Summary class before minor 1 week using Raspberry Pi as example

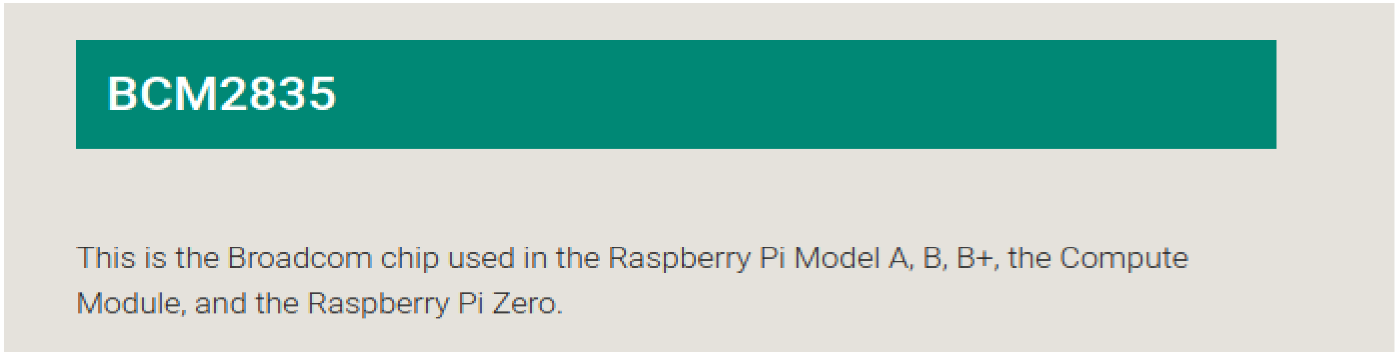
- How is hardware specified for an embedded platform?

  device trees
  https://www.raspberrypi.org/documentation/configuration/device-tree.md

- How is software written to work with different hardware peripherals?
  - device driver examples for
    - SPI (which can use polling or interrupt or DMA)

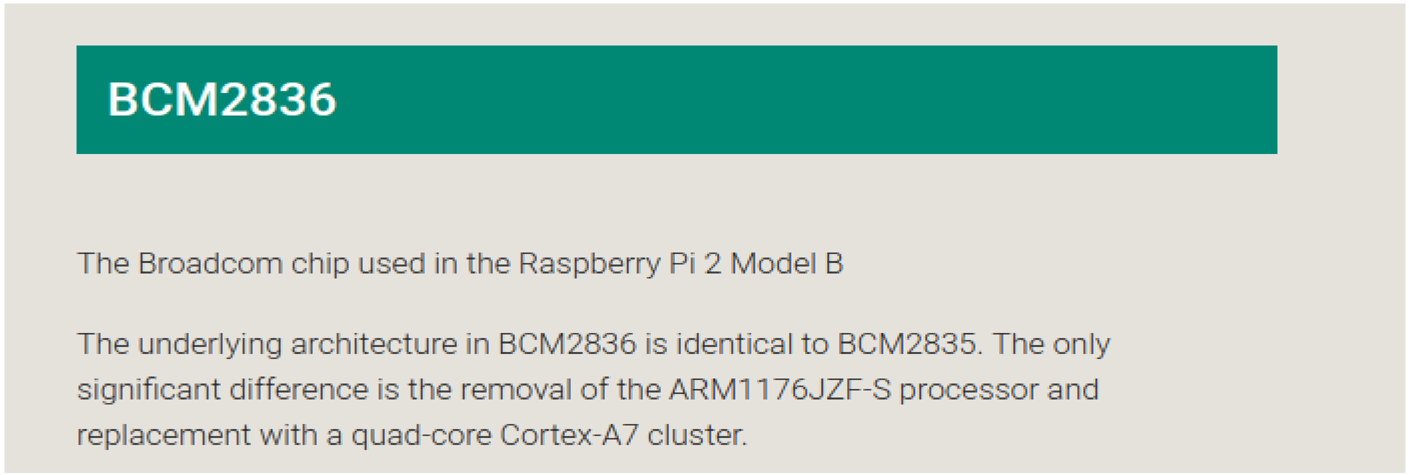
- User space interactions with the device drivers
- Some more details about DMA

Raspberry PI SoCs



BCM2835

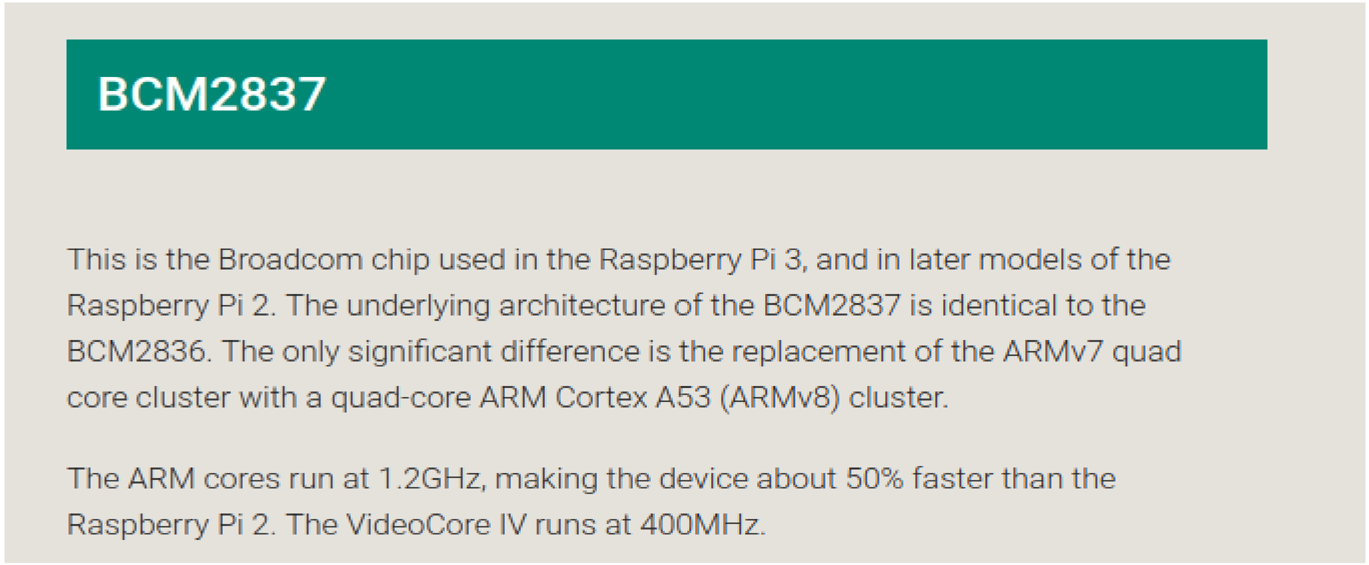
This is the Broadcom chip used in the Raspberry Pi Model A, B, B+, the Compute Module, and the Raspberry Pi Zero.



BCM2836

The Broadcom chip used in the Raspberry Pi 2 Model B

The underlying architecture in BCM2836 is identical to BCM2835. The only significant difference is the removal of the ARM1176JZF-S processor and replacement with a quad-core Cortex-A7 cluster.



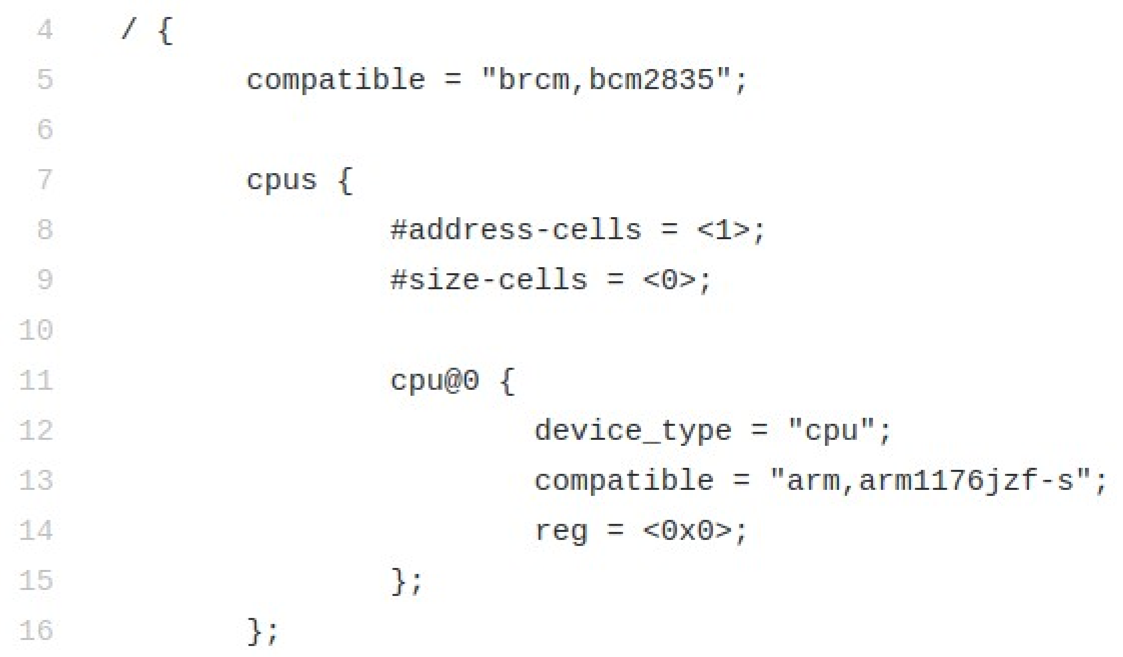
BCM2837

This is the Broadcom chip used in the Raspberry Pi 3, and in later models of the Raspberry Pi 2. The underlying architecture of the BCM2837 is identical to the BCM2836. The only significant difference is the replacement of the ARMv7 quad core cluster with a quad-core ARM Cortex A53 (ARMv8) cluster.

The ARM cores run at 1.2GHz, making the device about 50% faster than the Raspberry Pi 2. The VideoCore IV runs at 400MHz.

