MyPi Industrial Integrator Board NT

User Guide

Issue : 1.00

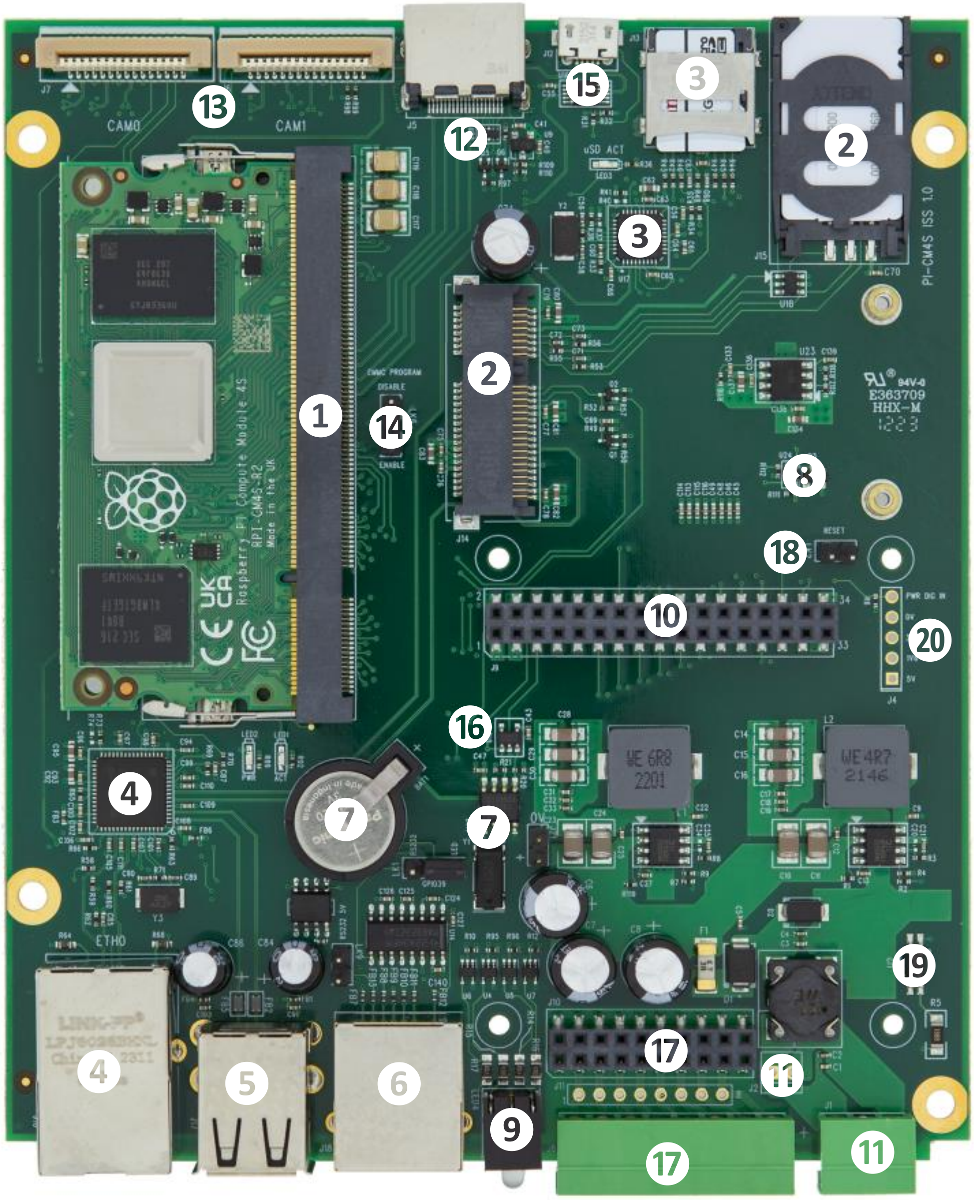
Dated : July 2023

Prepared By : Andrew O'Connell

FEATURES

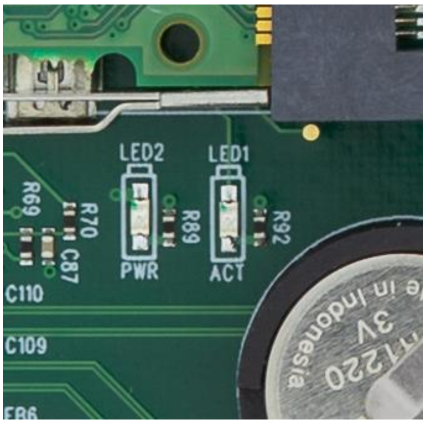
- Supports Raspberry Pi Compute Module 1/3/3+/4S variants
- 1 x 10/100 LAN
- 2 x USB 2.0 (external access)
- 1 x uSD Card Storage (USB Interfaced)
- 1 x mPCIe Interface (USB Interfaced) + SIM
- 1 x Front Faced RS232 Port
- 1 x Battery Backed RTC
- 1 x Board ID EEPROM (Preprogrammed)
- 2 x Camera Interfaces
- 1 x Optional Display Interface
- 1 x HDMI
- 1 x Opto-Isolated Digital Input
- 1 x Modular IO slot with 28 GPIO Pins
    - 2/4 x SPI
    - 1/5 x I2C
    - 2/5 x UART
    - SDIO Interface
    - 3 x GPCLK
    - 2 x PWM Channels
- 1 x 1.6second watchdog
- 2 x Bi-colour Status LEDs
- 9-28V Input
- Wide -20°C to +80°C Ambient operating temperature
- Core PCB Size : 125 x 142mm

BOARD IO FEATURES



1. Compute Module 1/3/3+/4S Socket
2. mPCIe Socket + Modem SIM Socket
3. USB µSD Card Interface + Socket
4. USB LAN9514 10/100 LAN + USB Interface
5. 2 x USB 2.0 Ports
6. RJ45 RS232 Port
7. I^{2}C DS1339U-33+ RTC + Battery Backup
8. External Watchdog
9. Dual Bi-colour LED
10. GPIO IO Card interface
11. Power In (9-28V DC)
12. HDMI Out
13. Dual Camera Interface
14. Programming Mode Selector Link
15. µUSB CM Programming port
16. I^{2}C ID EEPROM
17. Front IO Connector Connections
18. CPU Reset button
19. Digital Input
20. Power + Digital Input

HARDWARE CONFIGURATION LINKS



LED1 - ACT

This LED indicates 'Activity' functionality on the Pi unit, by default this indicates eMMC flash access on the module

LED2 - POWER (3.3V)

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

Fitted DISABLE Compute module programming forced as disabled

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

LK9 - RS232 Connector 5V power out

Removed DTR Line Floating

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

LK1 - LED1 RED or RS232 Out

Fitted 1-2 GPIO30 Conencted to front RS232 CTS
Fitted 2-3 GPIO39 Connected to LED1 RED

RASPBERRY PI COMPUTE MODULE PROGRAMMING

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

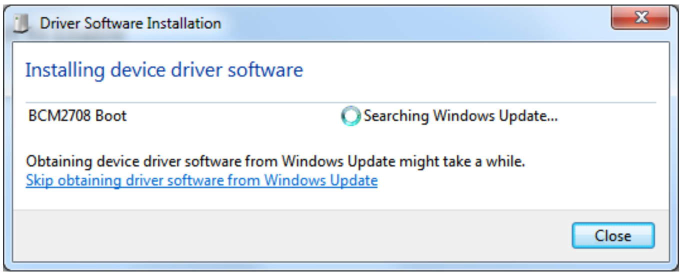
Demo kit units come complete with Compute modules that are pre-programmed with the demo Raspbian OS pre-installed; this section describes how to write a new disk image to the Compute Module.

