



# **MyPi Industrial CM3+ Integrator Board**

## **User Guide**

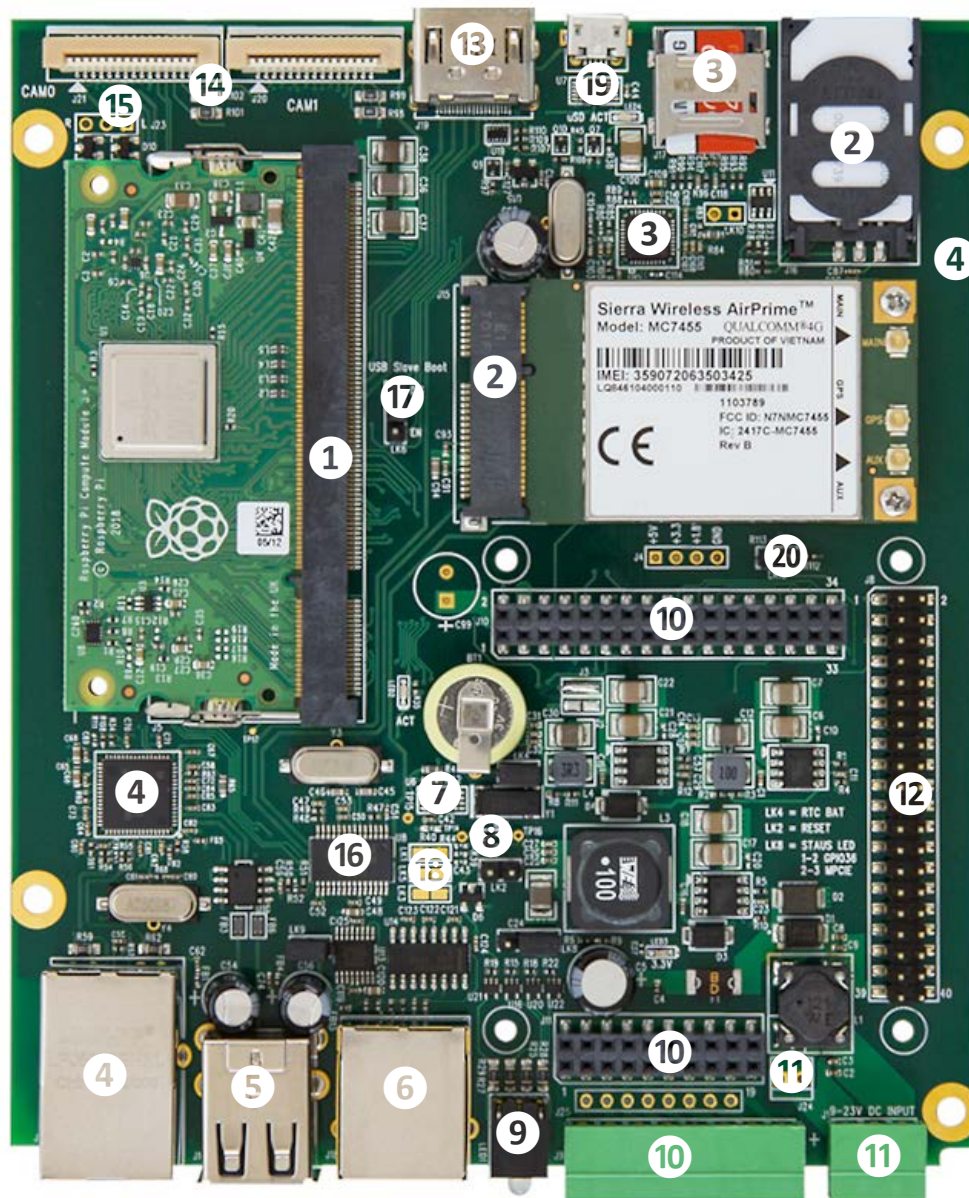
Issue : 1.00  
Dated : January 2022  
Prepared By : Andrew O'Connell

## FEATURES

---

- Raspberry Pi Compute Module 1 & 3/3+ compatible
- Integrated 10/100 Ethernet adapter
- 2 x USB 2.0 master interfaces
- Mini PCI-e Interface + SIM connector For use with cellular modems
- Ruggedised high speed USB interfaced  $\mu$ SD Card interface, providing additional storage separate from the Compute Module eMMC flash
- Integrated USB RS232 UART (RX/TX/RTS/CTS) using RJ45 connector for maximum range.
- Modular IO Cards for application specific IO solutions
- Dual high-resolution Raspberry Pi camera interfaces
- Raspberry Pi 2 HAT compatible I/O connector & mounting points
- Integrated battery backed real-time-clock (RTC)
- Additional 1.6s hardware watchdog for added system resilience
- HDMI out
- 2 x Bi-colour (red/green) front panel status LEDs
- 8Way 2-part 3.5mm Screw terminal connector for use with modular IO card outputs or HAT board
- Wide 7-24V (poly-fused and filtered) DC power input range
- Available in standard (0 to 70°C) and extended range (-25 to 80°C) temperature versions for demanding environments
- Core PCB Size : 125 x 142mm

## BOARD IO FEATURES



- |    |  |    |                                |
|----|--|----|--------------------------------|
| 1  | Compute Module 1/3/3+ Socket             | 11 | Power In (9-24V DC)            |
| 2  | mPCIe Socket + Modem SIM Socket          | 12 | Raspberry Pi HAT Connector     |
| 3  | USB $\mu$ SD Card Interface + Socket     | 13 | HDMI Out                       |
| 4  | USB LAN9512 10/100 LAN + USB Interface   | 14 | Dual Camera Interface          |
| 5  | 2 x USB 2.0 Ports                        | 15 | Audio Out                      |
| 6  | RJ45 RS232 Port (USB Interface)          | 16 | USB Hub                        |
| 7  | I2C DS1339U-33+ RTC + Battery Backup     | 17 | Programming Mode Selector Link |
| 8  | External Watchdog                        | 18 | Watchdog Selector Links        |
| 9  | Dual Bi-colour LED                       | 19 | $\mu$ USB CM Programming port  |
| 10 | GPIO IO Card interface + Front Connector | 20 | Modem Reset GPIO23 Link        |

## HARDWARE CONFIGURATION LINKS

---



### **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.



### **LK1 - WATCHDOG OUT**

Fitted Connect External Watchdog Out Line to RPI RUN/RESET Line

Open Default

### **LK5 - WATCHDOG ENABLE**

Fitted Connect External Watchdog Input Line to GPIO29\*

Open Default

### **LK3 - WATCHDOG INPUT**

Fitted Connect External Watchdog Enable to GPIO28\*

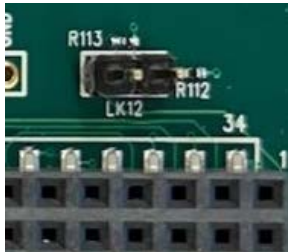
Open Default

**\*GPIO LINES ALSO USED FOR CAMERA 0**

**LK12 - mPCIe EMERGENCY RESET**

Fitted                      Fit to connect GPIO23 to PERST (pin 22) of mPCIe socket

Open                      Default



**LK6 - Compute Module Programming Mode (USB SLAVE BOOT MODE)**

Fitted DIS                Compute module programming forced as disabled

Fitted EN                Compute module programming enabled (fit USB programming cable in to activate)



**LK9 - RS232 Connector 5V power Out**

Fitted DIS DTR Line Floating

Fitted EN Fit to pull DTR RS232 line to +5V (default fitted)



**LK8 - Green Status LED 1 Drive Source**

Fitted 1-2 GPIO36 (default)

Fitted 2-3 mPCIe MODEM/WIFI Status LED Signal



# RASPBERRY PI COMPUTE MODULE PROGRAMMING

---

The unit as shipped is configured to allow the eMMC flash on the compute module to be re-programmed

