tpmupdate`

For more information see the github repository below

[https://github.com/Infineon/eltt2](https://github.com/Infineon/eltt2)

A hardware reset of the SLB9670 device is asserted at reboot/power up by the Compute Module

# I2C 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 barcoded sticker affixed to the PCB.

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

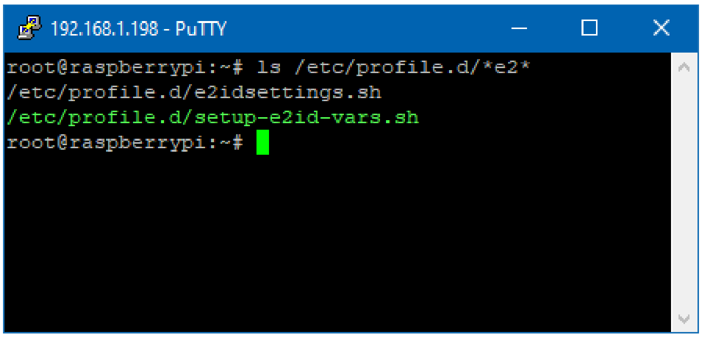
```
192.168.1.155 - PuTTY
root@raspberrypi:~# i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- 0c -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: UU -- -- -- -- -- -- -- -- -- -- -- -- -- -- 2f
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- UU -- -- -- -- -- --
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**

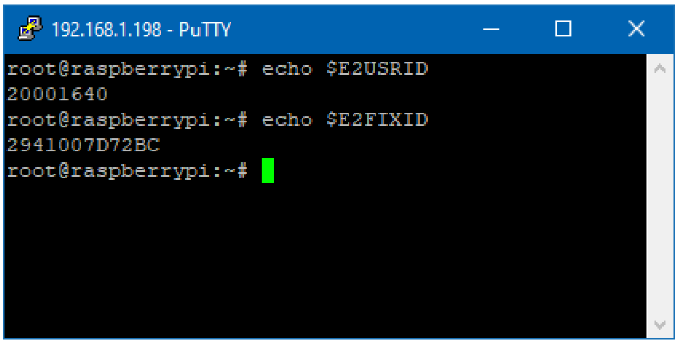
```
192.168.1.198 - PuTTY
root@raspberrypi:~# i2cdump -y 1 0x50
No size specified (using byte-data access)
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 20 00 16 40 ff ff ff ff ff ff ff ff ff ff ff ff     .?@............
10: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
20: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
30: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
40: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
50: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
60: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
70: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
80: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
90: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
a0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
b0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
c0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
d0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
e0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
f0: ff ff ff ff ff ff ff ff ff ff 29 41 00 7d 72 bc    ..........)A.}r?
root@raspberrypi:~#
```

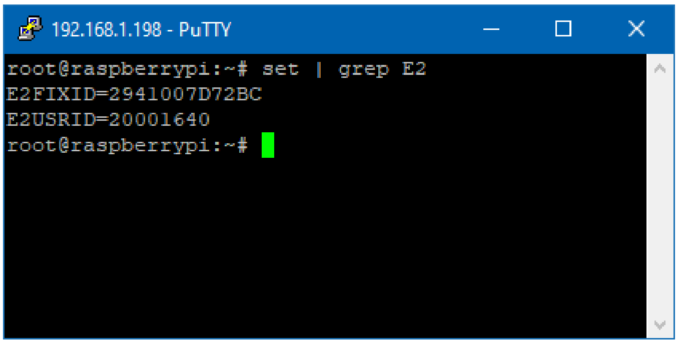
For convenience a script to create two bash environment variables has been created in **/etc/profile.d**

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

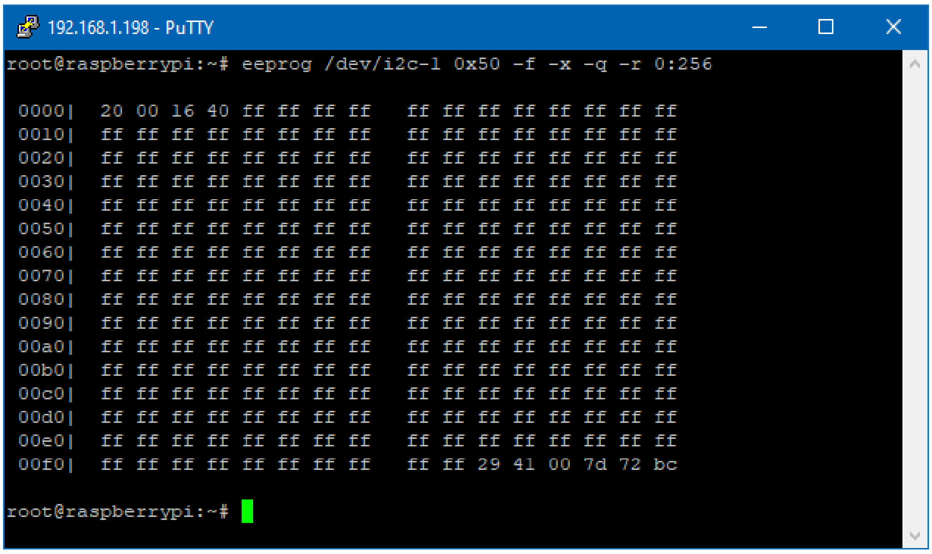



These environment variables can be used in scripting by any user




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**)

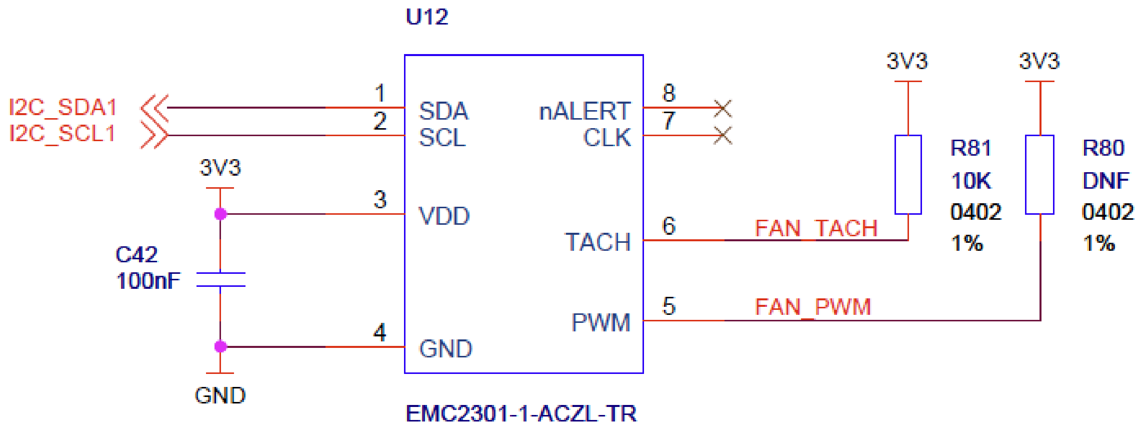


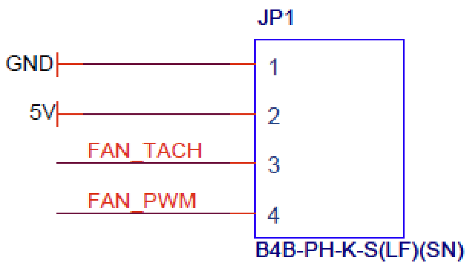
192.168.1.198 - PuTTY

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

 0000|  20 00 16 40 ff ff ff ff   ff ff ff ff ff ff ff ff
 0010|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 0020|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 0030|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 0040|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 0050|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 0060|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 0070|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 0080|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 0090|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 00a0|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 00b0|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 00c0|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 00d0|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 00e0|  ff ff ff ff ff ff ff ff   ff ff ff ff ff ff ff ff
 00f0|  ff ff ff ff ff ff ff ff   ff ff 29 41 00 7d 72 bc

root@raspberrypi:~#
```

