Using the I²S peripherals on Raspberry Pi SBCs

Colophon

Release	1
Build date	20/02/2026
Build version	e3526fb0ec27

Legal disclaimer notice

Document version history

Release	Date	Description
1	10 Feb 2026	Initial release

Scope of document

This document applies to the following Raspberry Pi products:

Single Board Computers / SBCs

Pi Zero			Pi Zero 2		Pi 1		Pi 2		Pi 3	Pi 4	Pi 5
-	W	H	W	H	A	B	A	B	B	-	-
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Compute Modules

CM1	CM3	CM4	CM5
✓	✓	✓	✓

Keyboard Computers

Pi 400	Pi 500	Pi 500+
✓	✓	✓

Introduction

This document describes how to use the I²S interfaces on Raspberry Pi Ltd single-board computers (SBCs). It covers basic connection, setup, and how to define your own device tree for more unusual I²S devices.

What is I²S?

I²S, or Inter-IC Sound, is a serial bus interface standard used to digitally transmit audio data between integrated circuits (ICs). It is a three- or four-line protocol that separates the clock and data signals, making it ideal for digital audio components like microcontrollers, digital-to-analogue converters (DACs), and digital signal processors (DSPs). Unlike its similar-sounding counterpart, I²C, which is a general-purpose communication protocol, I²S is specifically designed for audio.

Producers and consumers

I²S devices can be either producers or consumers. The main difference is that the I²S producer generates the clock signals and controls the timing, while the consumer derives its clock from the producer's signals. You can't connect two producers together on a single I²S bus; a single producer is paired with one or more consumers for audio communication. Both modes can transmit and receive data, but only the producer is responsible for providing the timing signals.

All Raspberry Pi Ltd SBCs can act as either a producer or a consumer.

Lanes

I²S lanes are the individual signal lines used in the I²S digital audio interface (DAI). There are normally three required lanes and one optional lane:

SD (serial data), required

Carries the audio samples. Usually one data line for stereo (left and right are time-multiplexed). Some devices add extra SD lines for multichannel audio.

BCLK/SCK (bit clock/serial clock), required

Tells the receiver when to read each bit. One clock pulse per audio data bit. Bit rate = sample rate × bits per sample × number of channels.

WS/LRCK/FS (word select/left-right clock/frame sync), required

Indicates whether the current SD bits belong to the left or right audio channel. Also matches the audio sample rate (e.g. 44.1 kHz, 48 kHz).

MCLK (master clock), optional

A high-frequency clock (e.g. 256× or 512× the sample rate). Needed by some digital-to-analogue converters (DACs) and analogue-to-digital converters (ADCs) for their internal oversampling filters. Not required in all I²S implementations and NOT supported on any Raspberry Pi SBCs.

Summary

Table 1.
I²S lanes

Lane	Name(s)	Purpose
SD	Serial data	Audio bitstream
BCLK/SCK	Bit clock/serial clock	Synchronises each audio bit
WS/LRCK	Word select/left-right clock	Indicates left versus right channel
MCLK	Master clock	Optional system clock for codecs (not supported)

Channels

Channels refer to audio channels carried over the data line (SD). Usually, I²S uses one data lane, and channels sent are time-multiplexed:

- When WS/LRCK = 0, the bits on the data line correspond to the left channel
- When WS/LRCK = 1, the bits on the data line correspond to the right channel

So the hardware only needs one SD line for stereo.

If you need more than two channels, there are two options: some hardware solutions implement multiple data lines, often referred to as SD0, SD1, and SD2; alternatively, it's possible to divide a single SD line into multiple time slots, with each slot representing one audio channel. This is known as time-division multiplexing, or TDM. Raspberry Pi 5 uses the former approach; TDM is NOT supported on any Raspberry Pi SBCs.

Input and output

I²S is a unidirectional, clock-synchronous digital audio bus, and the lanes/pins are either inputs (receiving signals) or outputs (driving signals).

An I²S output generates the I²S signals and sends audio data out of the device. All of the lanes referred to in the 'Lanes' section are driven by the peripheral. Typical output devices include microcontrollers, USB audio interfaces, digital audio players, and the Raspberry Pi SBC itself.

An I²S input listens to the I²S signals and accepts audio data into the device. Typical examples might be ADCs, DSPs, codecs, etc. A Raspberry Pi SBC can also act as an I²S receiver.

An I²S output must be directly connected to an I²S input — you cannot connect an output to another output, or an input to another input. In addition, there should only be one driver for the clock signal. In most cases, producers are output devices and consumers are input devices.

The I²S peripherals

On Raspberry Pi 4 Model B and earlier, the I²S peripheral is built into the main system-on-chip (SoC). All of these models have the same level of I²S support. On Raspberry Pi 5, the I²S peripheral is implemented on the RP1 I/O controller and is an entirely different device. The Linux kernel drivers insulate end users from this difference, so they all appear broadly the same.