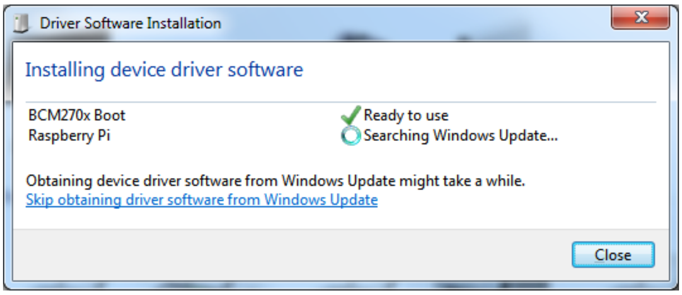
First of all download the windows USB boot installer; this will install the device drivers as well as a program we'll use later called RPi-Boot

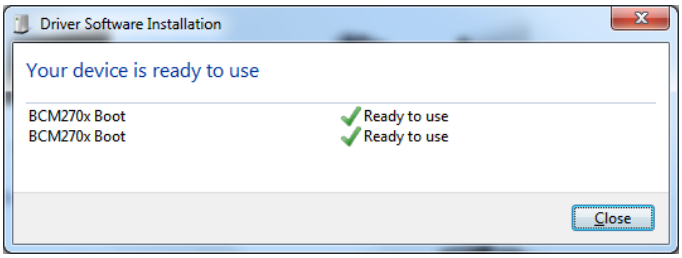
[Raspberry Pi RPI-BOOT Software Download Link]

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:

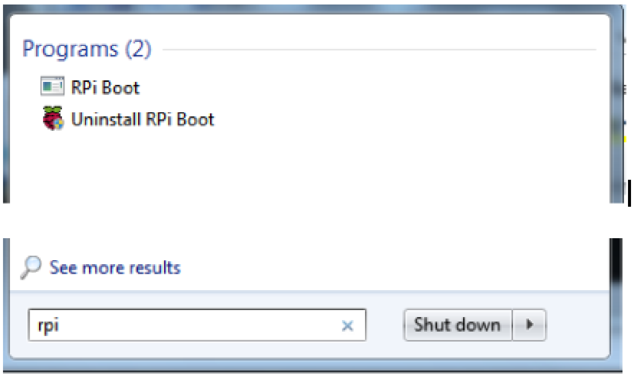




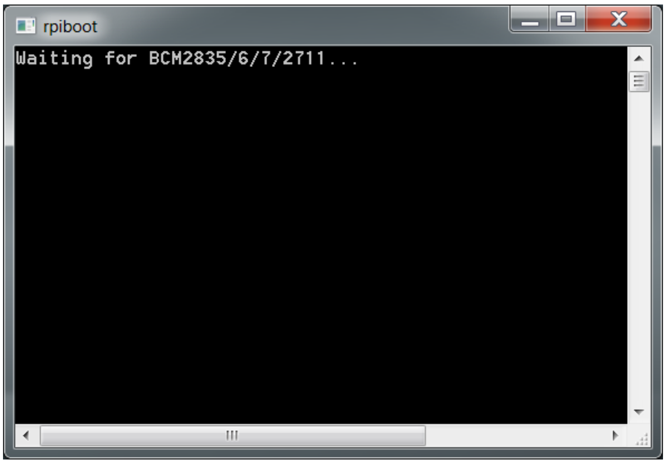


Once that sequence has finished Windows has now installed the required drivers and you can power off the unit for a moment whilst we get the PC side ready for the next step.

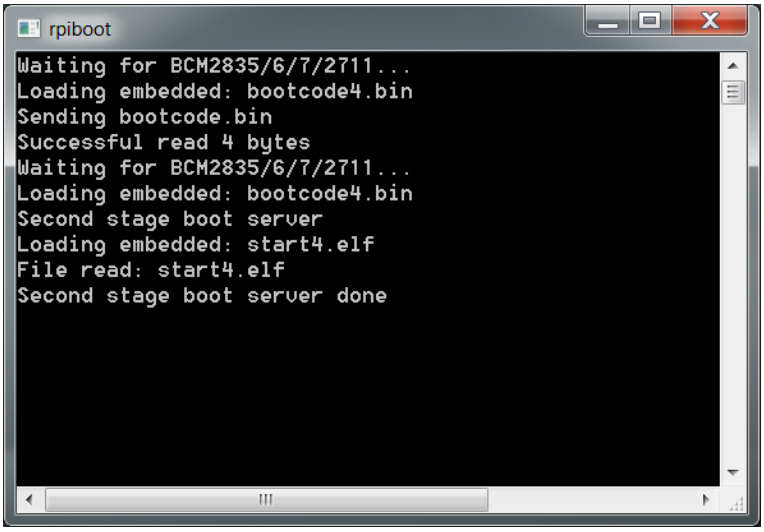
Making sure you have the unit powered off start up RPi Boot, this is easiest done via the start menu, we have found this needs to be run as 'Administrator' privilege mode for correct operation



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



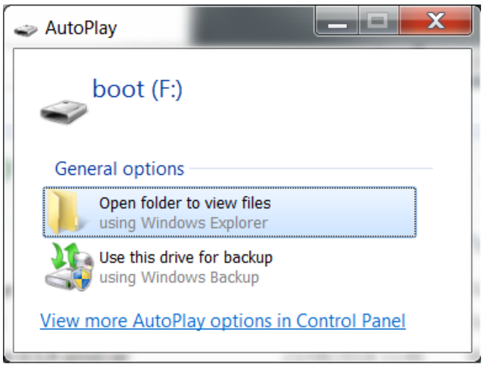
Power up the unit and RPi-Boot will configure the unit to appear as a flash drive:



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

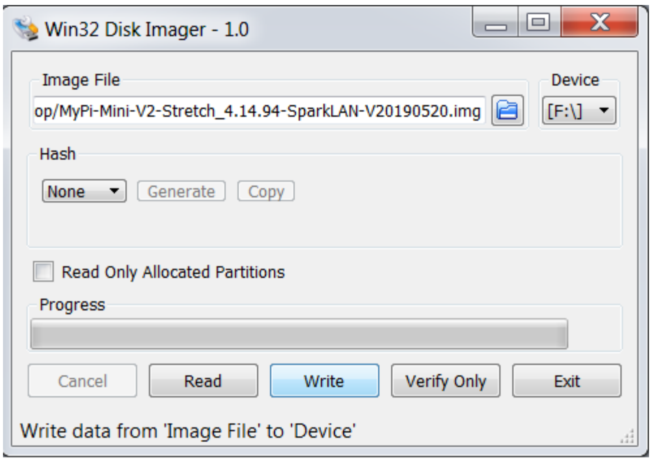
This process writes a disk image, containing the partition table as well as both FAT boot partition and Linux EXT partitions, ***over the entire disk.***

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 USB interfaced LAN/Modem/SD Card Reader 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 MAP

GPIO	Usage	/dev OS Shortcut	Default Pull	Active State
GPIO0	CAM1-SDA			
GPIO1	CAM1-SCL			
GPIO16	MCPIE-AIRPLANE	pcie-wdis	LOW	HIGH
GPIO17	MPCIE-RESET	pcie-reset	LOW	HIGH
GPIO18	WDOG TOGGLE			
GPIO19	WDOG EN		HIGH	LOW
GPIO20	CAM0 POWER/SHUTDOWN			
GPIO21	CAM1 POWER/SHUTDOWN			
GPIO28	CAM0-SDA			
GPIO29	CAM0-SCL			
GPIO34	SD-RESET	sd-disable	HIGH	LOW
GPIO35	LED-RED 2	led-red	HIGH	LOW
GPIO36	FRONT RS232			
GPIO37	FRONT RS232			
GPIO38	FRONT RS232			
GPIO39	FRONT RS232 | LED-RED 1			
GPIO44	LAN-RESET	lan-disable	HIGH	LOW
GPIO45	LED-GREEN 2	led-red	HIGH	LOW
GPIO46	HDMI HPD			
GPIO47	Pi Act LED			

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

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
```

The operation of **GPIO39** is set by LK1, this controls whether the GPIO is sent to the RS232 converter (CTS) or to the bottom Red LED

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/1vUbiLdCWlmordp_iGmQrE-gl3MKcxvDS/view?usp=sharing

CM4S USB INTERFACE

The USB interface on Compute Module 4S the USB port needs to be manually enabled

This is achieved by adding the below directive to **/boot/config.txt**

```
otg_mode=1
```

Without this setting the board will boot without USB Connectivity i.e. Ethernet, SD card interface and Modem will not work.

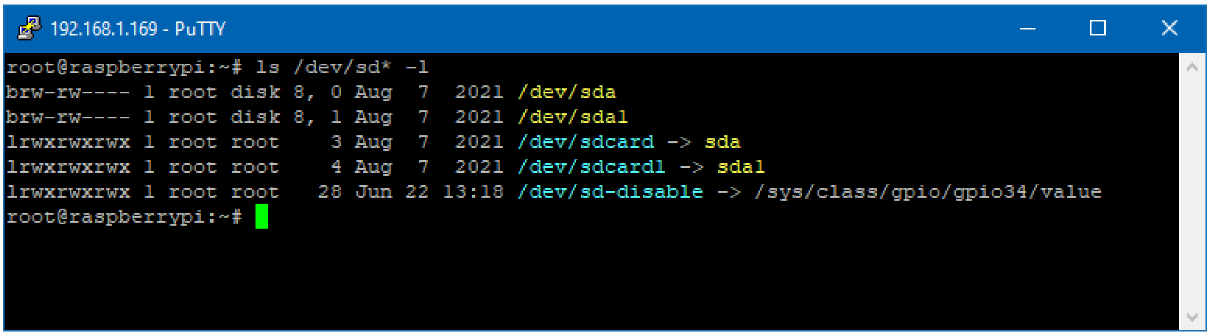
Our produced OS images have this setting enabled so there is no more to do.

For **/boot** firmware files dated >= 22nd August 2022, which were released with OS Kernel version 5.15.60, this has been automatically applied.

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)



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.

The reset line is active low