# I2C FAN CONTROLLER

Integrated onto the main board is a Microchip EMC2301 PWM Fan controller




This device occupies address 0x2F and can be operated by either basic I2C commands or a more comprehensive kernel driver

**<u>I2C Method</u>**

Turn fan off:

```
i2cset -y 1 0x2f 0x30 0x00
```

Turn fan on 100%:

```
i2cset -y 1 0x2f 0x30 0xff
```

We have created a bash script to operate this and get RPM information, see link below :

https://github.com/mypiandrew/fanctrl

**Kernel Driver Method**

A 3rd party driver here can be installed to allow more control over the fan operation

https://github.com/neg2led/cm4io-fan

# I2C REAL TIME CLOCK

A DS1338Z-33+ Real Time Clock with battery backup cell is integrated onto the board, this is configured by the below device tree overlay

```
dtoverlay=i2c-rtc,ds1307,addr=0x68
```

Further OS integration to remove the **fake-hwclock** functionality, and ensure the system reads/writes to the hwclock, has also been done.

A good primer on this can be found here :

https://learn.adafruit.com/adding-a-real-time-clock-to-raspberry-pi/set-rtc-time

A symbolic link is also setup to allow quick access to the devices battery backed NVRAM

```
rtc_nvram           -> /sys/class/rtc/rtc0/device/nvram
```

Example usage :

```
# echo “battery backed up ram” > /dev/rtc_nvram

# cat /dev/rtc_nvram
battery backed up ram
```

The 12mm coin cell is CR1216 type, eg. VARTA 6216401501

# WATCHDOG

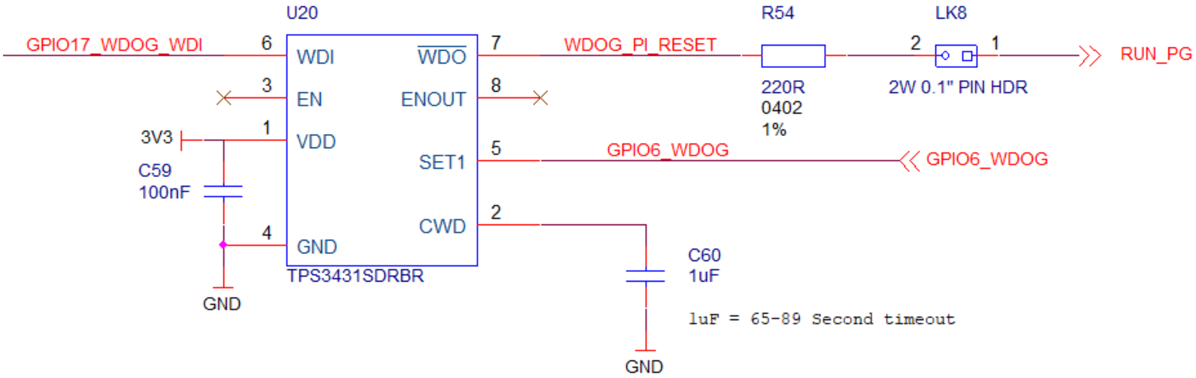
The on-board external watchdog with a 60 Second delay timer, provided by a Texas Instruments TPS3431 part, the reset output of this is connected to the Raspberry Pi Compute module's reset pin.

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 enabled by push on link LK8 and driven by GPIO16 (WD Enable) and GPIO17 (WD Input).

**Once these push on links are fitted the watchdog is enabled by default covering the whole boot cycle.**

***Note this watchdog does not work on the CM5 as there is no external RUN_PG signal on the CM5***




Once the watchdog is enabled the WD Input pin on the device must be togged H-L-H at least once per watchdog time-out period (60 seconds) and the low level pulse period must be >1uS long for the transition to be valid.

If the device sees a valid low-to-high transition on the input pin the internal 60 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 and hard reset the system.

The watchdog can be disabled completely by either physically removing LK6 (optionally removing LK5&7 additionally) or by driving GPIO16 low.

**/etc/init.d/mypi-init.sh** sets up symbolic links for these GPIO Lines, see this file for more details.

Watchdog integration can be directly done inside application code by writing directly to the GPIO lines or can be done via a kernel GPIO watchdog process via the watchdog package which provides more varied sources for monitoring.

The OS files for the CM4 setup have been placed in the **/root/watchdog** folder on the demo OS image.

## External Watchdog OS Integration

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

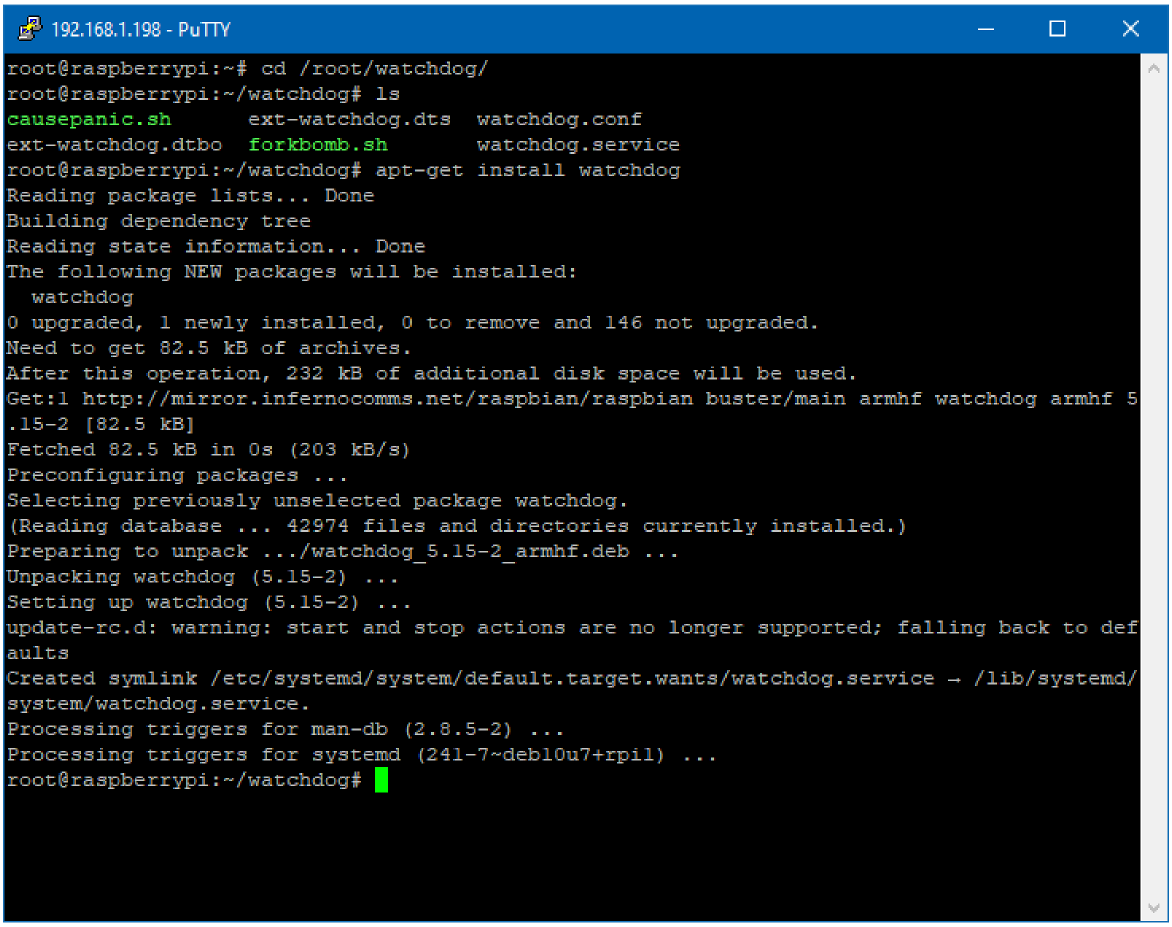
In this configuration if the target file is not updated the system will attempt an "orderly" reset as it performs some basic "clean-up" tasks prior to finally stopping the watchdog input line toggling, and so causing the Raspberry Pi Compute Module's reset line (aka RUN pin) to be momentarily pulled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

