CONFIG.TXT

As it's an embedded platform, the Raspberry Pi doesn't have a BIOS like you'd find on a conventional PC. The various system configuration parameters, which would traditionally be edited and stored using a BIOS, are stored in an optional text file named `config.txt`. This is read by the GPU before the ARM CPU and Linux are initialised, therefore it must be located on the first (boot) partition of your SD card, alongside `bootcode.bin` and `start.elf`. This file is normally accessible as `/boot/config.txt` from Linux and must be edited as root, but from Windows or OS X it's seen as a file in the only accessible part of the card. If you need to apply some of the config settings below, but you don't have a `config.txt` on your boot partition yet, then simply create it as a new text file.

Any changes will only take effect after you've rebooted your Raspberry Pi. After Linux has booted you can get the current active settings with the following commands:

`vcgencmd get_config <config>` - displays a specific config value, e.g. `vcgencmd get_config arm_freq`.

`vcgencmd get_config int` - lists all the integer config options that are set (non-zero).

`vcgencmd get_config str` - lists all the string config options that are set (non-null).

Note that there's a small number of config settings that can't be retrieved using `vcgencmd`.

FILE FORMAT

As `config.txt` is read by the early-stage boot firmware it has a very simple file format. The format is a single `property=value` statement on each line, where `value` is either an integer or a string. Comments may be added, or existing config values may be commented out and disabled, by starting a line with the `#` character.

Here is an example file:

```
# Force the monitor to HDMI mode so that sound will be sent over
HDMI cable
hdmi_drive=2
# Set monitor mode to DMT
hdmi_group=2
# Set monitor resolution to 1024x768 XGA 60Hz (HDMI_DMT_XGA_60)
hdmi_mode=16
```

```
# Make display smaller to stop text spilling off the screen
overscan_left=20
overscan_right=12
overscan_top=10
overscan_bottom=10
```

MEMORY

GPU_MEM

GPU memory in megabytes. Sets the memory split between the CPU and GPU; the CPU gets the remaining memory. Minimum value is `16`; maximum value is `192`, `448` or `944`, depending on whether you're using a 256M, 512MB or 1024MB Pi. The default value is `64`.

Setting `gpu_mem` to low values may automatically disable certain firmware features, as there are some things the GPU simply can't do with too little memory. So if a certain feature you're trying to use isn't working, try setting a larger GPU memory split.

Using `gpu_mem_256`, `gpu_mem_512` and `gpu_mem_1024` allows you to swap the same SD card between 256MB, 512MB and 1024MB Pis without having to edit `config.txt` each time:

GPU_MEM_256

GPU memory in megabytes for the 256MB Raspberry Pi (ignored if memory size is not 256MB). This overrides `gpu_mem`. The maximum value is `192` and the default is not set.

GPU_MEM_512

GPU memory in megabytes for the 512MB Raspberry Pi (ignored if memory size is not 512MB). This overrides `gpu_mem`. The maximum value is `448` and the default is not set.

GPU_MEM_1024

GPU memory in megabytes for the 1024MB Raspberry Pi 2 (ignored if memory size is not 1024MB). This overrides `gpu_mem`. The maximum value is `944` and the default is not set.

DISABLE_L2CACHE

Setting this to `1` disables the CPU's access to the GPU's L2 cache; requires a corresponding L2 disabled kernel. Default value is `0`.

DISABLE_PVT

Setting this to `1` disables adjusting the refresh rate of RAM every 500ms; this action measures the RAM's temperature. Default value is `0`.

CMA - DYNAMIC MEMORY SPLIT

The firmware and kernel as of 19th November 2012 supports CMA (Contiguous Memory Allocator), which means the memory split between CPU and GPU is managed dynamically at runtime. However, this is not officially supported.

You can find an example config.txt here.

CMA_LWM

When the GPU has less than `cma_lwm` (low-watermark) megabytes of memory available, it will request some from the CPU.

CMA_HWM

When the GPU has more than `cma_hwm` (high-watermark) megabytes of memory available, it will release some to the CPU.

The following options need to be in `cmdline.txt` for CMA to work:

```
coherent_pool=6M smsc95xx.turbo_mode=N
```

CAMERA

DISABLE_CAMERA_LED

Setting this to `1` prevents the red camera LED from turning on when recording video or taking a still picture. Useful for preventing reflections when the camera is facing a window.

