The config.txt file

What is `config.txt`?

Edit this on GitHub

The Raspberry Pi uses a configuration file instead of the BIOS you would expect to find on a conventional PC. The system configuration parameters, which would traditionally be edited and stored using a BIOS, are stored instead in an optional text file named `config.txt`. This is read by the GPU before the ARM CPU and Linux are initialised. It must therefore 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 the `root` user. From Windows or OS X it is visible 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, simply create it as a new text file.

Any changes will only take effect after you have rebooted your Raspberry Pi. After Linux has booted, you can view the current active settings using the following commands:

- `vcgencmd get_config <config>`: this displays a specific config value, e.g. `vcgencmd get_config arm_freq`.
- `vcgencmd get_config int`: this lists all the integer config options that are set (non-zero).
- `vcgencmd get_config str`: this lists all the string config options that are set (non-null).

NOTE

> There are some config settings that cannot be retrieved using `vcgencmd`.

File Format

The `config.txt` file is read by the early-stage boot firmware, so 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.

There is a 98-character line length limit (previously 78) for entries - any characters past this limit will be ignored.

Here is an example file:

```
# Enable audio (loads snd_bcm2835)
dtparam=audio=on

# Automatically load overlays for detected cameras
camera_auto_detect=1

# Automatically load overlays for detected DSI displays
display_auto_detect=1

# Enable DRM VC4 V3D driver
dtoverlay=vc4-kms-v3d
max_framebuffers=2

# Disable compensation for displays with overscan
disable_overscan=1
```

Advanced Features

`include`

Causes the content of the specified file to be inserted into the current file.

For example, adding the line `include extraconfig.txt` to `config.txt` will include the content of `extraconfig.txt` file in the `config.txt` file.

Include directives are not supported by bootcode.bin or the EEPROM bootloader

Conditional Filters

Conditional filters are covered in the conditionals section.

`autoboot.txt`

Edit this on GitHub

`autoboot.txt` is an optional configuration file that can be used to specify the `boot_partition` number. This is sometimes used with NOOBS to bypass the boot menu selection and boot a specific partition.

This can also be used in conjunction with the `tryboot` feature to implement A/B booting for OS upgrades.

`autoboot.txt` is limited to 512 bytes and supports the `[all]`, `[none]` and `[tryboot]` conditional filters.

See also TRYBOOT boot flow.

boot_partition

Specifies the partition number for booting unless the partition number was already specified as parameter to the `reboot` command (e.g. `sudo reboot 2`).

The [tryboot] filter

This filter passes if the system was booted with the `tryboot` flag set.

```
sudo reboot "0 tryboot"
```

tryboot_a_b

Set this property to `1` to load the normal `config.txt` and `boot.img` files instead of `tryboot.txt` and `tryboot.img` when the `tryboot` flag is set. This enables the `tryboot` switch to be made at the partition level rather than the file-level without having to modify configuration files in the A/B partitions.

A/B boot example

In the following example partition `3` would be contain the pending OS upgrade and would be tested by rebooting in `tryboot` mode (`sudo reboot "0 tryboot"`). If the OS determines that the upgrade was successful then it would replace `autoboot.txt` swapping the partition numbers. Otherwise, if the system is reboot (e.g. watchdog or cold boot) then system would boot from partition `2` as usual.

Since the `autoboot.txt` file is a single sector it will normally be possible to update this with a single sector update to the SD/EMMC.

`autoboot.txt`

```
[all]
tryboot_a_b=1
boot_partition=2
[tryboot]
boot_partition=3
```

Common Options

Common Display Options

`disable_overscan`

The default value for `disable_overscan` is `0` which gives default values of overscan for the left, right, top, and bottom edges of `48` for HD CEA modes, `32` for SD CEA modes, and `0` for DMT modes.

Set `disable_overscan` to `1` to disable the default values of overscan that are set by the firmware.

`hdmi_enable_4kp60` (Raspberry Pi 4 Only)

By default, when connected to a 4K monitor, the Raspberry Pi 4B, 400 and CM4 will select a 30Hz refresh rate. Use this option to allow selection of 60Hz refresh rates.

IMPORTANT

It is not possible to output 4Kp60 on both micro HDMI ports simultaneously.

WARNING

Setting `hdmi_enable_4kp60` will increase power consumption and the temperature of your Raspberry Pi.

Common Hardware Configuration Options

`camera_auto_detect`

With this setting enabled (set to `1`), the firmware will automatically load overlays for cameras that it recognises.

IMPORTANT

New Raspberry Pi OS images from Bullseye onwards come with this setting by default.

`display_auto_detect`

With this setting enabled (set to `1`), the firmware will automatically load overlays for displays that it recognises.

IMPORTANT

New Raspberry Pi OS images from Bullseye onwards come with this setting by default.

`dtoverlay`

The `dtoverlay` option requests the firmware to load a named Device Tree overlay - a configuration file that can enable kernel support for built-in and external hardware. For example, `dtoverlay=vc4-kms-v3d` loads an overlay that enables the kernel graphics driver.

As a special case, if called with no value - `dtoverlay=` - it marks the end of a list of overlay parameters. If used before any other `dtoverlay` or `dtparam` setting it prevents the loading of any HAT overlay.

For more details, see DTBs, overlays and config.txt.

`dtparam`

Device Tree configuration files for Raspberry Pis support a number of parameters for such things as enabling I2C and SPI interfaces. Many DT overlays are configurable via the use of parameters. Both types of parameters can be supplied using the `dtparam` setting. In addition, overlay parameters can be appended to the `dtoverlay` option, separated by commas, but beware the line length limit - previously 78 characters, now 98 characters.

For more details, see DTBs, overlays and config.txt.

`arm_boost` (Raspberry Pi 4 Only)

All Raspberry Pi 400s and newer revisions of the Raspberry Pi 4B are equipped with a second switch-mode power supply for the SoC voltage rail, and this allows the default turbo-mode clock to be increased from 1.5GHz to 1.8GHz. This change should be safe for all such boards, but to avoid unrequested changes for existing installations this change must be accepted by setting `arm_boost=1`.

IMPORTANT

> New Raspberry Pi OS images from Bullseye onwards come with this setting by default.

Onboard Analogue Audio (3.5mm Jack)

Edit this on GitHub

The onboard audio output uses config options to change the way the analogue audio is driven, and whether some firmware features are enabled or not.

`audio_pwm_mode`

`audio_pwm_mode=1` selects legacy low-quality analogue audio from the 3.5mm AV jack.

`audio_pwm_mode=2` (the default) selects high quality analogue audio using an advanced modulation scheme.

NOTE

> This option uses more GPU compute resources and can interfere with some use cases.

`disable_audio_dither`

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

`enable_audio_dither`

Audio dither (see disable_audio_dither above) is normally disabled when the audio samples are larger than 16 bits. Set this option to `1` to force the use of dithering for all bit depths.

`pwm_sample_bits`

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

Boot Options

Edit this on GitHub

`start_file`, `fixup_file`

These options specify the firmware files transferred to the VideoCore GPU prior to booting.

`start_file` specifies the VideoCore firmware file to use. `fixup_file` specifies the file used to fix up memory locations used in the `start_file` to match the GPU memory split. Note that the `start_file` and the `fixup_file` are a matched pair - using unmatched files will stop the board from booting. This is an advanced option, so we advise that you use `start_x` and `start_debug` rather than this option.

`start_x`, `start_debug`

These provide a shortcut to some alternative `start_file` and `fixup_file` settings, and are the recommended methods for selecting firmware configurations.

`start_x=1` implies

```
start_file=start_x.elf
fixup_file=fixup_x.dat
```

On the Raspberry Pi 4, if the files `start4x.elf` and `fixup4x.dat` are present, these files will be used instead.

`start_debug=1` implies

```
start_file=start_db.elf
fixup_file=fixup_db.dat
```

`start_x=1` should be specified when using the camera module. Enabling the camera via `raspi-config` will set this automatically.

disable_commandline_tags

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

cmdline

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

kernel

`kernel` is the alternative filename on the boot partition to use when loading the kernel. The default value on the Raspberry Pi 1, Zero and Zero W, and Raspberry Pi Compute Module 1 is `kernel.img`. The default value on the Raspberry Pi 2, 3, 3+ and Zero 2 W, and Raspberry Pi Compute Modules 3 and 3+ is `kernel7.img`. The default value on the Raspberry Pi 4 and 400, and Raspberry Pi Compute Module 4 is `kernel7l.img`.

arm_64bit

If set to non-zero, forces the kernel loading system to assume a 64-bit kernel, starts the processors up in 64-bit mode, and sets `kernel8.img` to be the kernel image loaded, unless

there is an explicit `kernel` option defined in which case that is used instead. Defaults to 0 on all platforms.

