Designing for Raspberry Pi Compute Modules and cameras

Raspberry Pi Ltd

2023-12-11: githash: 4c61fd9-clean

Colophon

build-date: 2023-12-11
build-version: githash: 4c61fd9-clean

Legal Disclaimer Notice

TECHNICAL AND RELIABILITY DATA FOR RASPBERRY PI PRODUCTS (INCLUDING DATASHEETS) AS MODIFIED FROM TIME TO TIME ("RESOURCES") ARE PROVIDED BY RASPBERRY PI LTD ("RPL") "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN NO EVENT SHALL RPL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE RESOURCES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

RPL reserves the right to make any enhancements, improvements, corrections or any other modifications to the RESOURCES or any products described in them at any time and without further notice.

The RESOURCES are intended for skilled users with suitable levels of design knowledge. Users are solely responsible for their selection and use of the RESOURCES and any application of the products described in them. User agrees to indemnify and hold RPL harmless against all liabilities, costs, damages or other losses arising out of their use of the RESOURCES.

RPL grants users permission to use the RESOURCES solely in conjunction with the Raspberry Pi products. All other use of the RESOURCES is prohibited. No licence is granted to any other RPL or other third party intellectual property right.

HIGH RISK ACTIVITIES. Raspberry Pi products are not designed, manufactured or intended for use in hazardous environments requiring fail safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, weapons systems or safety-critical applications (including life support systems and other medical devices), in which the failure of the products could lead directly to death, personal injury or severe physical or environmental damage ("High Risk Activities"). RPL specifically disclaims any express or implied warranty of fitness for High Risk Activities and accepts no liability for use or inclusions of Raspberry Pi products in High Risk Activities.

Raspberry Pi products are provided subject to RPL's Standard Terms. RPL's provision of the RESOURCES does not expand or otherwise modify RPL's Standard Terms including but not limited to the disclaimers and warranties expressed in them.

Document version history

Release	Date	Description
1.0	11 Dec 2023	• Initial release

Scope of document

This document applies to the following Raspberry Pi products:

Pi Zero			Pi 1				Pi 2		Pi 3			Pi 4	Pi 400	Pi 5	CM1	CM3	CM4	Pico
Zero	W	H	A	B	A+	B+	A	B	B	A+	B+	All	All	All	All	All	All	All
															*	*	*	

Introduction

When designing a baseboard for Raspberry Pi Compute Modules that requires the use of the CSI-2 camera ports, there are some constraints you need to consider during the design process.

NOTE

Raspberry Pi does not recommend that new designs are based around Raspberry Pi Compute Module 1, 3, or 3+. Raspberry Pi Compute Module 4 and newer are the recommended devices for new designs. While this document applies to all models of Compute Module, it is targeted mainly at CM4-based devices.

This document explains what constraints exist, and how to design with them.

This white paper assumes that the Compute Module is running Raspberry Pi OS, and is fully up to date with the latest firmware and kernels. Raspberry Pi Ltd strongly advises moving to the new `libcamera` API framework for any new work. The older camera APIs will not be available on future products.

Overview of the system

Camera interfaces

The SoCs on the Compute Modules have two Camera Serial Interfaces version 2 (CSI-2) hardware interfaces (see https://www.mipi.org/specifications/csi-2), and can therefore support two active CSI-2 cameras. CSI-2 is a protocol specification; it does not cover any software topics.

NOTE

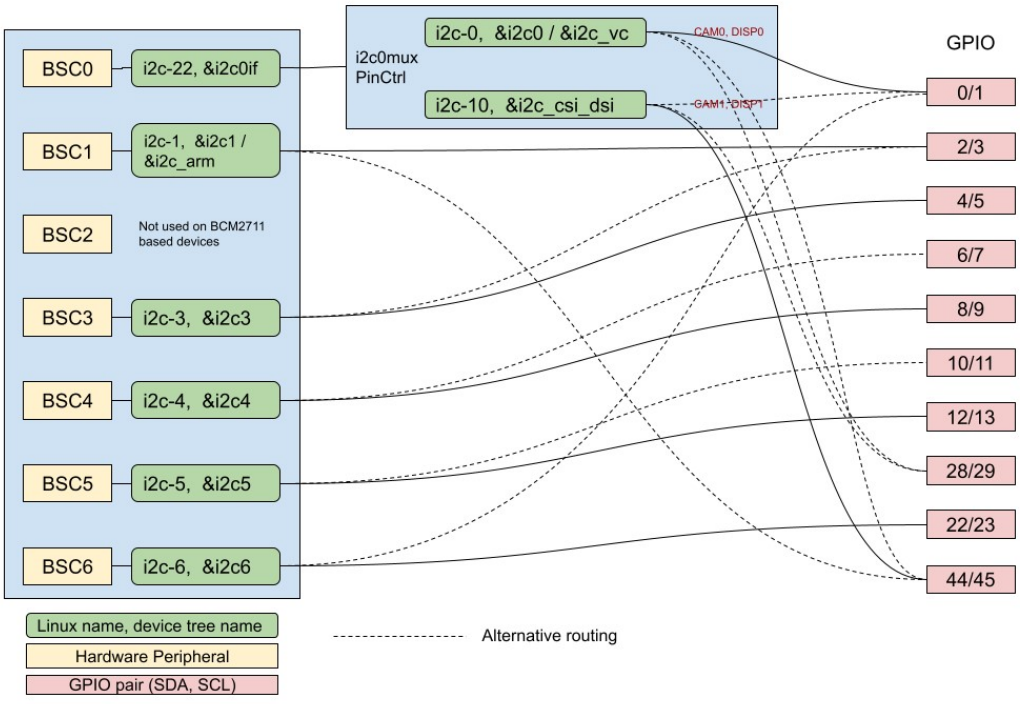
Although only two cameras can be active at any one time, CSI-2 muxes are supported, so more cameras can be attached, with only two able to supply images at any one time.