The watchdog system is configured by 3 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**

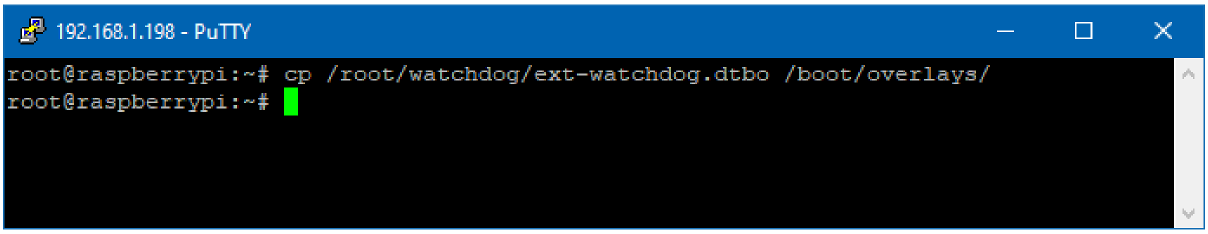
The configuration files for these are stored in **/root/watchdog** on the demo image

The watchdog OS package needs to be installed



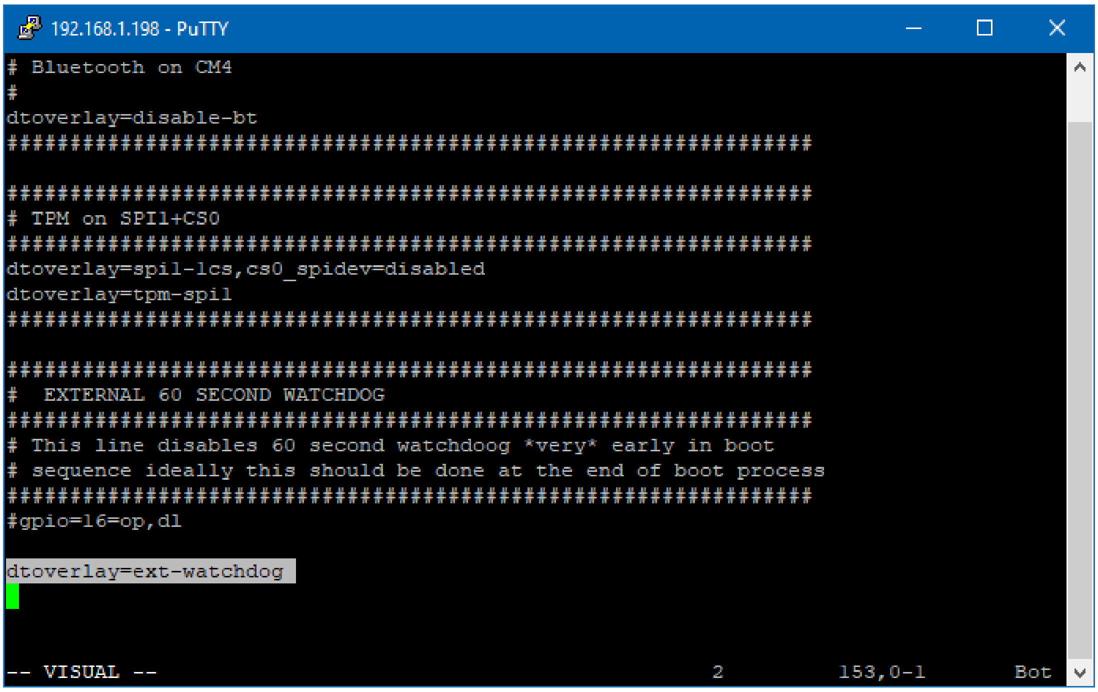

```
# apt-get install watchdog
```

Copy the device tree overlay for this module to the /boot/overlays folder if not already done so to configure the kernel to add a GPIO watchdog  to the system



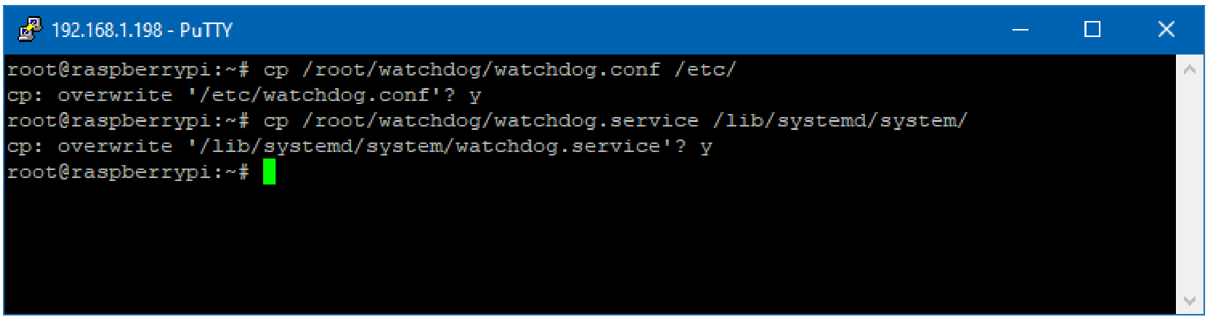

Add the below line to the end of **/boot/config.txt** to enable the overlay

```
dtoverlay=ext-watchdog
```