PI Zero device tree hierarchy

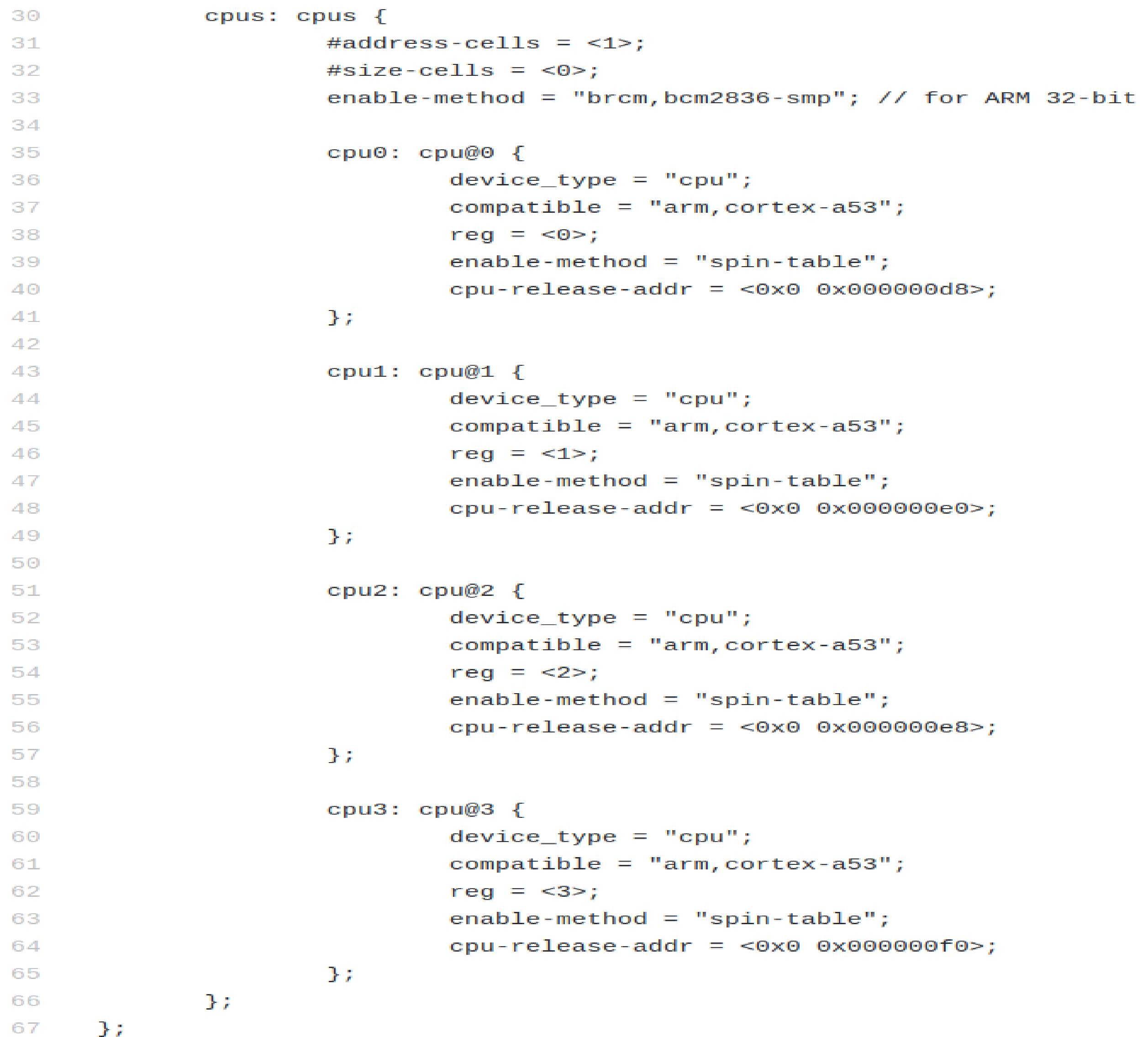
https://github.com/torvalds/linux/blob/master/arch/arm/boot/dts/bcm2835.dtsi
https://github.com/torvalds/linux/blob/master/arch/arm/boot/dts/bcm2835-rpi.dtsi
https://github.com/torvalds/linux/blob/master/arch/arm/boot/dts/bcm283x-rpi-usb-otg.dtsi
https://github.com/torvalds/linux/blob/master/arch/arm/boot/dts/bcm2835-rpi-zero-w.dts



```
/ {
        compatible = "brcm,bcm2835";

        cpus {
                #address-cells = <1>;
                #size-cells = <0>;

                cpu@0 {
                        device_type = "cpu";
                        compatible = "arm,arm1176jzf-s";
                        reg = <0x0>;
                };
        };
```

PI 3B device tree hierarchy

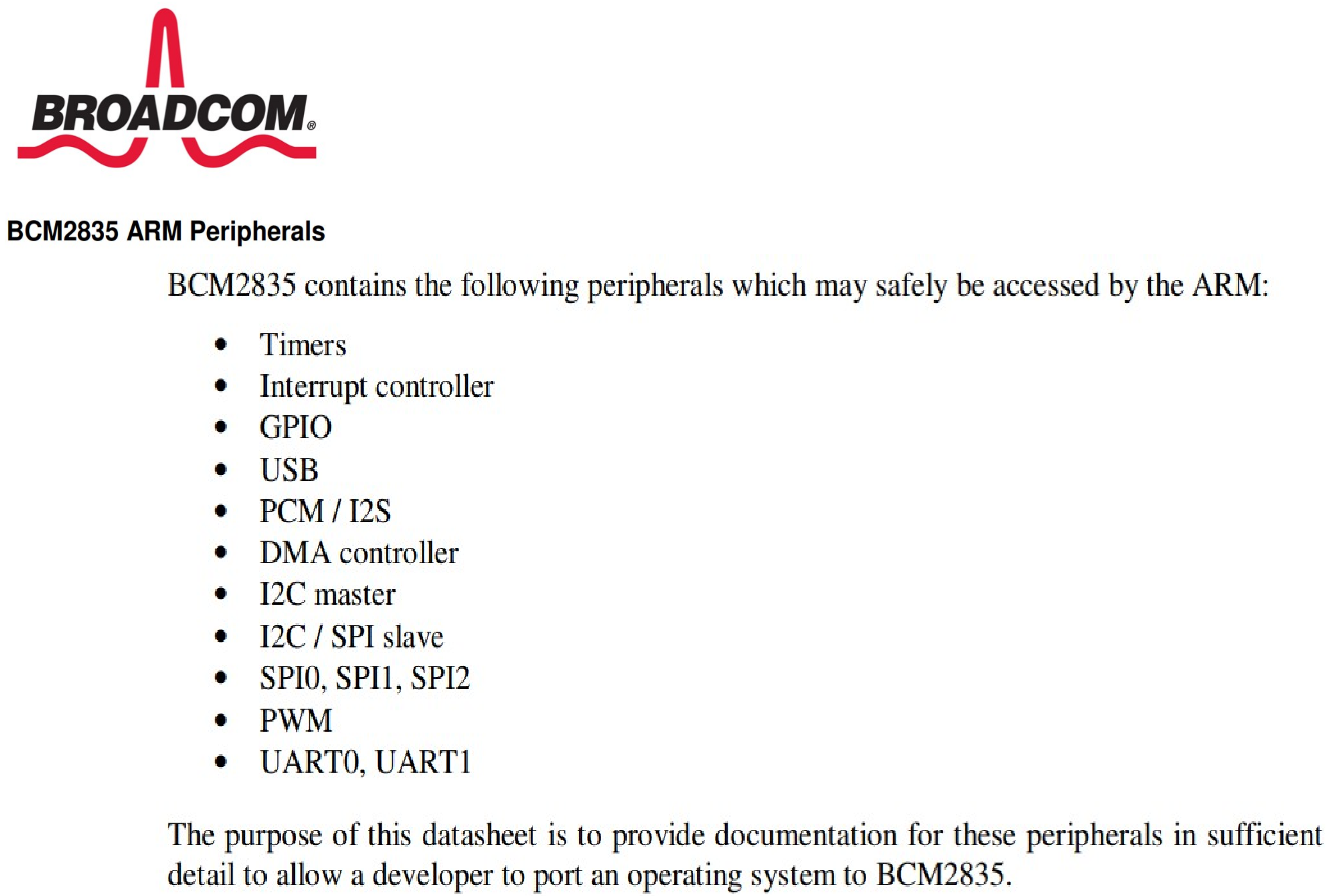
https://github.com/torvalds/linux/blob/master/arch/arm/boot/dts/bcm2837.dtsi
https://github.com/torvalds/linux/blob/master/arch/arm/boot/dts/bcm2835-rpi.dtsi
https://github.com/torvalds/linux/blob/master/arch/arm/boot/dts/bcm283x-rpi-usb-otg.dtsi
https://github.com/torvalds/linux/blob/master/arch/arm/boot/dts/bcm2837-rpi-3-b.dts



```
        cpus: cpus {
                #address-cells = <1>;
                #size-cells = <0>;
                enable-method = "brcm,bcm2836-smp"; // for ARM 32-bit

                cpu0: cpu@0 {
                        device_type = "cpu";
                        compatible = "arm,cortex-a53";
                        reg = <0>;
                        enable-method = "spin-table";
                        cpu-release-addr = <0x0 0x000000d8>;
                };

                cpu1: cpu@1 {
                        device_type = "cpu";
                        compatible = "arm,cortex-a53";
                        reg = <1>;
                        enable-method = "spin-table";
                        cpu-release-addr = <0x0 0x000000e0>;
                };

                cpu2: cpu@2 {
                        device_type = "cpu";
                        compatible = "arm,cortex-a53";
                        reg = <2>;
                        enable-method = "spin-table";
                        cpu-release-addr = <0x0 0x000000e8>;
                };

                cpu3: cpu@3 {
                        device_type = "cpu";
                        compatible = "arm,cortex-a53";
                        reg = <3>;
                        enable-method = "spin-table";
                        cpu-release-addr = <0x0 0x000000f0>;
                };
        };
};
```