NOTE

64-bit kernels may be uncompressed image files or a gzip archive of an image (which can still be called kernel8.img; the bootloader will recognize the archive from the signature bytes at the beginning).

NOTE

The 64-bit kernel will only work on the Raspberry Pi 3, 3+, 4, 400, Zero 2 W and 2B rev 1.2, and Raspberry Pi Compute Modules 3, 3+ and 4.

arm_control

WARNING

This setting is **DEPRECATED**, use `arm_64bit` instead to enable 64-bit kernels.

Sets board-specific control bits.

armstub

`armstub` is the filename on the boot partition from which to load the ARM stub. The default ARM stub is stored in firmware and is selected automatically based on the Raspberry Pi model and various settings.

The stub is a small piece of ARM code that is run before the kernel. Its job is to set up low-level hardware like the interrupt controller before passing control to the kernel.

arm_peri_high

Set `arm_peri_high` to `1` to enable "High Peripheral" mode on the Raspberry Pi 4. It is set automatically if a suitable DTB is loaded.

NOTE

Enabling "High Peripheral" mode without a compatible device tree will make your system fail to boot. Currently ARM stub support is missing, so you will also need to load a suitable file using `armstub`.

kernel_address

`kernel_address` is the memory address to which the kernel image should be loaded. 32-bit kernels are loaded to address `0x8000` by default, and 64-bit kernels to address `0x200000`. If `kernel_old` is set, kernels are loaded to the address `0x0`.

kernel_old

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

ramfsfile

`ramfsfile` is the optional filename on the boot partition of a `ramfs` to load.

NOTE

> Newer firmware supports the loading of multiple `ramfs` files. You should separate the multiple file names with commas, taking care not to exceed the 80-character line length limit. All the loaded files are concatenated in memory and treated as a single `ramfs` blob. More information is available on the forums.

ramfsaddr

`ramfsaddr` is the memory address to which the `ramfsfile` should be loaded.

initramfs

The `initramfs` command specifies both the ramfs filename **and** the memory address to which to load it. It performs the actions of both `ramfsfile` and `ramfsaddr` in one parameter. The address can also be `followkernel` (or `0`) to place it in memory after the kernel image. Example values are: `initramfs initramf.gz 0x00800000` or `initramfs init.gz followkernel`. As with `ramfsfile`, newer firmwares allow the loading of multiple files by comma-separating their names.

NOTE

> This option uses different syntax from all the other options, and you should not use a `=` character here.

init_uart_baud

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

init_uart_clock

`init_uart_clock` is the initial UART clock frequency. The default value is `48000000` (48MHz). Note that this clock only applies to UART0 (ttyAMA0 in Linux), and that the maximum baudrate for the UART is limited to 1/16th of the clock. The default UART on the Raspberry Pi 3 and Raspberry Pi Zero is UART1 (ttyS0 in Linux), and its clock is the core VPU clock - at least 250MHz.

bootcode_delay

The `bootcode_delay` command delays 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, for example if the Raspberry Pi and monitor are powered from the same source, but the monitor takes longer to start up than the Raspberry Pi. Try setting this value if the display detection is wrong on initial boot, but is correct if you soft-reboot the Raspberry Pi without removing power from the monitor.

boot_delay

The `boot_delay` command instructs to 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

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

disable_poe_fan

By default, a probe on the I2C bus will happen at startup, even when a PoE HAT is not attached. Setting this option to 1 disables control of a PoE HAT fan through I2C (on pins ID_SD & ID_SC). If you are not intending to use a PoE HAT doing this is useful if you need to minimise boot time.

disable_splash

If `disable_splash` is set to `1`, the rainbow splash screen will not be shown on boot. The default value is `0`.

enable_gic (Raspberry Pi 4 Only)

On the Raspberry Pi 4B, if this value is set to `0` then the interrupts will be routed to the ARM cores using the legacy interrupt controller, rather than via the GIC-400. The default value is `1`.

enable_uart

`enable_uart=1` (in conjunction with `console=serial0` in `cmdline.txt`) requests that the kernel creates a serial console, accessible using GPIOs 14 and 15 (pins 8 and 10 on the 40-pin header). Editing `cmdline.txt` to remove the line `quiet` enables boot messages from the kernel to also appear there. See also `uart_2ndstage`.

force_eeprom_read

Set this option to `0` to prevent the firmware from trying to read an I2C HAT EEPROM (connected to pins ID_SD & ID_SC) at powerup. See also `disable_poe_fan`.

os_prefix

`os_prefix` is an optional setting that allows you to choose between multiple versions of the kernel and Device Tree files installed on the same card. Any value in `os_prefix` is prepended to (stuck in front of) the name of any operating system files loaded by the firmware, where "operating system files" is defined to mean kernels, initramfs, cmdline.txt, .dtbs and overlays. The prefix would commonly be a directory name, but it could also be part of the filename such as "test-". For this reason, directory prefixes must include the trailing `/` character.

In an attempt to reduce the chance of a non-bootable system, the firmware first tests the supplied prefix value for viability - unless the expected kernel and .dtb can be found at the new location/name, the prefix is ignored (set to ""). A special case of this viability test is applied to overlays, which will only be loaded from `${os_prefix}${overlay_prefix}` (where the default value of `overlay_prefix` is "overlays/") if `${os_prefix}${overlay_prefix}README` exists, otherwise it ignores `os_prefix` and treats overlays as shared.

(The reason the firmware checks for the existence of key files rather than directories when checking prefixes is twofold - the prefix may not be a directory, and not all boot methods support testing for the existence of a directory.)

NOTE

> Any user-specified OS file can bypass all prefixes by using an absolute path (with respect to the boot partition) - just start the file path with a `/`, e.g. `kernel=/my_common_kernel.img`.

See also `overlay_prefix` and `upstream_kernel`.

otg_mode (Raspberry Pi 4 Only)

USB On-The-Go (often abbreviated to OTG) is a feature that allows supporting USB devices with an appropriate OTG cable to configure themselves as USB hosts. On older Raspberry Pis, a single USB 2 controller was used in both USB host and device mode.

Raspberry Pi 4B and Raspberry Pi 400 (not CM4 or CM4IO) add a high performance USB 3 controller, attached via PCIe, to drive the main USB ports. The legacy USB 2 controller is still available on the USB-C power connector for use as a device (`otg_mode=0`, the default).

`otg_mode=1` requests that a more capable XHCI USB 2 controller is used as another host controller on that USB-C connector.

NOTE

> Because CM4 and CM4IO don't include the external USB 3 controller, Raspberry Pi OS images set `otg_mode=1` on CM4 for better performance.

overlay_prefix

Specifies a subdirectory/prefix from which to load overlays - defaults to `overlays/` (note the trailing `/`). If used in conjunction with `os_prefix`, the `os_prefix` comes before the `overlay_prefix`, i.e. `dtoverlay=disable-bt` will attempt to load `${os_prefix}${overlay_prefix}disable-bt.dtbo`.

NOTE

> Unless `${os_prefix}${overlay_prefix}README` exists, overlays are shared with the main OS (i.e. `os_prefix` is ignored).

sha256

If set to non-zero, enables the logging of SHA256 hashes for loaded files (the kernel, initramfs, Device Tree .dtb file and overlays), as generated by the `sha256sum` utility. The logging output goes to the UART if enabled, and is also accessible via `sudo vcdbg log msg`. This option may be useful when debugging booting problems, but at the cost of potentially adding *many* seconds to the boot time. Defaults to 0 on all platforms.

uart_2ndstage

Setting `uart_2ndstage=1` causes the second-stage loader (`bootcode.bin` on devices prior to the Raspberry Pi 4, or the boot code in the EEPROM for Raspberry Pi 4 devices) and the

main firmware (`start*.elf`) to output diagnostic information to UART0.

Be aware that output is likely to interfere with Bluetooth operation unless it is disabled (`dtoverlay=disable-bt`) or switched to the other UART (`dtoverlay=miniuart-bt`), and if the UART is accessed simultaneously to output from Linux then data loss can occur leading to corrupted output. This feature should only be required when trying to diagnose an early boot loading problem.

upstream_kernel

If `upstream_kernel=1` is used, the firmware sets `os_prefix` to "upstream/", unless it has been explicitly set to something else, but like other `os_prefix` values it will be ignored if the required kernel and .dtb file can't be found when using the prefix.

The firmware will also prefer upstream Linux names for DTBs (`bcm2837-rpi-3-b.dtb` instead of `bcm2710-rpi-3-b.dtb`, for example). If the upstream file isn't found the firmware will load the downstream variant instead and automatically apply the "upstream" overlay to make some adjustments. Note that this process happens *after* the `os_prefix` has been finalised.