The CSI-2 ports are used to transfer the image data from the camera sensor to the Compute Module. In addition, the camera must also be connected to one of the Compute Module's I2C or SPI ports; these interfaces are used for command data that is sent to the camera to set it up, start it streaming, etc.

Raspberry Pi Ltd sells several CSI-2-based cameras that cover most use cases. All of these cameras are fully supported at the hardware and software levels. When used with the Raspberry Pi Compute Module 4 IO board, and using a default version of Raspberry Pi OS, the cameras work with only minor configuration changes.

I2C routing

The most complex part of designing a board using Raspberry Pi Compute Module 4 and a camera is probably the I2C routing. There are many options available, and this complexity can cause problems when designing both the PCB and the accompanying software. The following diagram gives an overview of the I2C routing that is possible using the BCM2711 SoC on Compute Module 4.



The Broadcom Serial Control bus is a proprietary bus compliant with the Philips® I2C bus/interface version 2.1 January 2000. The BCM2711 has seven BSC devices, which we shall refer to as I2C devices in this document. More details of the BSC hardware blocks can be found in chapter 3 of the BCM2711 datasheet, here:

https://datasheets.raspberrypi.com/bcm2711/bcm2711-peripherals.pdf

NOTE

On devices before Raspberry Pi Compute Module 4, there are only three BSC devices, BSC0-2. BSC2 is reserved for use by the HDMI DDC channel.

Control of the routing is via device tree. The base device tree files will define the default routing, but that can be overridden with `dtoverlay` commands, or a custom base file can be used for specific boards.

The use of the `pinctrl-mux` on BSC0 requires further explanation. The SoCs used in the earlier Raspberry Pi models only had three I2C ports: `i2c-1` exposed on the GPIO header, `i2c-2` used for HDMI DDC communications, and `i2c-0` used by the firmware for HAT probing (on GPIOs 0 and 1), for the camera, and for the display. To avoid removing any of the functionality in switching to Linux kernel control of the camera and display, multiplexing was necessary to provide enough ports for the required camera and display peripherals. The kernel provides the `pinctrl-mux` module to handle exactly this situation where the hardware supports multiple routings of hardware blocks to physical pins.

Although there is only one piece of I2C hardware (BSC0), the driver multiplexes the output to provide two sub-devices, named `i2C-0` and `i2c-10`. BSC0 is regarded as a parent device (which appears as i2c-22 to Linux; this should not be used directly). This means the throughput is reduced as the drivers need to reassign the GPIO muxing appropriately for each transfer, but since these lines are usually used to control the cameras and DSI displays which are low-bandwidth, this is an acceptable compromise. The principles of driver muxing are explained here: https://www.kernel.org/doc/html/latest/i2c/i2c-topology.html.

NOTE

The 28/29 mapping is largely irrelevant on BCM2711-based devices with Ethernet, as those are routed to the Ethernet PHY rather than exposed. They could be used on a Raspberry Pi Compute Module 4S device which does not have Ethernet support.

I2C dtoverlay options

If you are using the Raspberry Pi-supplied device tree files, then you can use the `dtoverlay` command in config.txt to adjust the routing as necessary.

To reassign pins to the alternative routing:

```
dtoverlay = i2c<N>, pins_A_B
```

e.g.

```
dtoverlay = i2c1, pins_44_45
```

To swap the GPIO assignments on the i2c-0 and i2c-10 devices, use:

```
dtoverlay=cm_swap_i2c
```

This results in `i2c-10` on GPIOs 0/1 and `ic2-0` on GPIOs 44/45.

Installing one or two cameras on the CM 4 IO board

NOTE

Although standard, standalone Raspberry Pi SBCs have camera autodetection (enabled by default by the `camera_auto_detect=1` line in `config.txt`), this has no effect on the Compute Modules, as the firmware does not include camera support. It is expected that Compute Module 4 users will either set up the camera using entries in the `config.txt` file as described below, or create a dedicated base device tree file for their custom board.