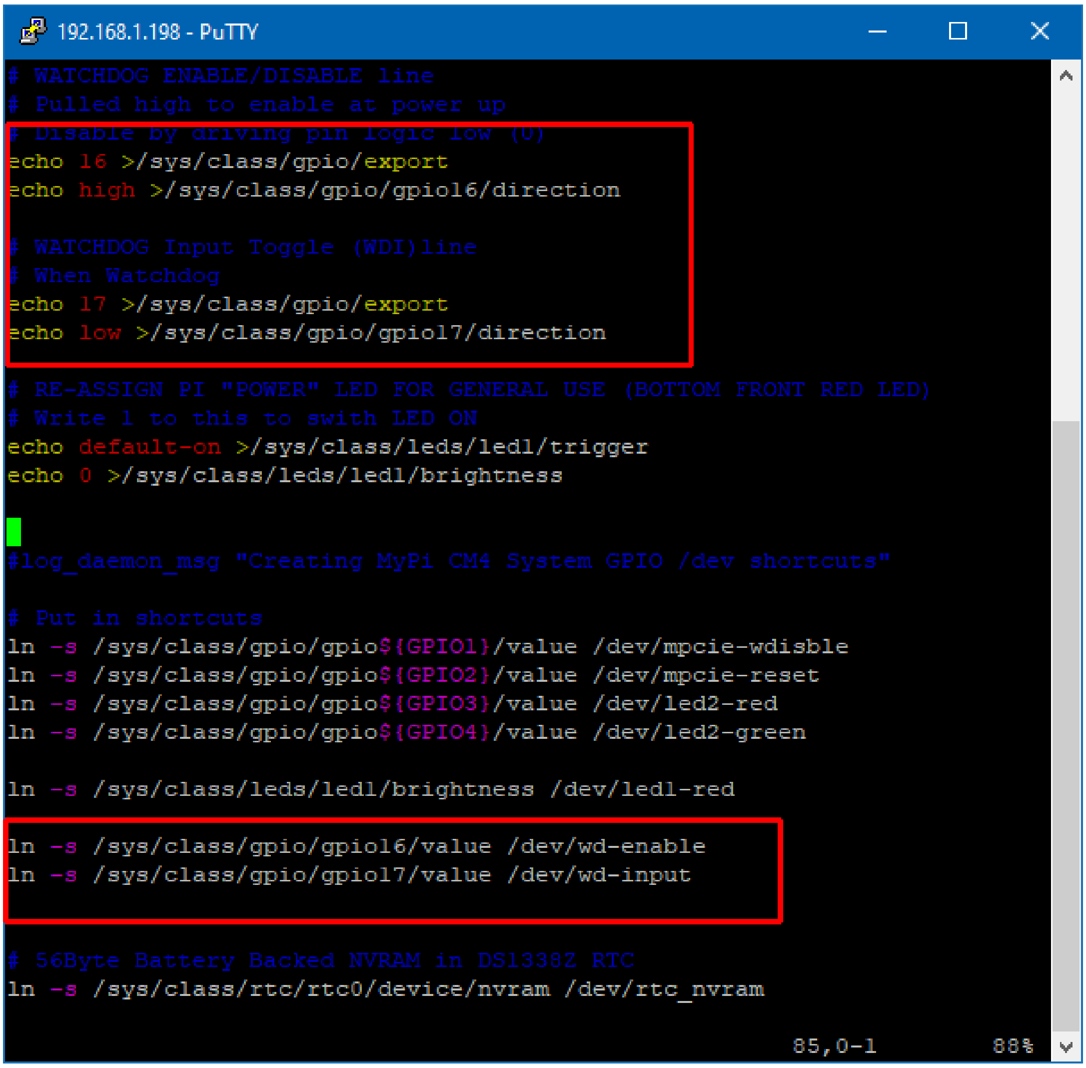

Install the configuration files for the watchdog service and the service file that starts the watchdog OS service then enables the watchdog.

```
cp /root/watchdog/watchdog.conf    /etc/
cp /root/watchdog/watchdog.service /lib/systemd/system/watchdog.service
```



Finally we need to remove the reference to the watchdog from **/etc/init.d/mypi-init.sh** to allow the watchdog and service file to claim the relevant IO lines.

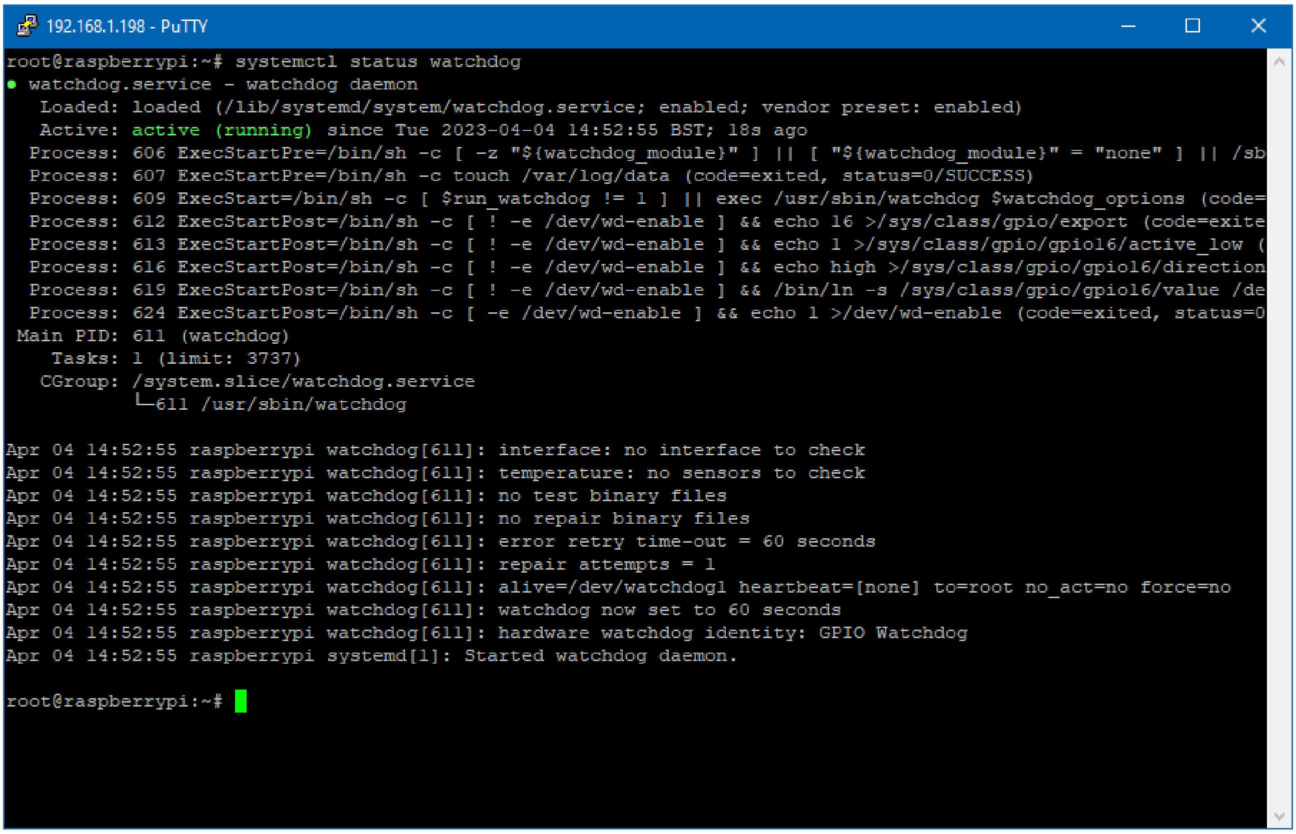
Do this by adding a **#** to the start of of each of the lines highlighted below




```
# WATCHDOG ENABLE/DISABLE line
# Pulled high to enable at power up
# Disable by driving pin logic low (0)
echo 16 >/sys/class/gpio/export
echo high >/sys/class/gpio/gpio16/direction

# WATCHDOG Input Toggle (WDI)line
# When Watchdog
echo 17 >/sys/class/gpio/export
echo low >/sys/class/gpio/gpio17/direction

# RE-ASSIGN PI "POWER" LED FOR GENERAL USE (BOTTOM FRONT RED LED)
# Write 1 to this to swith LED ON
echo default-on >/sys/class/leds/led1/trigger
echo 0 >/sys/class/leds/led1/brightness


#log_daemon_msg "Creating MyPi CM4 System GPIO /dev shortcuts"

# Put in shortcuts
ln -s /sys/class/gpio/gpio${GPIO1}/value /dev/mpcie-wdisble
ln -s /sys/class/gpio/gpio${GPIO2}/value /dev/mpcie-reset
ln -s /sys/class/gpio/gpio${GPIO3}/value /dev/led2-red
ln -s /sys/class/gpio/gpio${GPIO4}/value /dev/led2-green

ln -s /sys/class/leds/led1/brightness /dev/led1-red

ln -s /sys/class/gpio/gpio16/value /dev/wd-enable
ln -s /sys/class/gpio/gpio17/value /dev/wd-input

# 56Byte Battery Backed NVRAM in DS1338Z RTC
ln -s /sys/class/rtc/rtc0/device/nvram /dev/rtc_nvram

                                                      85,0-1        88%
```