GPIO Control

Edit this on GitHub

gpio

The `gpio` directive allows GPIO pins to be set to specific modes and values at boot time in a way that would previously have needed a custom `dt-blob.bin` file. Each line applies the same settings (or at least makes the same changes) to a set of pins, either a single pin (`3`), a range of pins (`3-4`), or a comma-separated list of either (`3-4,6,8`). The pin set is followed by an `=` and one or more comma-separated attributes from this list:

- `ip` - Input
- `op` - Output
- `a0-a5` - Alt0-Alt5
- `dh` - Driving high (for outputs)
- `dl` - Driving low (for outputs)
- `pu` - Pull up
- `pd` - Pull down

- `pn/np` - No pull

`gpio` settings are applied in order, so those appearing later override those appearing earlier.

Examples:

```
# Select Alt2 for GPIO pins 0 to 27 (for DPI24)
gpio=0-27=a2

# Set GPIO12 to be an output set to 1
gpio=12=op,dh

# Change the pull on (input) pins 18 and 20
gpio=18,20=pu

# Make pins 17 to 21 inputs
gpio=17-21=ip
```

The `gpio` directive respects the "[…]" section headers in `config.txt`, so it is possible to use different settings based on the model, serial number, and EDID.

GPIO changes made through this mechanism do not have any direct effect on the kernel — they don't cause GPIO pins to be exported to the sysfs interface, and they can be overridden by pinctrl entries in the Device Tree as well as utilities like `raspi-gpio`.

Note also that there is a delay of a few seconds between power being applied and the changes taking effect — longer if booting over the network or from a USB mass storage device.

`enable_jtag_gpio`

Setting `enable_jtag_gpio=1` selects Alt4 mode for GPIO pins 22-27, and sets up some internal SoC connections, thus enabling the JTAG interface for the ARM CPU. It works on all models of Raspberry Pi.

Pin #	Function
GPIO22	ARM_TRST
GPIO23	ARM_RTCK
GPIO24	ARM_TDO
GPIO25	ARM_TCK
GPIO26	ARM_TDI
GPIO27	ARM_TMS

Overclocking Options

Edit this on GitHub

The kernel has a CPUFreq driver with the "powersave" governor enabled by default, switched to "ondemand" during boot, when raspi-config is installed. With "ondemand" governor, CPU frequency will vary with processor load. You can adjust the minimum values with the `*_min` config options or disable dynamic clocking by applying a static scaling governor ("powersave" or "performance") or with `force_turbo=1`.

Overclocking and overvoltage will be disabled at runtime when the SoC reaches `temp_limit` (see below), which defaults to 85°C, in order to cool down the SoC. You should not hit this limit with Raspberry Pi 1 and Raspberry Pi 2, but you are more likely to with Raspberry Pi 3 and Raspberry Pi 4. Overclocking and overvoltage are also disabled when an undervoltage situation is detected.

NOTE

For more information see the section on frequency management and thermal control.

WARNING

Setting any overclocking parameters to values other than those used by raspi-config may set a permanent bit within the SoC, making it possible to detect that your Raspberry Pi has been overclocked. The specific circumstances where the overclock bit is set are if `force_turbo` is set to `1` and any of the `over_voltage_*` options are set to a value > `0`. See the blog post on Turbo Mode for more information.

Overclocking

Option	Description
arm_freq	Frequency of the ARM CPU in MHz.
arm_boost	Increases `arm_freq` to the highest supported frequency for the board-type and firmware. Set to `1` to enable.
gpu_freq	Sets `core_freq`, `h264_freq`, `isp_freq`, `v3d_freq` and `hevc_freq` together
core_freq	Frequency of the GPU processor core in MHz, influences CPU performance because it drives the L2 cache and memory bus; the L2 cache benefits only Raspberry Pi Zero / Raspberry Pi Zero W / Raspberry Pi 1, there is a small benefit for SDRAM on Raspberry Pi 2 / Raspberry Pi 3. See section below for use on the Raspberry Pi 4.
h264_freq	Frequency of the hardware video block in MHz; individual override of the `gpu_freq` setting

Option	Description
isp_freq	Frequency of the image sensor pipeline block in MHz; individual override of the `gpu_freq` setting
v3d_freq	Frequency of the 3D block in MHz; individual override of the `gpu_freq` setting
hevc_freq	Frequency of the High Efficiency Video Codec block in MHz; individual override of the `gpu_freq` setting. Raspberry Pi 4 only.
sdram_freq	Frequency of the SDRAM in MHz. SDRAM overclocking on Raspberry Pi 4B is not currently supported
over_voltage	CPU/GPU core upper voltage limit. The value should be in the range [-16,8] which equates to the range [0.95V,1.55V] ([0.8,1.4V] on Raspberry Pi 1) with 0.025V steps. In other words, specifying -16 will give 0.95V (0.8V on Raspberry Pi 1) as the maximum CPU/GPU core voltage, and specifying 8 will allow up to 1.55V (1.4V on Raspberry Pi 1). For defaults see table below. Values above 6 are only allowed when `force_turbo=1` is specified: this sets the warranty bit if `over_voltage_*` > `0` is also set.
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.
over_voltage_sdram_i	SDRAM I/O voltage adjustment. [-16,8] equates to [0.8V,1.4V] with 0.025V steps.
over_voltage_sdram_p	SDRAM phy voltage adjustment. [-16,8] equates to [0.8V,1.4V] with 0.025V steps.
force_turbo	Forces turbo mode frequencies even when the ARM cores are not busy. Enabling this may set the warranty bit if `over_voltage_*` is also set.
initial_turbo	Enables turbo mode from boot for the given value in seconds, or until cpufreq sets a frequency. The maximum value is `60`.

Option	Description
arm_freq_min	Minimum value of `arm_freq` used for dynamic frequency clocking. Note that reducing this value below the default does not result in any significant power savings and is not currently supported.
core_freq_min	Minimum value of `core_freq` used for dynamic frequency clocking.
gpu_freq_min	Minimum value of `gpu_freq` used for dynamic frequency clocking.
h264_freq_min	Minimum value of `h264_freq` used for dynamic frequency clocking.
isp_freq_min	Minimum value of `isp_freq` used for dynamic frequency clocking.
v3d_freq_min	Minimum value of `v3d_freq` used for dynamic frequency clocking.
hevc_freq_min	Minimum value of `hevc_freq` used for dynamic frequency clocking.
sdram_freq_min	Minimum value of `sdram_freq` used for dynamic frequency clocking.
over_voltage_min	Minimum value of `over_voltage` used for dynamic frequency clocking. The value should be in the range [-16,8] which equates to the range [0.8V,1.4V] with 0.025V steps. In other words, specifying -16 will give 0.8V as the CPU/GPU core idle voltage, and specifying 8 will give a minimum of 1.4V.
temp_limit	Overheat protection. This sets the clocks and voltages to default when the SoC reaches this value in degree Celsius. Values over 85 are clamped to 85.
temp_soft_limit	**3A+/3B+ only**. CPU speed throttle control. This sets the temperature at which the CPU clock speed throttling system activates. At this temperature, the clock speed is reduced from 1400MHz to 1200MHz. Defaults to `60`, can be raised to a maximum of `70`, but this may cause instability.

This table gives the default values for the options on various Raspberry Pi models, all frequencies are stated in MHz.

Option	Pi 0/W	Pi1	Pi2	Pi3	Pi3A+/Pi3B+	CM4 & Pi4B ⇐ R1.3	Pi4B R1.4	Pi 400	Pi Zero 2 W
arm_freq	1000	700	900	1200	1400	1500	1500 or 1800 if arm_boost=1	1800	1000
core_freq	400	250	250	400	400	500	500	500	400
h264_freq	300	250	250	400	400	500	500	500	300
isp_freq	300	250	250	400	400	500	500	500	300
v3d_freq	300	250	250	400	400	500	500	500	300
hevc_freq	N/A	N/A	N/A	N/A	N/A	500	500	500	N/A
sdram_freq	450	400	450	450	500	3200	3200	3200	450
arm_freq_min	700	700	600	600	600	600	600	600	600
core_freq_min	250	250	250	250	250	200	200	200	250
gpu_freq_min	250	250	250	250	250	250	250	250	250
h264_freq_min	250	250	250	250	250	250	250	250	250
isp_freq_min	250	250	250	250	250	250	250	250	250
v3d_freq_min	250	250	250	250	250	250	250	250	250
sdram_freq_min	400	400	400	400	400	3200	3200	3200	400

This table gives defaults for options that are the same across all models.