```
$ 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/usbpwrctl.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
Mechanical Key			
17	-	18	GND
19	-	20	WDIS# (GPIO23)
21	GND	22	PERST# (GPIO39)
23	-	24	3.3V
25	-	26	GND
27	GND	28	-
29	GND	30	-
31	-	32	-
33	-	34	GND
35	GND	36	USB_D+
37	GND	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

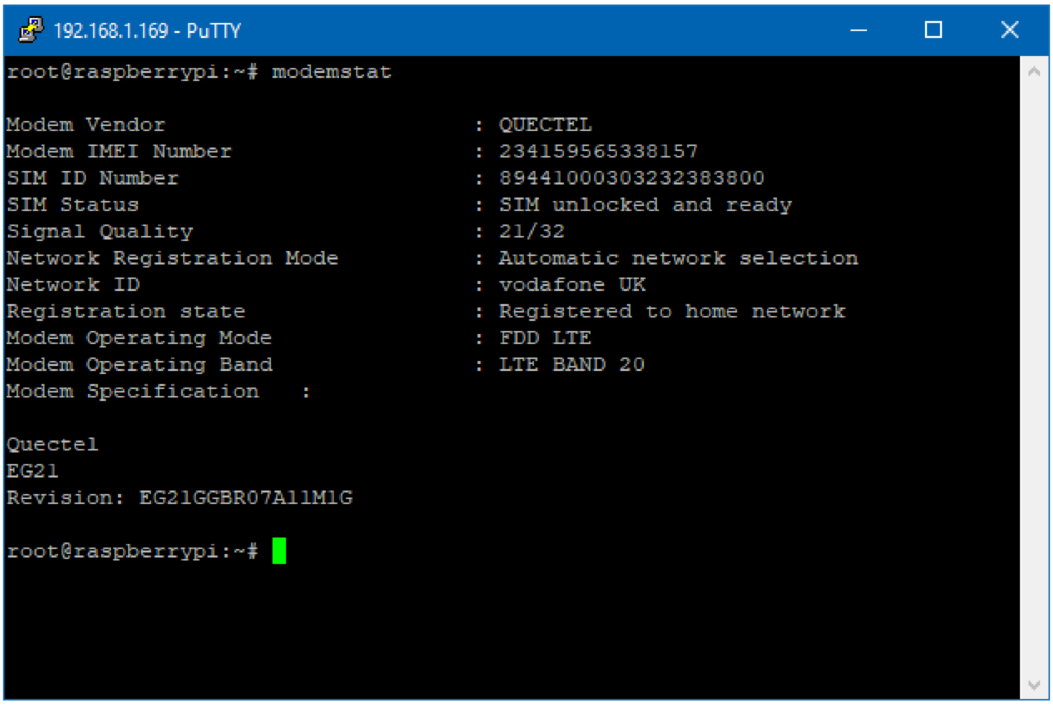
See the below link to pages from the main modem documentation section for details on how to operate modems :

https://www.embeddedpi.com/documentation/3g-4g-modems

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

See web page below for more details

https://www.embeddedpi.com/modemstat



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

192.168.1.169 - 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

192.168.1.169 - PuTTY

```
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#
```

mPCIe IO Cards

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

- LoRa (Microchip RN2483/RN2903) or RAK Lora Concentrators
- Bluetooth 4.0 BLE (Silicon Labs/BlueGiga BLE112)
- Bluetooth 5 (Laird BL652)
- enOcean TCM310
- ZIGBEE/802.15.4 (TI CC2652 ZIGBEE 3.0)
- 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

Whilst the CM4S has multiple UARTs in this section we will focus on the core 2 UARTs that are applicable to all CM versions.

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

UART0 can also be directed to appear as an RS232 port on the front face of the board instead of the GPIO Slot

Further serial ports can be added via either the mPCIe port or plugging additional adapters into the front USB ports.

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

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.

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**
30				**CTS0**		
31				**RTS0**		
32				**TX0**		**TX1**
33				**RX0**		**RX1**
36					**TX0**	
37					**RX0**	
38					**RTS0**	
39					**CTS0**	

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

UART0 Configuration Options

UART0 can be configured in one of four modes, depending on the use case, **only one mode at a time** can be used

```
# MODE 1 : Enable TX/RX on pins 32/33 of IO Card Slot
dtoverlay=uart0,txd0_pin=32,rxd0_pin=33,pin_func=7
```

```
# MODE 2 : Enable CTS/RTS/TX/RX on pins 30/31/32/33 of IO Card Slot
dtoverlay=uart0-full
```

```
# MODE 3 : Assign to Front Face RS232 Port (RX/TX Only)
dtoverlay=uart0,txd0_pin=36,rxd0_pin=37,pin_func=6
```

```
# Mode 4 : Assign to Front Face RS232 Port with full RX/TX/RTS/CTS
dtoverlay=uart0-full-front
```

NOTE : **uart0-full** and **uart0-full-front** are non-standard overlays that enable the additional RTS/CRS lines for the com port. The source files can be found in the **/root/device-tree** folder

UART1 Configuration Options

UART1 can be configured in one of two modes, depending on the use case, **only one mode at a time** can be used

```
# MODE 1 : Enable ttyS0 TX/RX to pins 14/15
dtoverlay=uart1,txd1_pin=14,rxd1_pin=15
```

```
# MODE 2 : Enable ttyS0 TX/RX to pins 32/33
# ** ONLY IF UART0 IS USED IN MODE 3/4 **
dtoverlay=uart1,txd1_pin=32,rxd1_pin=33,pin_func=6
```

Pi UART Port Order Fix

By default the Raspberry Pi Firmware will swap the assignment of the two UART ports to match the usage of the integrated Bluetooth receiver on Pi3+ models

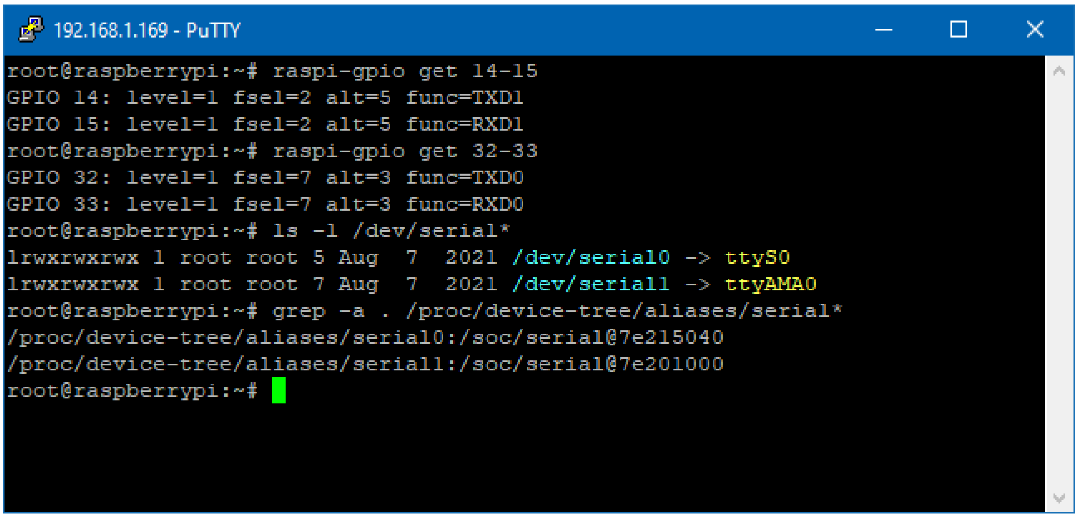
Unless changed can cause problems with unintended interaction between the two ports, especially if the serial console is enabled.