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

```
# systemctl status watchdog
```



```
192.168.1.198 - PuTTY
root@raspberrypi:~# systemctl status watchdog
● watchdog.service - watchdog daemon
   Loaded: loaded (/lib/systemd/system/watchdog.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2023-04-04 14:52:55 BST; 18s ago
  Process: 606 ExecStartPre=/bin/sh -c [ -z "${watchdog_module}" ] || [ "${watchdog_module}" = "none" ] || /sb
  Process: 607 ExecStartPre=/bin/sh -c touch /var/log/data (code=exited, status=0/SUCCESS)
  Process: 609 ExecStart=/bin/sh -c [ $run_watchdog != 1 ] || exec /usr/sbin/watchdog $watchdog_options (code=
  Process: 612 ExecStartPost=/bin/sh -c [ ! -e /dev/wd-enable ] && echo 16 >/sys/class/gpio/export (code=exite
  Process: 613 ExecStartPost=/bin/sh -c [ ! -e /dev/wd-enable ] && echo 1 >/sys/class/gpio/gpio16/active_low (
  Process: 616 ExecStartPost=/bin/sh -c [ ! -e /dev/wd-enable ] && echo high >/sys/class/gpio/gpio16/direction
  Process: 619 ExecStartPost=/bin/sh -c [ ! -e /dev/wd-enable ] && /bin/ln -s /sys/class/gpio/gpio16/value /de
  Process: 624 ExecStartPost=/bin/sh -c [ -e /dev/wd-enable ] && echo 1 >/dev/wd-enable (code=exited, status=0
 Main PID: 611 (watchdog)
    Tasks: 1 (limit: 3737)
   CGroup: /system.slice/watchdog.service
           └─611 /usr/sbin/watchdog

Apr 04 14:52:55 raspberrypi watchdog[611]: interface: no interface to check
Apr 04 14:52:55 raspberrypi watchdog[611]: temperature: no sensors to check
Apr 04 14:52:55 raspberrypi watchdog[611]: no test binary files
Apr 04 14:52:55 raspberrypi watchdog[611]: no repair binary files
Apr 04 14:52:55 raspberrypi watchdog[611]: error retry time-out = 60 seconds
Apr 04 14:52:55 raspberrypi watchdog[611]: repair attempts = 1
Apr 04 14:52:55 raspberrypi watchdog[611]: alive=/dev/watchdog1 heartbeat=[none] to=root no_act=no force=no
Apr 04 14:52:55 raspberrypi watchdog[611]: watchdog now set to 60 seconds
Apr 04 14:52:55 raspberrypi watchdog[611]: hardware watchdog identity: GPIO Watchdog
Apr 04 14:52:55 raspberrypi systemd[1]: Started watchdog daemon.

root@raspberrypi:~#
```

**/etc/watchdog.conf** contains the configuration for the external watchdog process, this has been configured in this example to monitor the file **/var/log/data** and check to see its timestamp updates at least once every 30 seconds

Other options are available :

[https://manpages.debian.org/testing/watchdog/watchdog.8.en.html](https://manpages.debian.org/testing/watchdog/watchdog.8.en.html)
[https://manpages.debian.org/testing/watchdog/watchdog.conf.5.en.html](https://manpages.debian.org/testing/watchdog/watchdog.conf.5.en.html)

192.168.1.198 - PuTTY

```
root@raspberrypi:~/watchdog# cat /etc/watchdog.conf
####################################################################################
##
##  Webpage below givs full config options
##
##  https://www.crawford-space.co.uk/old_psc/watchdog/watchdog-configure.html
##
###################################################################################
#ping                    = 172.31.14.1
#ping                    = 172.26.1.255
#interface               = eth0
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:~/watchdog#
```

Note that the watchdog device we have configured is **/dev/watchdog1** and not **/dev/watchdog0** which is the internal Pi CPU watchdog)

If the file **/var/log/data** 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