ONBOARD ANALOGUE AUDIO (3.5MM JACK)

The onboard audio output has a few config options that alter the behaviour of how the analogue audio is driven and whether some firmware features are enabled or not.

DISABLE_AUDIO_DITHER

By default, a 1.0LSB dither is applied to the audio stream if it's routed to the analogue audio output. This can create audible background "hiss" in some situations, such as if the ALSA volume is set to a low level. Set this to `1` to disable dither application.

PWM_SAMPLE_BITS

Adjust the bit depth of the analogue audio output. The default bit depth is `11`. Selecting bit depths below `8` will result in nonfunctional audio - settings below `8` result in a PLL frequency too low to support. Generally only useful as a demonstration of how bit depth affects quantisation noise.

VIDEO

COMPOSITE VIDEO MODE OPTIONS

SDTV_MODE

Defines the TV standard used for composite video output over the yellow RCA jack; the default value is `0`.

sdtv_mode	result
0	Normal NTSC
1	Japanese version of NTSC – no pedestal
2	Normal PAL
3	Brazilian version of PAL – 525/60 rather than 625/50, different subcarrier

SDTV_ASPECT

This defines the aspect ratio for composite video output. The default value is `1`.

sdtv_aspect	result
1	4:3
2	14:9
3	16:9

SDTV_DISABLE_COLOURBURST

Setting this to `1` disables colour burst on composite video output. The picture will be displayed in monochrome, but it may possibly be sharper.

HDMI MODE OPTIONS

HDMI_SAFE

Setting this to `1` uses "safe mode" settings to try to boot with maximum HDMI compatibility. This is the same as setting the following parameters:

```
hdmi_force_hotplug=1
hdmi_ignore_edid=0xa5000080
config_hdmi_boost=4
hdmi_group=2
hdmi_mode=4
disable_overscan=0
overscan_left=24
overscan_right=24
```

```
overscan_top=24
overscan_bottom=24
```

HDMI_IGNORE_EDID

Setting this to `0xa5000080` enables the ignoring of EDID/display data if your display doesn't have an accurate EDID. It requires this unusual value to ensure that it doesn't get triggered accidentally.

HDMI_EDID_FILE

Setting this to `1` will cause the GPU to read EDID data from the `edid.dat` file, located in the boot partition, instead of reading it from the monitor. More information is available here.

HDMI_FORCE_EDID_AUDIO

Setting this to `1` pretends that all audio formats are supported by the display, allowing passthrough of DTS/AC3 even when not reported as supported.

HDMI_IGNORE_EDID_AUDIO

Setting this to `1` pretends that all audio formats are unsupported by the display. This means ALSA will default to the analogue audio (headphone) jack.

HDMI_FORCE_EDID_3D

Setting this to `1` pretends that all CEA modes support 3D, even when the EDID doesn't indicate support for them.

AVOID_EDID_FUZZY_MATCH

Setting this to `1` avoids "fuzzy matching" of modes described in the EDID. Instead, it will pick the standard mode with the matching resolution and closest framerate, even if blanking is wrong.

HDMI_IGNORE_CEC_INIT

Setting this to `1` will prevent the initial active source message being sent during bootup. This avoids bringing a CEC-enabled TV out of standby and channel switching when rebooting your Raspberry Pi.

HDMI_IGNORE_CEC

Setting this to `1` pretends that CEC is not supported at all by the TV. No CEC functions will be supported.

HDMI_PIXEL_ENCODING

Force the pixel encoding mode. By default, it will use the mode requested from the EDID, so it shouldn't need changing.

hdmi_pixel_encoding	result
0	default (RGB limited for CEA, RGB full for DMT)
1	RGB limited (16-235)
2	RGB full (0-255)
3	YCbCr limited (16-235)
4	YCbCr full (0-255)

HDMI_DRIVE

This allows you to choose between HDMI and DVI output modes.

hdmi_drive	result
1	Normal DVI mode (No sound)
2	Normal HDMI mode (Sound will be sent if supported and enabled)

CONFIG_HDMI_BOOST

Configures the signal strength of the HDMI interface; the default value is `0` and the maximum is `7`. Try `4` if you have interference issues with HDMI.