Option	Default
initial_turbo	0 (seconds)
temp_limit	85 (°C)
over_voltage	0 (1.35V, 1.2V on Raspberry Pi 1)
over_voltage_min	0 (1.2V)
over_voltage_sdram	0 (1.2V)
over_voltage_sdram_c	0 (1.2V)
over_voltage_sdram_i	0 (1.2V)
over_voltage_sdram_p	0 (1.2V)

The firmware uses Adaptive Voltage Scaling (AVS) to determine the optimum CPU/GPU core voltage in the range defined by `over_voltage` and `over_voltage_min`.

Specific to Raspberry Pi 4, Raspberry Pi 400 and CM4

The minimum core frequency when the system is idle must be fast enough to support the highest pixel clock (ignoring blanking) of the display(s). Consequently, `core_freq` will be boosted above 500 MHz if the display mode is 4Kp60.

Display option	**Max** `core_freq`
Default	500
hdmi_enable_4kp60	550

- Overclocking requires the latest firmware release.
- The latest firmware automatically scales up the voltage if the system is overclocked. Manually setting `over_voltage` disables automatic voltage scaling for overclocking.
- It is recommended when overclocking to use the individual frequency settings (`isp_freq`, `v3d_freq` etc) rather than `gpu_freq` because the maximum stable frequency will be different for ISP, V3D, HEVC etc.
- The SDRAM frequency is not configurable on Raspberry Pi 4.

`force_turbo`

By default (`force_turbo=0`) the "On Demand" CPU frequency driver will raise clocks to their maximum frequencies when the ARM cores are busy and will lower them to the minimum frequencies when the ARM cores are idle.

`force_turbo=1` overrides this behaviour and forces maximum frequencies even when the ARM cores are not busy.

`never_over_voltage`

Sets a bit in the OTP memory (one time programmable) that prevents the device from being overvoltaged. This is intended to lock the device down so the warranty bit cannot be set either inadvertently or maliciously by using an invalid overvoltage.

`disable_auto_turbo`

On Raspberry Pi 2 / Raspberry Pi 3, setting this flag will disable the GPU from moving into turbo mode, which it can do in particular load cases.

Clocks Relationship

The GPU core, CPU, SDRAM and GPU each have their own PLLs and can have unrelated frequencies. The h264, v3d and ISP blocks share a PLL.

To view the Raspberry Pi's current frequency in KHz, type: `cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq`. Divide the result by 1000 to find the value in MHz. Note that this frequency is the kernel *requested* frequency, and it is possible that any throttling (for example at high temperatures) may mean the CPU is actually running more slowly than reported. An instantaneous measurement of the actual ARM CPU frequency can be retrieved using the vcgencmd `vcgencmd measure_clock arm`. This is displayed in Hertz.

Monitoring Core Temperature

To view the Raspberry Pi's temperature, type `cat /sys/class/thermal/thermal_zone0/temp`. Divide the result by 1000 to find the value in degrees Celsius. Alternatively, there is a vcgencmd, `vcgencmd measure_temp` that interrogates the GPU directly for its temperature.

Whilst hitting the temperature limit is not harmful to the SoC, it will cause CPU throttling. A heatsink can help to control the core temperature and therefore performance. This is especially useful if the Raspberry Pi is running inside a case. Airflow over the heatsink will make cooling more efficient.

With firmware from 12th September 2016 or later, when the core temperature is between 80'C and 85'C, a warning icon showing a red half-filled thermometer will be displayed, and the ARM cores will be throttled back. If the temperature exceeds 85'C, an icon showing a fully-filled thermometer will be displayed, and both the ARM cores and the GPU will be throttled back.

For the Raspberry Pi 3 Model B+, the PCB technology has been changed to provide better heat dissipation and increased thermal mass. In addition, a soft temperature limit has been introduced, with the goal of maximising the time for which a device can "sprint" before reaching the hard limit at 85°C. When the soft limit is reached, the clock speed is reduced from 1.4GHz to 1.2GHz, and the operating voltage is reduced slightly. This reduces the rate of temperature increase: we trade a short period at 1.4GHz for a longer period at 1.2GHz. By default, the soft limit is 60°C, and this can be changed via the `temp_soft_limit` setting in config.txt.

See the page on warning icons for more details.

Monitoring Voltage

It is essential to keep the supply voltage above 4.8V for reliable performance. Note that the voltage from some USB chargers/power supplies can fall as low as 4.2V. This is because they are usually designed to charge a 3.7V LiPo battery, not to supply 5V to a computer.

To monitor the Raspberry Pi's PSU voltage, you will need to use a multimeter to measure between the VCC and GND pins on the GPIO. More information is available in power.

If the voltage drops below 4.63V (+-5%), recent versions of the firmware will show a yellow lightning bolt symbol on the display to indicate a lack of power, and a message indicating the low voltage state will be added to the kernel log.

See the page on warning icons for more details.

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. This will temporarily disable all overclocking, allowing you to boot successfully and then edit your settings.

Conditional Filters

Edit this on GitHub

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

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

The `[all]` filter

The `[all]` filter is the most basic filter. It resets all previously set filters and allows any settings listed below it to be applied to all hardware. It is usually a good idea to add an `[all]` filter at the end of groups of filtered settings to avoid unintentionally combining filters (see below).

Model Filters

The conditional model filters are applied according to the following table.

Filter	Applicable model(s)
[pi1]	Model 1A, Model 1B, Model 1A+, Model 1B+, Compute Module 1
[pi2]	Model 2B (BCM2836- or BCM2837-based)
[pi3]	Model 3B, Model 3B+, Model 3A+, Compute Module 3, Compute Module 3+

Filter	Applicable model(s)
[pi3+]	Model 3A+, Model 3B+
[pi4]	Model 4B, Pi 400, Compute Module 4, Compute Module 4S
[pi400]	Pi 400
[cm4]	Compute Module 4
[cm4s]	Compute Module 4S
[pi0]	Zero, Zero W, Zero 2 W
[pi0w]	Zero W
[pi02]	Zero 2 W
[board-type=Type]	Filter by `Type` number - see Raspberry Pi Revision Codes E.g `[board-type=0x14]` would match CM4.

These are particularly useful for defining different `kernel`, `initramfs`, and `cmdline` settings, as the Raspberry Pi 1 and Raspberry Pi 2 require different kernels. They can also be useful to define different overclocking settings, as the Raspberry Pi 1 and Raspberry Pi 2 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 Raspberry Pi 2 hardware only.

It is important to note that the Raspberry Pi Zero W will see the contents of `[pi0w]` AND `[pi0]`. Likewise, a Raspberry Pi 3B+ sees `[pi3+]` AND `[pi3]`, and a Raspberry Pi 400 sees `[pi400]` AND `[pi4]`. If you want a setting to apply only to Raspberry Pi Zero, Raspberry Pi 3B or Raspberry Pi 4B, you need to follow it (order is important) with a setting in the `[pi0w]`, `[pi3+]` or `[pi400]` section that reverts it.

The [none] filter

The `[none]` filter prevents any settings that follow from being applied to any hardware. Although there is nothing that you can't do without `[none]`, it can be a useful way to keep groups of unused settings in config.txt without having to comment out every line.

The [tryboot] filter

This filter succeeds if the `tryboot` reboot flag was set.

It is intented for use in autoboot.txt to select a different `boot_partition` in `tryboot` mode for fail-safe OS updates.

The `[EDID=*]` filter

When switching between multiple monitors while using a single SD card in your Raspberry 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 an attached 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:

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

This forces 1920x1080 DVT mode for the specified 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 Raspberry Pi must be able to read its EDID information to find the correct name. Hotplugging a different monitor into the Raspberry Pi after boot will not select different settings.

On the Raspberry Pi 4, if both HDMI ports are in use, then the EDID will be checked against both of them, and subsequent configuration applied only to the first matching device. You can determine the EDID names for both ports by first running `tvservice -l` in a terminal window to list all attached devices and then using the returned numerical IDs in `tvservice -v <id> -n` to find the EDID name for a specific display ID.

The Serial Number Filter

Sometimes settings should only be applied to a single specific Raspberry 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 is not possible, provided that you don't swap monitors between your Raspberry Pis. For example, if your monitor doesn't supply a usable EDID name, or if you are using composite output (for which EDID cannot be read).

To view the serial number of your Raspberry 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 Raspberry Pi:

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

The GPIO Filter

You can also filter depending on the state of a GPIO. For example

```
[gpio4=1]
# Settings here are applied if GPIO 4 is high

[gpio2=0]
# Settings here are applied if GPIO 2 is low

[all]
# settings here are applied to all hardware
```

The `[HDMI:*]` Filter

NOTE

> This filter is for the Raspberry Pi 4 only.