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

```
192.168.1.198 - PuTTY
root@raspberrypi:~# tail  -f /var/log/syslog
Apr  4 13:40:47 raspberrypi systemd[1]: Started User Manager for UID 0.
Apr  4 13:40:47 raspberrypi systemd[1]: Started Session 1 of user root.
Apr  4 13:40:47 raspberrypi watchdog[610]: file /var/log/data was not changed in 71 seconds (more than 30)
Apr  4 13:40:48 raspberrypi watchdog[610]: file /var/log/data was not changed in 72 seconds (more than 30)
Apr  4 13:40:49 raspberrypi watchdog[610]: file /var/log/data was not changed in 73 seconds (more than 30)
Apr  4 13:40:50 raspberrypi watchdog[610]: file /var/log/data was not changed in 74 seconds (more than 30)
Apr  4 13:40:51 raspberrypi watchdog[610]: file /var/log/data was not changed in 75 seconds (more than 30)
Apr  4 13:40:52 raspberrypi watchdog[610]: file /var/log/data was not changed in 76 seconds (more than 30)
Apr  4 13:40:53 raspberrypi watchdog[610]: file /var/log/data was not changed in 77 seconds (more than 30)
Apr  4 13:40:54 raspberrypi watchdog[610]: file /var/log/data was not changed in 78 seconds (more than 30)
Apr  4 13:40:55 raspberrypi watchdog[610]: file /var/log/data was not changed in 79 seconds (more than 30)
Apr  4 13:40:56 raspberrypi watchdog[610]: file /var/log/data was not changed in 80 seconds (more than 30)
^C
root@raspberrypi:~# touch /var/log/data
root@raspberrypi:~# tail  -f /var/log/syslog
Apr  4 13:40:49 raspberrypi watchdog[610]: file /var/log/data was not changed in 73 seconds (more than 30)
Apr  4 13:40:50 raspberrypi watchdog[610]: file /var/log/data was not changed in 74 seconds (more than 30)
Apr  4 13:40:51 raspberrypi watchdog[610]: file /var/log/data was not changed in 75 seconds (more than 30)
Apr  4 13:40:52 raspberrypi watchdog[610]: file /var/log/data was not changed in 76 seconds (more than 30)
Apr  4 13:40:53 raspberrypi watchdog[610]: file /var/log/data was not changed in 77 seconds (more than 30)
Apr  4 13:40:54 raspberrypi watchdog[610]: file /var/log/data was not changed in 78 seconds (more than 30)
Apr  4 13:40:55 raspberrypi watchdog[610]: file /var/log/data was not changed in 79 seconds (more than 30)
Apr  4 13:40:56 raspberrypi watchdog[610]: file /var/log/data was not changed in 80 seconds (more than 30)
Apr  4 13:40:57 raspberrypi watchdog[610]: file /var/log/data was not changed in 81 seconds (more than 30)
Apr  4 13:40:58 raspberrypi watchdog[610]: file /var/log/data was not changed in 82 seconds (more than 30)
^C
root@raspberrypi:~# date
Tue  4 Apr 13:41:25 BST 2023
root@raspberrypi:~#
```

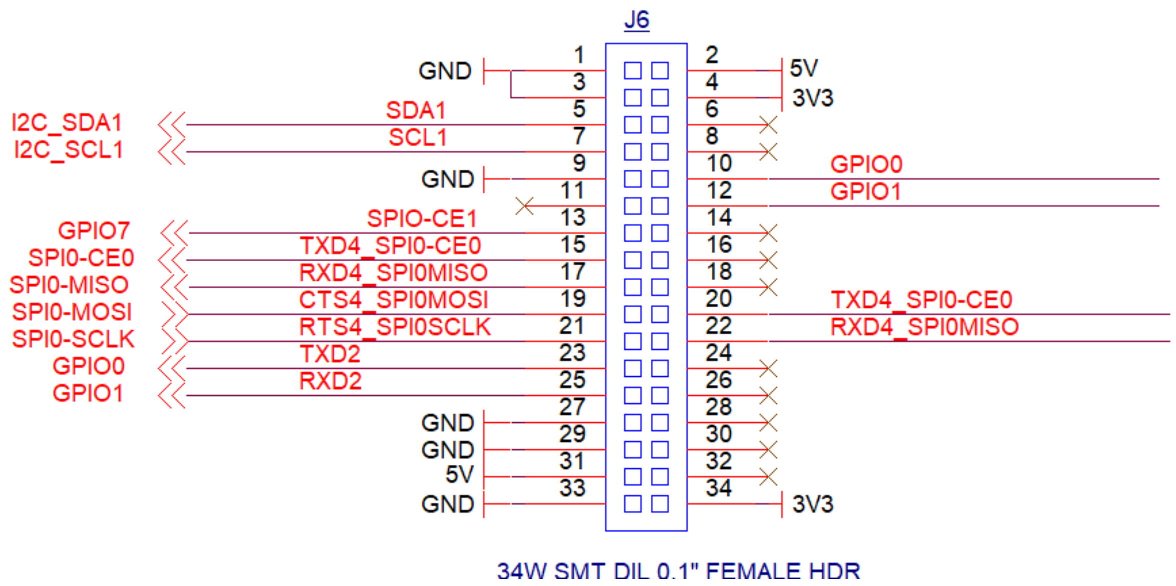
To test the system operation in the event of a kernel fault run the below to provoke a kernel panic

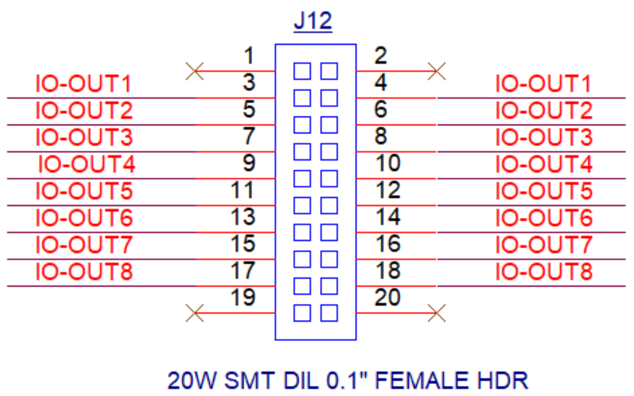
```
# echo c > /proc/sysrq-trigger
```

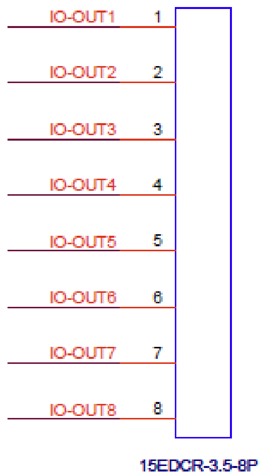
Alternately a recursive "fork bomb" which causes all CPU resources to be used can be provoked using the command below

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

# GPIO CARD SLOT




Note that the green 8 way plug in screw terminal connector is uncommitted and is defined by the signals connected to IO-OUT on the 20way connector giving rise to a truly flexible IO interface solution.

Template files for this card can be downloaded from the website and is the same form factor as the CM3 based integrator board.

**Note that 'double height' IO cards require 21mm headers minimum to clear the LAN port**

**Note that the GPIO 3V has a recommended maximum capacity of 500mA**

*The IO Card slot on the board supports the ISO-ADC,ISO-485(-DUO),ISO-CAN(-DUO) & ISO-SENSOR-KIT/SENSOR-KIT-2*