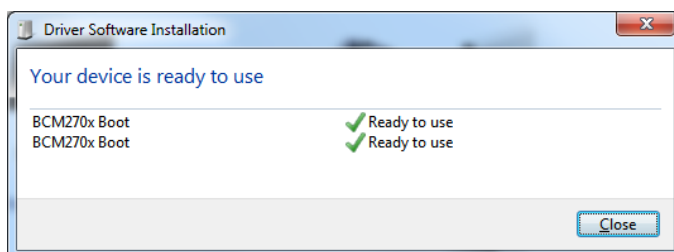
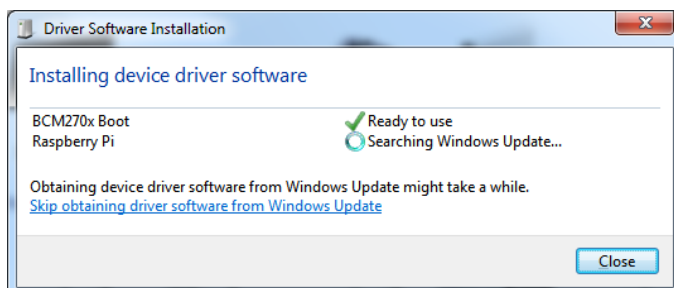
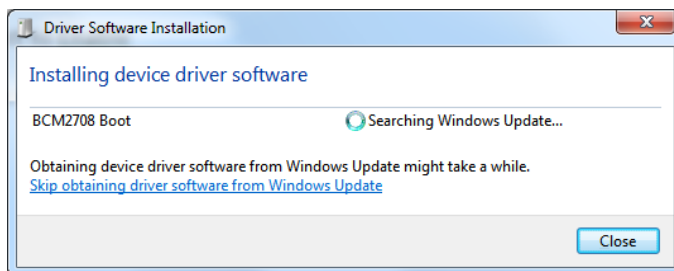
Demo kit units come complete with Compute modules that are pre-programmed with the demo Raspbian OS pre-installed, this section describes how to write a new disk image to the Compute Module.

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](#)

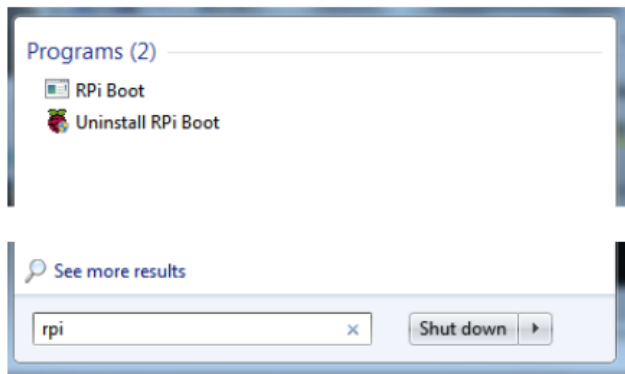
Connect the mini USB connector to the Windows PC using the supplied USB A to micro USB B data cable, fit the programming mode jumper link (LK6) to EN and then power up the unit.

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

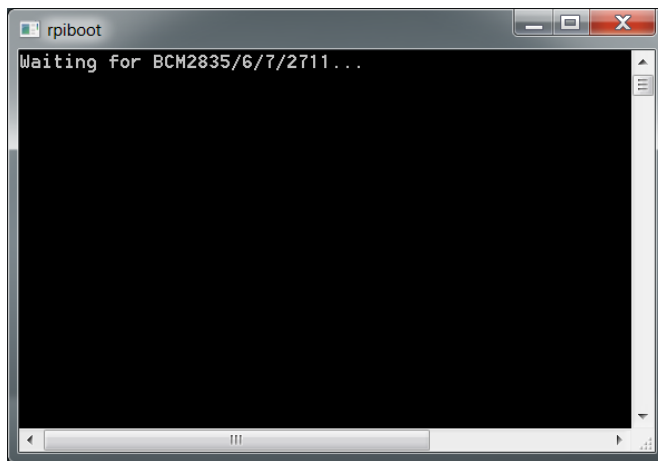


Once that sequence has finished Windows has now installed the required drivers and you can power off the unit for a moment whilst we get the PC side ready for the next step.

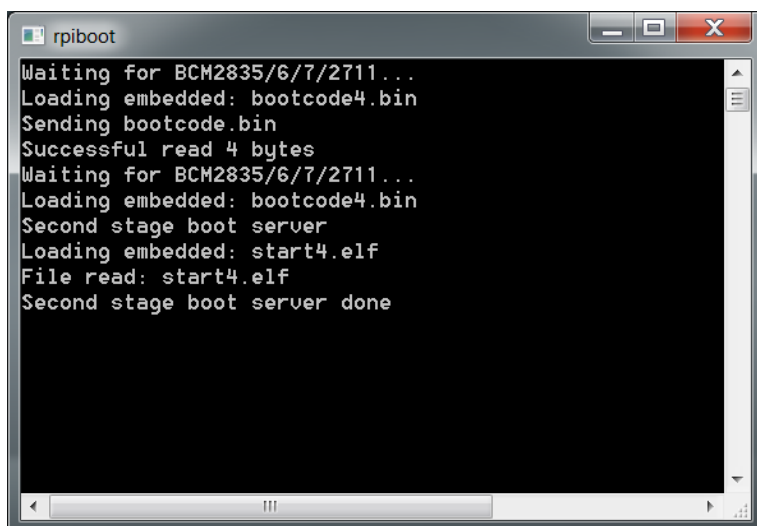
Making sure you have the unit powered off start up RPi Boot, this is easiest done via the start menu, we have found this needs to be run as 'Administrator' privilege mode for correct operation



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 :

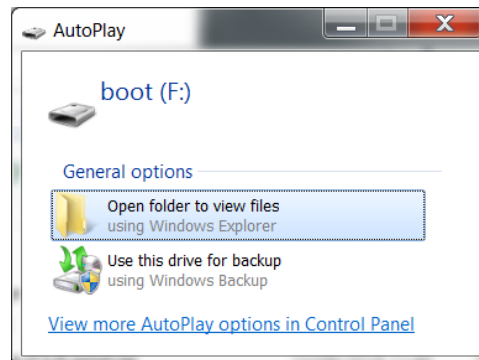


When done the compute module will alternate into mass storage mode (so behaving just as though it's a USB memory stick) and windows will then recognise the module as an external drive.



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

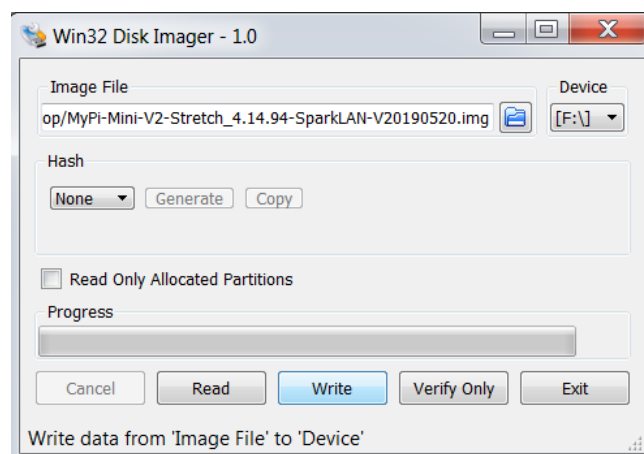
This process writes a disk image, containing the partition table as well as both FAT boot partition and Linux EXT partitions, **over the entire disk**.

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 the data on the computer's hard disk!**

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



Once complete power off the unit and set the **USB Boot** jumper link back to **Disabled**, and finally remove the USB cable.