HDMI_GROUP

This defines the HDMI output group to be either CEA (Consumer Electronics Association, the standard typically used by TVs) or DMT (Display Monitor Timings, the standard typically used by monitors). This setting should be used in conjunction with `hdmi_mode`.

hdmi_group	result
0	Auto-detect from EDID
1	CEA
2	DMT

HDMI_MODE

This, together with `hdmi_group`, defines the HDMI output format.

For setting a custom display mode not listed here, see this thread.

These values are valid if `hdmi_group=1` (CEA):

hdmi_mode	resolution	frequency	notes
1	VGA (640x480)		
2	480p	60Hz	
3	480p	60Hz	16:9 aspect ratio
4	720p	60Hz	
5	1080i	60Hz	
6	480i	60Hz	
7	480i	60Hz	16:9 aspect ratio
8	240p	60Hz	
9	240p	60Hz	16:9 aspect ratio
10	480i	60Hz	pixel quadrupling
11	480i	60Hz	pixel quadrupling, 16:9 aspect ratio
12	240p	60Hz	pixel quadrupling
			pixel quadrupling, 16:9 aspect

13	240p	60Hz	ratio
14	480p	60Hz	pixel doubling
15	480p	60Hz	pixel doubling, 16:9 aspect ratio
16	1080p	60Hz	
17	576p	50Hz	
18	576p	50Hz	16:9 aspect ratio
19	720p	50Hz	
20	1080i	50Hz	
21	576i	50Hz	
22	576i	50Hz	16:9 aspect ratio
23	288p	50Hz	
24	288p	50Hz	16:9 aspect ratio
25	576i	50Hz	pixel quadrupling
26	576i	50Hz	pixel quadrupling, 16:9 aspect ratio
27	288p	50Hz	pixel quadrupling
28	288p	50Hz	pixel quadrupling, 16:9 aspect ratio
29	576p	50Hz	pixel doubling
30	576p	50Hz	pixel doubling, 16:9 aspect ratio
31	1080p	50Hz	

32	1080p	24Hz	
33	1080p	25Hz	
34	1080p	30Hz	
35	480p	60Hz	pixel quadrupling
36	480p	60Hz	pixel quadrupling, 16:9 aspect ratio
37	576p	50Hz	pixel quadrupling
38	576p	50Hz	pixel quadrupling, 16:9 aspect ratio
39	1080i	50Hz	reduced blanking
40	1080i	100Hz	
41	720p	100Hz	
42	576p	100Hz	
43	576p	100Hz	16:9 aspect ratio
44	576i	100Hz	
45	576i	100Hz	16:9 aspect ratio
46	1080i	120Hz	
47	720p	120Hz	
48	480p	120Hz	
49	480p	120Hz	16:9 aspect ratio
50	480i	120Hz	
51	480i	120Hz	16:9 aspect ratio

52	576p	200Hz	
53	576p	200Hz	16:9 aspect ratio
54	576i	200Hz	
55	576i	200Hz	16:9 aspect ratio
56	480p	240Hz	
57	480p	240Hz	16:9 aspect ratio
58	480i	240Hz	
59	480i	240Hz	16:9 aspect ratio

In the table above, the modes with a 16:9 aspect ratio are a widescreen variant of a mode which usually has 4:3 aspect ratio. Pixel doubling and quadrupling indicates a higher clock rate, with each pixel repeated two or four times respectively.

These values are valid if `hdmi_group=2` (DMT):

hdmi_mode	resolution	frequency	notes
1	640x350	85Hz	
2	640x400	85Hz	
3	720x400	85Hz	
4	640x480	60Hz	
5	640x480	72Hz	
6	640x480	75Hz	
7	640x480	85Hz	

8	800x600	56Hz	
9	800x600	60Hz	
10	800x600	72Hz	
11	800x600	75Hz	
12	800x600	85Hz	
13	800x600	120Hz	
14	848x480	60Hz	
15	1024x768	43Hz	incompatible with the Raspberry Pi
16	1024x768	60Hz	
17	1024x768	70Hz	
18	1024x768	75Hz	
19	1024x768	85Hz	
20	1024x768	120Hz	
21	1152x864	75Hz	
22	1280x768		reduced blanking
23	1280x768	60Hz	
24	1280x768	75Hz	
25	1280x768	85Hz	
26	1280x768	120Hz	reduced blanking
27	1280x800		reduced blanking
28	1280x800	60Hz	
29	1280x800	75Hz	