SoC Peripheral Hardware Description

https://www.raspberrypi.org/app/uploads/2012/02/BCM2835-ARM-Peripherals.pdf



BROADCOM®

BCM2835 ARM Peripherals

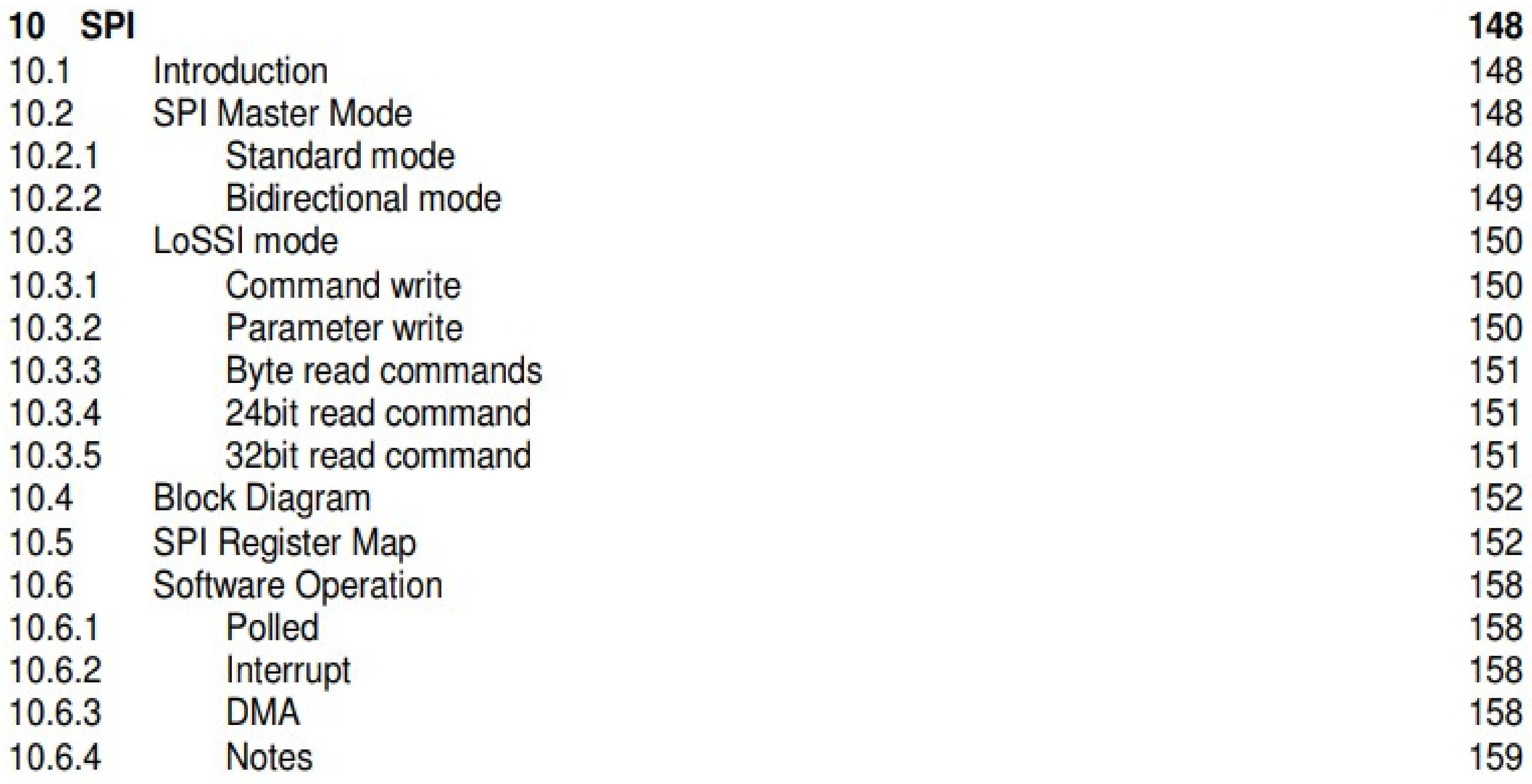
BCM2835 contains the following peripherals which may safely be accessed by the ARM:

- Timers
- Interrupt controller
- GPIO
- USB
- PCM / I2S
- DMA controller
- I2C master
- I2C / SPI slave
- SPI0, SPI1, SPI2
- PWM
- UART0, UART1

The purpose of this datasheet is to provide documentation for these peripherals in sufficient detail to allow a developer to port an operating system to BCM2835.

SPI in datasheet

https://www.raspberrypi.org/app/uploads/2012/02/BCM2835-ARM-Peripherals.pdf



SPI driver code supports all three (along with helper functions)

https://github.com/torvalds/linux/blob/master/drivers/spi/spi-bcm2835.c

```c
static int bcm2835_spi_transfer_one_poll(struct spi_master *master,
                                        struct spi_device *spi,
                                        struct spi_transfer *tfr,
                                        u32 cs,
                                        unsigned long long xfer_time_us)
```

```c
static int bcm2835_spi_transfer_one_irq(struct spi_master *master,
                                       struct spi_device *spi,
                                       struct spi_transfer *tfr,
                                       u32 cs)
```

```c
static int bcm2835_spi_transfer_one_dma(struct spi_master *master,
                                       struct spi_device *spi,
                                       struct spi_transfer *tfr,
                                       u32 cs)
```

SPI driver code chooses among three

https://github.com/torvalds/linux/blob/master/drivers/spi/spi-bcm2835.c

```c
static int bcm2835_spi_transfer_one(struct spi_master *master,
                                    struct spi_device *spi,
                                    struct spi_transfer *tfr)
```

```c
        /* calculate the estimated time in us the transfer runs */
        xfer_time_us = (unsigned long long)tfr->len
                * 9 /* clocks/byte - SPI-HW waits 1 clock after each byte */
                * 1000000;
        do_div(xfer_time_us, spi_used_hz);

        /* for short requests run polling*/
        if (xfer_time_us <= BCM2835_SPI_POLLING_LIMIT_US)
                return bcm2835_spi_transfer_one_poll(master, spi, tfr,
                                                     cs, xfer_time_us);

        /* run in dma mode if conditions are right */
        if (master->can_dma && bcm2835_spi_can_dma(master, spi, tfr))
                return bcm2835_spi_transfer_one_dma(master, spi, tfr, cs);

        /* run in interrupt-mode */
        return bcm2835_spi_transfer_one_irq(master, spi, tfr, cs);
```

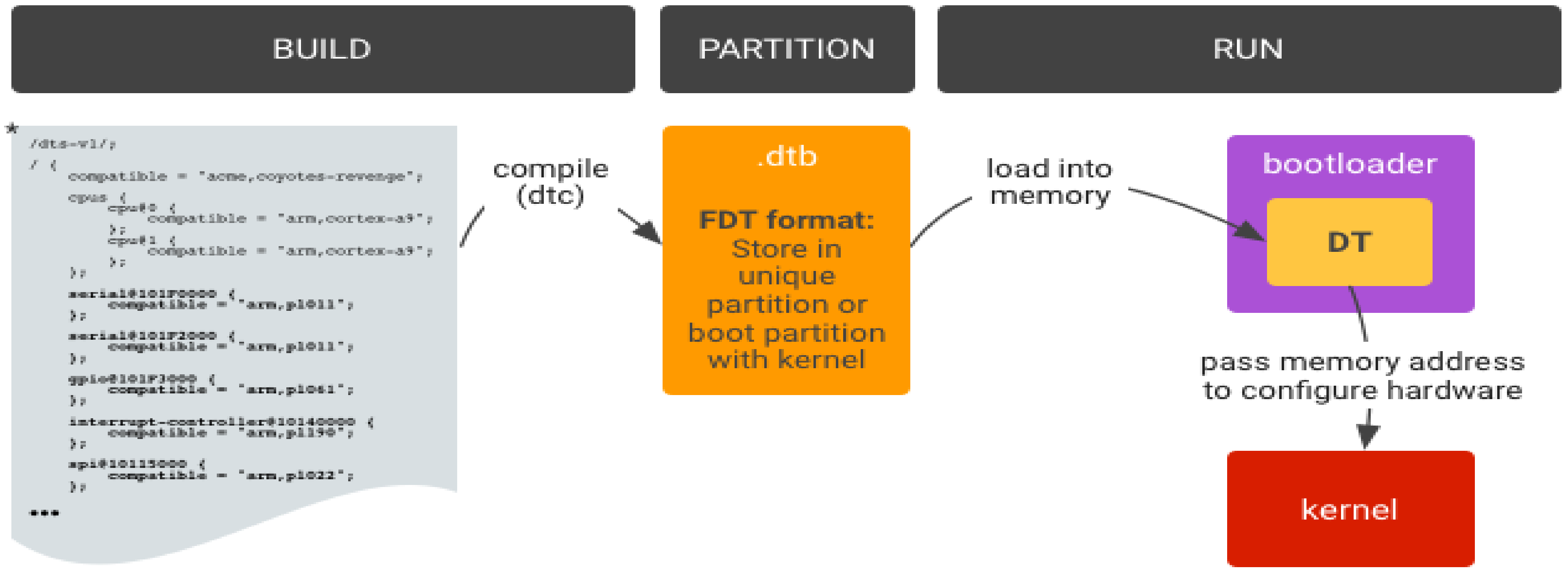
But none of the files in the device tree hieararchy mentions anything about SPI or DMA controller

Description from Android Source Documentation

https://source.android.com/devices/architecture/dto/

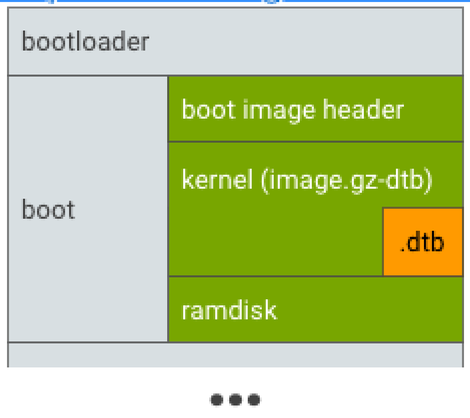
- A device tree (DT) is a data structure of named nodes and properties that describe non-discoverable hardware.
- Operating systems, such as the Linux kernel used in Android, use DTs to support a wide range of hardware configurations used by Android-powered devices.
- Hardware vendors supply their own DT source files, which Linux then compiles into the Device Tree Blob (DTB) file used by the bootloader.
- A device tree overlay (DTO) enables a central device tree blob (DTB) to be overlaid on the device tree.
- A bootloader using DTO can maintain the system-on-chip (SoC) DT and dynamically overlay a device-specific DT, adding nodes to the tree and making changes to properties in the existing tree.