**Failing to do this will prevent the on-board USB hub from working when the board is rebooted due to CM's USB master being still switched over to the programming socket and not the internal bus**

The same utility can also create snapshot images of the current image config to save time, although note this is a straight binary dump of the entire disk not just the parts with files in so the image files end up quite big and take a long time to read/write

Created images can be cleaned and compressed using **pishrink** utility to speed up programming time

<https://github.com/Drewsif/PiShrink/>

## SYSTEM GPIO

These GPIO lines that have been reserved for system use, most are not accessible from other interface connectors

The startup file `/etc/init.d/mypi.sh` exports these for OS usage and creates shortcut entries in `/dev` for easy reference

GPIO	/dev Shortcut	Description	Active	Default
0	-	CAM1 I2C (I2C0)	-	-
1	-	CAM1 I2C (I2C0)	-	-
20	-	PSU Connector Middle Pin via LK11 (3.3V Logic)	-	-
21	-	CAM1 Power Enable	-	-
5	-	CAM1 LED Indicator	-	-
22	-	CAM0 Power Enable	-	-
4	-	CAM0 LED Indicator	-	-
28	-	CAM0 I2C / Hardware Watchdog In	-	-
29	-	CAM0 I2C / Hardware Watchdog Enable	-	-
34	sd-disable	USB SD Card /Reset (Disable)	Low	High
35	led-red	Status LED Red	High	Low
36	led-green	Status LED Green	High	Low
39	pcie-wdis	mPCIe Wireless Disable (Airplane mode)	High	Low
23	pcie-reset	mPCIe Modem Emergency Reset (fit LK12)	High	Low
44	lan-disable	USB LAN /Reset (Disable)	Low	High
40	-	Audio R	-	-
45	-	Audio L	-	-
46	-	HDMI HPD	-	-
47	-	Pi Compute Module Activity LED	-	-

GPIO Example usage using created /dev shortcuts:

```
$ echo 1 >/dev/sd-disable # Reset/Disable SD Interface Chip
$ echo 0 >/dev/sd-disable # Enable SD Interface Chip

$ echo 1 >/dev/lan-disable # Reset/Disable LAN Interface Chip
$ echo 0 >/dev/lan-disable # Enable LAN Interface Chip

$ echo 1 >/dev/pcie-wdis # Disable RF output from mPCIe card
$ echo 0 >/dev/pcie-wdis # Enable RF output from mPCIe card

$ echo 1 >/dev/pcie-reset # Reset/Disable mPCIe card
$ echo 0 >/dev/pcie-reset # Enable mPCIe card

$ echo 1 >/dev/led1-red # Switch Red Status LED on
$ echo 0 >/dev/led1-red # Switch Red Status LED off

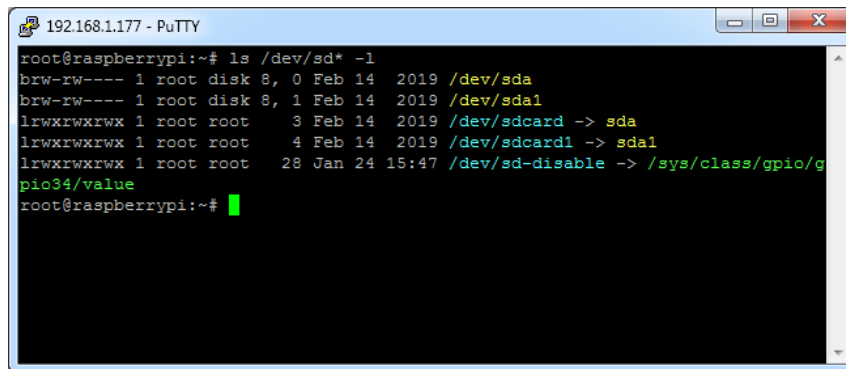
$ echo 1 >/dev/led2-green # Switch Green Status LED on
$ echo 0 >/dev/led2-green # Switch Green Status LED off
```

# USB SD CARD INTERFACE

---

The on-board micro SD Card is interfaced to the Raspberry Pi Compute Module using on-board Microchip USB2240 SD card interface controller, providing fast access to secondary storage for datalogging.

Configuration file **/etc/udev/rules.d/8-sdcard.rules** creates the below **/dev** shortcuts for the main SD Card and any partitions contained (once a card is fitted)



```
192.168.1.177 - PuTTY
root@raspberrypi:~# ls -l /dev/sd* -l
brw-rw---- 1 root disk 8, 0 Feb 14 2019 /dev/sda
brw-rw---- 1 root disk 8, 1 Feb 14 2019 /dev/sda1
lrwxrwxrwx 1 root root 3 Feb 14 2019 /dev/sdcard -> sda
lrwxrwxrwx 1 root root 4 Feb 14 2019 /dev/sdcard1 -> sda1
lrwxrwxrwx 1 root root 28 Jan 24 15:47 /dev/sd-disable -> /sys/class/gpio/gpio34/value
root@raspberrypi:~#
```

The **/dev/sdcardx** reference can then be used in **/etc/fstab** to mount the partitions, rather than the **/dev/sdx** reference to avoid clashing with other USB interfaced media

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>

The SD Card interface chip gets a power up reset pulse, the below lines optionally allow you direct control over the chip's reset signal. Disabling the chip also reduces the system power draw.

```
$ echo 1 >/dev/sd-disable # Reset/Disable SD Interface Chip
$ echo 0 >/dev/sd-disable # Enable SD Interface Chip
```

## USB 10/100 LAN + USB CONTROLLER

---

Integrated on-board is an Microchip LAN9512 device, this is connected to the Raspberry Pi via the on board USB HUB port which provides 2 additional downstream USB ports, which are brought out to the front face USB ports.

There are two scripts that are helpful:

**/usr/local/bin/resetbyauthorized.sh**

This script allows you to issue a software reset command to a USB peripheral by supplying the **vendorid** & **productid** identifiers (can be found using lsusb)

**/usr/local/bin/usbprctl.sh**

This script allows you to switch the power off/on to either/both of the front USB ports

The LAN chip gets a power up reset pulse, the below lines optionally allow you direct control over the LAN chip reset signal.

Disabling the LAN chip also reduces the power draw of the system significantly.

Note that you should disable/bring down any LAN related interface (e.g. eth0) before disabling the port to avoid OS related problems.

```
$ echo 1 >/dev/lan-disable # Reset/Disable LAN Interface Chip
$ echo 0 >/dev/lan-disable # Enable LAN Interface Chip
```

## USB MINI-PCIE INTERFACE

---

The Integrated mPCIe socket 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
<b>Mechanical Key</b>			
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

The mPCIe USB signals are connected to the on-board USB hub chip.

The WWAN/WLAN LED signals can be optionally connected to the front top green bi-colour LED, to indicate modem network registration/data transmission status, by setting LK8 to position 2-3.

## Modem Compatibility/Operation

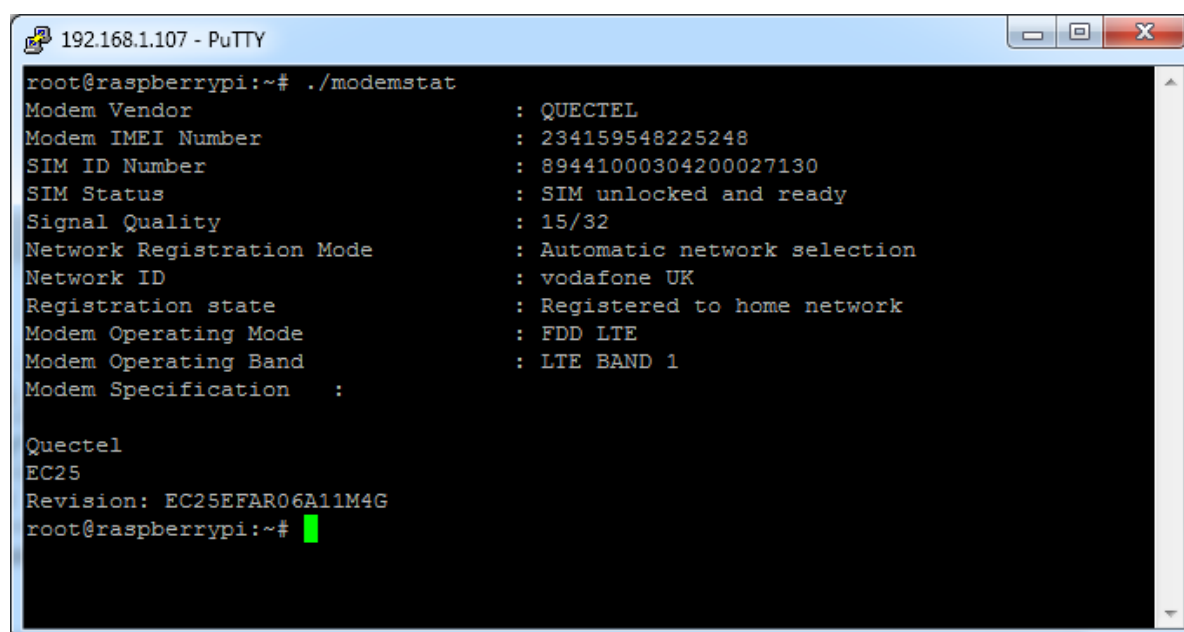
See the below link to pages from the main modem documentation section for details on how to operate modems :