30	1280x800	85Hz	
31	1280x800	120Hz	reduced blanking
32	1280x960	60Hz	
33	1280x960	85Hz	
34	1280x960	120Hz	reduced blanking
35	1280x1024	60Hz	
36	1280x1024	75Hz	
37	1280x1024	85Hz	
38	1280x1024	120Hz	reduced blanking
39	1360x768	60Hz	
40	1360x768	120Hz	reduced blanking
41	1400x1050		reduced blanking
42	1400x1050	60Hz	
43	1400x1050	75Hz	
44	1400x1050	85Hz	
45	1400x1050	120Hz	reduced blanking
46	1440x900		reduced blanking
47	1440x900	60Hz	
48	1440x900	75Hz	
49	1440x900	85Hz	
50	1440x900	120Hz	reduced blanking
51	1600x1200	60Hz	

52	1600x1200	65Hz	
53	1600x1200	70Hz	
54	1600x1200	75Hz	
55	1600x1200	85Hz	
56	1600x1200	120Hz	reduced blanking
57	1680x1050		reduced blanking
58	1680x1050	60Hz	
59	1680x1050	75Hz	
60	1680x1050	85Hz	
61	1680x1050	120Hz	reduced blanking
62	1792x1344	60Hz	
63	1792x1344	75Hz	
64	1792x1344	120Hz	reduced blanking
65	1856x1392	60Hz	
66	1856x1392	75Hz	
67	1856x1392	120Hz	reduced blanking
68	1920x1200		reduced blanking
69	1920x1200	60Hz	
70	1920x1200	75Hz	
71	1920x1200	85Hz	
72	1920x1200	120Hz	reduced blanking

73	1920x1440	60Hz	
74	1920x1440	75Hz	
75	1920x1440	120Hz	reduced blanking
76	2560x1600		reduced blanking
77	2560x1600	60Hz	
78	2560x1600	75Hz	
79	2560x1600	85Hz	
80	2560x1600	120Hz	reduced blanking
81	1366x768	60Hz	
82	1920x1080	60Hz	1080p
83	1600x900		reduced blanking
84	2048x1152		reduced blanking
85	1280x720	60Hz	720p
86	1366x768		reduced blanking

Note that there is a pixel clock limit, which means the highest supported mode is 1920x1200 at 60Hz with reduced blanking.

WHICH VALUES ARE VALID FOR MY MONITOR?

Your HDMI monitor may support only a limited set of formats. To find out which formats are supported, use the following method:

1. Set the output format to VGA 60Hz (`hdmi_group=1` and `hdmi_mode=1`)

and boot up your Raspberry Pi

2. Enter the following command to give a list of CEA supported modes:
   `/opt/vc/bin/tvservice -m CEA`
3. Enter the following command to give a list of DMT supported modes:
   `/opt/vc/bin/tvservice -m DMT`
4. Enter the following command to show your current state:
   `/opt/vc/bin/tvservice -s`
5. Enter the following commands to dump more detailed information from your monitor:
   `/opt/vc/bin/tvservice -d edid.dat; /opt/vc/bin/edidparser edid.dat`

The `edid.dat` should also be provided when troubleshooting problems with the default HDMI mode.

CUSTOM MODE

If your monitor requires a mode that is not in one of the tables above, then it's possible to define a custom CVT mode for it instead:

```
hdmi_cvt=<width> <height> <framerate> <aspect> <margins>
<interlace> <rb>
```

Value	Default	Description
width	(required)	width in pixels
height	(required)	height in pixels
framerate	(required)	framerate in Hz

aspect	3	aspect ratio 1=4:3, 2=14:9, 3=16:9, 4=5:4, 5=16:10, 6=15:9
margins	0	0=margins disabled, 1=margins enabled
interlace	0	0=progressive, 1=interlaced
rb	0	0=normal, 1=reduced blanking

Fields at the end can be omitted to use the default values.