The Raspberry Pi 4 has two HDMI ports, and for many `config.txt` commands related to HDMI, it is necessary to specify which HDMI port is being referred to. The HDMI

conditional filters subsequent HDMI configurations to the specific port.

```
[HDMI:0]
  hdmi_group=2
  hdmi_mode=45
[HDMI:1]
  hdmi_group=2
  hdmi_mode=67
```

An alternative `variable:index` syntax is available on all port-specific HDMI commands. You could use the following, which is the same as the previous example:

```
hdmi_group:0=2
hdmi_mode:0=45
hdmi_group:1=2
hdmi_mode:1=67
```

Combining Conditional Filters

Filters of the same type replace each other, so `[pi2]` overrides `[pi1]`, because it is 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
Raspberry 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.

Legacy Options

Edit this on GitHub

The remaining groups of `config.txt` options are considered legacy settings, either because they relate to older software such as the firmware graphics driver, or because they have been deprecated or removed altogether.

Memory Options

gpu_mem

Specifies how much memory, in megabytes, to reserve for the exclusive use of the GPU: the remaining memory is allocated to the ARM CPU for use by the OS. For Raspberry Pis with less than 1GB of memory, the default is `64`; for Raspberry Pis with 1GB or more of memory the default is `76`.

IMPORTANT

Unlike GPUs found on x86 machines, where increasing memory can improve 3D performance, the architecture of the VideoCore means **there is no performance advantage from specifying values larger than is necessary**, and in fact it can harm performance.

To ensure the best performance of Linux, you should set `gpu_mem` to the lowest possible value. If a particular graphics feature is not working correctly, try increasing the value of `gpu_mem`, being mindful of the recommended maximums shown below.

On the Raspberry Pi 4 the 3D component of the GPU has its own memory management unit (MMU), and does not use memory from the `gpu_mem` allocation. Instead memory is allocated dynamically within Linux. This allows a smaller value to be specified for `gpu_mem` on the Raspberry Pi 4, compared to previous models.

On legacy kernels, the memory allocated to the GPU is used for display, 3D, Codec and camera purposes as well as some basic firmware housekeeping. The maximums specified below assume you are using all these features. If you are not, then smaller values of gpu_mem should be used.

The recommended maximum values are as follows:

total RAM	`gpu_mem` **recommended maximum**
256MB	`128`
512MB	`384`
1GB or greater	`512`, `76` on the Raspberry Pi 4

IMPORTANT

The default camera stack (libcamera2) on Raspberry Pi OS - Bullseye uses Linux CMA memory to allocate buffers instead of GPU memory so there is no benefit in increasing the GPU memory size.

It is possible to set `gpu_mem` to larger values, however this should be avoided since it can cause problems, such as preventing Linux from booting. The minimum value is `16`, however this disables certain GPU features.

You can also use `gpu_mem_256`, `gpu_mem_512`, and `gpu_mem_1024` to allow swapping the same SD card between Raspberry Pis with different amounts of RAM without having to edit `config.txt` each time:

gpu_mem_256

The `gpu_mem_256` command sets the GPU memory in megabytes for Raspberry Pis with 256MB of memory. (It is ignored if memory size is not 256MB). This overrides `gpu_mem`.

gpu_mem_512

The `gpu_mem_512` command sets the GPU memory in megabytes for Raspberry Pis with 512MB of memory. (It is ignored if memory size is not 512MB). This overrides `gpu_mem`.

gpu_mem_1024

The `gpu_mem_1024` command sets the GPU memory in megabytes for Raspberry Pis with 1GB or more of memory. (It is ignored if memory size is smaller than 1GB). This overrides `gpu_mem`.

total_mem

This parameter can be used to force a Raspberry Pi to limit its memory capacity: specify the total amount of RAM, in megabytes, you wish the Raspberry Pi to use. For example, to make a 4GB Raspberry Pi 4B behave as though it were a 1GB model, use the following:

```
total_mem=1024
```

This value will be clamped between a minimum of 128MB, and a maximum of the total memory installed on the board.

disable_l2cache

Setting this to `1` disables the CPU's access to the GPU's L2 cache and requires a corresponding L2 disabled kernel. Default value on BCM2835 is `0`. On BCM2836, BCM2837, and BCM2711, the ARMs have their own L2 cache and therefore the default is `1`. The standard Raspberry Pi `kernel.img` and `kernel7.img` builds reflect this difference in cache setting.

Licence Key and Codec Options

Edit this on GitHub

Hardware decoding of additional codecs on the Raspberry Pi 3 and earlier models can be enabled by purchasing a licence that is locked to the CPU serial number of your Raspberry Pi.

On the Raspberry Pi 4, the hardware codecs for MPEG2 or VC1 are permanently disabled and cannot be enabled even with a licence key; on the Raspberry Pi 4, thanks to its increased processing power compared to earlier models, MPEG2 and VC1 can be decoded in software via applications such as VLC. Therefore, a hardware codec licence key is not needed if you're using a Raspberry Pi 4.

`decode_MPG2`

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

`decode_WVC1`

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

If you have multiple Raspberry Pis and you've bought a codec licence for each of them, you can list up to eight 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 Raspberry Pis without having to edit `config.txt` each time.

Video Options

Edit this on GitHub

HDMI Mode

NOTE

> Because the Raspberry Pi 4 and Raspberry Pi 400 have two HDMI ports, some HDMI commands can be applied to either port. You can use the syntax `<command>:<port>`, where port is 0 or 1, to specify which port the setting should apply to. If no port is specified, the default is 0. If you specify a port number on a command that does not require a port number, the port is ignored. Further details on the syntax and alternatives mechanisms can be found in the HDMI sub-section of the conditionals section of the documentation.

In order to support dual 4k displays, the Raspberry Pi 4 has updated video hardware, which imposes minor restrictions on the modes supported.

hdmi_safe

Setting `hdmi_safe` to `1` will lead to "safe mode" settings being used 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 `hdmi_ignore_edid` to `0xa5000080` enables the ignoring of EDID/display data if your display does not have an accurate EDID. It requires this unusual value to ensure that it is not triggered accidentally.

hdmi_edid_file

Setting `hdmi_edid_file` 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 on the forums.

hdmi_edid_filename

On the Raspberry Pi 4B, you can use the `hdmi_edid_filename` command to specify the filename of the EDID file to use, and also to specify which port the file is to be applied to. This also requires `hdmi_edid_file=1` to enable EDID files.

For example:

```
hdmi_edid_file=1
hdmi_edid_filename:0=FileForPortZero.edid
hdmi_edid_filename:1=FileForPortOne.edid
```

hdmi_force_edid_audio

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

hdmi_ignore_edid_audio

Setting `hdmi_ignore_edid_audio` 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 `hdmi_force_edid_3d` to `1` pretends that all CEA modes support 3D, even when the EDID does not indicate support for this.

hdmi_ignore_cec_init

Setting `hdmi_ignore_cec_init` to `1` will stop the initial active source message being sent during bootup. This prevents a CEC-enabled TV from coming out of standby and channel-switching when you are rebooting your Raspberry Pi.

hdmi_ignore_cec

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

cec_osd_name

The `cec_osd_name` command sets the initial CEC name of the device. The default is Raspberry Pi.

hdmi_pixel_encoding

The `hdmi_pixel_encoding` command forces the pixel encoding mode. By default, it will use the mode requested from the EDID, so you shouldn't need to change it.

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_max_pixel_freq

The pixel frequency is used by the firmware and KMS to filter HDMI modes. Note, this is not the same as the frame rate. It specifies the maximum frequency that a valid mode can have, thereby culling out higher frequency modes. So for example, if you wish to disable all 4K modes, you could specify a maximum frequency of 200000000, since all 4K modes have frequencies greater than this.

hdmi_blanking

The `hdmi_blanking` command controls what happens when the operating system asks for the display to be put into standby mode, using DPMS, to save power. If this option is not set or set to 0, the HDMI output is blanked but not switched off. In order to mimic the behaviour of other computers, you can set the HDMI output to switch off as well by setting this option to 1: the attached display will go into a low power standby mode.

NOTE

On the Raspberry Pi 4, setting `hdmi_blanking=1` will not cause the HDMI output to be switched off, since this feature has not yet been implemented. This feature may cause issues when using applications which don't use the framebuffer, such as `omxplayer`.

hdmi_blanking	result
0	HDMI output will be blanked
1	HDMI output will be switched off and blanked

hdmi_drive

The `hdmi_drive` command 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 minimum value is `0` and the maximum is `11`.

The default value for the original Model B and A is `2`. The default value for the Model B+ and all later models is `5`.

If you are seeing HDMI issues (speckling, interference) then try `7`. Very long HDMI cables may need up to `11`, but values this high should not be used unless absolutely necessary.

This option is ignored on the Raspberry Pi 4.

hdmi_group

The `hdmi_group` command 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

Together with `hdmi_group`, `hdmi_mode` defines the HDMI output format. Format mode numbers are derived from the CTA specification.

To set a custom display mode not listed here, see more information on the forums.