<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>



```
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.

Combined versions for SIMCOM SIM7xxx and Quectel EC2x modems are pre-installed on the demo image

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:~#
```

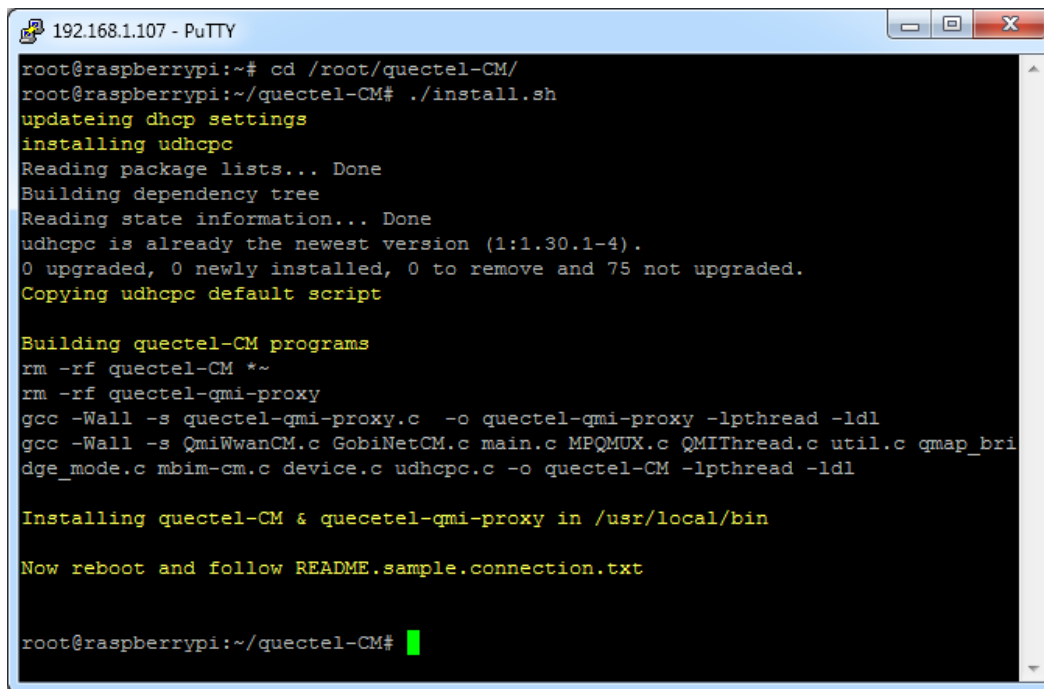


## QUECTEL-CM example

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`

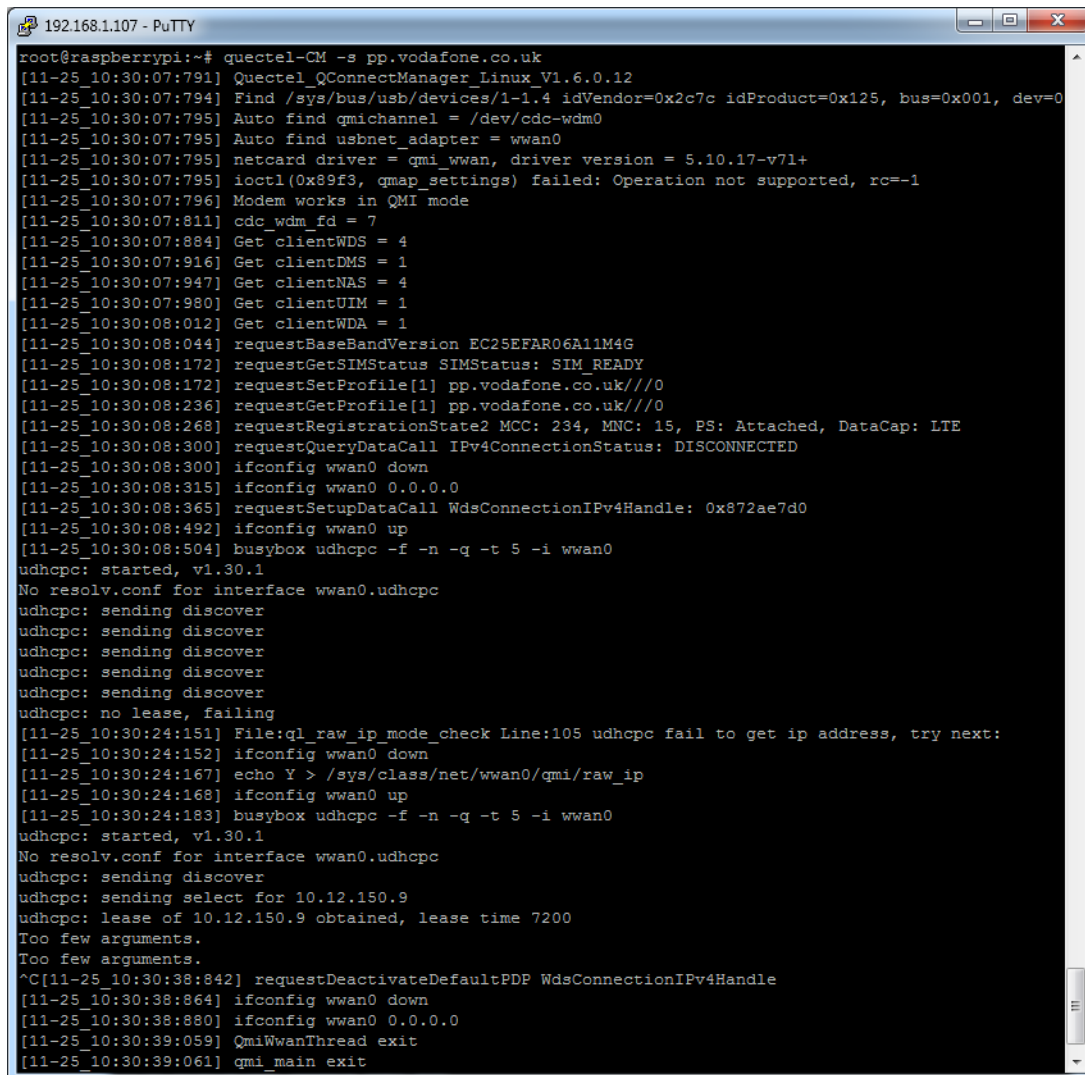
**Example 2:** `./quectel-CM -s pp.vodafone.co.uk`

**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:~# quetcml-CM -s pp.vodafone.co.uk
[11-25_10:30:07:791] Quetcml_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 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
[11-25_10:30:24:151] File:ql_raw_ip_mode_check Line:105 udhcpc 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 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 10.12.150.9
udhcpc: lease of 10.12.150.9 obtained, lease time 7200
Too few arguments.
Too few arguments.
^C[11-25_10:30:38:842] requestDeactivateDefaultPDF 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/ttyS2**

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

## COM PORTS

There are three on board serial ports available on the MyPi base board

UART0&1 are direct from the RPi module and available on the GPIO IO card slot

An additional serial port is available via the front RJ45 connector which derived from an on board FTDI 230XS USB-UART device.

Further serial ports can be added via either the mPCIe port or plugging additional adapters into the front USB ports, see the table below for

Name	OS Port	Type	RTS/CTS?
UART0	/dev/ttyAMA0	PL011 Full UART	Yes**
UART1	/dev/ttyS0	Mini UART	No
USB-SERIAL0	/dev/ttyS1*	On Board FT230XS USB UART	Yes
USB-SERIAL1	/dev/ttyS2*	mPCIe FT230XS USB UART	Yes
USB-SERIAL2	/dev/ttyS3*	Top USB Port FTDI USB Serial Adapter	Yes
USB-SERIAL3	/dev/ttyS4*	Bottom USB Port FTDI USB Serial Adapter	Yes

\* Will appear as a ttyUSBx port, udev rules will create /dev/ttySx symlink short-cuts – see **/etc/udev/rules.d**

\*\* RTS/CTS lines optional and configured via device-tree overlays (not suitable for 485 flow control)

Note that **UART1** was originally intended as a serial console rather than a full featured UART, as a result it has a few quirks in the settings it can reliably use (including baud rates being affected by the clock rate of the CPU). For this reason care should be taken to assess its suitability for usage, for more information see this web page : [HERE](#)

**UART0** is the preferred choice when using for serial communications between devices due to having a more complete feature set.

