config.txt

What is config.txt?

Edit this on GitHub

Instead of the BIOS found on a conventional PC, Raspberry Pi devices use a configuration file called `config.txt`. The GPU reads `config.txt` before the Arm CPU and Linux initialise. Raspberry Pi OS looks for this file in the **boot partition**, located at `/boot/firmware/`.

NOTE

Prior to Raspberry Pi OS *Bookworm*, Raspberry Pi OS stored the boot partition at `/boot/`.

You can edit `config.txt` directly from your Raspberry Pi OS installation. You can also remove the storage device and edit files in the boot partition, including `config.txt`, from a separate computer.

Changes to `config.txt` only take effect after a reboot. You can view the current active settings using 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 non-zero integer config options (non-zero)

`vcgencmd get_config str`
: lists all non-null string config options

NOTE

Not all config settings can be retrieved using `vcgencmd`.

File format

The `config.txt` file is read by the early-stage boot firmware, so it uses a very simple file format: **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 for entries. Raspberry Pi OS ignores any characters past this limit.

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
```

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.

NOTE

The `bootcode.bin` or EEPROM bootloaders do not support the `include` directive. Settings which are handled by the bootloader will only take effect if they are specified in `config.txt` (rather than any additional included file):

- `bootcode_delay`,
- `gpu_mem`, `gpu_mem_256`, `gpu_mem_512`, `gpu_mem_1024`,
- `total_mem`,
- `sdram_freq`,
- `start_x`, `start_debug`, `start_file`, `fixup_file`,
- `uart_2ndstage`.

Conditional filtering

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 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 a parameter to the `reboot` command (e.g. `sudo reboot 2`).

Partition numbers start at `1` and the MBR partitions are `1` to `4`. Specifying partition `0` means boot from the `default` partition which is the first bootable FAT partition.

Bootable partitions must be formatted as FAT12, FAT16 or FAT32 and contain a `start.elf` file (or `config.txt` file on Raspberry Pi 5) in order to be classed as be bootable by the bootloader.

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.

Example update flow for A/B booting

The following pseudo-code shows how a hypothetical OS `Update service` could use `tryboot` in `autoboot.txt` to perform a fail-safe OS upgrade.

Initial `autoboot.txt`:

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

Installing the update

- System is powered on and boots to partition 2 by default
- An `Update service` downloads the next version of the OS to partition 3
- The update is tested by rebooting to `tryboot` mode `reboot "0 tryboot"` where `0` means the default partition

Committing or cancelling the update

- System boots from partition 3 because the `[tryboot]` filter evaluates to true in `tryboot mode`
- If tryboot is active (`/proc/device-tree/chosen/bootloader/tryboot == 1`)
    - If the current boot partition (`/proc/device-tree/chosen/bootloader/partition`) matches the `boot_partition` in the `[tryboot]` section of `autoboot.txt`
        - The `Update Service` validates the system to verify that the update was successful
        - If the update was successful
            - Replace `autoboot.txt` swapping the `boot_partition` configuration
            - Normal reboot - partition 3 is now the default boot partition
        - Else
            - `Update Service` marks the update as failed e.g. it removes the update files.
            - Normal reboot - partition 2 is still the default boot partition because the `tryboot` flag is automatically cleared
        - End if
    - End If
- End If

Updated `autoboot.txt`:

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

NOTE

> It's not mandatory to reboot after updating `autoboot.txt`. However, the `Update Service` must be careful to avoid overwriting the current partition since `autoboot.txt` has already been modified to commit the last update. For more information, see Device Tree parameters.

Common options

Common display options

`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. Raspberry Pi 4 does not support 4Kp60 output on both micro HDMI ports simultaneously. Setting `hdmi_enable_4kp60` increases power consumption and temperature.

Common hardware configuration options

`camera_auto_detect`

With this setting enabled (in Raspberry Pi OS it is enabled by default), the firmware will automatically load overlays for CSI cameras that it recognises. Set `camera_auto_detect=0` to disable the setting.

`display_auto_detect`

With this setting enabled (it is enabled by default in Raspberry Pi OS), the firmware will automatically load overlays for DSI displays that it recognises. Set `display_auto_detect=0` to disable.

`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=` - the option 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 keep in mind the line length limit of 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 is enabled by default in Raspberry Pi OS. Set `arm_boost=0` to disable.

`power_force_3v3_pwm` (Raspberry Pi 5 Only)

Forces PWM when using a 3V3 power supply. Set `power_force_3v3_pwm=0` to disable.

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 on some models.

`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.

HDMI audio

By default, HDMI audio output is enabled on all Raspberry Pi models with HDMI output.

To disable HDMI audio output, append `,noaudio` to the end of the `dtoverlay=vc4-kms-v3d` line in `/boot/firmware/config.txt`:

```
dtoverlay=vc4-kms-v3d,noaudio
```

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.

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.

NOTE

Cut-down firmware (`start*cd.elf` and `fixup*cd.dat`) cannot be selected this way - the system will fail to boot. The only way to enable the cut-down firmware is to specify `gpu_mem=16`. The cut-down firmware removes support for codecs, 3D and debug logging as well as limiting the initial early-boot framebuffer to 1080p @16bpp - although KMS can replace this with up to 32bpp 4K framebuffer(s) at a later stage as with any firmware.

NOTE

The Raspberry Pi 5 firmware is self-contained in the bootloader EEPROM.

`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 for 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 `kernel8.img`, or `kernel7l.img` if `arm_64bit` is set to 0.

The Raspberry Pi 5 firmware defaults to loading `kernel_2712.img` because this image contains optimisations specific to Raspberry Pi 5 (e.g. 16K page-size). If this file is not present, then the common 64-bit kernel (`kernel8.img`) will be loaded instead.

arm_64bit

If set to 1, the kernel will be started in 64-bit mode. Setting to 0 selects 32-bit mode.

In 64-bit mode, the firmware will choose an appropriate kernel (e.g. `kernel8.img`), unless there is an explicit `kernel` option defined, in which case that is used instead.

Defaults to 1 on Pi 4s (Pi 4B, Pi 400, CM4 and CM4S), and 0 on all other platforms. However, if the name given in an explicit `kernel` option matches one of the known kernels then `arm_64bit` will be set accordingly.

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). 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. Raspberry Pi 5 only supports a 64-bit kernel, so this parameter has been removed for that device.

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 the `=` character here.

auto_initramfs

If `auto_initramfs` is set to `1`, look for an `initramfs` file using the same rules as the kernel selection.

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, this is a helpful way 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_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 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, and 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).

Configuration Properties

Raspberry Pi 5 requires a `config.txt` file to be present to indicate that the partition is bootable.

`boot_ramdisk`

If this property is set to `1` then the bootloader will attempt load a ramdisk file called `boot.img` containing the boot filesystem. Subsequent files (e.g. `start4.elf`) are read from the ramdisk instead of the original boot file system.

The primary purpose of `boot_ramdisk` is to support `secure-boot`, however, unsigned `boot.img` files can also be useful to Network Boot or `RPIBOOT` configurations.

- The maximum size for a ramdisk file is 96MB.
- `boot.img` files are raw disk `.img` files. The recommended format is a plain FAT32 partition with no MBR.
- The memory for the ramdisk filesystem is released before the operating system is started.
- If TRYBOOT is selected then the bootloader will search for `tryboot.img` instead of `boot.img`.
- See also autoboot.txt.

For more information about `secure-boot` and creating `boot.img` files please see USBBOOT.

Default: `0`

`boot_load_flags`

Experimental property for custom firmware (bare metal).

Bit 0 (0x1) indicates that the .elf file is custom firmware. This disables any compatibility checks (e.g. is USB MSD boot supported) and resets PCIe before starting the executable.

Not relevant on Raspberry Pi 5 because there is no `start.elf` file.

Default: `0x0`

`pciex4_reset`

Raspberry Pi 5 only.

By default, the PCIe x4 controller used by `RP1` is reset before starting the operating system. If this parameter is set to `0` then the reset is disabled allowing operating system or bare metal code to inherit the PCIe configuration setup from the bootloader.

Default: `1`

`uart_2ndstage`

If `uart_2ndstage` is `1` then enable debug logging to the UART. This option also automatically enables UART logging in `start.elf`. This is also described on the Boot options page.

The `BOOT_UART` property also enables bootloader UART logging but does not enable UART logging in `start.elf` unless `uart_2ndstage=1` is also set.

Default: `0`

`erase_eeprom`

If `erase_eeprom` is set to `1` then `recovery.bin` will erase the entire SPI EEPROM instead of flashing the bootloader image. This property has no effect during a normal boot.

Default: `0`

`eeprom_write_protect`

Configures the EEPROM `write status register`. This can be set either to mark the entire EEPROM as write-protected, or to clear write-protection.

This option must be used in conjunction with the EEPROM `/WP` pin which controls updates to the EEPROM `Write Status Register`. Pulling `/WP` low (CM4 `EEPROM_nWP` or on a Raspberry Pi 4 `TP5`) does NOT write-protect the EEPROM unless the `Write Status Register` has also been configured.

See the Winbond W25x40cl or Winbond W25Q16JV datasheets for further details.

`eeprom_write_protect` settings in `config.txt` for `recovery.bin`.

Value	Description
1	Configures the write protect regions to cover the entire EEPROM.
0	Clears the write protect regions.
-1	Do nothing.

NOTE

`flashrom` does not support clearing of the write-protect regions and will fail to update

the EEPROM if write-protect regions are defined.

On Raspberry Pi 5 `/WP` is pulled low by default and consequently write-protect is enabled as soon as the `Write Status Register` is configured. To clear write-protect pull `/WP` high by connecting `TP14` and `TP1`.

Default: `-1`

`os_check`

On Raspberry Pi 5 the firmware automatically checks for a compatible Device Tree file before attempting to boot from the current partition. Otherwise, older non-compatible kernels would be loaded and then hang. To disable this check (e.g. for bare-metal development), set `os_check=0` in config.txt

Default: `1`

`bootloader_update`

This option may be set to 0 to block self-update without requiring the EEPROM configuration to be updated. This is sometimes useful when updating multiple Raspberry Pis via network boot because this option can be controlled per Raspberry Pi (e.g. via a serial number filter in `config.txt`).