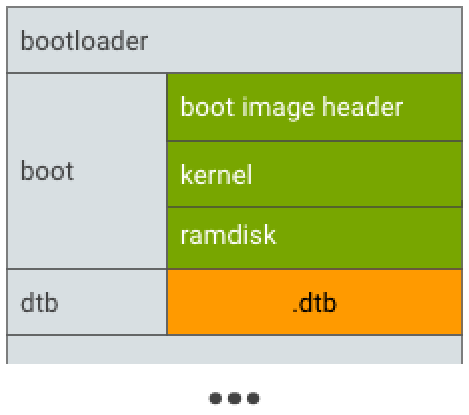
Loading a device tree in bootloader involves building, partitioning, and running.


BUILD
PARTITION
RUN
*
compile (dtc)
.dtb
FDT format: Store in unique partition or boot partition with kernel
load into memory
bootloader
DT
pass memory address to configure hardware
kernel

```
/dts-v1/;
/ {
    compatible = "acme,coyotes-revenge";
    cpus {
        cpu@0 {
            compatible = "arm,cortex-a9";
        };
        cpu@1 {
            compatible = "arm,cortex-a9";
        };
    };
    serial@101F0000 {
        compatible = "arm,pl011";
    };
    serial@101F2000 {
        compatible = "arm,pl011";
    };
    gpio@101F3000 {
        compatible = "arm,pl061";
    };
    interrupt-controller@10140000 {
        compatible = "arm,pl190";
    };
    spi@10115000 {
        compatible = "arm,pl022";
    };
...
```

*http://elinux.org/Device_Tree_Usage#Devices

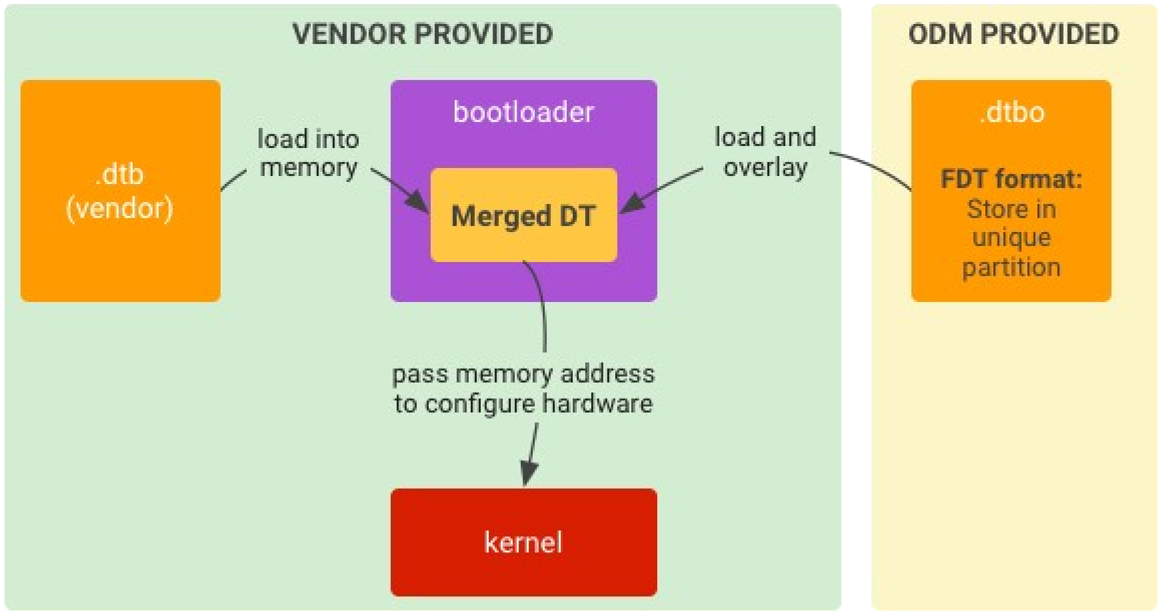




Possible partitions

.dtb, .dtbo merging is needed

- Typical runtime implementation for device tree overlay in bootloader.
  - Load .dtb from storage into memory.
  - Load .dtbo from storage into memory.
  - Overlay .dtb with .dtbo to be a merged DT.
  - Start kernel given the memory address of the merged DT.



Similar procedure in PI

https://github.com/eq-3/RaspberryMatic/blob/master/linux-4.1/arch/arm/boot/dts/overlays/README

- Device Tree makes it possible to support many hardware configurations with a single kernel and without the need to explicitly load or blacklist kernel modules. This isn't a "pure" Device Tree configuration
  - (c.f. MACH_BCM2835) - some on-board devices are still configured by the board support code, but the intention is to eventually reach that goal.

- On Raspberry Pi, Device Tree usage is controlled from /boot/config.txt. By default, the Raspberry Pi kernel boots with device tree enabled. You can completely disable DT usage (for now) by adding:

  device_tree=

  to your config.txt, which should cause your Pi to revert to the old way of doing things after a reboot.

- In /boot you will find a .dtb for each base platform. This describes the hardware that is part of the Raspberry Pi board.
- The loader (start.elf and its siblings) selects the .dtb file appropriate for the platform by name, and reads it into memory.
- At this point, all of the optional interfaces (i2c, i2s, spi) are disabled, but they can be enabled using Device Tree parameters:

  dtparam=i2c=on,i2s=on,spi=on

- Configuring additional, optional hardware is done using **Device Tree overlays**

DT overlay example in PI

- Overlays are loaded using the "dtoverlay" directive. As an example, consider the popular lirc-rpi module, the Linux Infrared Remote Control driver.
- In the pre-DT world this would be loaded from /etc/modules, with an explicit "modprobe lirc-rpi" command, or programmatically by lircd.
- With DT enabled, this becomes a line in config.txt:

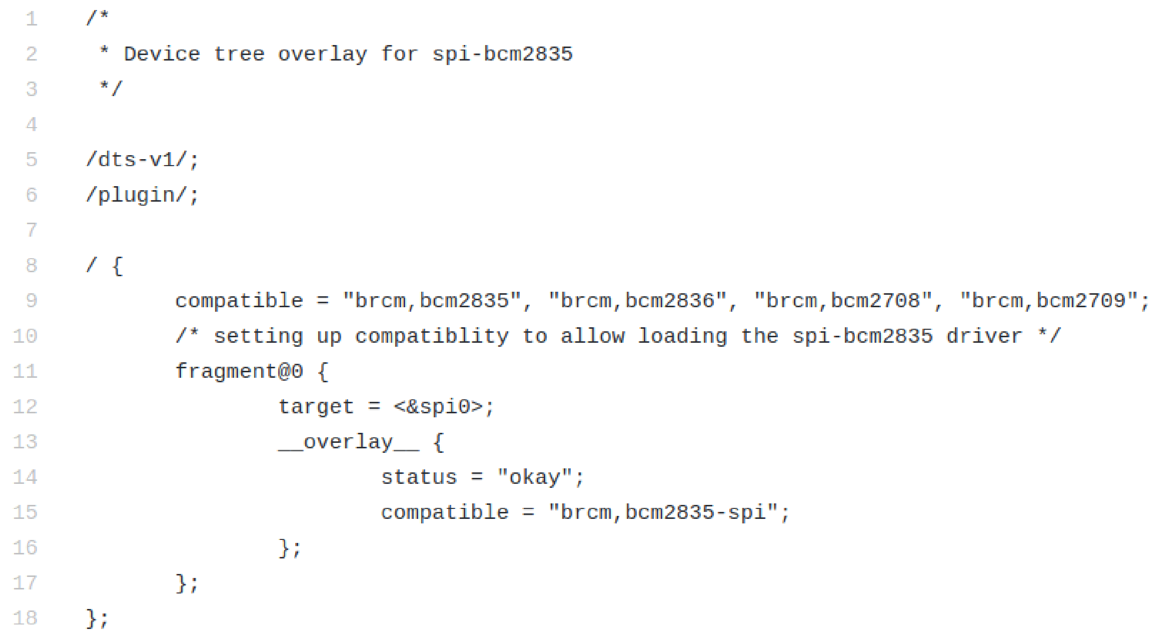
```
dtoverlay=lirc-rpi
```

- This causes the file /boot/overlays/lirc-rpi-overlay.dtb to be loaded. By default it will use GPIOs 17 (out) and 18 (in), but this can be modified using DT parameters:

```
dtoverlay=lirc-rpi,gpio_out_pin=17,gpio_in_pin=13
```