To fix the issue use the overlay below

```
dtoverlay=uart_swap
```

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

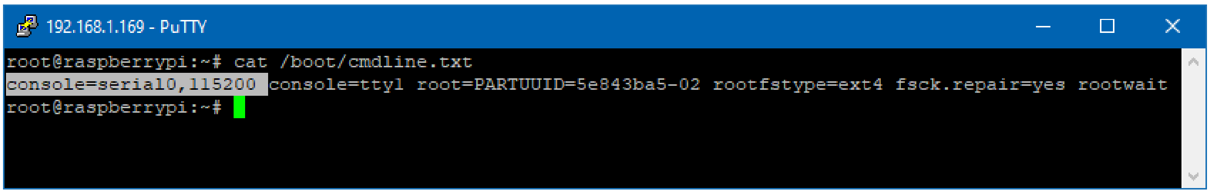
Correct operation can then be confirmed by using the checks below



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

Serial Console

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



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,

Pin	Signal
1	RTS
2	5V / N.C. – LK9 link
3	TX
4	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 topic can be found here :

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

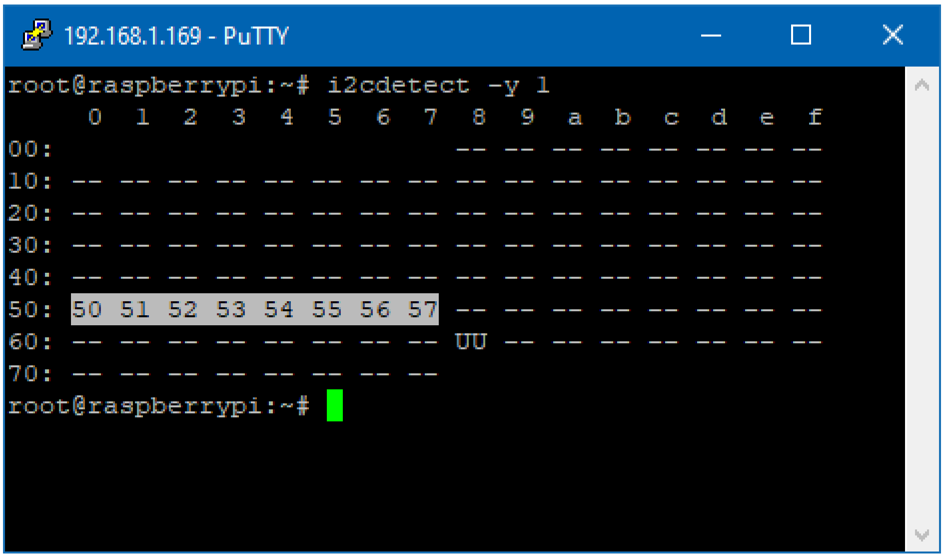
I2C USER EEPROM

A 256Byte EEPROM for user ID storage

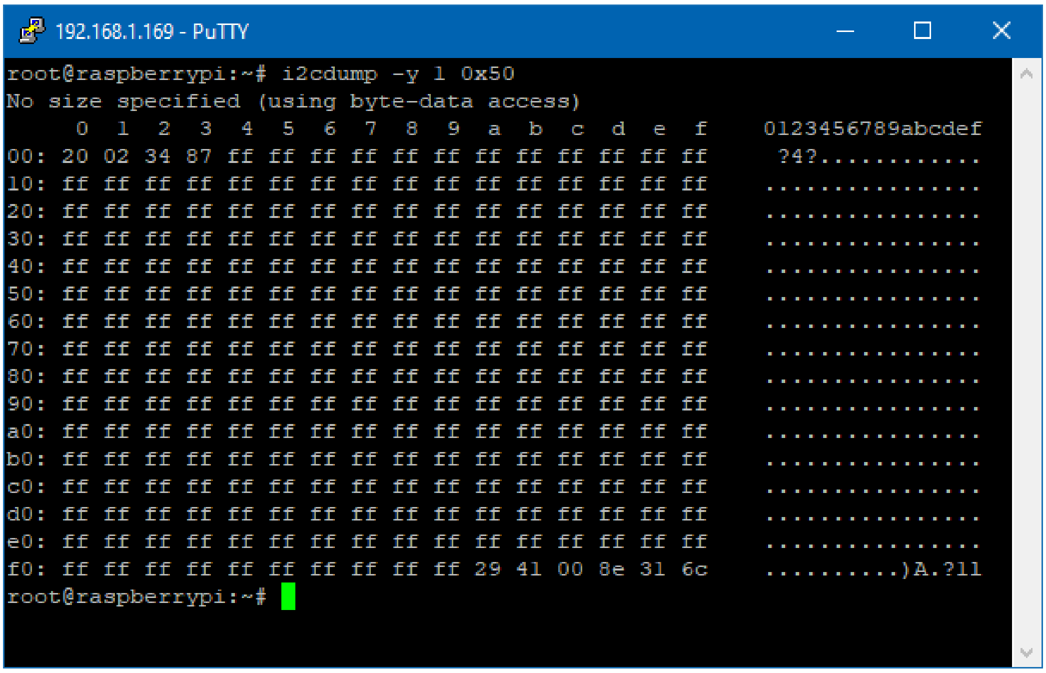
The lower 128Byte has read/write access for user storage, the first 4 hex bytes have been programmed with an ID code visible on the barcoded sticker affixed to the PCB.

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

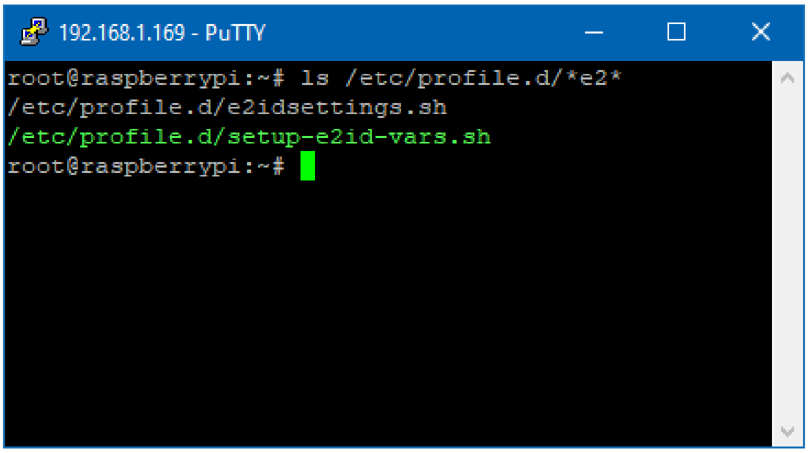


The EEPROM can be accessed for read/write operations using **i2c-tools** utilities, such as **i2cdump**

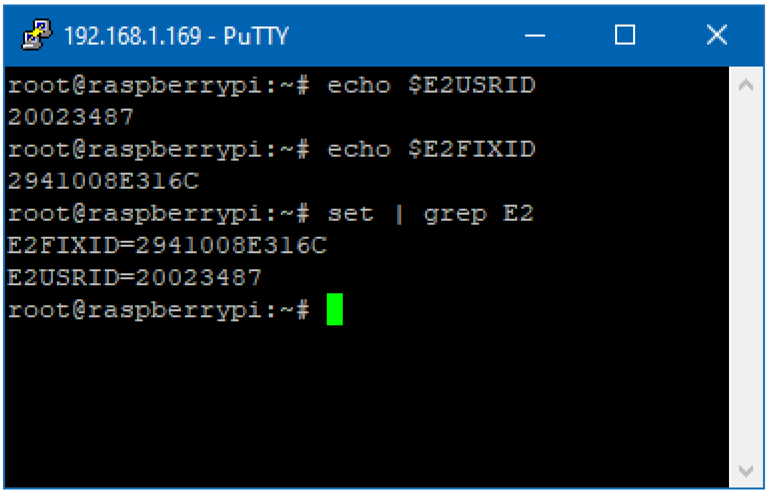


For convenience a script to create two bash environment variables has been created in **/etc/profile.d**

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



These environment variables can be used in scripting by root 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**)

192.168.1.169 - PuTTY

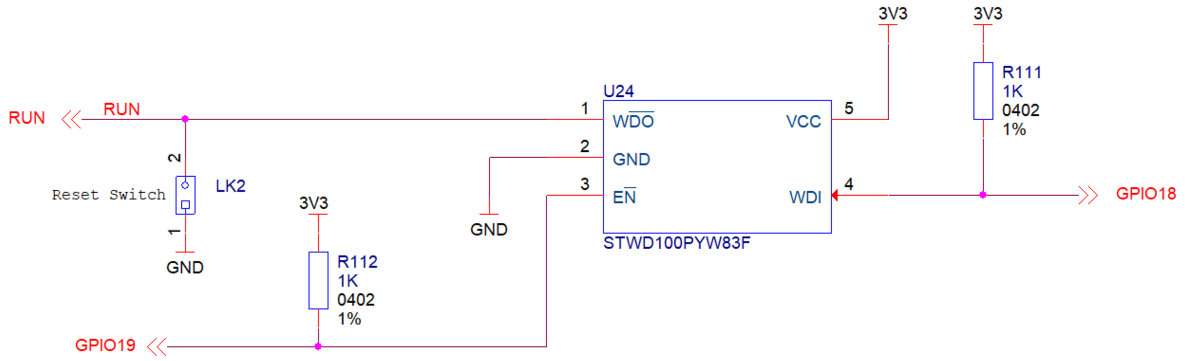
```
root@raspberrypi:~/eeprom# ./eeprog /dev/i2c-1 0x50 -f -x -q -r 0:256

 0000|  20 02 34 87 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 8e 31 6c

root@raspberrypi:~/eeprom#
```

WATCHDOG

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

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

The external watchdog device is driven by GPIO19 (/WD Enable) and GPIO18 (/WD Input), by default the watchdog is disabled as GPIO19 is pulled high by default pin state.