NOTE

> Not all modes are available on all models.

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

hdmi_mode	Resolution	Frequency	Screen Aspect	Notes
1	VGA (640x480)	60Hz	4:3	
2	480p	60Hz	4:3	
3	480p	60Hz	16:9	
4	720p	60Hz	16:9	
5	1080i	60Hz	16:9	
6	480i	60Hz	4:3	
7	480i	60Hz	16:9	
8	240p	60Hz	4:3	
9	240p	60Hz	16:9	

---

hdmi_mode	Resolution	Frequency	Screen Aspect	Notes
10	480i	60Hz	4:3	pixel quadrupling
11	480i	60Hz	16:9	pixel quadrupling
12	240p	60Hz	4:3	pixel quadrupling
13	240p	60Hz	16:9	pixel quadrupling
14	480p	60Hz	4:3	pixel doubling
15	480p	60Hz	16:9	pixel doubling
16	1080p	60Hz	16:9	
17	576p	50Hz	4:3	
18	576p	50Hz	16:9	
19	720p	50Hz	16:9	
20	1080i	50Hz	16:9	
21	576i	50Hz	4:3	
22	576i	50Hz	16:9	
23	288p	50Hz	4:3	
24	288p	50Hz	16:9	
25	576i	50Hz	4:3	pixel quadrupling
26	576i	50Hz	16:9	pixel quadrupling
27	288p	50Hz	4:3	pixel quadrupling
28	288p	50Hz	16:9	pixel quadrupling
29	576p	50Hz	4:3	pixel doubling
30	576p	50Hz	16:9	pixel doubling
31	1080p	50Hz	16:9	
32	1080p	24Hz	16:9	
33	1080p	25Hz	16:9	
34	1080p	30Hz	16:9	

hdmi_mode	Resolution	Frequency	Screen Aspect	Notes
35	480p	60Hz	4:3	pixel quadrupling
36	480p	60Hz	16:9	pixel quadrupling
37	576p	50Hz	4:3	pixel quadrupling
38	576p	50Hz	16:9	pixel quadrupling
39	1080i	50Hz	16:9	reduced blanking
40	1080i	100Hz	16:9	
41	720p	100Hz	16:9	
42	576p	100Hz	4:3	
43	576p	100Hz	16:9	
44	576i	100Hz	4:3	
45	576i	100Hz	16:9	
46	1080i	120Hz	16:9	
47	720p	120Hz	16:9	
48	480p	120Hz	4:3	
49	480p	120Hz	16:9	
50	480i	120Hz	4:3	
51	480i	120Hz	16:9	
52	576p	200Hz	4:3	
53	576p	200Hz	16:9	
54	576i	200Hz	4:3	
55	576i	200Hz	16:9	
56	480p	240Hz	4:3	
57	480p	240Hz	16:9	
58	480i	240Hz	4:3	
59	480i	240Hz	16:9	
60	720p	24Hz	16:9	
61	720p	25Hz	16:9	

hdmi_mode	Resolution	Frequency	Screen Aspect	Notes
62	720p	30Hz	16:9	
63	1080p	120Hz	16:9	
64	1080p	100Hz	16:9	
65	Custom			
66	720p	25Hz	64:27	Pi 4
67	720p	30Hz	64:27	Pi 4
68	720p	50Hz	64:27	Pi 4
69	720p	60Hz	64:27	Pi 4
70	720p	100Hz	64:27	Pi 4
71	720p	120Hz	64:27	Pi 4
72	1080p	24Hz	64:27	Pi 4
73	1080p	25Hz	64:27	Pi 4
74	1080p	30Hz	64:27	Pi 4
75	1080p	50Hz	64:27	Pi 4
76	1080p	60Hz	64:27	Pi 4
77	1080p	100Hz	64:27	Pi 4
78	1080p	120Hz	64:27	Pi 4
79	1680x720	24Hz	64:27	Pi 4
80	1680x720	25z	64:27	Pi 4
81	1680x720	30Hz	64:27	Pi 4
82	1680x720	50Hz	64:27	Pi 4
83	1680x720	60Hz	64:27	Pi 4
84	1680x720	100Hz	64:27	Pi 4
85	1680x720	120Hz	64:27	Pi 4
86	2560x720	24Hz	64:27	Pi 4
87	2560x720	25Hz	64:27	Pi 4
88	2560x720	30Hz	64:27	Pi 4
89	2560x720	50Hz	64:27	Pi 4
90	2560x720	60Hz	64:27	Pi 4
91	2560x720	100Hz	64:27	Pi 4

hdmi_mode	Resolution	Frequency	Screen Aspect	Notes
92	2560x720	120Hz	64:27	Pi 4
93	2160p	24Hz	16:9	Pi 4
94	2160p	25Hz	16:9	Pi 4
95	2160p	30Hz	16:9	Pi 4
96	2160p	50Hz	16:9	Pi 4
97	2160p	60Hz	16:9	Pi 4
98	4096x2160	24Hz	256:135	Pi 4
99	4096x2160	25Hz	256:135	Pi 4
100	4096x2160	30Hz	256:135	Pi 4
101	4096x2160	50Hz	256:135	Pi 4[1]
102	4096x2160	60Hz	256:135	Pi 4[1]
103	2160p	24Hz	64:27	Pi 4
104	2160p	25Hz	64:27	Pi 4
105	2160p	30Hz	64:27	Pi 4
106	2160p	50Hz	64:27	Pi 4
107	2160p	60Hz	64:27	Pi 4

1. Only available with an overclocked core frequency: set `core_freq_min=600` and `core_freq=600`. See overclocking.

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	Screen Aspect	Notes
1	640x350	85Hz		
2	640x400	85Hz	16:10	
3	720x400	85Hz		
4	640x480	60Hz	4:3	
5	640x480	72Hz	4:3	
6	640x480	75Hz	4:3	
7	640x480	85Hz	4:3	
8	800x600	56Hz	4:3	

hdmi_mode	Resolution	Frequency	Screen Aspect	Notes
9	800x600	60Hz	4:3	
10	800x600	72Hz	4:3	
11	800x600	75Hz	4:3	
12	800x600	85Hz	4:3	
13	800x600	120Hz	4:3	
14	848x480	60Hz	16:9	
15	1024x768	43Hz	4:3	incompatible with the Raspberry Pi
16	1024x768	60Hz	4:3	
17	1024x768	70Hz	4:3	
18	1024x768	75Hz	4:3	
19	1024x768	85Hz	4:3	
20	1024x768	120Hz	4:3	
21	1152x864	75Hz	4:3	
22	1280x768	60Hz	15:9	reduced blanking
23	1280x768	60Hz	15:9	
24	1280x768	75Hz	15:9	
25	1280x768	85Hz	15:9	
26	1280x768	120Hz	15:9	reduced blanking
27	1280x800	60	16:10	reduced blanking
28	1280x800	60Hz	16:10	
29	1280x800	75Hz	16:10	
30	1280x800	85Hz	16:10	
31	1280x800	120Hz	16:10	reduced blanking
32	1280x960	60Hz	4:3	
33	1280x960	85Hz	4:3	
34	1280x960	120Hz	4:3	reduced blanking

hdmi_mode	Resolution	Frequency	Screen Aspect	Notes
35	1280x1024	60Hz	5:4	
36	1280x1024	75Hz	5:4	
37	1280x1024	85Hz	5:4	
38	1280x1024	120Hz	5:4	reduced blanking
39	1360x768	60Hz	16:9	
40	1360x768	120Hz	16:9	reduced blanking
41	1400x1050	60Hz	4:3	reduced blanking
42	1400x1050	60Hz	4:3	
43	1400x1050	75Hz	4:3	
44	1400x1050	85Hz	4:3	
45	1400x1050	120Hz	4:3	reduced blanking
46	1440x900	60Hz	16:10	reduced blanking
47	1440x900	60Hz	16:10	
48	1440x900	75Hz	16:10	
49	1440x900	85Hz	16:10	
50	1440x900	120Hz	16:10	reduced blanking
51	1600x1200	60Hz	4:3	
52	1600x1200	65Hz	4:3	
53	1600x1200	70Hz	4:3	
54	1600x1200	75Hz	4:3	
55	1600x1200	85Hz	4:3	
56	1600x1200	120Hz	4:3	reduced blanking
57	1680x1050	60Hz	16:10	reduced blanking
58	1680x1050	60Hz	16:10	
59	1680x1050	75Hz	16:10	

hdmi_mode	Resolution	Frequency	Screen Aspect	Notes
60	1680x1050	85Hz	16:10	
61	1680x1050	120Hz	16:10	reduced blanking
62	1792x1344	60Hz	4:3	
63	1792x1344	75Hz	4:3	
64	1792x1344	120Hz	4:3	reduced blanking
65	1856x1392	60Hz	4:3	
66	1856x1392	75Hz	4:3	
67	1856x1392	120Hz	4:3	reduced blanking
68	1920x1200	60Hz	16:10	reduced blanking
69	1920x1200	60Hz	16:10	
70	1920x1200	75Hz	16:10	
71	1920x1200	85Hz	16:10	
72	1920x1200	120Hz	16:10	reduced blanking
73	1920x1440	60Hz	4:3	
74	1920x1440	75Hz	4:3	
75	1920x1440	120Hz	4:3	reduced blanking
76	2560x1600	60Hz	16:10	reduced blanking
77	2560x1600	60Hz	16:10	
78	2560x1600	75Hz	16:10	
79	2560x1600	85Hz	16:10	
80	2560x1600	120Hz	16:10	reduced blanking
81	1366x768	60Hz	16:9	NOT on Raspberry Pi 4
82	1920x1080	60Hz	16:9	1080p
83	1600x900	60Hz	16:9	reduced blanking

hdmi_mode	Resolution	Frequency	Screen Aspect	Notes
84	2048x1152	60Hz	16:9	reduced blanking
85	1280x720	60Hz	16:9	720p
86	1366x768	60Hz	16:9	reduced blanking