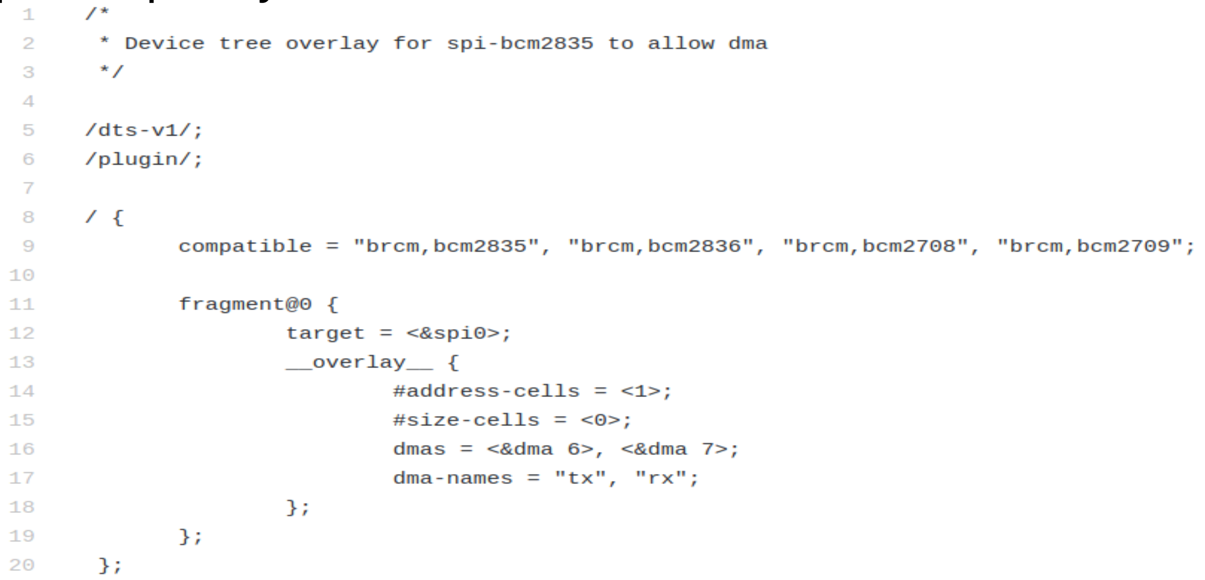
DT overlays for SPI on PI

https://github.com/eq-3/RaspberryMatic/blob/master/linux-4.1/arch/arm/boot/dts/overlays/spi-bcm2835-overlay.dts



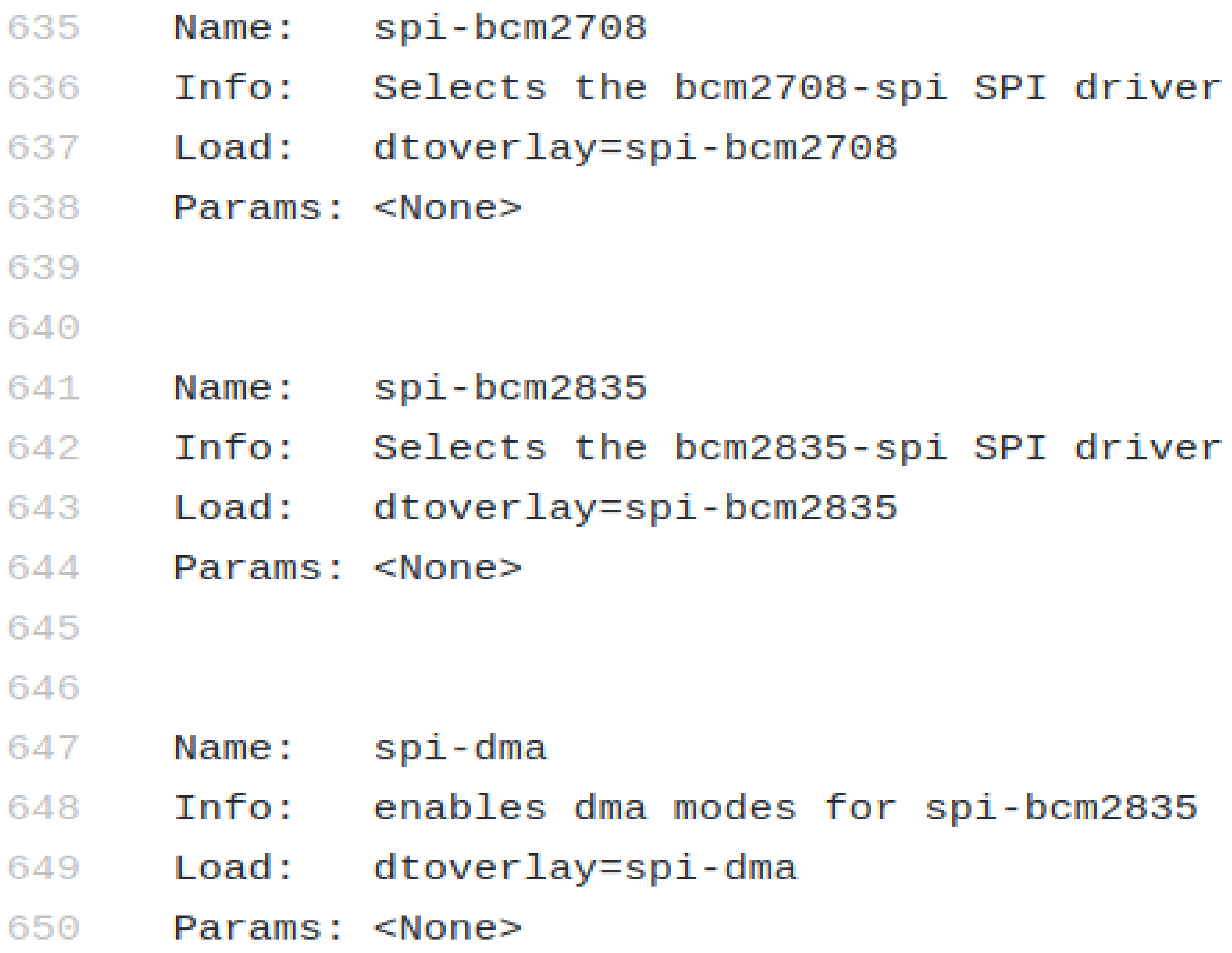
```
/*
 * Device tree overlay for spi-bcm2835
 */

/dts-v1/;
/plugin/;

/ {
        compatible = "brcm,bcm2835", "brcm,bcm2836", "brcm,bcm2708", "brcm,bcm2709";
        /* setting up compatiblity to allow loading the spi-bcm2835 driver */
        fragment@0 {
                target = <&spi0>;
                __overlay__ {
                        status = "okay";
                        compatible = "brcm,bcm2835-spi";
                };
        };
};
```

https://github.com/eq-3/RaspberryMatic/blob/master/linux-4.1/arch/arm/boot/dts/overlays/spi-dma-overlay.dts



```
/*
 * Device tree overlay for spi-bcm2835 to allow dma
 */

/dts-v1/;
/plugin/;

/ {
        compatible = "brcm,bcm2835", "brcm,bcm2836", "brcm,bcm2708", "brcm,bcm2709";

        fragment@0 {
                target = <&spi0>;
                __overlay__ {
                        #address-cells = <1>;
                        #size-cells = <0>;
                        dmas = <&dma 6>, <&dma 7>;
                        dma-names = "tx", "rx";
                };
        };
};
```

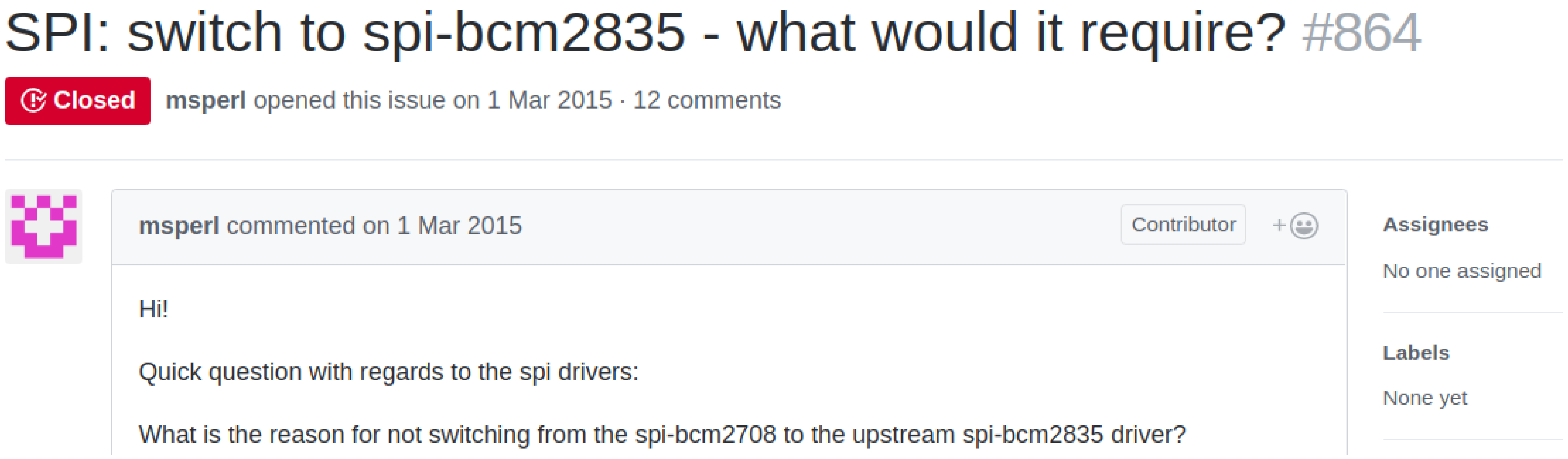
Choosing an appropriate overlay



```
Name:   spi-bcm2708
Info:   Selects the bcm2708-spi SPI driver
Load:   dtoverlay=spi-bcm2708
Params: <None>


Name:   spi-bcm2835
Info:   Selects the bcm2835-spi SPI driver
Load:   dtoverlay=spi-bcm2835
Params: <None>


Name:   spi-dma
Info:   enables dma modes for spi-bcm2835
Load:   dtoverlay=spi-dma
Params: <None>
```

Discussions over choosing default driver

https://github.com/raspberrypi/linux/issues/864



SPI: switch to spi-bcm2835 - what would it require? #864

Closed msperl opened this issue on 1 Mar 2015 · 12 comments

msperl commented on 1 Mar 2015 Contributor

Hi!

Quick question with regards to the spi drivers:

What is the reason for not switching from the spi-bcm2708 to the upstream spi-bcm2835 driver?