Default: `1`

Secure Boot configuration properties

WHITE PAPER

How to use Raspberry Pi Secure Boot

This whitepaper describes how to implement secure boot on devices based on Raspberry Pi 4. For an overview of our approach to implementing secure boot implementation, please see the Raspberry Pi 4 Boot Security whitepaper. The secure boot system is intended for use with `buildroot`-based OS images; using it with Raspberry Pi OS is not recommended or supported.

The following `config.txt` properties are used to program the `secure-boot` OTP settings. These changes are irreversible and can only be programmed via `RPIBOOT` when flashing the bootloader EEPROM image. This ensures that `secure-boot` cannot be set remotely or by accidentally inserting a stale SD card image.

For more information about enabling `secure-boot` please see the Secure Boot readme and the Secure Boot tutorial in the USBBOOT repo.

program_pubkey

If this property is set to `1` then `recovery.bin` will write the hash of the public key in the EEPROM image to OTP. Once set, the bootloader will reject EEPROM images signed with different RSA keys or unsigned images.

Default: `0`

revoke_devkey

If this property is set to `1` then `recovery.bin` will write a value to OTP that prevents the ROM from loading old versions of the second stage bootloader which do not support `secure-boot`. This prevents `secure-boot` from being turned off by reverting to an older release of the bootloader.

Default: `0`

program_rpiboot_gpio

Since there is no dedicated `nRPIBOOT` jumper on Raspberry Pi 4B or Raspberry Pi 400, an alternative GPIO must be used to select `RPIBOOT` mode by pulling the GPIO low. Select a single GPIO from the following options:

- `2`
- `4`
- `5`
- `6`
- `7`
- `8`

This property does not depend on `secure-boot`, but verify that this GPIO configuration does not conflict with any HATs which might pull the GPIO low during boot.

Since for safety this property can only be programmed via `RPIBOOT`, the bootloader EEPROM must first be cleared using `erase_eeprom`. This causes the BCM2711 ROM to failover to `RPIBOOT` mode, which then allows this option to be set.

Default:

program_jtag_lock

If this property is set to `1` then `recovery.bin` will program an OTP value that prevents VideoCore JTAG from being used. This option requires that `program_pubkey` and `revoke_devkey` are also set. This option can prevent failure analysis, and should only be set after the device has been fully tested.

Default: `0`

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, addressing 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 apply 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 "[…]" conditional filters 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 `pinctrl`.

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.

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 the 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 newer. 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. This makes it possible to detect that your Raspberry Pi was once overclocked. The overclock bit sets when `force_turbo` is set to `1` and any of the `over_voltage_*` options are set to a value of more than `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; and there is a small benefit for SDRAM on Raspberry Pi 2 and Raspberry Pi 3. See section below for use on Raspberry Pi 4.
`h264_freq`	Frequency of the hardware video block in MHz; individual override of the `gpu_freq` setting.
`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. On Raspberry Pi 5, V3D is independent of

Option	Description
	`core_freq`, `isp_freq` and `hevc_freq`.
`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 4 or newer is not 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 the 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. Not supported on Raspberry Pi 4 or later devices.
`over_voltage_sdram_i`	SDRAM I/O voltage adjustment. [-16,8] equates to [0.8V,1.4V] with 0.025V steps. Not supported on Raspberry Pi 4 or later devices.
`over_voltage_sdram_p`	SDRAM phy voltage adjustment. [-16,8] equates to [0.8V,1.4V] with 0.025V steps. Not supported on Raspberry Pi 4 or later devices.
`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`.
`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

Option	Description
	8 will give a minimum of 1.4V. This setting is deprecated on Raspberry Pi 4 and Raspberry Pi 5.
`over_voltage_delta`	On Raspberry Pi 4 and Raspberry Pi 5 the over_voltage_delta parameter adds the given offset in microvolts to the number calculated by the DVFS algorithm.
`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	Pi 5
`arm_freq`	1000	700	900	1200	1400	1500	1500 or 1800 if `arm_boost`=1	1800	1000	2400
`core_freq`	400	250	250	400	400	500	500	500	400	910
`h264_freq`	300	250	250	400	400	500	500	500	300	N/A
`isp_freq`	300	250	250	400	400	500	500	500	300	910
`v3d_freq`	300	250	250	400	400	500	500	500	300	910
`hevc_freq`	N/A	N/A	N/A	N/A	N/A	500	500	500	N/A	910
`sdram_freq`	450	400	450	450	500	3200	3200	3200	450	4267
`arm_freq_min`	700	700	600	600	600	600	600	600	600	1500
`core_freq_min`	250	250	250	250	250	200	200	200	250	500
`gpu_freq_min`	250	250	250	250	250	250	250	250	250	500
`h264_freq_min`	250	250	250	250	250	250	250	250	250	N/A
`isp_freq_min`	250	250	250	250	250	250	250	250	250	500
`v3d_freq_min`	250	250	250	250	250	250	250	250	250	500
`sdram_freq_min`	400	400	400	400	400	3200	3200	3200	400	4267

This table gives defaults for options which 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

NOTE

There is no need to use `hdmi_enable_4kp60` on Raspberry Pi 5; it supports dual-4Kp60 displays by default.

- 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 or later devices.

`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.

Clocks relationship

Raspberry Pi 4

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

WHITE PAPER

Cooling a Raspberry Pi device

This white paper goes through the reasons why your Raspberry Pi may get hot and why you might want to cool it back down, offering options on the cooling process.

To view the temperature of a Raspberry Pi, run the following command:

```
$ cat /sys/class/thermal/thermal_zone0/temp
```

Divide the result by 1000 to find the value in degrees Celsius. Alternatively, you can use `vcgencmd measure_temp` to report the GPU temperature.

Hitting the temperature limit is not harmful to the SoC, but it will cause the CPU to throttle. A heat sink 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 heat sink will make cooling more efficient.

When the core temperature is between 80°C and 85°C, the ARM cores will be throttled back. If the temperature exceeds 85°C, 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. This can be changed via the `temp_soft_limit` setting in `config.txt`.

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 the power section of the documentation.

If the voltage drops below 4.63V (±5%), the ARM cores and the GPU will be throttled back, and a message indicating the low voltage state will be added to the kernel log.

The Raspberry Pi 5 PMIC has built in ADCs that allow the supply voltage to be measured. To view the current supply voltage, run the following command:

```
$ vcgencmd pmic_read_adc EXT5V_V
```

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 apply 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+
[pi3+]	Model 3A+, Model 3B+ (also sees [pi3] contents)
[pi4]	Model 4B, Pi 400, Compute Module 4, Compute Module 4S
[pi5]	Raspberry Pi 5
[pi400]	Pi 400 (also sees [pi4] contents)
[cm4]	Compute Module 4 (also sees [pi4] contents)
[cm4s]	Compute Module 4S (also sees [pi4] contents)
[pi0]	Zero, Zero W, Zero 2 W
[pi0w]	Zero W (also sees [pi0] contents)
[pi02]	Zero 2 W (also sees [pi0w] and [pi0] contents)
[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.

NOTE

> Some models of Raspberry Pi (Zero W, Zero 2 W, Model 3B+, Pi 400, Compute Module 4 and Compute Module 4S) see the settings for multiple filters (as listed in the table above). This means that if you want a setting to apply only to (e.g.) a Model 4B without *also* applying that setting to a Pi 400, then the setting in the `[pi4]` section would need to be reverted by an alternate setting in a following `[pi400]` section - the ordering of such sections is significant. Alternatively, you could use a `[board-type=0x11]` filter which has a one-to-one mapping to different hardware products.

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 intended 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, you need to follow a few steps. Run the following command to see which output devices you have on your Raspberry Pi:

```
$ ls -1 /sys/class/drm/card?-HDMI-A-?/edid
```

On a Raspberry Pi 4, this will print something like:

```
/sys/class/drm/card1-HDMI-A-1/edid
/sys/class/drm/card1-HDMI-A-2/edid
```

You then need to run `edid-decode` against each of these filenames, for example:

```
$ edid-decode /sys/class/drm/card1-HDMI-A-1/edid
```

If there's no monitor connected to that particular output device, it'll tell you the EDID was empty; otherwise it will serve you **lots** of information about your monitor's capabilities. You need to look for the lines specifying the `Manufacturer` and the `Display Product Name`. The "EDID name" is then `<Manufacturer>-<Display Product Name>`, with any spaces in either string replaced by underscores. For example, if your `edid-decode` output included:

```
....
  Vendor & Product Identification:
    Manufacturer: DEL
....
    Display Product Name: 'DELL U2422H'
....
```

The EDID name for this monitor would be `DEL-DELL_U2422H`.

You can then use this as a conditional-filter to specify settings that only apply when this particular monitor is connected:

```
[EDID=DEL-DELL_U2422H]
cmdline=cmdline_U2422H.txt
[all]
```

These settings apply only at boot. 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 filter will be checked against both of them, and configuration from all matching conditional filters will be applied.

NOTE

> This setting is not available on Raspberry Pi 5.

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 (from which EDID cannot be read).

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

```
$ cat /proc/cpuinfo
```

A 16-digit hex value will be displayed near the bottom of the output. Your Raspberry Pi's serial number is the last eight hex-digits. For example, if you see:

```
Serial          : 0000000012345678
```

The serial number is `12345678`.

NOTE

> On some Raspberry Pi models, the first 8 hex-digits contain values other than `0`. Even in this case, only use the last eight hex-digits as the serial number.

You can define settings that will only be applied to this specific Raspberry Pi:

```
[0x12345678]
# settings here apply only to the Raspberry Pi with this serial

[all]
# settings here apply to all hardware
```

The GPIO filter

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

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

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

[all]
# settings here apply to all hardware
```

Combine 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 by listing them one after the other, for example:

```
# settings here apply to all hardware

[EDID=VSC-TD2220]
# settings here apply only if monitor VSC-TD2220 is connected

[pi2]
# settings here apply only if monitor VSC-TD2220 is connected *and* on a Raspbe
rry Pi 2

[all]
# settings here apply to all hardware
```

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

Memory options

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.