NOTE

There is a pixel clock limit. The highest supported mode on models prior to the Raspberry Pi 4 is 1920x1200 at 60Hz with reduced blanking, whilst the Raspberry Pi 4 can support up to 4096x2160 (known as 4k) at 60Hz. Also note that if you are using both HDMI ports of the Raspberry Pi 4 for 4k output, then you are limited to 30Hz on both.

`hdmi_timings`

This allows setting of raw HDMI timing values for a custom mode, selected using `hdmi_group=2` and `hdmi_mode=87`.

```
hdmi_timings=<h_active_pixels> <h_sync_polarity> <h_front_porch> <h_sync_pulse>
<h_back_porch> <v_active_lines> <v_sync_polarity> <v_front_porch> <v_sync_pulse
> <v_back_porch> <v_sync_offset_a> <v_sync_offset_b> <pixel_rep> <frame_rate> <
interlaced> <pixel_freq> <aspect_ratio>
```

```
<h_active_pixels> = horizontal pixels (width)
<h_sync_polarity> = invert hsync polarity
<h_front_porch>   = horizontal forward padding from DE active edge
<h_sync_pulse>    = hsync pulse width in pixel clocks
<h_back_porch>    = vertical back padding from DE active edge
<v_active_lines>  = vertical pixels height (lines)
<v_sync_polarity> = invert vsync polarity
<v_front_porch>   = vertical forward padding from DE active edge
<v_sync_pulse>    = vsync pulse width in pixel clocks
<v_back_porch>    = vertical back padding from DE active edge
<v_sync_offset_a> = leave at zero
<v_sync_offset_b> = leave at zero
<pixel_rep>       = leave at zero
<frame_rate>      = screen refresh rate in Hz
<interlaced>      = leave at zero
<pixel_freq>      = clock frequency (width*height*framerate)
<aspect_ratio>    = *
```

* The aspect ratio can be set to one of eight values (choose the closest for your screen):

```
HDMI_ASPECT_4_3 = 1
HDMI_ASPECT_14_9 = 2
HDMI_ASPECT_16_9 = 3
HDMI_ASPECT_5_4 = 4
HDMI_ASPECT_16_10 = 5
HDMI_ASPECT_15_9 = 6
```

```
HDMI_ASPECT_21_9 = 7
HDMI_ASPECT_64_27 = 8
```

`hdmi_force_mode`

Setting to `1` will remove all other modes except the ones specified by `hdmi_mode` and `hdmi_group` from the internal list, meaning they will not appear in any enumerated lists of modes. This option may help if a display seems to be ignoring the `hdmi_mode` and `hdmi_group` settings.

`edid_content_type`

Forces the EDID content type to a specific value.

The options are:

- `0` = `EDID_ContentType_NODATA`, content type none.
- `1` = `EDID_ContentType_Graphics`, content type graphics, ITC must be set to 1
- `2` = `EDID_ContentType_Photo`, content type photo
- `3` = `EDID_ContentType_Cinema`, content type cinema
- `4` = `EDID_ContentType_Game`, content type game

Which Values are Valid for my Monitor?

Your HDMI monitor may only support 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 Raspberry Pi use this by default, you must add some additional settings. For example, the following selects an 800 × 480 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.

Composite Video Mode

The table below describes where composite video output can be found on each model of Raspberry Pi computer:

model	composite output
Raspberry Pi 1 A and B	RCA jack
Raspberry Pi Zero	Unpopulated TV header

---

model	composite output
Raspberry Pi Zero 2 W	Test pads on underside of board
All other models	3.5mm AV jack

NOTE

Composite video output is not available on the Raspberry Pi 400.

`sdtv_mode`

The `sdtv_mode` command defines the TV standard used for composite video output:

sdtv_mode	result
0 (default)	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
16	Progressive scan NTSC
18	Progressive scan PAL

`sdtv_aspect`

The `sdtv_aspect` command 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 `sdtv_disable_colourburst` to `1` disables colourburst on composite video output. The picture will be displayed in monochrome, but it may appear sharper.

`enable_tvout`

Set to `1` to enable composite video output, or `0` to disable. On Raspberry Pi 4, composite output is only available if you set this to `1`. Composite output is not available on the Raspberry Pi 400.

On all models except Raspberry Pi 4 and Raspberry Pi 400, composite output will be enabled if HDMI output is disabled. HDMI output is disabled when no HDMI display is detected, or `hdmi_ignore_hotplug=1` is set. Set `enable_tvout=0` to prevent composite being enabled when HDMI is disabled.

Model	Default
Pi 4 and 400	0
All other models	1

LCD Displays and Touchscreens

`ignore_lcd`

By default the Raspberry Pi Touch Display is used when it is detected on the I2C bus. `ignore_lcd=1` will skip this detection phase, and therefore the LCD display will not be used.

`display_default_lcd`

If a Raspberry Pi Touch Display is detected it will be used as the default display and will show the framebuffer. Setting `display_default_lcd=0` will ensure the LCD is not the default display, which usually implies the HDMI output will be the default. The LCD can still be used by choosing its display number from supported applications, for example, omxplayer.

`lcd_framerate`

Specify the framerate of the Raspberry Pi Touch Display, in Hertz/fps. Defaults to 60Hz.

`lcd_rotate`

This flips the display using the LCD's inbuilt flip functionality, which is a cheaper operation that using the GPU-based rotate operation.

For example, `lcd_rotate=2` will compensate for an upside down display.

`disable_touchscreen`

Enable/disable the touchscreen.

`disable_touchscreen=1` will disable the touchscreen on the official Raspberry Pi Touch Display.

`enable_dpi_lcd`

Enable LCD displays attached to the DPI GPIOs. This is to allow the use of third-party LCD displays using the parallel display interface.

`dpi_group`, `dpi_mode`, `dpi_output_format`

The `dpi_group` and `dpi_mode` `config.txt` parameters are used to set either predetermined modes (DMT or CEA modes as used by HDMI above). A user can generate custom modes in much the same way as for HDMI (see `dpi_timings` section).

`dpi_output_format` is a bitmask specifying various parameters used to set up the display format.

`dpi_timings`

This allows setting of raw DPI timing values for a custom mode, selected using `dpi_group=2` and `dpi_mode=87`.

```
dpi_timings=<h_active_pixels> <h_sync_polarity> <h_front_porch> <h_sync_pulse>
 <h_back_porch> <v_active_lines> <v_sync_polarity> <v_front_porch> <v_sync_puls
e> <v_back_porch> <v_sync_offset_a> <v_sync_offset_b> <pixel_rep> <frame_rate>
 <interlaced> <pixel_freq> <aspect_ratio>
```

```
<h_active_pixels> = horizontal pixels (width)
<h_sync_polarity> = invert hsync polarity
<h_front_porch>   = horizontal forward padding from DE active edge
<h_sync_pulse>    = hsync pulse width in pixel clocks
<h_back_porch>    = vertical back padding from DE active edge
<v_active_lines>  = vertical pixels height (lines)
<v_sync_polarity> = invert vsync polarity
<v_front_porch>   = vertical forward padding from DE active edge
<v_sync_pulse>    = vsync pulse width in pixel clocks
<v_back_porch>    = vertical back padding from DE active edge
<v_sync_offset_a> = leave at zero
<v_sync_offset_b> = leave at zero
<pixel_rep>       = leave at zero
<frame_rate>      = screen refresh rate in Hz
<interlaced>      = leave at zero
<pixel_freq>      = clock frequency (width*height*framerate)
<aspect_ratio>    = *
```

* The aspect ratio can be set to one of eight values (choose the closest for your screen):

```
HDMI_ASPECT_4_3 = 1
HDMI_ASPECT_14_9 = 2
HDMI_ASPECT_16_9 = 3
```

```
HDMI_ASPECT_5_4 = 4
HDMI_ASPECT_16_10 = 5
HDMI_ASPECT_15_9 = 6
HDMI_ASPECT_21_9 = 7
HDMI_ASPECT_64_27 = 8
```

Generic Display Options

`hdmi_force_hotplug`

Setting `hdmi_force_hotplug` 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 `hdmi_ignore_hotplug` to `1` pretends that the HDMI hotplug signal is not asserted, so it appears that a HDMI display is not attached. HDMI output will therefore be disabled, even if a monitor is connected.