Note that this simply *creates* the mode (group 2 mode 87); in order to make the Pi use this by default, you must add some additional settings. As an example, the following selects an 800x480 resolution and enables audio drive:

```
hdmi_cvt=800 480 60 6
hdmi_group=2
hdmi_mode=87
hdmi_drive=2
```

This may not work if your monitor does not support standard CVT timings.

GENERIC DISPLAY OPTIONS

HDMI_FORCE_HOTPLUG

Setting this to `1` pretends that the HDMI hotplug signal is asserted, so it appears that a HDMI display is attached. In other words, HDMI output mode will be used, even if no HDMI monitor is detected.

HDMI_IGNORE_HOTPLUG

Setting this to `1` pretends that the HDMI hotplug signal is not asserted, so it appears that a HDMI display is not attached. In other words, composite output mode will be used, even if an HDMI monitor is detected.

DISABLE_OVERSCAN

Set to `1` to disable overscan.

OVERSCAN_LEFT

Specifies the number of pixels to skip on the left edge of the screen. Increase this value if the text flows off the left edge of the screen; decrease it if there's a black border between the left edge of the screen and the text.

OVERSCAN_RIGHT

Specifies the number of pixels to skip on the right edge of the screen.

OVERSCAN_TOP

Specifies the number of pixels to skip on the top edge of the screen.

OVERSCAN_BOTTOM

Specifies the number of pixels to skip on the bottom edge of the screen.

FRAMEBUFFER_WIDTH

Specifies the console framebuffer width in pixels. The default is the display width minus the total horizontal overscan.

FRAMEBUFFER_HEIGHT

Specifies the console framebuffer height in pixels. The default is the display height minus the total vertical overscan.

FRAMEBUFFER_DEPTH

Specifies the console framebuffer depth in bits per pixel. The default value is `16`.

framebuffer_depth	result	notes
8	8bit framebuffer	Default RGB palette makes screen unreadable
16	16bit framebuffer	
24	24bit framebuffer	May result in a corrupted display
32	32bit framebuffer	May need to be used in conjunction with `framebuffer_ignore_alpha=1`

FRAMEBUFFER_IGNORE_ALPHA

Set to `1` to disable the alpha channel. Can help with the display of a 32bit `framebuffer_depth`.

TEST_MODE

Displays a test image and sound during boot (but only over the composite video and analogue audio outputs) for the given number of seconds, before continuing to

boot the OS as normal. This is used as a manufacturing test; the default value is `0`.

DISPLAY_ROTATE

Can be used to rotate or flip the screen orientation; the default value is `0`.

display_rotate	result
0	no rotation
1	rotate 90 degrees clockwise
2	rotate 180 degrees clockwise
3	rotate 270 degrees clockwise
0x10000	horizontal flip
0x20000	vertical flip

Note that the 90 and 270 degree rotation options require additional memory on the GPU, so these won't work with the 16MB GPU split.

LICENCE KEYS/CODECS

Hardware decoding of additional codecs can be enabled by purchasing a licence that is locked to the CPU serial number of your Raspberry Pi.

DECODE_MPG2

Licence key to allow hardware MPEG-2 decoding, e.g. `decode_MPG2=0x12345678`.

DECODE_WVC1

Licence key to allow hardware VC-1 decoding, e.g. `decode_WVC1=0x12345678`.

If you've got multiple Raspberry Pis and you've bought a codec licence for each of them, you can list up to 8 licence keys in a single `config.txt`, for example `decode_MPG2=0x12345678,0xabcdabcd,0x87654321`. This enables you to swap the same SD card between the different Pis without having to edit `config.txt` each time.

BOOT

DISABLE_COMMANDLINE_TAGS

Set to `1` to stop `start.elf` from filling in ATAGS (memory from `0x100`) before launching the kernel.

CMDLINE

The alternative filename on the boot partition to read the kernel command line string from; the default value is `cmdline.txt`.

KERNEL

The alternative filename on the boot partition to use when loading the kernel; the default value is `kernel.img`.

KERNEL_ADDRESS

The memory address into which the kernel image should be loaded.

KERNEL_OLD

Set to `1` to load the kernel at the memory address `0x0`.