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 (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 GPIO19 (/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]

GPIO line /dev shortcuts for direct control over these lines can be enabled via the script in

/root/ext-watchdog/wd-setup.sh

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

192.168.1.169 - PuTTY

```
root@raspberrypi:~/ext-watchdog# cp ext-watchdog.dtbo /boot/overlays/
root@raspberrypi:~/ext-watchdog# apt-get install watchdog
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  watchdog
0 upgraded, 1 newly installed, 0 to remove and 151 not upgraded.
Need to get 71.7 kB of archives.
After this operation, 205 kB of additional disk space will be used.
Get:1 http://archive.raspberrypi.org/debian bullseye/main armhf watchdog armhf 5.16-2
~rpt2 [71.7 kB]
Fetched 71.7 kB in 0s (661 kB/s)
Preconfiguring packages ...
Selecting previously unselected package watchdog.
(Reading database ... 46476 files and directories currently installed.)
Preparing to unpack .../watchdog_5.16-2~rpt2_armhf.deb ...
Unpacking watchdog (5.16-2~rpt2) ...
Setting up watchdog (5.16-2~rpt2) ...
update-rc.d: warning: start and stop actions are no longer supported; falling back to
 defaults
Created symlink /etc/systemd/system/default.target.wants/watchdog.service → /lib/syst
emd/system/watchdog.service.
wd_keepalive.service is a disabled or a static unit, not starting it.
Processing triggers for man-db (2.9.4-2) ...
root@raspberrypi:~/ext-watchdog#
```

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

```
dtoverlay=ext-watchdog
```

Copy the new configuration files to the target folders

192.168.1.169 - PuTTY

```
Preparing to unpack .../watchdog_5.16-2~rpt2_armhf.deb ...
Unpacking watchdog (5.16-2~rpt2) ...
Setting up watchdog (5.16-2~rpt2) ...
update-rc.d: warning: start and stop actions are no longer supported; falling back to
 defaults
Created symlink /etc/systemd/system/default.target.wants/watchdog.service → /lib/syst
emd/system/watchdog.service.
wd_keepalive.service is a disabled or a static unit, not starting it.
Processing triggers for man-db (2.9.4-2) ...
root@raspberrypi:~/ext-watchdog# vim /boot/config.txt
root@raspberrypi:~/ext-watchdog# cp watchdog.conf /etc/
cp: overwrite '/etc/watchdog.conf'? y
root@raspberrypi:~/ext-watchdog# cp watchdog.service /lib/systemd/system/
cp: overwrite '/lib/systemd/system/watchdog.service'? y
root@raspberrypi:~/ext-watchdog#
```

The watchdog service file has been altered as shown below to start the watchdog process, then enable the watchdog and during boot

192.168.1.169 - PuTTY

```
Description=watchdog daemon
Conflicts=wd_keepalive.service
After=multi-user.target
OnFailure=wd_keepalive.service

[Service]
Type=forking
EnvironmentFile=/etc/default/watchdog
ExecStartPre=/bin/sh -c '[ -z "${watchdog_module}" ] || [ "${watchdog_module}" = "none" ] || /sbin/modpro
be $watchdog_module'
ExecStartPre=/bin/sh -c 'touch /var/log/data'
ExecStart=/bin/sh -c '[ $run_watchdog != 1 ] || exec /usr/sbin/watchdog $watchdog_options'
ExecStartPost=-/bin/sh -c 'echo 19 >/sys/class/gpio/export'
ExecStartPost=-/bin/sh -c 'echo 1 >/sys/class/gpio/gpio19/active_low'
ExecStartPost=-/bin/sh -c 'echo high >/sys/class/gpio/gpio19/direction'
ExecStartPost=-/bin/ln -s /sys/class/gpio/gpio19/value /dev/wdog-enable
ExecStartPost=/bin/sh -c 'echo 1 >/dev/wdog-enable'
ExecStopPost=/bin/sh -c '[ $run_wd_keepalive != 1 ] || false'

[Install]
WantedBy=default.target

root@raspberrypi:~/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**

192.168.1.169 - PuTTY

```
root@raspberrypi:~/ext-watchdog# cat /etc/watchdog.conf
#ping                   = 172.31.14.1
#ping                   = 172.26.1.255
#interface              = eth0
file                    = /var/log/data
change                  = 30

watchdog-device         = /dev/watchdog1

# 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


# 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:~/ext-watchdog#
```

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.

192.168.1.169 - 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 Thu 2023-06-22 13:52:05 BST; 7s ago
    Process: 1455 ExecStartPre=/bin/sh -c [ -z "${watchdog_module}" ] || [ "${watchdog_module}" = "none">
    Process: 1456 ExecStartPre=/bin/sh -c touch /var/log/data (code=exited, status=0/SUCCESS)
    Process: 1458 ExecStart=/bin/sh -c [ $run_watchdog != 1 ] || exec /usr/sbin/watchdog $watchdog_optio>
    Process: 1462 ExecStartPost=/bin/sh -c echo 19 >/sys/class/gpio/export (code=exited, status=0/SUCCES>
    Process: 1464 ExecStartPost=/bin/sh -c echo 1 >/sys/class/gpio/gpio19/active_low (code=exited, statu>
    Process: 1467 ExecStartPost=/bin/sh -c echo high >/sys/class/gpio/gpio19/direction (code=exited, sta>
    Process: 1469 ExecStartPost=/bin/ln -s /sys/class/gpio/gpio19/value /dev/wdog-enable (code=exited, s>
    Process: 1472 ExecStartPost=/bin/sh -c echo 1 >/dev/wdog-enable (code=exited, status=0/SUCCESS)
   Main PID: 1460 (watchdog)
      Tasks: 1 (limit: 1631)
        CPU: 77ms
     CGroup: /system.slice/watchdog.service
             └─1460 /usr/sbin/watchdog

Jun 22 13:52:05 raspberrypi watchdog[1460]:  interface: no interface to check
Jun 22 13:52:05 raspberrypi watchdog[1460]:  temperature: no sensors to check
Jun 22 13:52:05 raspberrypi watchdog[1460]:  no test binary files
Jun 22 13:52:05 raspberrypi watchdog[1460]:  no repair binary files
Jun 22 13:52:05 raspberrypi watchdog[1460]:  error retry time-out = 60 seconds
Jun 22 13:52:05 raspberrypi watchdog[1460]:  repair attempts = 1
Jun 22 13:52:05 raspberrypi watchdog[1460]:  alive=/dev/watchdog1 heartbeat=[none] to=root no_act=no for>
Jun 22 13:52:05 raspberrypi watchdog[1460]: watchdog now set to 60 seconds
Jun 22 13:52:05 raspberrypi watchdog[1460]: hardware watchdog identity: GPIO Watchdog
Jun 22 13:52:05 raspberrypi systemd[1]: Started watchdog daemon.

root@raspberrypi:~# ls /dev/watchdog* -l
crw------- 1 root root  10, 130 Aug  7  2021 /dev/watchdog
crw------- 1 root root 250,   0 Aug  7  2021 /dev/watchdog0
crw------- 1 root root 250,   1 Aug  7  2021 /dev/watchdog1
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

192.168.1.169 - PuTTY