## GPIO Card Slot Pin Functions

| Pin | Signal | ALT0 | ALT3 | ALT4 | ALT5 | Pin | Signal | ALT0 | ALT3 | ALT4 | ALT5 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | GND | | | | | 2 | +5V | | | | |
| 3 | GND | | | | | 4 | +3.3V | | | | |
| 5 | GPIO2 | SDA1 | | | SDA3 | 6 | NC | | | | |
| 7 | GPIO3 | SCL1 | | | SCL3 | 8 | NC | | | | |
| 9 | GND | | | | | 10 | GPIO0 | SDA0 | | TXD-2 | SDA6 |
| 11 | NC | | | | | 12 | GPIO1 | SCL0 | | RXD-2 | SCL6 |
| 13 | GPIO7 | SPI0-CE1 | SPI4-SCLK | RTS-3 | SCL4 | 14 | NC | | | | |
| 15 | GPIO8 | SPI0-CE0 | I2CSL CE | TXD-4 | SDA4 | 16 | NC | | | | |
| 17 | GPIO9 | SPI0-MISO | I2CSL SDI | RXD-4 | SCL4 | 18 | NC | | | | |
| 19 | GPIO10 | SPI0-MOSI | I2CSL SDA | CTS-4 | SDA5 | 20 | GPIO8 | SPI0-CE0 | I2CSL CE | TXD-4 | SDA4 |
| 21 | GPIO11 | SPI0-SCLK | I2CSL SCL | RTS-4 | SCL5 | 22 | GPIO9 | SPI0-MISO | I2CSL SDI | RXD-4 | SCL4 |
| 23 | GPIO0 | SDA0 | | TXD-2 | SDA6 | 24 | NC | | | | |
| 25 | GPIO1 | SCL0 | | RXD-2 | SCL6 | 26 | NC | | | | |
| 27 | GND | | | | | 28 | NC | | | | |
| 29 | GND | | | | | 30 | NC | | | | |
| 31 | +5V | | | | | 32 | NC | | | | |
| 33 | GND | | | | | 34 | +3.3V | | | | |

## Full Pi GPIO Function Listing

| GPIO | Pull | Notes | ALT0 | ALT3 | ALT4 | ALT5 |
|---|---|---|---|---|---|---|
| GPIO0 | High | ASIO | SDA0 | | | |
| GPIO1 | High | ASIO | SCL0 | | | |
| GPIO2 | High | IO-CONNECTOR | SDA1 | | | |
| GPIO3 | High | IO-CONNECTOR | SCL1 | | | |
| GPIO4 | High | XBEE-TX3 | | | TXD-3 | |
| GPIO5 | High | XBEE-RX3 | | | RXD-3 | |
| GPIO6 | High | WDOG-DISABLE | | | | |
| GPIO7 | High | IO-CONNECTOR | SPI0-CE1 | | | SCL4 |
| GPIO8 | High | IO-CONNECTOR | SPI0-CE0 | I2CSL CE_N | TXD-4 | SDA4 |
| GPIO9 | Low | IO-CONNECTOR | SPI0-MISO | I2CSL SDI | RXD-4 | SCL4 |
| GPIO10 | Low | IO-CONNECTOR | SPI0-MOSI | I2CSL SDA | CTS-4 | SDA5 |
| GPIO11 | Low | IO-CONNECTOR | SPI0-SCLK | I2CSL SCL | RTS-4 | SCL5 |
| GPIO12 | Low | RS485-TX5 | | | TXD-5 | |
| GPIO13 | Low | RS485-RX5 | | | RXD-5 | |
| GPIO14 | Low | RS232-TX0 | TXD-0 | | | |
| GPIO15 | Low | RS232-RX0 | RXD-0 | | | |
| GPIO16 | Low | POWER-LATCH | | | | |
| GPIO17 | Low | WDOG-INPUT | | | | |
| GPIO18 | Low | On Board TPM | | | SPI1-CE0 | |
| GPIO19 | Low | On Board TPM | | | SPI1-MISO | |
| GPIO20 | Low | On Board TPM | | | SPI1-MOSI | |
| GPIO21 | Low | On Board TPM | | | SPI1-SCLK | |
| GPIO22 | Low | SD-CARD | | SD1_CLK | | |
| GPIO23 | Low | SD-CARD | | SD1_CMD | | |
| GPIO24 | Low | SD-CARD | | SD1_DAT0 | | |
| GPIO25 | Low | SD-CARD | | SD1_DAT1 | | |
| GPIO26 | Low | SD-CARD | | SD1_DAT2 | | |
| GPIO27 | Low | SD-CARD | | SD1_DAT3 | | |

# 10/100/1000 ETHERNET INTERFACE

The integrated Gigabit has been brought out to the main face of the card, this takes the place of the RJ45 Serial port on the CM3 Integrator board, so as to allow easy hardware migration.

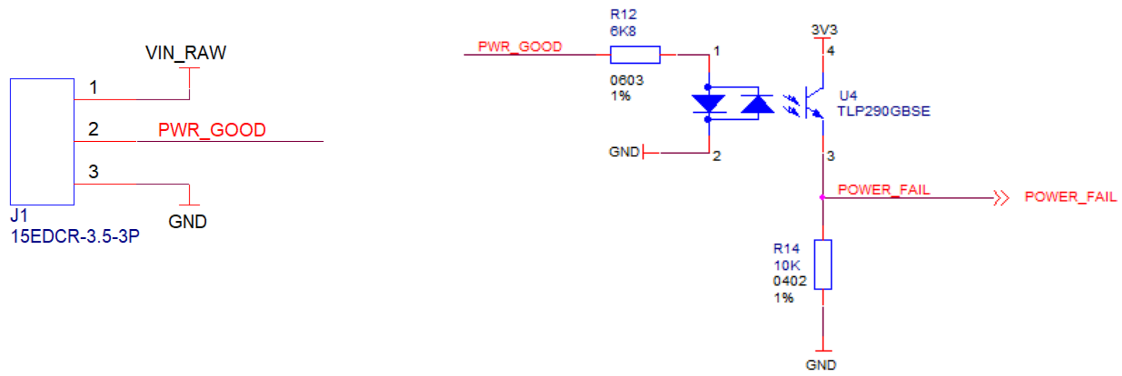
Note that the Gigabit interface takes its MAC address from the CM4's serial number.

# POWER-GOOD DIGITAL INPUT

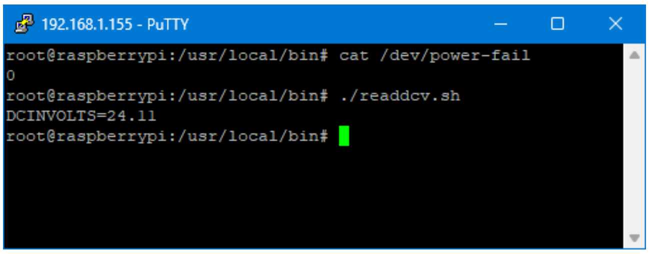
The power input connector performs several functions but is there to provide power in gating to allow the use of an external DC UPS or another type of activation source.

The middle pin provides a Power-Good indication, which can be used in conjunction with an external DC UPS to indicate when the UPS is running on batteries or mains power.  This logical state can be read by

The board has an ADC to allow monitoring of the DC input voltage at J1, see the **readdcv.sh** script for more details.