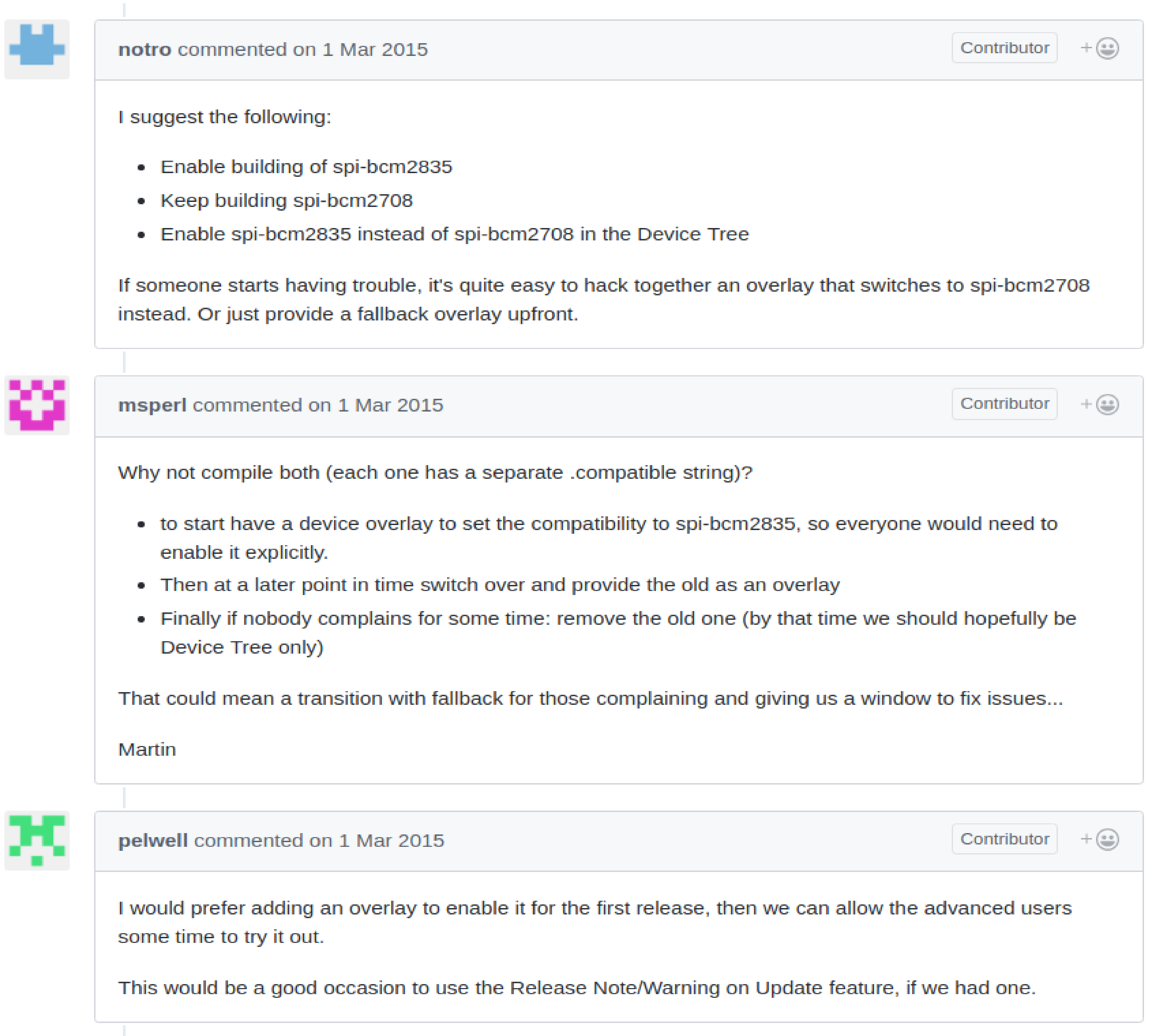
Assignees

No one assigned

Labels

None yet

Different suggestions using DT



notro commented on 1 Mar 2015 Contributor

I suggest the following:

- Enable building of spi-bcm2835
- Keep building spi-bcm2708
- Enable spi-bcm2835 instead of spi-bcm2708 in the Device Tree

If someone starts having trouble, it's quite easy to hack together an overlay that switches to spi-bcm2708 instead. Or just provide a fallback overlay upfront.

msperl commented on 1 Mar 2015 Contributor

Why not compile both (each one has a separate .compatible string)?

- to start have a device overlay to set the compatibility to spi-bcm2835, so everyone would need to enable it explicitly.
- Then at a later point in time switch over and provide the old as an overlay
- Finally if nobody complains for some time: remove the old one (by that time we should hopefully be Device Tree only)

That could mean a transition with fallback for those complaining and giving us a window to fix issues...

Martin

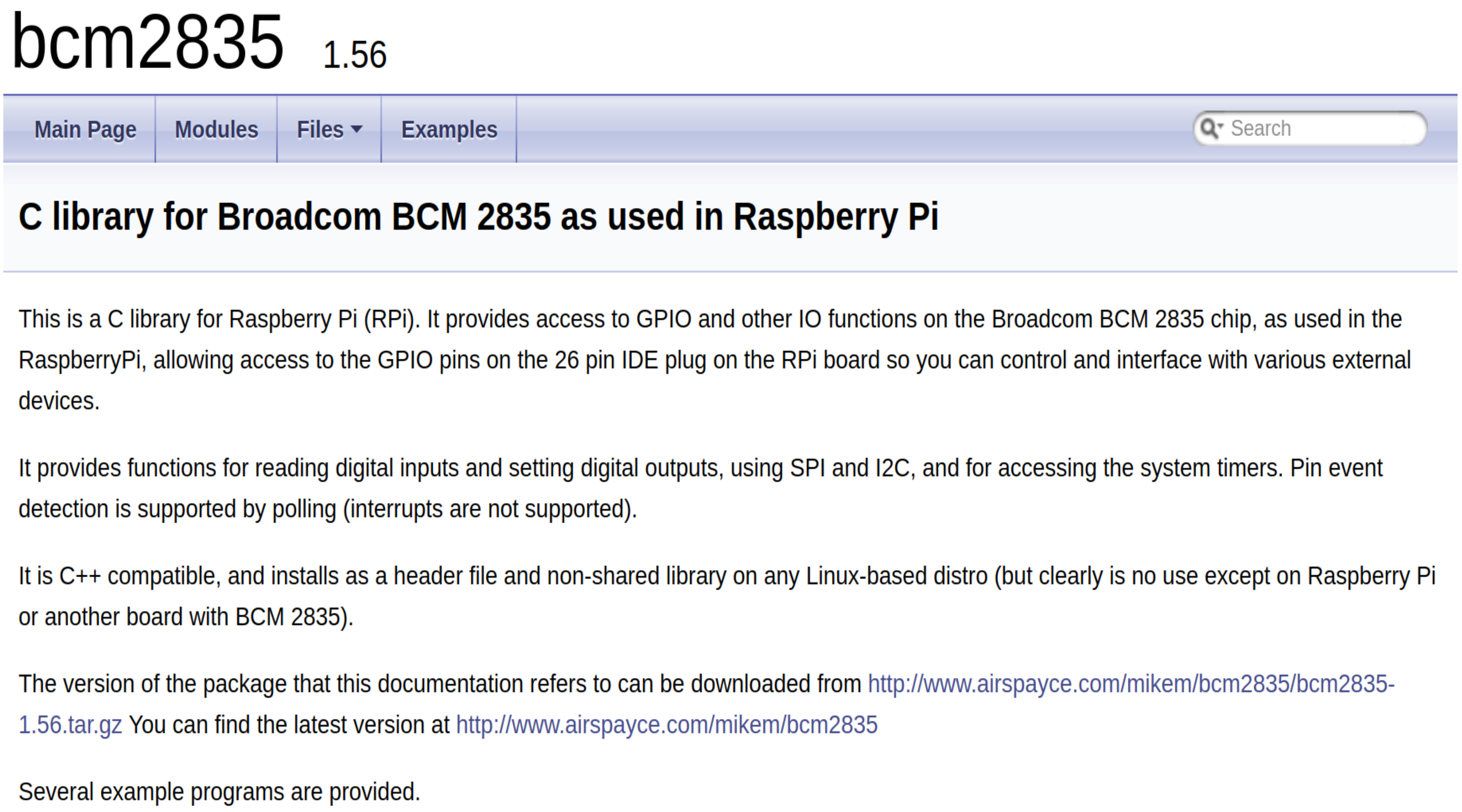
pelwell commented on 1 Mar 2015 Contributor

I would prefer adding an overlay to enable it for the first release, then we can allow the advanced users some time to try it out.

This would be a good occasion to use the Release Note/Warning on Update feature, if we had one.

User Space Libraries

http://www.airspayce.com/mikem/bcm2835/



bcm2835 1.56

C library for Broadcom BCM 2835 as used in Raspberry Pi

This is a C library for Raspberry Pi (RPi). It provides access to GPIO and other IO functions on the Broadcom BCM 2835 chip, as used in the RaspberryPi, allowing access to the GPIO pins on the 26 pin IDE plug on the RPi board so you can control and interface with various external devices.

It provides functions for reading digital inputs and setting digital outputs, using SPI and I2C, and for accessing the system timers. Pin event detection is supported by polling (interrupts are not supported).