**USB-SERIAL0** is permanently enabled as RS232 via the RJ45 socket on the front face (wired to Cisco console cable format) to give maximum cable distance.

The GPIO pins UART0 & 1 appear on are user-definable via device tree overlays

GPIO	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
14	TX0					TX1
15	RX0					TX1
16				CTS0		
17				RTS0		
30				CTS0		
31				RTS0		
32				TX0		TX1
33				RX0		RX1

The standard setup of the system assigns UART0 to pins 32/33, UART1 to 14/15

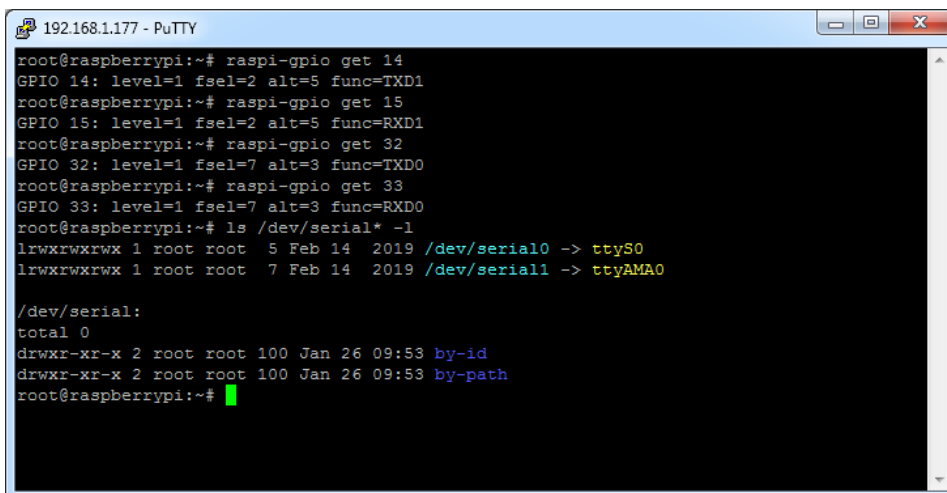
The lines below in `/boot/config.txt` configure the system to this setup

```
dtoverlay=mypi-uart0,txd0_pin=32,rxid0_pin=33
dtoverlay=uart1,txd1_pin=14,rxid1_pin=15
core_freq=250
dtoverlay=uart_swap
enable_uart=1
```

The last 3 lines are important as these prevent baud rate issues, and ensure correct port assignments from the defaults (relating to the Bluetooth on the Pi3).

The source file for the non-standard overlays can be found in `/root/device-tree`

This can be confirmed by using the `raspi-gpio` tool

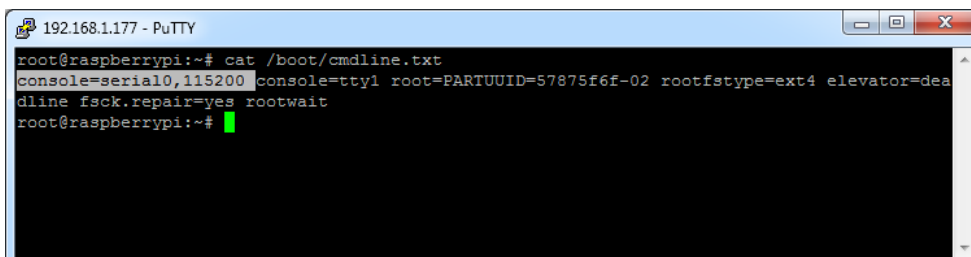


```
192.168.1.177 - PuTTY
root@raspberrypi:~# raspi-gpio get 14
GPIO 14: level=1 fsel=2 alt=5 func=TXD1
root@raspberrypi:~# raspi-gpio get 15
GPIO 15: level=1 fsel=2 alt=5 func=RXD1
root@raspberrypi:~# raspi-gpio get 32
GPIO 32: level=1 fsel=7 alt=3 func=TXD0
root@raspberrypi:~# raspi-gpio get 33
GPIO 33: level=1 fsel=7 alt=3 func=RXD0
root@raspberrypi:~# ls /dev/serial* -l
lrwxrwxrwx 1 root root 5 Feb 14 2019 /dev/serial0 -> ttyS0
lrwxrwxrwx 1 root root 7 Feb 14 2019 /dev/serial1 -> ttyAMA0

/dev/serial:
total 0
drwxr-xr-x 2 root root 100 Jan 26 09:53 by-id
drwxr-xr-x 2 root root 100 Jan 26 09:53 by-path
root@raspberrypi:~#
```

Note that serial0 (which is the console port) is correctly allocated to ttyS0

To remove the serial console edit `/boot/cmdline.txt` and remove `console=serial0,115200` from the front



```
192.168.1.177 - PuTTY
root@raspberrypi:~# cat /boot/cmdline.txt
console=serial0,115200 console=tty1 root=PARTUUID=57875f6f-02 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait
root@raspberrypi:~#
```

Next disable the system serial console service and reboot the unit

```
# systemctl disable serial-getty@ttyS0.service
```

## RJ45 Serial Port

The front RJ45 RS232 Serial port is wired as below table shows, the bottom Bi Colour LED also shows Rx/Tx Activity



Pin	Signal
1	RTS
2	5V / N.C. – LK9 link
3	TX
	GND
5	GND
6	RX
7	-
8	CTS

This can either be converted back to a D9-M Serial connector as the below wiring scheme shows

RJ45 Pin	Signal	Direction	D9-M Pin	Signal
1	RTS	➔	7	RTS
2	5V / N.C. – LK9 link	➔	4	DTR
3	TX	➔	3	TX
4	GND	-	5	GND
5	GND	-	-	N/C
6	RX	➔	2	RX
7	-	-	6	DSR
8	CTS	➔	8	CTS
-	-	-	9	RI
-	-	-	1	DCD

*Direction shown is from the PCB Connector*

Alternatively use compatible Cisco 72-3383-01 RJ45 - DB9F (Cross-Over/Null Modem) Console Cable:

RJ45 Pin	Signal	Direction	D9-F Pin	Signal
1	RTS	➔	8	CTS
2	5V / N.C. – LK9 link	➔	6	DSR
3	TX	➔	2	RX
4	GND	-	5	GND
5	GND	-	5	N/C
6	RX	➔	3	TX
7	-	-	4	DTR
8	CTS	➔	7	RTS