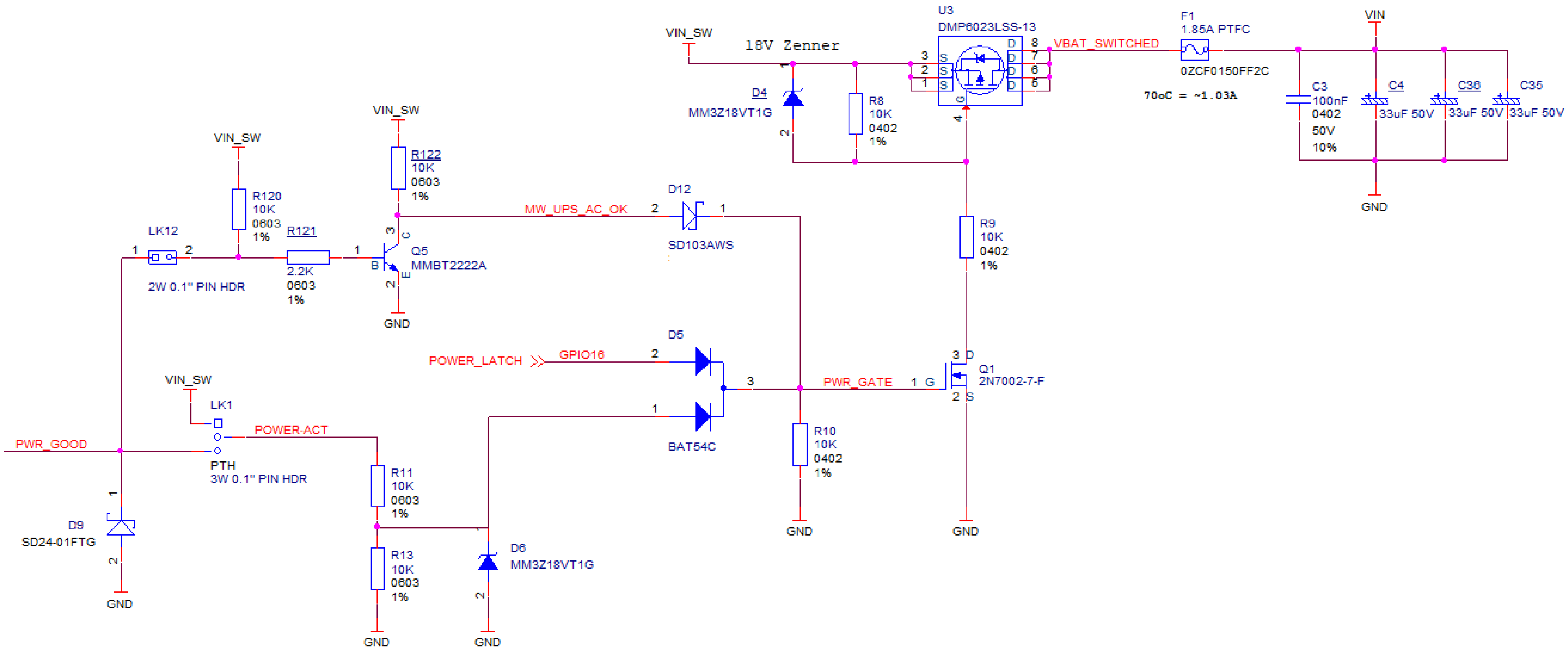



We have created a simple bash script to make the process of reading the DC Voltage easy and report the logical level of the **POWER_FAIL** input state (I2C-GPIO3)




This is configured via the below lines in **config.txt** and the setting of LK1 & LK2 – see over

```
################################################################
#  POWER LATCH
################################################################
# This line latches the power gate FET on early in the boot
# process, this defaults off so will give power off at shutdown
# when in UPS mode with middle power pin power-good input
################################################################
gpio=16=op,dh

# UPS (Battery) input voltage ADC
dtoverlay=ads1015,addr=0x49,cha_gain=0,chb_gain=0,chc_gain=0,chd_gain=0
```




| LK1 | LK12 | Operating Mode |
| --- | --- | --- |
| Fitted 1-2 | Not Fitted | Disable gated start-up, U3 FET Switch is always enabled and the board always powers up when power is applied |
| Fitted 2-3 | Not Fitted | Enabled gated start-up, U3 FET Switch is enabled only when DC in connector Power-Good signal is >5V |
| Not Fitted | Fitted | Enabled gated start-up, U3 FET Switch is enabled only when DC in connector Power-Good signal is 0V* |

*LK12 Operation is setup for use with MeanWell DRC-40A UPS with AC-OK +ve input connected to POWER-GOOD signal

# OS CONFIGURATION FILES

This is a list of the files altered from a base install to create the demo OS image

```
/boot/dt-blob.bin
/boot/config.txt
/boot/cmdline.txt
/boot/overlays/tpm-spi1.dtbo
/boot/overlays/ext-watchdog.dtbo
/boot/overlays/sdhost-custom.dtbo.dtbo
/boot/overlays/disablepcie.dtbo
/boot/overlays/novchiq.dtbo

/usr/local/bin/tpmtool
/usr/local/bin/tpmupdate
/usr/local/bin/modemstat
/usr/local/bin/qmi-network-raw

/usr/local/bin/eeprog
/usr/local/bin/setup-gpio.sh
/usr/local/bin/setup-adc.sh
/usr/local/bin/readdcv.sh
/usr/local/bin/fanctl.sh
/usr/local/bin/tpminfo.sh

/usr/local/bin/mbpoll

/etc/udev/rules.d/8-sdcard.rules
/etc/udev/rules.d/10-ftdi-usbserial.rules
/etc/udev/rules.d/20-modem-ec2x.rules
/etc/udev/rules.d/20-modem-7xxxx.rules
/etc/udev/rules.d/30-uart-symlinks.rules
/etc/udev/rules.d/40-i2c-gpio.rules
/etc/udev/rules.d/50-i2c-adc.rules

/etc/udev/rules.d/modem-rules/*

/etc/issue
/etc/modules

/root/backups/*
/root/boot-config/*
/root/image-create/*
/root/overlays/*
/root/stresscpu.sh
/root/tpm-toolkit/*
/root/watchdog/*
/root/device-tree/*
/root/ext-watchdog/*
/root/startup-shortcuts/*
/root/quectel-CM/*
```

# RASPBERRY PI DOCUMENTATION

Raspberry Pi have produced a comprehensive knowledge base on how to configure and control various aspects of the Compute Module and it's OS.

## https://www.raspberrypi.com/documentation

# SCHEMATICS

A reduced schematic set can be provided on request, please contact your technical support representative for more details.

# Pi Module Power Consumption Tests

Tests were done on the 5V power consumption of the Pi module in a headless/Lite configuration running various stress test programs:

**Test Setup**

- Gigabit Ethernet plugged in
- No front port USB devices plugged in.

Note Internal 5V PSU is 3.5A Continuous 5A Peak, readings were highest steady state observed

### CM4

| Test | 5V Current Draw |
|---|---|
| Idle | 199mA |
| `sysbench --num-threads=4 --test=cpu --cpu-max-prime=20000 --validate run` | 585mA |
| `stress --cpu 8 --io 4 --vm 2 --vm-bytes 128M --timeout 30s` | 587mA |

### CM5

| Test | 5V Current Draw |
|---|---|
| Idle | 428mA |
| `sysbench --num-threads=4 --test=cpu --cpu-max-prime=20000 --validate run` | 970mA |
| `stress --cpu 8 --io 4 --vm 2 --vm-bytes 128M --timeout 30s` | 1.2A |

Note the CM5 module operates **significantly** hotter than the CM4 so active cooling using a heatsink and cpu/case fan operation is recommended.

# 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