RAMFSFILE

Optional filename on the boot partition of a ramfs to load. More information is available here.

RAMFSADDR

The memory address into which the `ramfsfile` should be loaded.

INITRAMFS

This specifies both the ramfs filename **and** the memory address to load it at; it performs the actions of both `ramfsfile` and `ramfsaddr` in one parameter. Example values are: `initramfs initramf.gz 0x00800000`. **NOTE:** This option uses different syntax to all the other options; you should not use a `=` character here.

INIT_UART_BAUD

The initial UART baud rate; the default value is `115200`.

INIT_UART_CLOCK

The initial UART clock frequency; the default value is `3000000` (3MHz).

INIT_EMMC_CLOCK

The initial eMMC clock frequency; the default value is `100000000` (100MHz).

BOOTCODE_DELAY

Wait for a given number of seconds in `bootcode.bin` before loading `start.elf`; the default value is `0`.

This is particularly useful to insert a delay before reading the EDID of the monitor, which can help if the Pi and monitor are powered from the same source but the monitor takes longer to start up than the Pi. Try setting this value if the display detection is wrong on initial boot, but is correct if you soft-reboot the Pi without removing power from the monitor.

BOOT_DELAY

Wait for a given number of seconds in `start.elf` before loading the kernel; the default value is `1`. The total delay in milliseconds is calculated as `(1000 x boot_delay) + boot_delay_ms`. This can be useful if your SD card needs a while to 'get ready' before Linux is able to boot from it.

BOOT_DELAY_MS

Wait for a given number of milliseconds in `start.elf`, together with `boot_delay`, before loading the kernel. The default value is `0`.

DISABLE_SPLASH

If set to `1`, don't show the rainbow splash screen on boot. The default value is `0`.

DEVICE TREE

There are several `config.txt` parameters related to device tree setup, and these are documented separately here.

OVERCLOCKING

NOTE: Setting any overclocking parameters to values other than those used by raspi-config will set a permanent bit within the SoC, making it possible to detect that your Pi has been overclocked. This was originally set to detect a void warranty if the device had been overclocked. Since September 19th 2012, you can overclock your Pi without affecting your warranty; for more information see the blog post on *turbo mode*.

The latest kernel has a cpufreq kernel driver with the "ondemand" governor enabled by default. It has no effect if you have no overclock settings; if you overclock, the CPU frequency will vary with processor load. Non-default values are only used when needed according to the governor. You can adjust the minimum values with the `*_min` config options, or disable dynamic clocking with `force_turbo=1`; for more information see here.

Overclocking and overvoltage will be disabled at runtime when the SoC reaches 85°C, in order to cool it down. You should not hit the limit, even with maximum settings at 25°C ambient temperature; for more information see here.

OVERCLOCKING OPTIONS

Option	Description
arm_freq	Frequency of the ARM CPU in MHz. The default value is `700`.
gpu_freq	Sets `core_freq`, `h264_freq`, `isp_freq`, and `v3d_freq` together. The default value is `250`.
core_freq	Frequency of the GPU processor core in MHz. It has an impact on CPU performance, since it drives the L2 cache. The default value is `250`.
h264_freq	Frequency of the hardware video block in MHz. The default value is `250`.
isp_freq	Frequency of the image sensor pipeline block in MHz. The default value is `250`.
v3d_freq	Frequency of the 3D block in MHz. The default value is `250`.
avoid_pwm_pll	Don't dedicate a PLL to PWM audio. This will reduce analogue audio quality slightly. The spare PLL allows the `core_freq` to be set independently from the rest of the GPU, allowing for more control over overclocking. The default value is `0`.
sdram_freq	Frequency of the SDRAM in MHz. The default value is `400`.
	CPU/GPU core voltage adjustment. [-16,8] equates to