Edit the `config.txt` file using your favourite editor, and add the following Device Tree Overlay (`dtoverlay`) commands to the end of the file:

```
dtoverlay=imx477
dtoverlay=imx708,cam0
```

These overlay commands load the driver for the IMX477 (Raspberry Pi High Quality Camera) to cam1, and the driver for the IMX708 (Camera Module 3) to cam0. The CSI port for the camera is specified after the camera type. Omitting the port defaults to cam1.

Raspberry Pi provides the following drivers which can be used for all current and older Raspberry Pi cameras:

Sensor	Device tree name	Product
Omnivision OV5647	ov5647	Camera Module
Sony IMX219	imx219	Camera Module 2
Sony IMX708	imx708	Camera Module 3
Sony IMX477	imx477	High Quality Camera
Sony IMX296	imx296	Global Shutter Camera

Other drivers are also available for third-party boards.

Designing a custom baseboard for use with cameras

Constraints

The I2C diagram from a previous section shows which GPIOs can be assigned to which I2C driver. There is added complexity with the muxing on BSC0.

When using `libcamera` and the Linux kernel camera drivers, the I2C ports used for the camera and displays can also be used for other devices. This was not possible when using the older legacy camera drivers where the firmware controlled the I2C. However, the overall bandwidth available is reduced due to the muxing stage.

Assigning ports with device tree

It is possible to reassign the I2C ports used by `libcamera` to any of the other ports indicated on the diagram above; however, this will require custom overlays, as the standard device tree is set up for the defaults that Raspberry Pi Ltd uses in-house.

As an example of what you will need to change, here is an extract from the device tree overlay `imx219-overlay.dts` which is located in the Raspberry Pi Ltd Linux kernel source tree `https://github.com/raspberrypi/linux`. Device tree overlays are in the `linux/arch/arm/boot/dts/overlays` folder.

```
i2c_frag: fragment@100 {
    target = <&i2c_csi_dsi>;
    __overlay__ {
        #address-cells = <1>;
        #size-cells = <0>;
        status = "okay";

        #include "imx219.dtsi"

        vcm: ad5398@c {
            compatible = "adi,ad5398";
            reg = <0x0c>;
            status = "disabled";
            VANA-supply = <&cam1_reg>;
        };
    };
};
```

We can see that for the default camera (which is cam1), the I2C target is `i2c_csi_dsi`. Later on in the file, we have an overrides section:

```
    cam0 = <&i2c_frag>, "target:0=",<&i2c_csi_dsi0 >,
           <&csi_frag>, "target:0=",<&csi0>,
           <&clk_frag>, "target:0=",<&cam0_clk>,
           <&cam_node>, "clocks:0=",<&cam0_clk>,
```

```
                <&cam_node>, "VANA-supply:0=",<&cam0_reg>,
                <&vcm>, "VANA-supply:0=", <&cam0_reg>;
```

Looking at the `i2c_frag` we see it overrides `target` in the fragment with an offset of 0 (not used). This of course is the I2C controller, and it is set to `i2c_csi_dsi0`, the default for camera 0.

NOTE

`i2c_csi_dsi0` used to be named `i2c_vc` but has been changed for Raspberry Pi 5 compatibility.

You can develop your overlays to reassign the I2C controllers to reflect your hardware, as long as you keep to the routing rules shown in the diagram.

So, if you are using cam1, but want it on BSC3:

```
i2c_frag: fragment@100 {
    target = <&i2c3>;
    __overlay__ {
        #address-cells = <1>;
        #size-cells = <0>;
        status = "okay";

        #include "imx219.dtsi"

        vcm: ad5398@c {
            compatible = "adi,ad5398";
            reg = <0x0c>;
            status = "disabled";
            VANA-supply = <&cam1_reg>;
        };
    };
};
```

It is worth pointing out at this point how the CSI peripherals are managed.

```
csi_frag: fragment@101 {
    target = <&csi1>;
    csi: __overlay__ {
        status = "okay";
        brcm,media-controller;

        port {
            csi_ep: endpoint {
                remote-endpoint = <&cam_endpoint>;
                clock-lanes = <0>;
                data-lanes = <1 2>;
                clock-noncontinuous;
            };
        };
    };
};
```

Raspberry Pi's advice when developing your baseboard is to develop one overlay that defines all the attached hardware, rather than requiring multiple 'dtoverlay' lines to configure each part automatically or having a custom base DT file.

Example hardware schematics

The Raspberry Pi Compute Module 4 IO board schematics are an excellent reference to the hardware design required when building your own baseboard.

The schematics are in the datasheet, which can be found here:

https://datasheets.raspberrypi.com/cm4io/cm4io-datasheet.pdf