```
root@raspberrypi:~# tail /var/log/syslog
Jun 22 13:54:31 raspberrypi systemd[1]: Startup finished in 5.319s (kernel) + 24.290s (userspace) = 29.610s.
Jun 22 13:54:36 raspberrypi kernel: [   34.446764] cam-dummy-reg: disabling
Jun 22 13:54:42 raspberrypi systemd[1]: systemd-fsckd.service: Succeeded.
Jun 22 13:55:02 raspberrypi watchdog[1517]: file /var/log/data was not changed in 31 seconds (more than 30)
Jun 22 13:55:03 raspberrypi watchdog[1518]: file /var/log/data was not changed in 32 seconds (more than 30)
Jun 22 13:55:04 raspberrypi watchdog[1520]: file /var/log/data was not changed in 33 seconds (more than 30)
Jun 22 13:55:05 raspberrypi watchdog[1521]: file /var/log/data was not changed in 34 seconds (more than 30)
Jun 22 13:55:06 raspberrypi watchdog[1524]: file /var/log/data was not changed in 35 seconds (more than 30)
Jun 22 13:55:07 raspberrypi watchdog[1526]: file /var/log/data was not changed in 36 seconds (more than 30)
Jun 22 13:55:08 raspberrypi watchdog[1527]: file /var/log/data was not changed in 37 seconds (more than 30)
root@raspberrypi:~# touch /var/log/data
root@raspberrypi:~# tail /var/log/syslog
Jun 22 13:55:11 raspberrypi watchdog[1531]: file /var/log/data was not changed in 40 seconds (more than 30)
Jun 22 13:55:12 raspberrypi watchdog[1532]: file /var/log/data was not changed in 41 seconds (more than 30)
Jun 22 13:55:13 raspberrypi watchdog[1535]: file /var/log/data was not changed in 42 seconds (more than 30)
Jun 22 13:55:14 raspberrypi watchdog[1536]: file /var/log/data was not changed in 43 seconds (more than 30)
Jun 22 13:55:15 raspberrypi watchdog[1539]: file /var/log/data was not changed in 44 seconds (more than 30)
Jun 22 13:55:16 raspberrypi watchdog[1542]: file /var/log/data was not changed in 45 seconds (more than 30)
Jun 22 13:55:17 raspberrypi watchdog[1545]: file /var/log/data was not changed in 46 seconds (more than 30)
Jun 22 13:55:18 raspberrypi watchdog[1548]: file /var/log/data was not changed in 47 seconds (more than 30)
Jun 22 13:55:19 raspberrypi watchdog[1549]: file /var/log/data was not changed in 48 seconds (more than 30)
Jun 22 13:55:20 raspberrypi watchdog[1550]: file /var/log/data was not changed in 49 seconds (more than 30)
root@raspberrypi:~# date
Thu 22 Jun 13:55:33 BST 2023
root@raspberrypi:~# tail /var/log/syslog
Jun 22 13:55:11 raspberrypi watchdog[1531]: file /var/log/data was not changed in 40 seconds (more than 30)
Jun 22 13:55:12 raspberrypi watchdog[1532]: file /var/log/data was not changed in 41 seconds (more than 30)
Jun 22 13:55:13 raspberrypi watchdog[1535]: file /var/log/data was not changed in 42 seconds (more than 30)
Jun 22 13:55:14 raspberrypi watchdog[1536]: file /var/log/data was not changed in 43 seconds (more than 30)
Jun 22 13:55:15 raspberrypi watchdog[1539]: file /var/log/data was not changed in 44 seconds (more than 30)
Jun 22 13:55:16 raspberrypi watchdog[1542]: file /var/log/data was not changed in 45 seconds (more than 30)
Jun 22 13:55:17 raspberrypi watchdog[1545]: file /var/log/data was not changed in 46 seconds (more than 30)
Jun 22 13:55:18 raspberrypi watchdog[1548]: file /var/log/data was not changed in 47 seconds (more than 30)
Jun 22 13:55:19 raspberrypi watchdog[1549]: file /var/log/data was not changed in 48 seconds (more than 30)
Jun 22 13:55:20 raspberrypi watchdog[1550]: file /var/log/data was not changed in 49 seconds (more than 30)
root@raspberrypi:~#
```

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

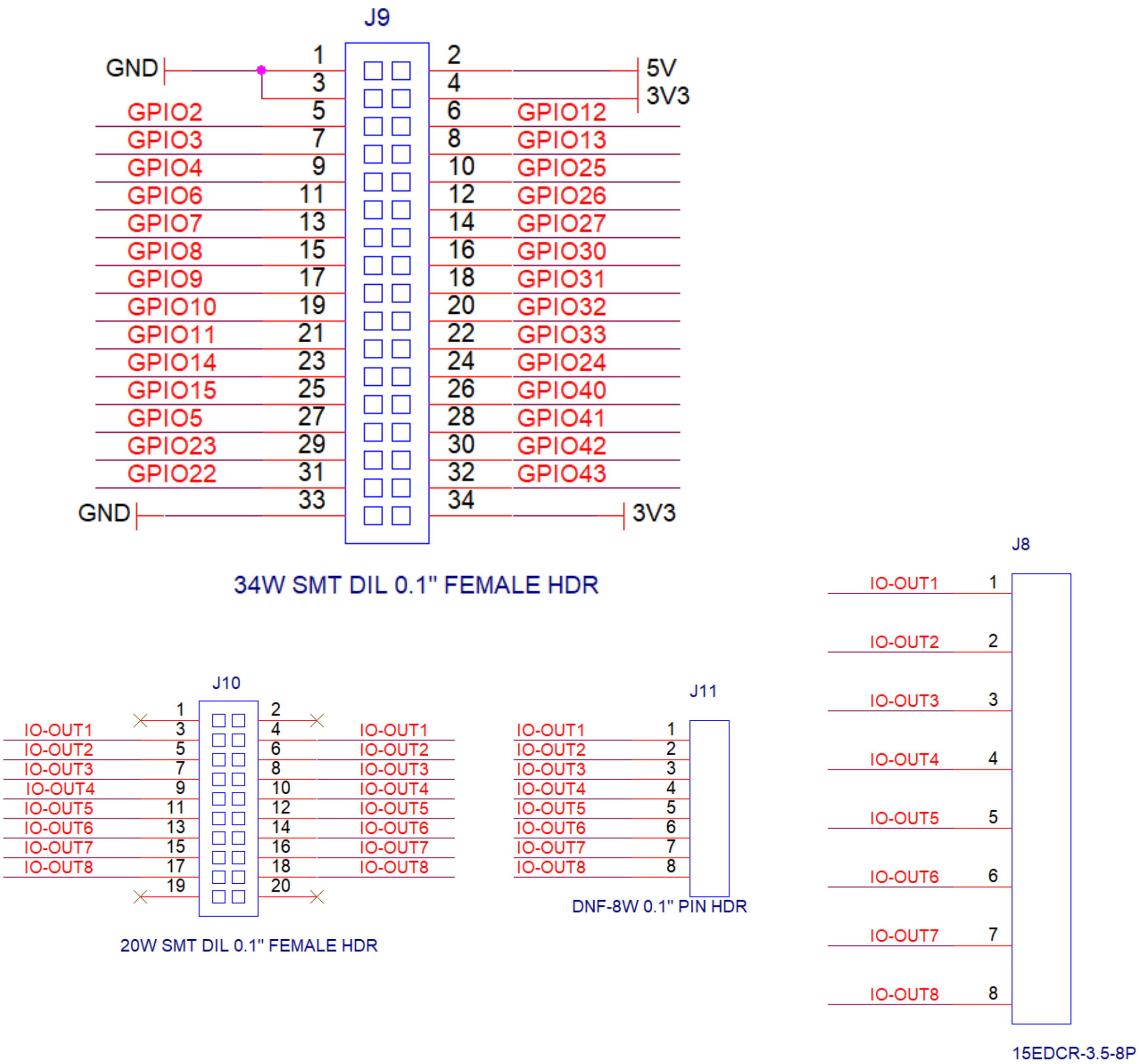
```
# echo c > /proc/sysrq-trigger
```

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

J9 Pin Out

Pin	Signal	Pin	Signal
1	GND	**2**	+5V
3	GND	**4**	+3.3V
5	GPIO2	**6**	GPIO12
7	GPIO3	**8**	GPIO13
9	GPIO4	**10**	GPIO25
11	GPIO6	**12**	GPIO26
13	GPIO7	**14**	GPIO27
15	GPIO8	**16**	GPIO30
17	GPIO9	**18**	GPIO31
19	GPIO10	**20**	GPIO32