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.

The Raspberry Pi 4 has permanently disabled hardware decoders for MPEG2 and VC1. These codecs cannot be enabled, so a hardware codec licence key is not needed. Software decoding of MPEG2 and VC1 files performs well enough for typical use cases.

The Raspberry Pi 5 has H.265 (HEVC) hardware decoding. This decoding is enabled by default, so a hardware codec licence key is not needed.

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

To control HDMI settings, use the Screen Configuration utility or KMS video settings in `cmdline.txt`.

HDMI Pipeline for Raspberry Pi 4

In order to support dual displays and modes up to 4Kp60, the Raspberry Pi 4 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 cannot support a timing where *any* of the horizontal timings are not divisible by 2. The firmware and Linux kernel filter out any mode that does not fulfil this criteria.

There is only one incompatible mode in the CEA and DMT standards: DMT mode 81, 1366x768 @ 60Hz. This mode has odd-numbered values for the horizontal sync and back porch timings and a width that indivisible by 8.

If your monitor has this resolution, Raspberry Pi 4 automatically drops down to the next mode advertised by the monitor; typically 1280x720.

HDMI Pipeline for Raspberry Pi 5

While Raspberry Pi 5 also works at 2 output pixels per clock cycle, it has special handling for odd timings and can handle these modes directly.

Composite video mode

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
Raspberry Pi Zero 2 W	Test pads on underside of board
Raspberry Pi 5	J7 pad next to HDMI socket
All other models	3.5mm AV jack

NOTE

Composite video output is not available on Raspberry Pi 400.

`enable_tvout`

Set to `1` to enable composite video output and `0` to disable. On Raspberry Pi 4 and 5, composite output is only available if you set this to `1`, which also disables HDMI output.

Composite output is not available on the Raspberry Pi 400.

Model	Default
Pi 4, 5 and 400	0
All other models	1

On all models except Raspberry Pi 4 and 5, HDMI output needs to be disabled in order for composite output to be enabled. HDMI output is disabled when no HDMI display is connected / detected. Set `enable_tvout=0` to prevent composite being enabled when HDMI is disabled.

To enable composite output, append `,composite` to the end of the `dtoverlay=vc4-kms-v3d` line in `/boot/firmware/config.txt`:

```
dtoverlay=vc4-kms-v3d,composite
```

By default, this outputs composite NTSC video. To choose a different mode, instead append the following to the single line in `/boot/firmware/cmdline.txt`:

```
vc4.tv_norm=<video_mode>
```

Replace the `<video_mode>` placeholder with one of the following values:

- `NTSC`
- `NTSC-J`
- `NTSC-443`
- `PAL`
- `PAL-M`
- `PAL-N`
- `PAL60`
- `SECAM`

LCD displays and touchscreens

`ignore_lcd`

By default, the Raspberry Pi Touch Display is used when detected on the I2C bus. `ignore_lcd=1` skips this detection phase. This prevents the LCD display from being used.

`disable_touchscreen`

Enables and disables the touchscreen.

`disable_touchscreen=1` disables the touchscreen component of the official Raspberry Pi Touch Display.

Generic display options

`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 (and overscan parameters) to the Linux kernel via settings on the kernel command line. In rare circumstances, the firmware can choose a mode not in the EDID that may be incompatible with the device. Use `disable_fw_kms_setup=1` to disable passing video mode parameters, which can avoid this problem. The Linux video mode system (KMS) instead parses the EDID itself and picks an appropriate mode.

NOTE

On Raspberry Pi 5, this parameter defaults to `1`.

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, for example when the camera is facing a window.

`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.