*Direction shown is from the PCB Connector*

## 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 line in **/boot/config.txt**

```
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>

# WATCHDOG

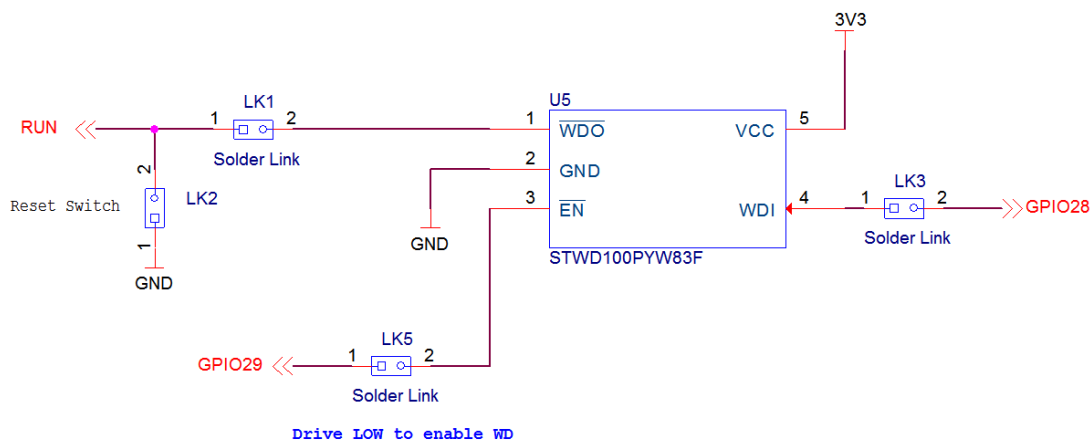
The on-board external 1.6 second watchdog is a single chip part provided by ST STWD100PYW83F, the reset output of this part is connected to the Raspberry Pi Compute module.

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.

In order to enable the watchdog the below solder links need to be bridged -- this is a factory setting so please request with your order if you require these links to be pre-made for production quantities.



The external watchdog device is driven by GPIO29 (/WD Enable) and GPIO28 (/WD Input), by default the watchdog is disabled as GPIO29 is pulled high by default pin state.



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 >1uS long for the watchdog pulse 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 GPIO29 (/WD Enable) to be pulled high (as the Pi CPU resets) and so disable the watchdog allowing the system to boot without further time out reset occurring.

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. The other on board devices have RC circuits to provide an initial power-up reset pulse.

When the system hard resets in this manner all GPIO lines will revert back to their default state, which will have implications for any GPIO driven IO devices.

Full datasheet for watchdog part: [Download Link](#)

## External Watchdog OS Integration

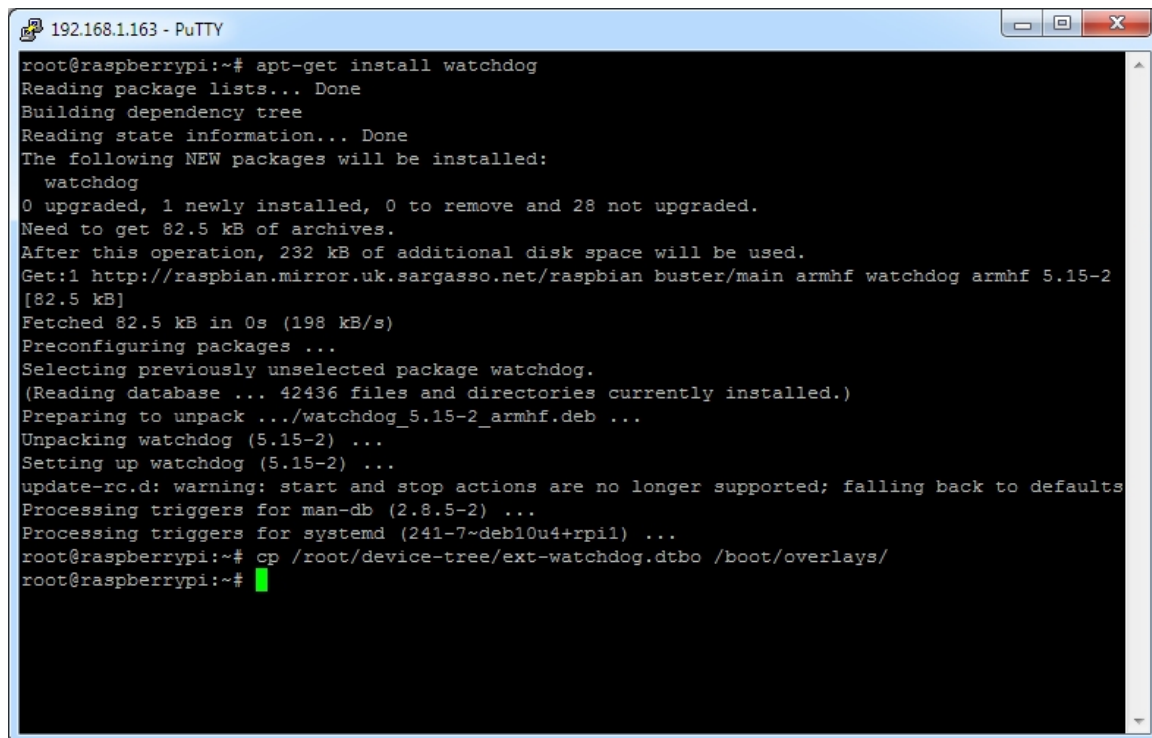
Integrated into the Raspbian kernel and OS there are pre-built utilities for configuring and managing watchdogs, in this example we will show how to configure the OS such that a file's last update timestamp will trigger a watchdog time out.

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 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**

Start by installing the requisite files and configuring them



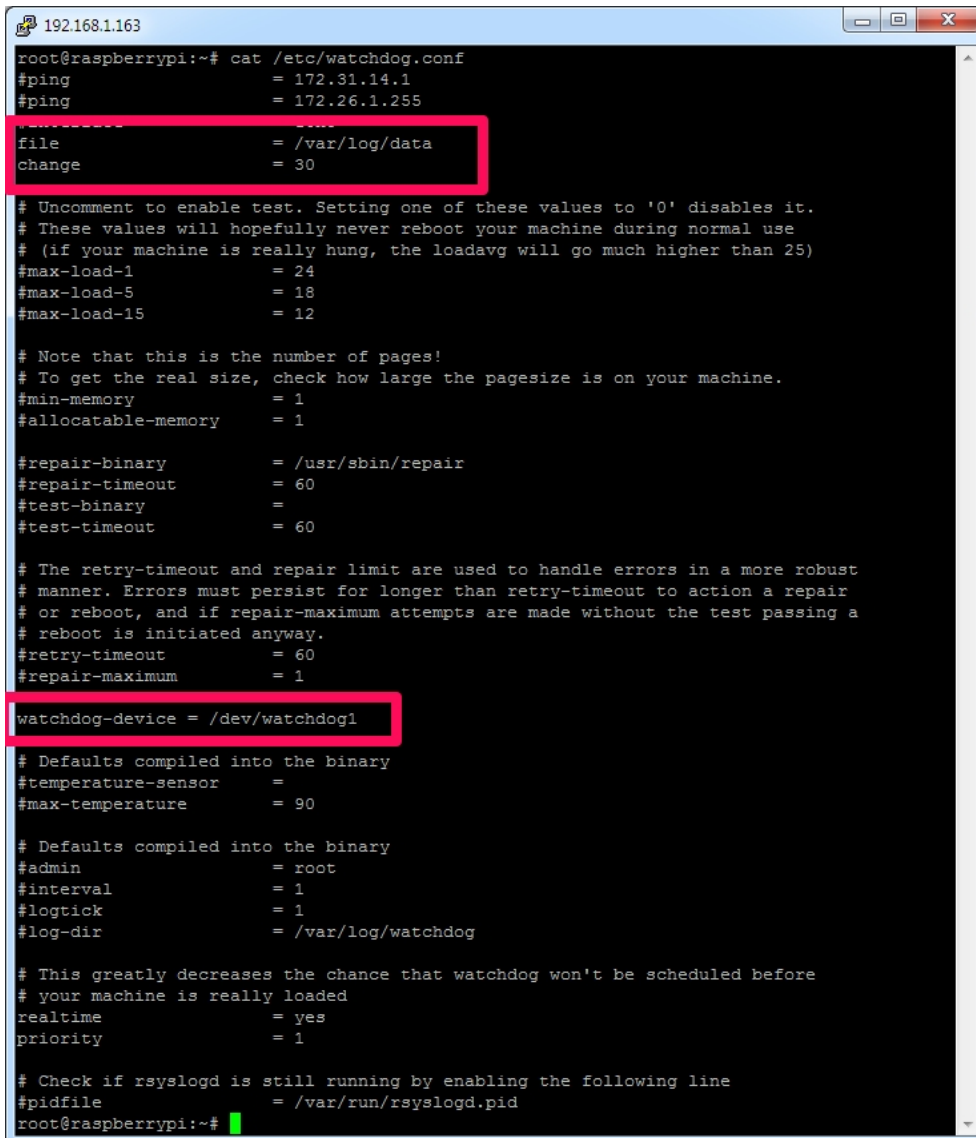
```
192.168.1.163 - PuTTY
root@raspberrypi:~# apt-get install watchdog
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  watchdog
0 upgraded, 1 newly installed, 0 to remove and 28 not upgraded.
Need to get 82.5 kB 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+rp1) ...
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
```



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`