over_voltage	[0.8V,1.4V] with 0.025V steps; in other words, specifying -16 will give 0.8V as the GPU/core voltage, and specifying 8 will give 1.4V. The default value is `0` (1.2V). Values above 6 are only allowed when `force_turbo` or `current_limit_override` are specified; this sets the warranty bit.
over_voltage_sdram	Sets `over_voltage_sdram_c`, `over_voltage_sdram_i`, and `over_voltage_sdram_p` together.
over_voltage_sdram_c	SDRAM controller voltage adjustment. [-16,8] equates to [0.8V,1.4V] with 0.025V steps. The default value is `0` (1.2V).
over_voltage_sdram_i	SDRAM I/O voltage adjustment. [-16,8] equates to [0.8V,1.4V] with 0.025V steps. The default value is `0` (1.2V).
over_voltage_sdram_p	SDRAM phy voltage adjustment. [-16,8] equates to [0.8V,1.4V] with 0.025V steps. The default value is `0` (1.2V).
force_turbo	Disables the dynamic cpufreq driver and minimum settings described below. Enables h264/v3d/isp overclocking options. The default value is `0`. Enabling this may set the warranty bit.
initial_turbo	Enables turbo mode from boot for the given value in seconds up to 60, or until cpufreq sets a frequency; for more information see here. This option can help

	with SD card corruption if the Pi is overclocked. The default value is `0`.
arm_freq_min	Minimum value of `arm_freq` used for dynamic frequency clocking. The default value is `700`.
core_freq_min	Minimum value of `core_freq` used for dynamic frequency clocking. The default value is `250`.
sdram_freq_min	Minimum value of `sdram_freq` used for dynamic frequency clocking. The default value is `400`.
over_voltage_min	Minimum value of `over_voltage` used for dynamic frequency clocking. The default value is `0`.
temp_limit	Overheat protection. This sets the clocks and voltages to default when the SoC reaches this value in Celsius. Setting this higher than the default voids your warranty; the default value is `85`.
current_limit_override	Disables SMPS current limit protection when set to `0x5A000020`; it requires this unusual value to ensure that it doesn't get triggered accidentally. This can help if you're currently hitting a reboot failure when specifying a value for overclocking that is too high. For more information see here. Changing this option may set the warranty bit.

FORCE_TURBO

`force_turbo=0`

This enables dynamic clocks and voltage for the CPU, GPU core, and SDRAM. When busy, the CPU frequency goes up to `arm_freq` and down to `arm_freq_min` on idle.

`core_freq` / `core_freq_min`, `sdram_freq` / `sdram_freq_min` and `over_voltage` / `over_voltage_min` behave in a similar manner. `over_voltage` is limited to 6 (1.35V). Non-default values for the h264/v3d/isp frequencies are ignored.

`force_turbo=1`

Disables dynamic frequency clocking, so that all frequencies and voltages stay high. Overclocking of h264/v3d/isp GPU parts is allowed, as well as setting `over_voltage` up to 8 (1.4V). For more information see here.

CLOCKS RELATIONSHIP

The GPU core, h264, v3d, and ISP blocks all share a PLL and therefore need to have related frequencies. The CPU, SDRAM and GPU each have their own PLLs and can have unrelated frequencies; for more information see here.

The frequencies are calculated as follows:

```
pll_freq = floor(2400 / (2 x core_freq)) x (2 x core_freq)
gpu_freq = pll_freq / [even number]
```

The effective `gpu_freq` is automatically rounded to the nearest even integer; asking for `core_freq=500` and `gpu_freq=300` will result in the divisor of

2000/300 = 6.666 => 6 and so result in a `gpu_freq` of 333.33MHz.

AVOID_PWM_PLL

Setting this to `1` will decouple a PLL from the PWM hardware. This will result in more hiss on the analogue audio output, but will allow you to set the `gpu_freq` independently of the `core_freq`.

MONITORING TEMPERATURE AND VOLTAGE

To view the Pi's temperature, type: `cat /sys/class/thermal/thermal_zone0/temp`. Divide the result by 1000 to get the value in Celsius.

To view the Pi's current frequency, type: `cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq`. Divide the result by 1000 to get the value in MHz.

To monitor the Pi's PSU voltage, you'll need a multimeter to measure between the TP1 and TP2 power supply test points; more information is available in power.

It's generally a good idea to keep the core temperature below 70 degrees and the voltage above 4.8V. Note that some USB power supplies fall as low as 4.2V; this is because they are usually designed to charge a 3.7V LiPo battery, rather than to supply 5V to a computer. If your overclocked Raspberry Pi is getting hot a heatsink can be helpful, especially if the Pi is to be run inside a case. A suitable heatsink is the self-adhesive BGA (ball-grid-array) 14x14x10 mm heatsink, available from RS Components.