21	GPIO11	**22**	GPIO33
23	GPIO14	**24**	**GPIO24**
25	GPIO15	**26**	**GPIO40**
27	**GPIO5**	**28**	GPIO41
29	**GPIO23**	**30**	GPIO42
31	**GPIO22**	**32**	GPIO43
33	GND	**34**	+3.3V

These GPIO Lines Differ From Older CM3 Integrator Board Version

J10 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

Compute Module 1-3+ = https://elinux.org/RPi_BCM2835_GPIOs

Compute Module 4S+ = https://elinux.org/RPi_BCM2711_GPIOs

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

IO Connector Pinout Peripherals - CM3/CM3+

Pin	Signal	ALT0	ALT3	ALT4	ALT5	Pin	Signal	ALT0	ALT3	ALT4	ALT5
1	GND					2	+5V				
3	GND					4	+3.3V				
5	GPIO2	SDA1				6	GPIO12	PWM0_0			
7	GPIO3	SCL1				8	GPIO13	PWM0_1			
9	GPIO4	GPCLK0				10	GPIO25	SD0_DAT1	SD1_DAT1		
11	GPIO6	GPCLK2				12	GPIO26	SD0_DAT2	SD1_DAT2		
13	GPIO7	SPI0-CE1				14	GPIO27	SD0_DAT3	SD1_DAT3		
15	GPIO8	SPI0-CE0				16	GPIO30		CTS-0		CTS-1
17	GPIO9	SPI0-MISO				18	GPIO31		RTS-0		RTS-1
19	GPIO10	SPI0-MOSI				20	GPIO32		TXD-0		TXD-1
21	GPIO11	SPI0-SCLK				22	GPIO33		RXD-0		RXD-1
23	GPIO14	TXD-0			TXD-1	24	GPIO24	SD0_DAT0	SD1_DAT0		
25	GPIO15	RXD-0			RXD-1	26	GPIO40	PWM1_0	SPI2-MISO	TXD-1	
27	GPIO5	GPCLK1				28	GPIO41	PWM1_1	SPI2-MOSI	RXD-1	
29	GPIO23	SD0_CMD	SD1_CMD			30	GPIO42	GPCLK1	SPI2-SCLK	RTS-1	
31	GPIO22	SD0_CLK	SD1_CLK			32	GPIO43	GPCLK2	SPI2-CE0	CTS-1	
33	GND					34	+3.3V				

Changed From Older CM3 MyPi Integrator Board Version

IO Connector Pinout Peripherals - CM4S

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	GPIO12	PWM0_0	SPI5-CE0	TXD-5	SDA5
7	GPIO3	SCL1			SCL3	8	GPIO13	PWM0_1	SPI5-MISO	RXD-5	SCL5
9	GPIO4	GPCLK0	SPI4-CE0	TXD-3	SDA3	10	GPIO25	SD0_DAT1	SD1_DAT1		SPI4-CE1
11	GPIO6	GPCLK2	SPI4-MOSI	CTS-3	SDA4	12	GPIO26	SD0_DAT2	SD1_DAT2		SPI5-CE1
13	GPIO7	SPI0-CE1	SPI4-SCLK	RTS-3	SCL4	14	GPIO27	SD0_DAT3	SD1_DAT3		
15	GPIO8	SPI0-CE0	I2CSL CE_N	TXD-4	SDA4	16	GPIO30		CTS-0		CTS-1
17	GPIO9	SPI0-MISO	I2CSL SDI	RXD-4	SCL4	18	GPIO31		RTS-0		RTS-1
19	GPIO10	SPI0-MOSI	I2CSL SDA	CTS-4	SDA5	20	GPIO32		TXD-0		TXD-1
21	GPIO11	SPI0-SCLK	I2CSL SCL	RTS-4	SCL5	22	GPIO33		RXD-0		RXD-1
23	GPIO14	TXD-0	SPI5-MOSI	CTS-5	TXD-1	24	GPIO24	SD0_DAT0	SD1_DAT0		
25	GPIO15	RXD-0	SPI5-SCLK	RTS-5	RXD-1	26	GPIO40	PWM1_0	SPI2-MISO	TXD-1	
27	GPIO5	GPCLK1	SPI5-MISO	RXD-3	SCL3	28	GPIO41	PWM1_1	SPI2-MOSI	RXD-1	
29	GPIO23	SD0_CMD	SD1_CMD		SCL6	30	GPIO42	GPCLK1	SPI2-SCLK	RTS-1	
31	GPIO22	SD0_CLK	SD1_CLK		SDA6	32	GPIO43	GPCLK2	SPI2-CE0	CTS-1	
33	GND					34	+3.3V				

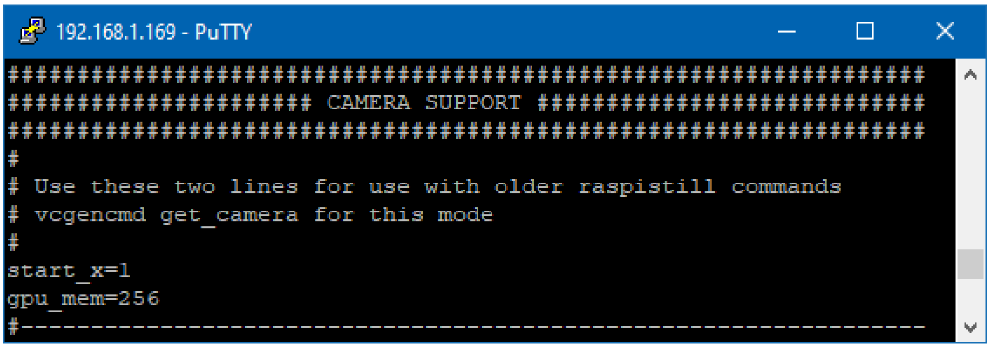
Changed From Older CM3 Integrator Board Version

DUAL CAMERA

Dual Camera support has the below pre-requisites

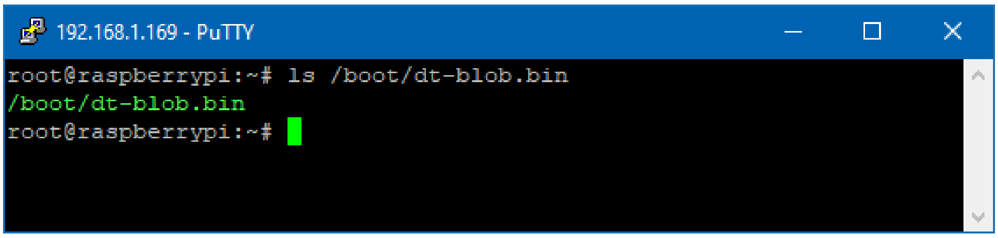
1. **System config.txt configuration settings**

Enable these two lines as shown below

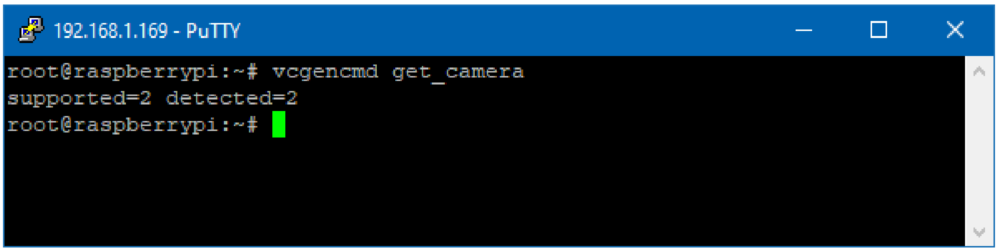


2. **System dt-blob.bin file configuring the camera setup**

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



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



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

J7 – 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	GPIO20 (Power Control)
12	NC
13	I2C0 SCL (GPIO29)
14	I2C0 SDA (GPIO28)
15	3.3V

J6 – CAM1 Connector

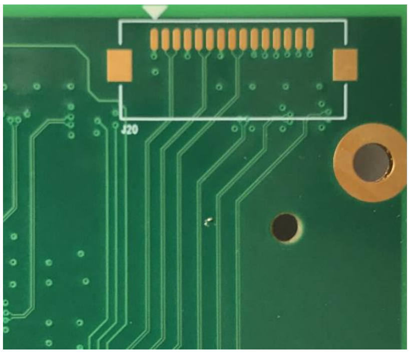
Pin	Signal
1	GND
2	CAM1_DN0
3	CAM1_DP0
4	GND
5	CAM1_DN1
6	CAM1_DP1
7	GND
8	CAM1_CN
9	CAM1_CP
10	GND
11	GPIO21 (Power Control)
12	NC
13	I2C0 SCL (GPIO1)
14	I2C0 SDA (GPIO0)
15	3.3V

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 J20 underneath the Camera connector



The display should be powered from +5V and 0V lines 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 the below section in the system device tree file:

These lines need adding/enabling in the system **dt-blob-dual-cameras.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 (**dt-blob-dual-cameras.dts**) can be found in **/root/devicetree**

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