```
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

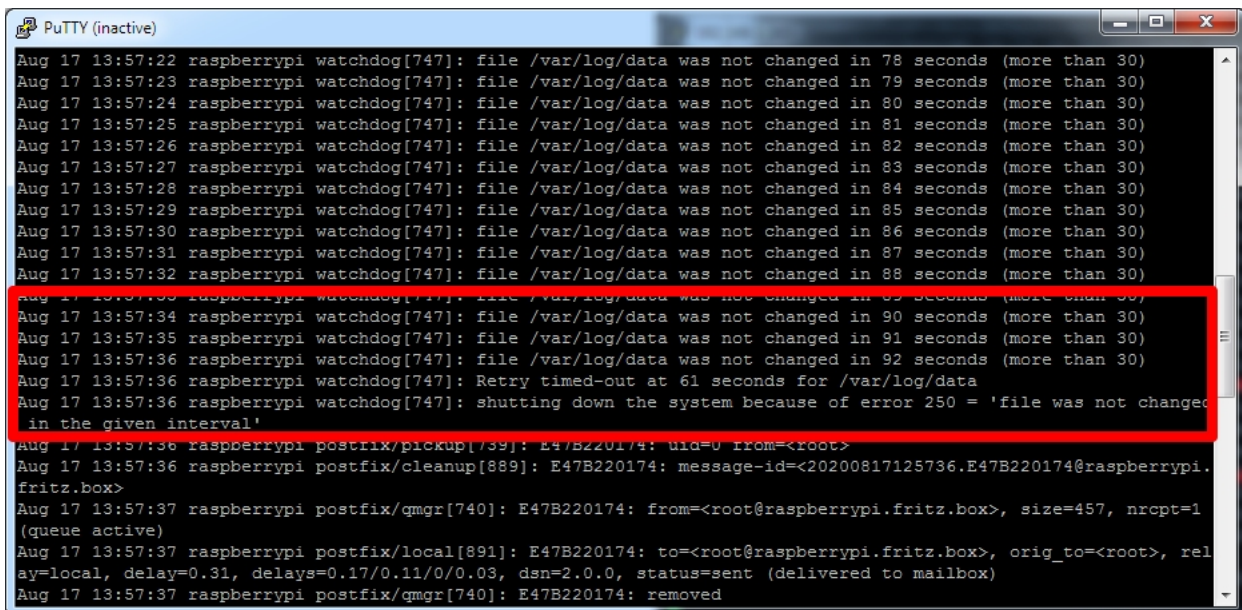
SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set a new password.

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 29 >/sys/class/gpio/export (code=exited, status=0/SUCCESS)
     Process: 750 ExecStartPost=/bin/sh -c echo 1 >/sys/class/gpio/gpio29/active_low (code=exited, status=0/SUCCESS)
     Process: 753 ExecStartPost=/bin/sh -c echo high >/sys/class/gpio/gpio29/direction (code=exited, status=0/SUCCESS)
     Process: 755 ExecStartPost=/bin/sh -c /sys/class/gpio/gpio29/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

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

# GPIO CARD SLOT

The IO Card slot on the board supports a variety of interface cards



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

<https://embeddedpi.com/documentation/mypi-io-card-pcb-template>

Note that 'double height' IO cards require 19mm headers minimum to clear the RJ45 COM port

## J10 Pin Out

Pin	Signal	Pin	Signal
1	GND	2	+5V
3	GND	4	+3.3V
5	GPIO2 (I2C1 SDA)	6	GPIO12 (PWM0)
7	GPIO3 (I2C1 SCL)	8	GPIO13 (PWM1)
9	GPIO4 (GPCLK0/CAM0)	10	GPIO25
11	GPIO6	12	GPIO26
13	GPIO7 (SPI0 CE1)	14	GPIO27
15	GPIO8 (SPI0 CE0)	16	GPIO30 (UART0 CTS)
17	GPIO9 (SPI0 MISO)	18	GPIO31 (UART0 RTS)
19	GPIO10 (SPI0 MOSI)	20	GPIO32 (UART0 TX)
21	GPIO11 (SPI0 SCLK)	22	GPIO33 (UART0 RX)
23	GPIO14 (UART1 TX)	24	GPIO37
25	GPIO15 (UART1 RX)	26	GPIO38
27	GPIO16	28	GPIO41
29	GPIO17	30	GPIO42
31	GPIO18	32	GPIO43
33	GND	34	+3.3V

## J11 Pin Out

Pin	Signal	Pin	Signal
1	-	2	-
3	IO-OUT 1	4	IO-OUT 1
5	IO-OUT 2	6	IO-OUT 2
7	IO-OUT 3	8	IO-OUT 3
9	IO-OUT 4	10	IO-OUT 4
11	IO-OUT 5	12	IO-OUT 5
13	IO-OUT 6	14	IO-OUT 6
15	IO-OUT 7	16	IO-OUT 7
17	IO-OUT 8	18	IO-OUT 8
19	-	20	-

## GPIO

The below website provides information on the different low level peripherals integrated into the GPIO lines

[https://elinux.org/RPi\\_BCM2835\\_GPIOs](https://elinux.org/RPi_BCM2835_GPIOs)

The **raspi-gpio** tool provides information and the ability to directly manipulate GPIO lines (bypassing the OS) for debug purposes.

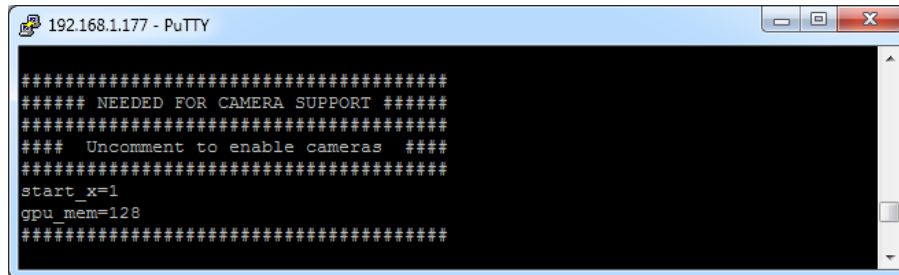


# DUAL CAMERA

---

Dual Camera support has the below pre-requisites

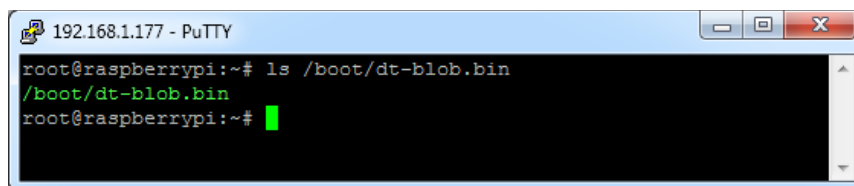
## 1. System config.txt configuration settings



```
#####  
##### NEEDED FOR CAMERA SUPPORT #####  
#####  
### Uncomment to enable cameras ###  
#####  
start_x=1  
gpu_mem=128  
#####
```

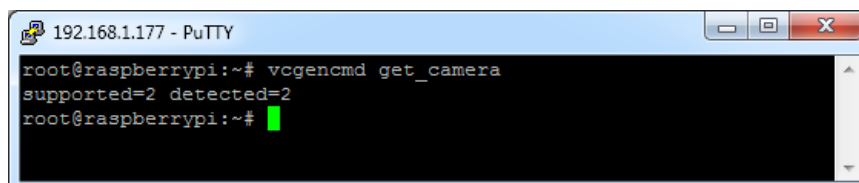
## 2. System dt-blob.bin file configuring the camera setup

This file is located in **/boot** and configures the control lines and interfaces used for camera setup



```
root@raspberrypi:~# ls /boot/dt-blob.bin  
/boot/dt-blob.bin  
root@raspberrypi:~#
```

With this in place the command below should report back accordingly.



```
root@raspberrypi:~# vcgencmd get_camera  
supported=2 detected=2  
root@raspberrypi:~#
```

Note : If 2 cameras are configured (as per default image) but only 1 camera is connected it will always be detected as camera 0 **regardless of which physical port the camera is plugged into.**

See device tree .dts source file in **/root/devicetree** for details on setup

#### J20 – CAM0 Connector

Pin	Signal
1	GND
2	CAM0_DN0
3	CAM0_DP0
4	GND
5	CAM0_DN1
6	CAM0_DP1
7	GND
8	CAM0_CN
9	CAM0_CP
10	GND
11	GPIO22
12	GPIO4
13	I2C0 SCL (GPIO29)
14	I2C0 SDA (GPIO28)
15	3.3V

