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 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 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.

Example update flow for A/B booting

The following pseudo code shows how an hypothetical OS `Update Service` could use `tryboot` + `autoboot.txt` to perform an 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
```

Notes * 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.. * See also: Device-tree parameters.

Common Options

Edit this on GitHub

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.

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

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

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).

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 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 nam 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 forur

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.