`overscan_left`

The `overscan_left` command specifies the number of pixels to add to the firmware default value of overscan on the left edge of the screen. The default value is `0`.

Increase this value if the text flows off the left edge of the screen; decrease it if there is a black border between the left edge of the screen and the text.

`overscan_right`

The `overscan_right` command specifies the number of pixels to add to the firmware default value of overscan on the right edge of the screen. The default value is `0`.

Increase this value if the text flows off the right edge of the screen; decrease it if there is a black border between the right edge of the screen and the text.

`overscan_top`

The `overscan_top` command specifies the number of pixels to add to the firmware default value of overscan on the top edge of the screen. The default value is `0`.

Increase this value if the text flows off the top edge of the screen; decrease it if there is a black border between the top edge of the screen and the text.

overscan_bottom

The `overscan_bottom` command specifies the number of pixels to add to the firmware default value of overscan on the bottom edge of the screen. The default value is `0`.

Increase this value if the text flows off the bottom edge of the screen; decrease it if there is a black border between the bottom edge of the screen and the text.

overscan_scale

Set `overscan_scale` to `1` to force any non-framebuffer layers to conform to the overscan settings. The default value is `0`.

NOTE: this feature is generally not recommended: it can reduce image quality because all layers on the display will be scaled by the GPU. Disabling overscan on the display itself is the recommended option to avoid images being scaled twice (by the GPU and the display).

framebuffer_width

The `framebuffer_width` command specifies the console framebuffer width in pixels. The default is the display width minus the total horizontal overscan.

framebuffer_height

The `framebuffer_height` command specifies the console framebuffer height in pixels. The default is the display height minus the total vertical overscan.

max_framebuffer_height, max_framebuffer_width

Specifies the maximum dimensions that the internal frame buffer is allowed to be.

framebuffer_depth

Use `framebuffer_depth` to specify the console framebuffer depth in bits per pixel. The default value is `16`.

framebuffer_depth	result	notes
8	8-bit framebuffer	Default RGB palette makes screen unreadable
16	16-bit framebuffer	
24	24-bit framebuffer	May result in a corrupted display

framebuffer_depth	result	notes
32	32-bit framebuffer	May need to be used in conjunction with `framebuffer_ignore_alpha=1`

framebuffer_ignore_alpha

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

framebuffer_priority

In a system with multiple displays, using the legacy (pre-KMS) graphics driver, this forces a specific internal display device to be the first Linux framebuffer (i.e. `/dev/fb0`).

The options that can be set are:

Display	ID
Main LCD	0
Secondary LCD	1
HDMI 0	2
Composite	3
HDMI 1	7

max_framebuffers

This configuration entry sets the maximum number of firmware framebuffers that can be created. Valid options are 0, 1, and 2. By default on devices before the Raspberry Pi 4 this is set to 1, so will need to be increased to 2 when using more than one display, for example HDMI and a DSI or DPI display. The Raspberry Pi 4 configuration sets this to 2 by default as it has two HDMI ports.

Generally in most cases it is safe to set this to 2, as framebuffers will only be created when an attached device is actually detected.

Setting this value to 0 can be used to reduce memory requirements when used in headless mode as it will prevent any framebuffers from being allocated.

test_mode

The `test_mode` command displays a test image and sound during boot (over the composite video and analogue audio outputs only) 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_hdmi_rotate`

Use `display_hdmi_rotate` to rotate or flip the HDMI display orientation. The default value is `0`.

display_hdmi_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 will not work with the 16MB GPU split.

If using the VC4 FKMS V3D driver (this is the default on the Raspberry Pi 4), then 90 and 270 degree rotations are not supported. The Screen Configuration utility provides display rotations for this driver.

`display_lcd_rotate`

For the legacy graphics driver (default on models prior to the Raspberry Pi 4), use `display_lcd_rotate` to rotate or flip the LCD orientation. Parameters are the same as `display_hdmi_rotate`. See also `lcd_rotate`.

`display_rotate`

`display_rotate` is deprecated in the latest firmware but has been retained for backwards compatibility. Please use `display_lcd_rotate` and `display_hdmi_rotate` instead.

Use `display_rotate` to rotate or flip the screen orientation. Parameters are the same as `display_hdmi_rotate`.

`disable_fw_kms_setup`

By default, the firmware parses the EDID of any HDMI attached display, picks an appropriate video mode, then passes the resolution and frame rate of the mode, along with overscan parameters, to the Linux kernel via settings on the kernel command line. In rare

circumstances, this can have the effect of choosing a mode that is not in the EDID, and may be incompatible with the device. You can use `disable_fw_kms_setup=1` to disable the passing of these parameters and avoid this problem. The Linux video mode system (KMS) will then parse the EDID itself and pick an appropriate mode.

Other Options

`dispmanx_offline`

Forces `dispmanx` composition to be done offline in two offscreen framebuffers. This can allow more `dispmanx` elements to be composited, but is slower and may limit screen framerate to typically 30fps.

Raspberry Pi 4 HDMI Pipeline

Edit this on GitHub

IMPORTANT

> When using the VC4 KMS graphics driver, the complete display pipeline is managed by Linux - this includes the HDMI outputs. These settings only apply to the legacy FKMS and firmware-based graphics driver.

In order to support dual displays, and modes up to 4k60, the Raspberry Pi 4 has updated the HDMI composition pipeline hardware in a number of ways. One of the major changes is that it generates 2 output pixels for every clock cycle.

Every HDMI mode has a list of timings that control all the parameters around sync pulse durations. These are typically defined via a pixel clock, and then a number of active pixels, a front porch, sync pulse, and back porch for each of the horizontal and vertical directions.

Running everything at 2 pixels per clock means that the Raspberry Pi 4 can not support a timing where *any* of the horizontal timings are not divisible by 2. The firmware and Linux kernel will filter out any mode that does not fulfill this criteria.

There is only one mode in the CEA and DMT standards that falls into this category - DMT mode 81, which is 1366x768 @ 60Hz. This mode has odd values for the horizontal sync and back porch timings. It's also an unusual mode for having a width that isn't divisible by 8.

If your monitor is of this resolution, then the Raspberry Pi 4 will automatically drop down to the next mode that is advertised by the monitor; this is typically 1280x720.

On some monitors it is possible to configure them to use 1360x768 @ 60Hz. They typically do not advertise this mode via their EDID so the selection can't be made automatically, but it can be manually chosen by adding

```
hdmi_group=2
hdmi_mode=87
hdmi_cvt=1360 768 60
```

to config.txt.

Timings specified manually via a `hdmi_timings=` line in `config.txt` will also need to comply with the restriction of all horizontal timing parameters being divisible by 2.

`dpi_timings=` are not restricted in the same way as that pipeline still only runs at a single pixel per clock cycle.

Camera Settings

Edit this on GitHub

disable_camera_led

Setting `disable_camera_led` to `1` prevents the red camera LED from turning on when recording video or taking a still picture. This is useful for preventing reflections when the camera is facing a window, for example.

awb_auto_is_greyworld

Setting `awb_auto_is_greyworld` to `1` allows libraries or applications that do not support the greyworld option internally to capture valid images and videos with NoIR cameras. It switches "auto" awb mode to use the "greyworld" algorithm. This should only be needed for NoIR cameras, or when the High Quality camera has had its IR filter removed.

Miscellaneous Options

Edit this on GitHub

avoid_warnings

The warning symbols can be disabled using this option, although this is not advised.

`avoid_warnings=1` disables the warning overlays. `avoid_warnings=2` disables the warning overlays, but additionally allows turbo mode even when low-voltage is present.

logging_level

Sets the VideoCore logging level. The value is a VideoCore-specific bitmask.

`max_usb_current`

WARNING

> This command is now deprecated and has no effect.

Originally certain models of Raspberry Pi limited the USB ports to a maximum of 600mA. Setting `max_usb_current=1` changed this default to 1200mA. However, all firmware now has this flag set by default, so it is no longer necessary to use this option.