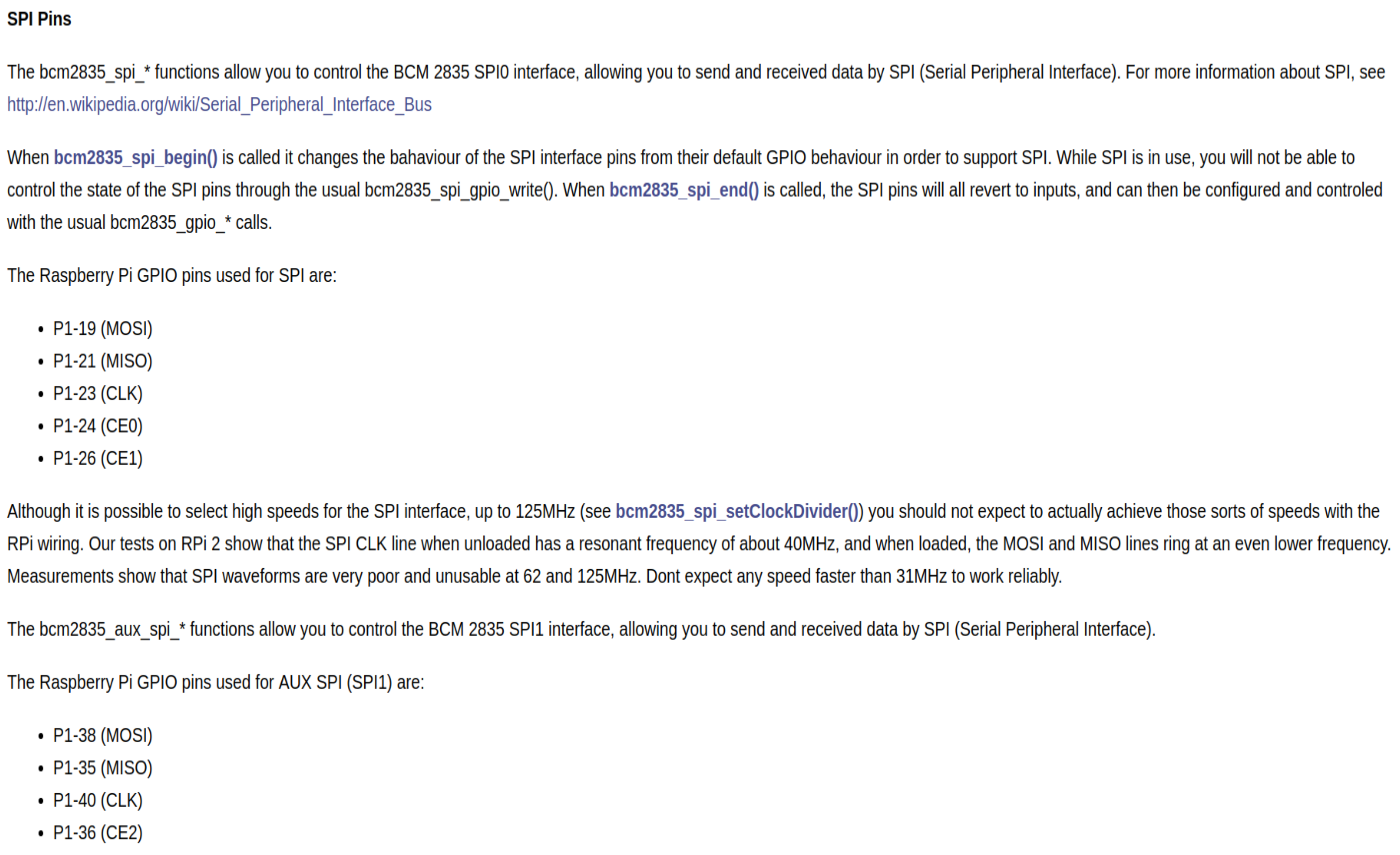
It is C++ compatible, and installs as a header file and non-shared library on any Linux-based distro (but clearly is no use except on Raspberry Pi or another board with BCM 2835).

The version of the package that this documentation refers to can be downloaded from http://www.airspayce.com/mikem/bcm2835/bcm2835-1.56.tar.gz You can find the latest version at http://www.airspayce.com/mikem/bcm2835

Several example programs are provided.

Using SPI from userspace with lib functions

http://www.airspayce.com/mikem/bcm2835/



SPI Pins

The bcm2835_spi_* functions allow you to control the BCM 2835 SPI0 interface, allowing you to send and received data by SPI (Serial Peripheral Interface). For more information about SPI, see http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

When **bcm2835_spi_begin()** is called it changes the bahaviour of the SPI interface pins from their default GPIO behaviour in order to support SPI. While SPI is in use, you will not be able to control the state of the SPI pins through the usual bcm2835_spi_gpio_write(). When **bcm2835_spi_end()** is called, the SPI pins will all revert to inputs, and can then be configured and controled with the usual bcm2835_gpio_* calls.

The Raspberry Pi GPIO pins used for SPI are:

- P1-19 (MOSI)
- P1-21 (MISO)
- P1-23 (CLK)
- P1-24 (CE0)
- P1-26 (CE1)

Although it is possible to select high speeds for the SPI interface, up to 125MHz (see **bcm2835_spi_setClockDivider()**) you should not expect to actually achieve those sorts of speeds with the RPi wiring. Our tests on RPi 2 show that the SPI CLK line when unloaded has a resonant frequency of about 40MHz, and when loaded, the MOSI and MISO lines ring at an even lower frequency. Measurements show that SPI waveforms are very poor and unusable at 62 and 125MHz. Dont expect any speed faster than 31MHz to work reliably.

The bcm2835_aux_spi_* functions allow you to control the BCM 2835 SPI1 interface, allowing you to send and received data by SPI (Serial Peripheral Interface).

The Raspberry Pi GPIO pins used for AUX SPI (SPI1) are:

- P1-38 (MOSI)
- P1-35 (MISO)
- P1-40 (CLK)
- P1-36 (CE2)

Root vs. Non-root and installation

Running as root

Prior to the release of Raspbian Jessie in Feb 2016, access to any peripheral device via /dev/mem on the RPi required the process to run as root. Raspbian Jessie permits non-root users to access the GPIO peripheral (only) via /dev/gpiomem, and this library supports that limited mode of operation.

If the library runs with effective UID of 0 (ie root), then **bcm2835_init()** will attempt to open /dev/mem, and, if successful, it will permit use of all peripherals and library functions.

If the library runs with any other effective UID (ie not root), then **bcm2835_init()** will attempt to open /dev/gpiomem, and, if successful, will only permit GPIO operations. In particular, **bcm2835_spi_begin()** and **bcm2835_i2c_begin()** will return false and all other non-gpio operations may fail silently or crash.

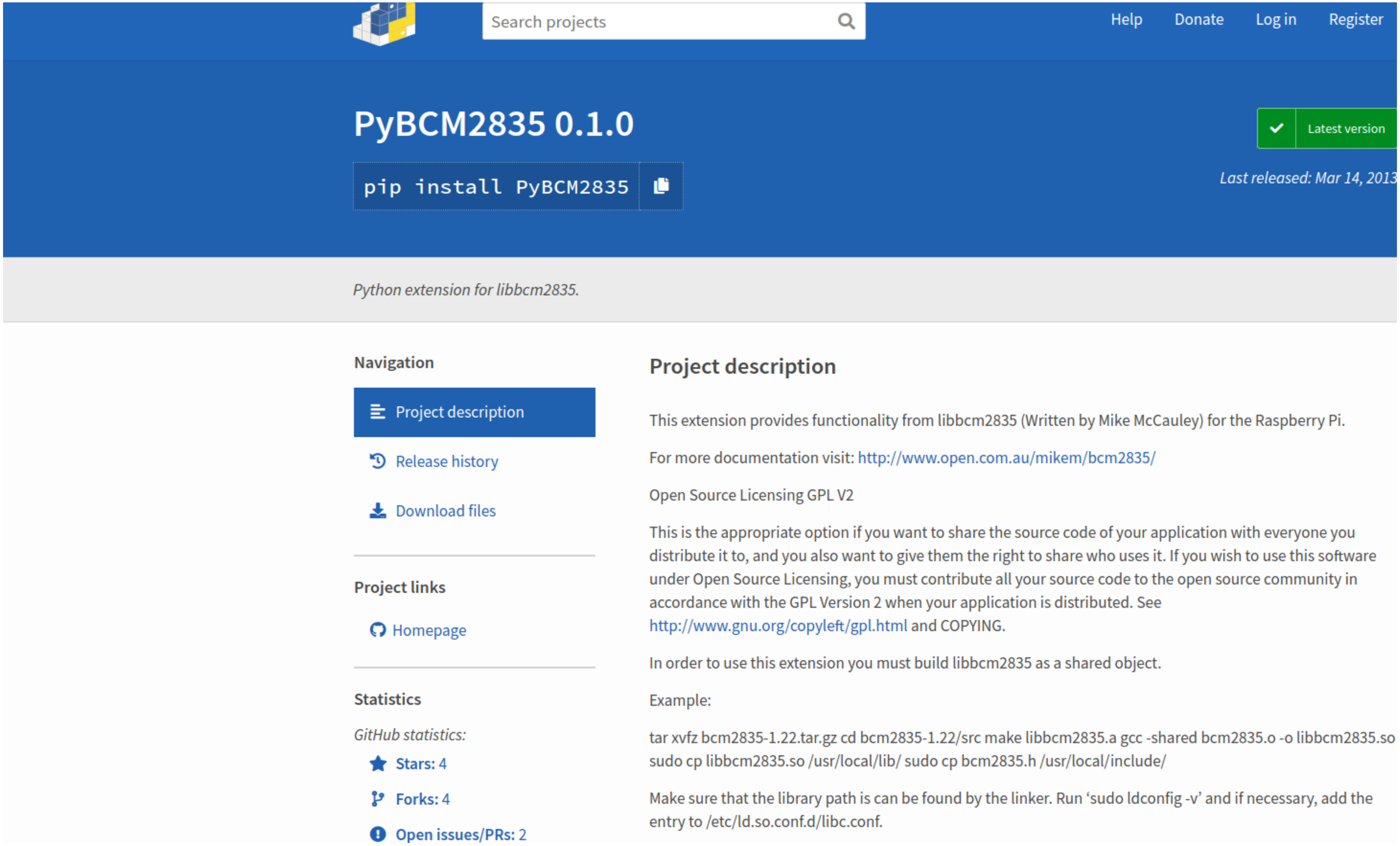
Installation

This library consists of a single non-shared library and header file, which will be installed in the usual places by make install

```
# download the latest version of the library, say bcm2835-1.xx.tar.gz, then:
tar zxvf bcm2835-1.xx.tar.gz
cd bcm2835-1.xx
./configure
make
sudo make check
sudo make install
```

Python bindings

https://pypi.org/project/PyBCM2835/



PyBCM2835 0.1.0

```
pip install PyBCM2835
```

Last released: Mar 14, 2013

Python extension for libbcm2835.

Statistics

GitHub statistics:

Stars: 4

Forks: 4

Open issues/PRs: 2

Project description

This extension provides functionality from libbcm2835 (Written by Mike McCauley) for the Raspberry Pi.