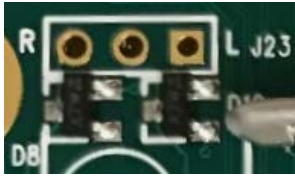
#### J21 – CAM1 Connector

Pin	Signal
1	GND
2	CAM1_DN0
3	CAM1_DP0
4	GND
5	CAM1_DN1
6	CAM1_DP1
7	GND
8	CAM1_CN
9	CAM1_CP
10	GND
11	GPIO21
12	GPIO5
13	I2C0 SCL (GPIO1)
14	I2C0 SDA (GPIO0)
15	3.3V

## AUDIO OUT

---

J23 provides the Audio Out connections



Pin	Signal
1	LEFT
2	GND
3	RIGHT

To enable the PCM audio output add the below lines to `/boot/config.txt` and reboot

```
dtoverlay=audio=on
dtoverlay=pwm-2chan,pin=40,func=4,pin2=45,func2=4
# Optional, improves audio quality
audio_pwm_mode=2
```

Then run the below commands as root user which will run a basic test on the output

```
# amixer sset "PCM" 95%
# speaker-test
```

Note that the PCM output audio requires additional amplification to drive anything other than a basic set of headphones

The below links are useful as a reference:

<https://learn.adafruit.com/adding-basic-audio-output-to-raspberry-pi-zero>

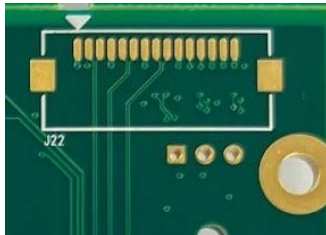
<https://learn.adafruit.com/adding-basic-audio-output-to-raspberry-pi-zero/pi-zero-pwm-audio>

## LCD OUT

The unit can drive the official Raspberry Pi LCD display, to do this the below connector part needs fitting to the solder side J22 connector:

1mm Pitch SMT 15 Way Right Angle Female FPC Connector      Molex 52271-1579

This fits to J22 underneath the Camera connector



The display should be powered from a +5V and 0V lines either from the HAT connector (J8) or from J4, connect these to the +5V and 0V lines on the display board.

**Do not power the LCD board from the front panel side USB connector as this interface isn't usually enabled early enough in the boot cycle.**

The display connects up as standard using the same type of FFC cable as normal, you may find a longer length of cable is helpful to locate the display and board apart from each other.

The only extra software configuration needed is to add the below section to the system device tree file:

```
192.168.1.104 - PuTTY

pin_defines {
    pin_define@HDMI_CONTROL_ATTACHED { type = "internal"; number = <46>; }; // HPD_N on GPIO46

    ////////////////////////////////////////////////////
    // DISPLAY CONFIG -- START
    ////////////////////////////////////////////////////

    pin_define@DISPLAY_SDA {
        type = "internal";
        number = <28>;
    };
    pin_define@DISPLAY_SCL {
        type = "internal";
        number = <29>;
    };
    pin_define@DISPLAY_I2C_PORT {
        type="internal";
        number=<0>;
    };

    ////////////////////////////////////////////////////
    // DISPLAY CONFIG -- END
    ////////////////////////////////////////////////////
}
```

These lines need adding to the system **dt-blob.dts** file

```
pin_define@DISPLAY_SDA {  
    type = "internal";  
    number = <28>;  
};  
pin_define@DISPLAY_SCL {  
    type = "internal";  
    number = <29>;  
};  
pin_define@DISPLAY_I2C_PORT {  
    type="internal";  
    number=<0>;  
};
```

The source device tree file for the board can be found in **/root/devicetree**

Pick the version of the **dt-blob-xxx.dts** file that matches the Compute Module being used.

With the section above in place recompile the device tree file with :

```
dtc -I dts -O dtb -o dt-blob.bin dt-blob-CM3plus.dts && cp dt-blob.bin /boot
```

On next reboot the display will take over from the HDMI output and display the standard rainbow output at boot any system messages.

## 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**

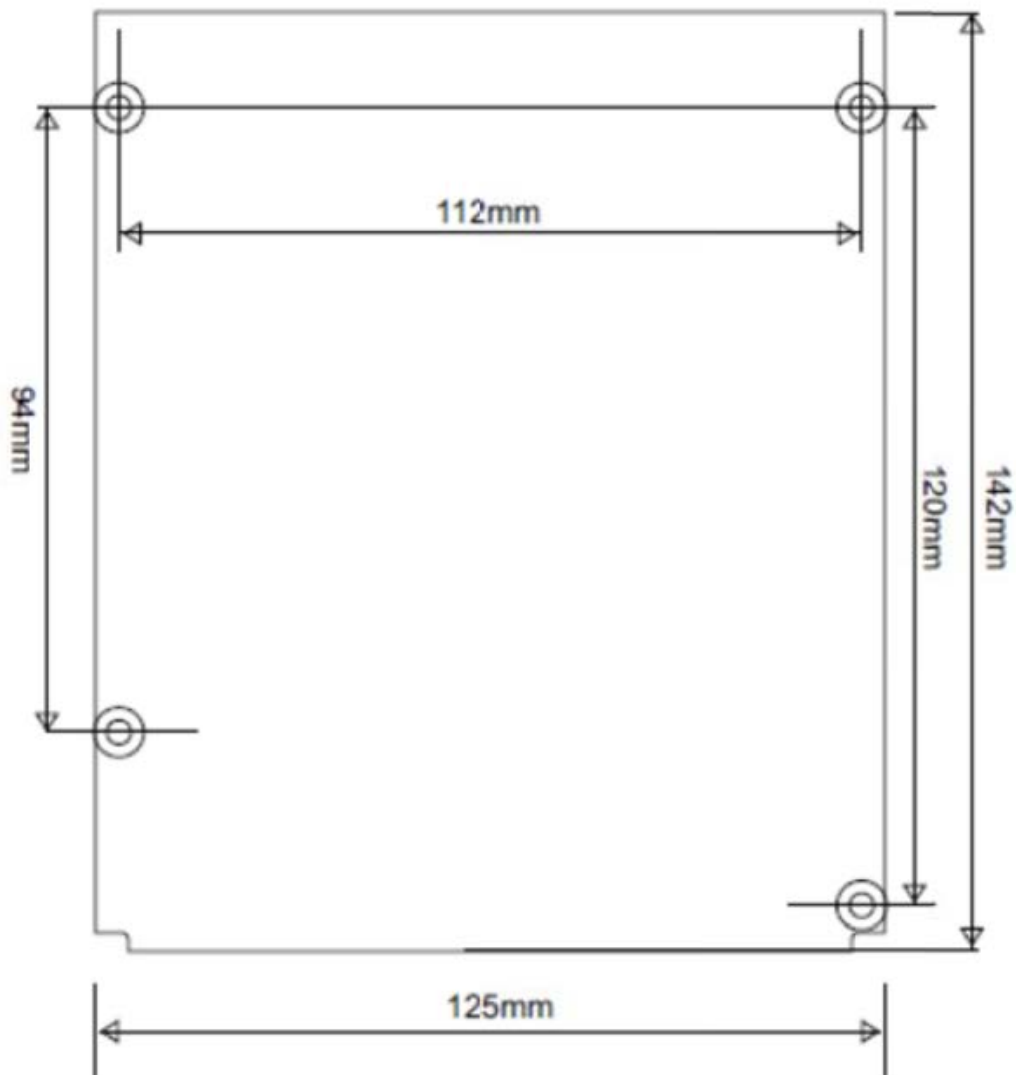
---

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

## DIMENSIONS

---

Below drawing shows the location of the mounting holes, please contact sales for CAD data.





## CHASSIS GROUND

---

With regards to the Integrator Board the following connections share a Chassis Ground net, which is separate from the main 0V line.

- 4 x M3.5 Mounting holes
- LAN RJ45 Shield
- USB Shield
- COM RJ45 Shield

**R64** position on the underside of the PCB provides an easy access point to either connect Chassis Ground directly to the main power supply DC IN 0V via a solder link, or fitting an 0805 size component.



The connection of R64 is dependent on the enclosure design and how the overall chassis ground is dealt with at a system level.

Connecting the chassis ground net to 0V provides termination for the LAN port and also ESD discharge route back to the power connector rather than through the mainboard ground.

## **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