```
dtc -I dts -O dtb -o dt-blob.bin dt-blob-dual-cameras.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.

POWER DRAW NOTES

Power consumption was measured in different modes to give guidance

Configuration #1

- No Cameras
- HDMI Connected
- No Modem
- PSU 12V
- LAN Connected

Readings

Mode	Power Draw
Steady State	182mA
tvservice -o	175mA
echo 1 >/dev/sd-disable	142mA
echo 1 >/dev/lan-disable	75mA

Configuration #1

- No Cameras
- HDMI Connected
- Modem Connected (Quectel EC21V) With SIM but idle
- PSU 12V
- LAN Connected

Readings

Mode	Power Draw
Steady State	200mA
tvservice -o	190mA
echo 1 >/dev/sd-disable	158mA
echo 1 >/dev/pcie-reset	158mA
echo 1 >/dev/lan-disable	96mA

LAN Connection should be brought down before LAN Chip is disconnected to avoid OS issues.

Note that the mPCIe Slot (Modem) and the SD card controller are both connected via the USB interface on the LAN chip.

When LAN chip is active removing LAN cable causes a drop of approximately 20mA

MIGRATION FROM OLDER CM3 INTEGRATOR BOARD

Migration from our older CM1/3/3+ Integrator board comprises of a small amount of GPIO lines that have changed function. This has been done to improve functionality and enhance compatibility with the CM4S feature set.

GPIO Configuration Changes

Function	Shortcut	OLD GPIO	NEW GPIO
Status LED (GREEN)	/dev/led-green	36	45
mPCIe Wireless Disable/Airplane	/dev/pcie-wdis	39	16
mPCIe Reset	/dev/pcie-reset	23	17
CAM0 Power Enable		22	20
CAM1 Power Enable		21	21
Hardware Watchdog Enable		28	19
Hardware Watchdog Input		29	18
IO SLOT-24		37	24
IO SLOT-26		38	40
IO SLOT-27		16	5
IO SLOT-29		17	23
IO SLOT-31		18	22
Front RS232 Port TX	ttyS1 (for ttyUSBx)	USB-TX (ttyUSBx)	GPIO36 (ttyAMA0)
Front RS232 Port RX	ttyS1 (for ttyUSBx)	USB-RX (ttyUSBx)	GPIO37 (ttyAMA0)
Front RS232 Port RTS	ttyS1 (for ttyUSBx)	USB-RTS (ttyUSBx)	GPIO38 (ttyAMA0)
Front RS232 Port CTS	ttyS1 (for ttyUSBx)	USB-CTS (ttyUSBx)	GPIO39 (ttyAMA0)

Functionality Changed/Removed

Function	Change Type	Notes
Audio Out	Removed	GPIO40/45 reused for board IO
Camera LED Indicator	Removed	GPIO4/5 reused for board IO
Raspberry Pi HAT Connector	Removed	Various GPIO reused for board IO
Power Connector DIO	Changed	Changed from GPIO20 to optional connection to GPIO43

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

A white paper on how to transition from Compute Module 3 to Compute Module 4S has been written by the Raspberry Pi Team and can be downloaded from the link below

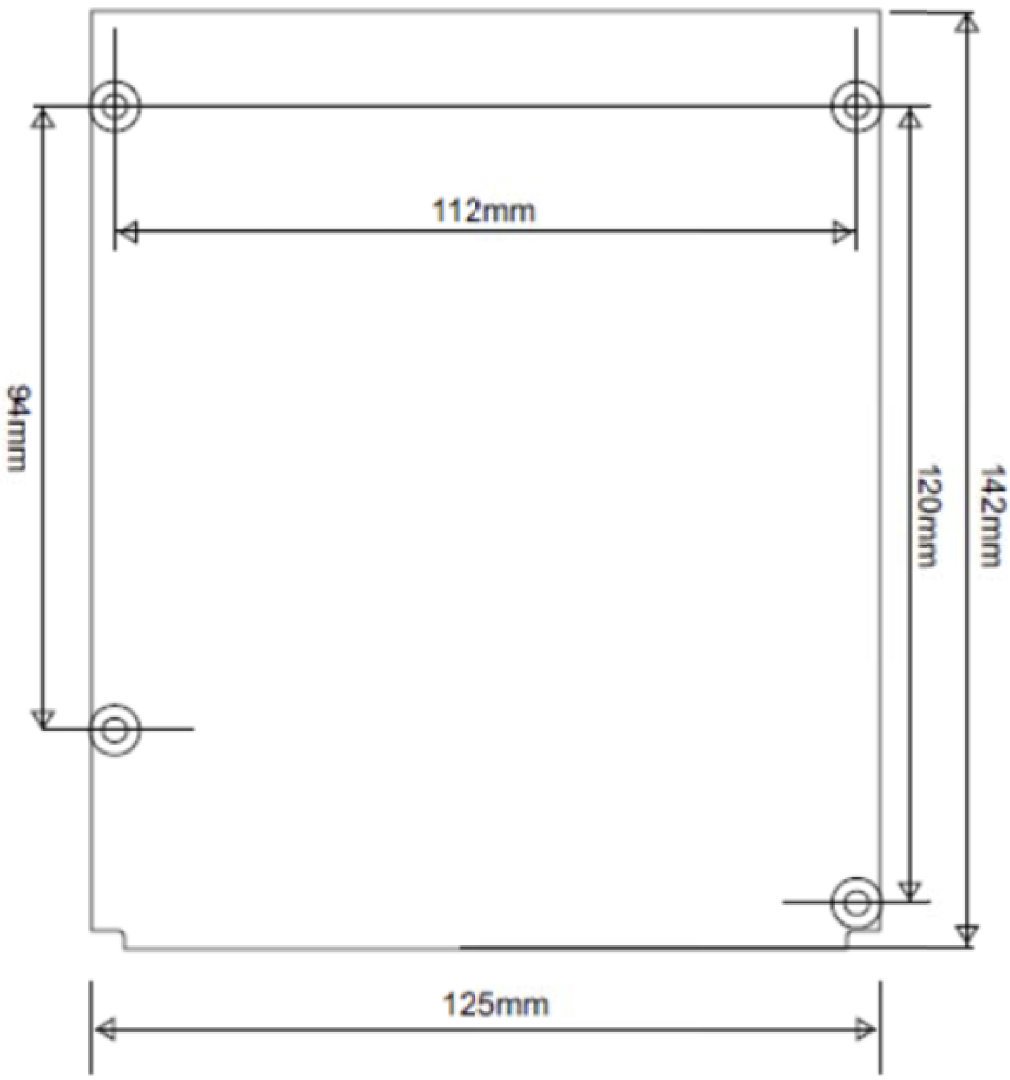
https://pip.raspberrypi.com/categories/685-whitepapers-app-notes/documents/RP-003478-WP/Transitioning-from-CM-3-to-CM-4S.pdf

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.

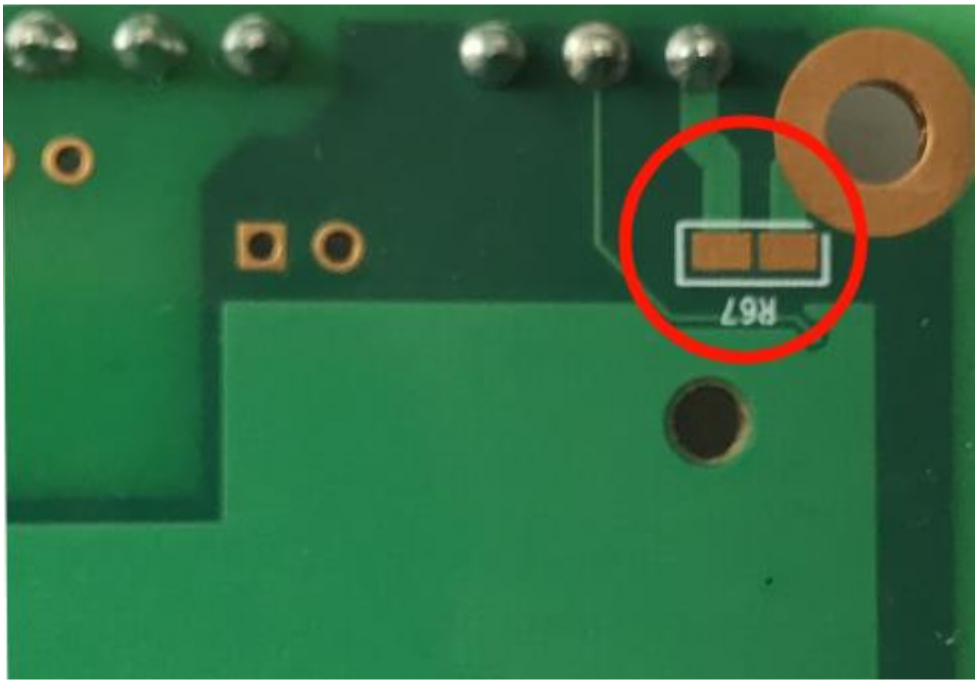


CHASSIS GROUND

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

R67 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 R67 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, so 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