OVERCLOCKING PROBLEMS

Most overclocking issues show up immediately with a failure to boot. If this occurs, hold down the `shift` key during the next boot which will temporarily disable all overclocking; this will allow you to boot successfully and then edit your settings.

CONDITIONAL FILTERS

When a single SD card (or card image) is only being used with one Pi and one monitor, it's easy to simply set `config.txt` as required for that specific combination and keep it that way, amending only when something changes.

However, if one Pi is swapped between different monitors, or if the SD card (or card image) is being swapped between multiple Pis, a single set of settings may no longer be sufficient. Conditional filters allow you to make certain sections of the config file used only in specific cases, allowing a single `config.txt` to create different configurations when read by different hardware.

THE `[ALL]` FILTER

This is the most basic filter: it resets all previously set filters and allows any settings listed below it to be applied to all hardware.

```
[all]
```

It's usually a good idea to add an `[all]` filter at the end of groups of filtered settings to avoid unintentionally combining filters (see below).

THE `[PI1]` AND `[PI2]` FILTERS

Any settings below a `[pi1]` filter will only be applied to Pi 1 (A, A+, B, B+) hardware. Any settings below a `[pi2]` filter will only be applied to Pi 2 hardware.

```
[pi1]
[pi2]
```

These are particularly useful for defining different `kernel`, `initramfs`, and `cmdline` settings, as the Pi 1 and Pi 2 require different kernels. They can also be useful to define different overclocking settings for each, since they have different default speeds. For example, to define separate `initramfs` images for each:

```
[pi1]
initramfs initrd.img-3.18.7+ followkernel
[pi2]
initramfs initrd.img-3.18.7-v7+ followkernel
[all]
```

Remember to use the `[all]` filter at the end, so that any subsequent settings aren't limited to Pi 2 hardware only.

THE `[EDID=*]` FILTER

When switching between multiple monitors while using a single SD card in your Pi, and where a blank config isn't sufficient to automatically select the desired resolution for each one, this allows specific settings to be chosen based on the monitors' EDID names.

To view the EDID name of a specific monitor, run the following command:

```
tvservice -n
```

This will print something like this:

```
device_name=VSC-TD2220
```

You can then specify settings that apply only to this monitor like so:

```
[EDID=VSC-TD2220]
hdmi_group=2
hdmi_mode=82
[all]
```

This forces 1920x1080 DVT mode for this monitor, without affecting any other monitors.

Note that these settings apply only at boot, so the monitor must be connected at boot time and the Pi must be able to read its EDID information to get the correct name. Hotplugging a different monitor after boot will not reselect different settings.

THE SERIAL NUMBER FILTER

Sometimes settings should only be applied to a single specific Pi, even if you swap the SD card to a different one. Examples include licence keys and overclocking settings (although the licence keys already support SD card swapping in a different way). You can also use this to select different display settings even if the EDID

identification above isn't possible for some reason, provided that you don't swap monitors between your Pis - for example, if your monitor doesn't supply a usable EDID name or if you're using composite output (for which EDID cannot be read).

To view the serial number of your Pi, run the following command:

```
cat /proc/cpuinfo
```

The serial will be shown as a 16-digit hex value at the bottom. For example, if you see:

```
Serial          : 0000000012345678
```

Then you can define settings that will only be applied to this specific Pi like so:

```
[0x12345678]
# settings here are applied only to the Pi with this serial
[all]
# settings here are applied to all hardware
```

COMBINING CONDITIONAL FILTERS

Filters of the same type replace each other, so `[pi2]` overrides `[pi1]`, as it's not possible for both to be true at once.

Filters of different types can be combined simply by listing them one after the other, for example:

```
# settings here are applied to all hardware
[EDID=VSC-TD2220]
# settings here are applied only if monitor VSC-TD2220 is connected
[pi2]
# settings here are applied only if monitor VSC-TD2220 is connected
*and* on a Pi 2
[all]
# settings here are applied to all hardware
```

Use the `[all]` filter to reset all previous filters and avoid unintentionally combining different filter types.