For more documentation visit: http://www.open.com.au/mikem/bcm2835/

Open Source Licensing GPL V2

This is the appropriate option if you want to share the source code of your application with everyone you distribute it to, and you also want to give them the right to share who uses it. If you wish to use this software under Open Source Licensing, you must contribute all your source code to the open source community in accordance with the GPL Version 2 when your application is distributed. See http://www.gnu.org/copyleft/gpl.html and COPYING.

In order to use this extension you must build libbcm2835 as a shared object.

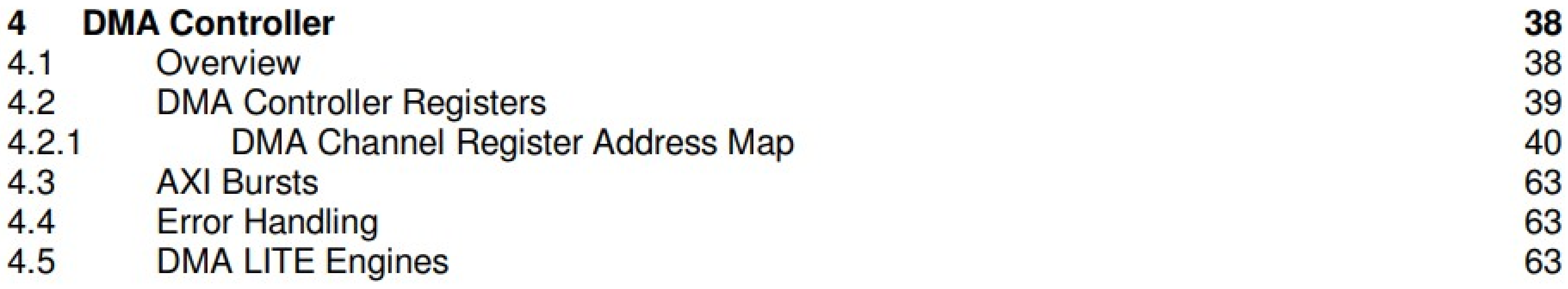
Example:

tar xvfz bcm2835-1.22.tar.gz cd bcm2835-1.22/src make libbcm2835.a gcc -shared bcm2835.o -o libbcm2835.so sudo cp libbcm2835.so /usr/local/lib/ sudo cp bcm2835.h /usr/local/include/

Make sure that the library path is can be found by the linker. Run ‘sudo ldconfig -v’ and if necessary, add the entry to /etc/ld.so.conf.d/libc.conf.

DMA controller in datasheet

https://www.raspberrypi.org/app/uploads/2012/02/BCM2835-ARM-Peripherals.pdf



Linux Kernel APIs that drivers can include

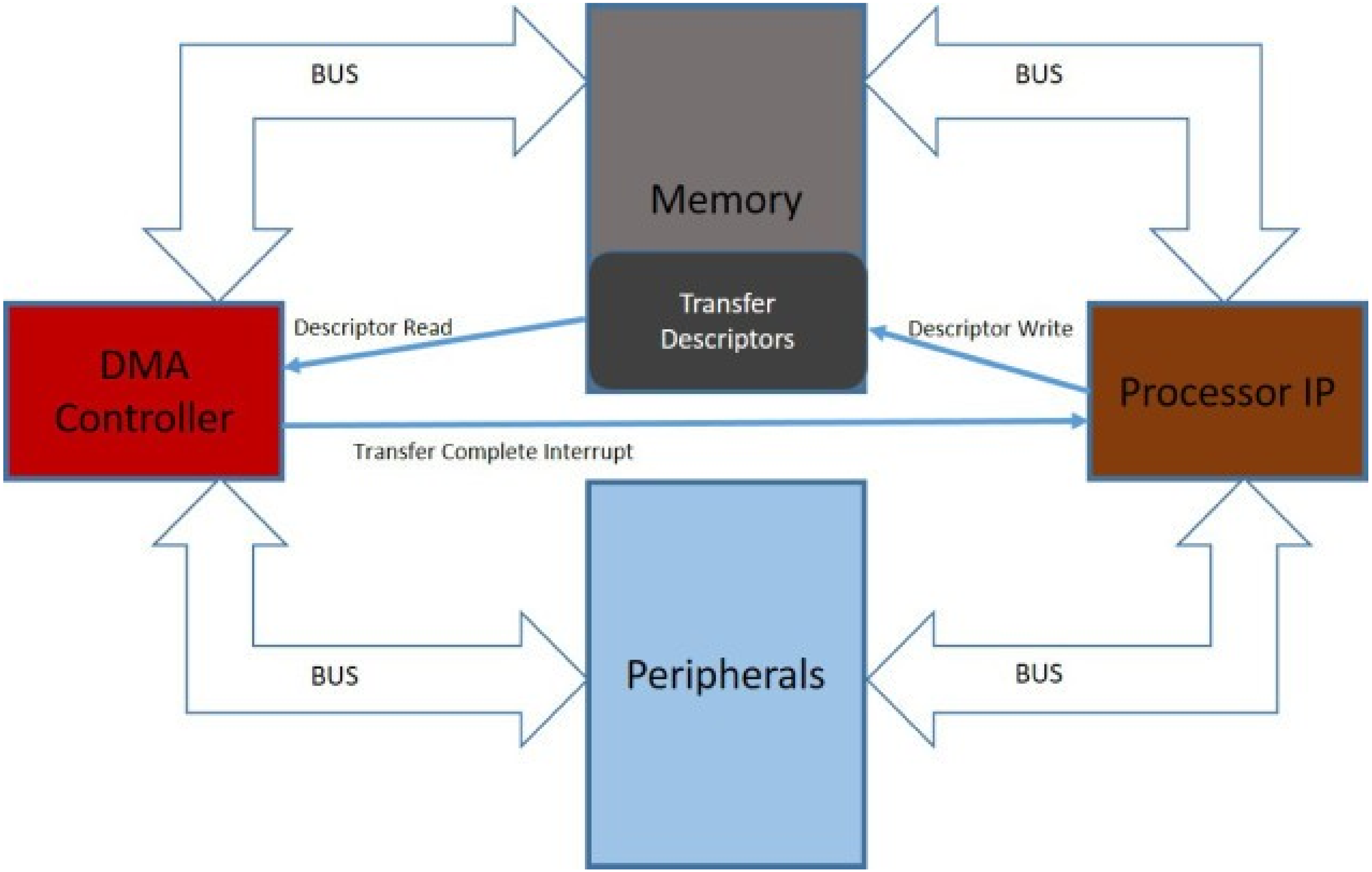
https://elixir.bootlin.com/linux/latest/source/include/linux/dmaengine.h

https://github.com/torvalds/linux/blob/master/include/linux/dma-mapping.h

DMA controller hardware

- Circuit implementing load-store loop
- Driver software runs on CPU
- Copies descriptor from RAM to controller device register to start copying
  - Source, destination
  - Burst size
  - Length of transfer
- Separate channels have separate hardware registers to provide concurrency
- Raises interrupt once transfer is done

Descriptor in RAM



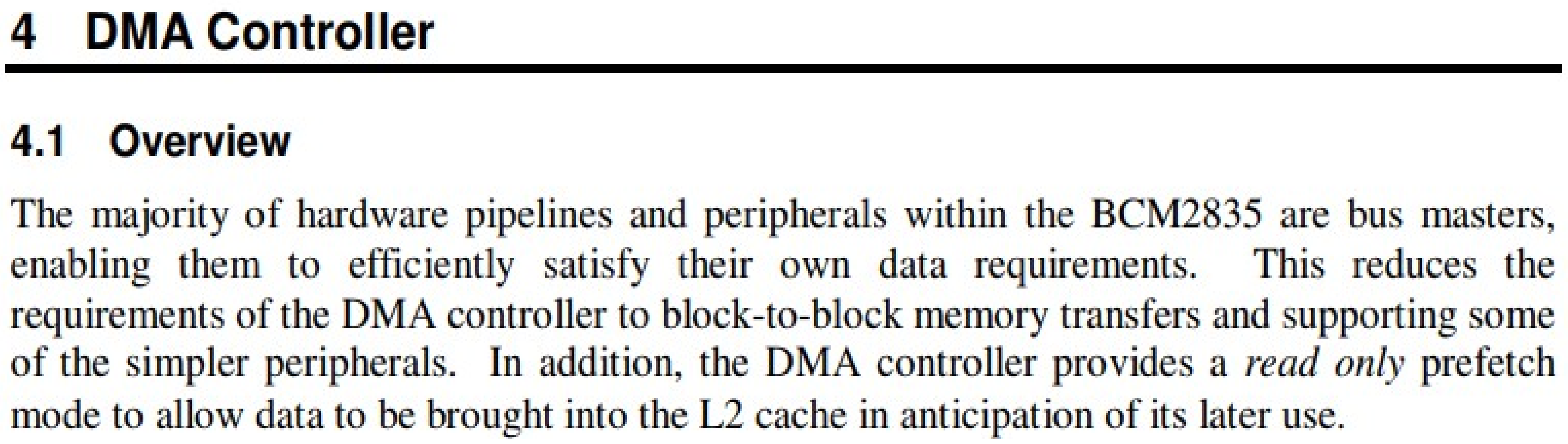
DMA controller driver code

https://github.com/torvalds/linux/blob/master/drivers/dma/bcm2835-dma.c

- Gives 14 DMA channels
- Calls many of the linux dma mapping api-s that we saw in Monday class
  - dma_pool_create, dma_pool_alloc
- Different peripherals use different channels
  - PCM/I2S audio
  - PWM DMA channel 5
  - SPI uses 2 DMA channels

https://www.raspberrypi.org/app/uploads/2012/02/BCM2835-ARM-Peripherals.pdf

First Party DMA or bus controller



4 DMA Controller

4.1 Overview

The majority of hardware pipelines and peripherals within the BCM2835 are bus masters, enabling them to efficiently satisfy their own data requirements. This reduces the requirements of the DMA controller to block-to-block memory transfers and supporting some of the simpler peripherals. In addition, the DMA controller provides a *read only* prefetch mode to allow data to be brought into the L2 cache in anticipation of its later use.