Hardware specifications

All Raspberry Pi SBCs support producer and consumer modes.

Raspberry Pi 4 Model B and earlier

There is one I2S peripheral on these boards, with one bidirectional data lane assigned to two GPIO pins to separate input and output.

GPIO assignments

Table 2.
GPIO mux ALT0

GPIO	I²S name	Raspberry Pi name
18	BCLK/SCK	PCM_CLK
19	WS/LRCK	PCM_FS
20	SD (in)	PCM_DIN
21	SD (out)	PCM_DOUT

Raspberry Pi 5

The RP1 I/O controller on Raspberry Pi 5 provides the I²S interfaces.

Warning

Although RP1 supports three I²S peripherals, only two are connected to usable pins on Raspberry Pi 5 and Raspberry Pi Compute Module 5.

Raspberry Pi 5 increases the number of data lanes on the peripheral to four:

- I2S0 is a clock producer with up to four bidirectional channels
- I2S1 is a clock consumer with up to four bidirectional channels

GPIO assignments

Table 3.
GPIO mux A2 (/dev/I2S0)

GPIO	I²S name	Raspberry Pi name
18	BCLK/SCK	I2S0_SCLK
19	WS/LRCK	I2S0_WS
20	SD (in)	I2S0_SDI[0]
21	SD (out)	I2S0_SDO[0]
22	SD (in)	I2S0_SDI[1]

GPIO	I²S name	Raspberry Pi name
23	SD (out)	I2S0_SDO[1]
24	SD (in)	I2S0_SDI[2]
25	SD (out)	I2S0_SDO[2]
26	SD (in)	I2S0_SDI[3]
27	SD (out)	I2S0_SDO[3]

Table 4.
GPIO mux A4 (/dev/I2S1)

GPIO	I²S name	Raspberry Pi name
18	BCLK/SCK	I2S1_SCLK
19	WS/LRCK	I2S1_WS
20	SD (in) channel 0	I2S1_SDI[0]
21	SD (out) channel 0	I2S1_SDO[0]
22	SD (in) channel 1	I2S1_SDI[1]
23	SD (out) channel 1	I2S1_SDO[1]
24	SD (in) channel 2	I2S1_SDI[2]
25	SD (out) channel 2	I2S1_SDO[2]
26	SD (in) channel 2	I2S1_SDI[3]
27	SD (out) channel 2	I2S1_SDO[3]

I2S0 and I2S1 occupy the same set of GPIO pins in the mux map. Users should select the I²S instance (producer or consumer) according to their attached codec. Raspberry Pi 5 cannot act as both at the same time.

Installing I^{2}S devices

Typical connections

Below is a connection table for a simple I^{2}S device based on an easily available PCM5102-based board. In this example, the Raspberry Pi is a producer and the attached device is a consumer.

Figure 1.
PCM5102-based device

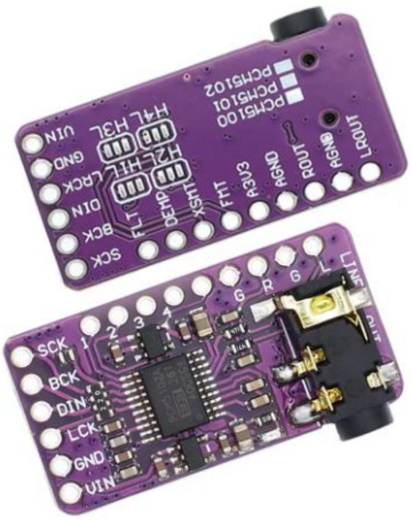

Table 5.
GPIO mux A4 (/dev/I2S1)

DAC board	GPIO header	GPIO number
SCK	N/C	
BCK	12	18
DIN	40	21
LRCK	35	19
GND	6	GND
VIN	2	5V

> **Note**
>
> PCM5102 is an I^{2}S DAC chip. It is often used in audio modules, providing excellent sound quality with a high dynamic range and signal-to-noise ratio. The chip decodes digital audio signals like I^{2}S and outputs them as stereo analogue signals, which can be used with an external amplifier or a headphone jack. Though boards using PCM5102 are commonly available, it is not recommended for new products.

Software/device tree

A device tree is a data structure that describes the hardware in a computer system to the operating system's kernel, allowing the kernel to be generic rather than hardcoded for specific hardware. It's a standardised way to describe hardware components like CPUs, memory, buses, and peripherals, enabling the kernel to effectively manage them. This approach ensures that a single kernel image can support various hardware configurations while keeping drivers independent of specific hardware details.

Raspberry Pi OS uses the device tree extensively to target the Linux kernel at the specific hardware being used, e.g. the model of Raspberry Pi SBC.

Raspberry Pi Ltd also uses device tree overlays to customise the kernel at run time, overriding any board-specific definitions. To set up a Raspberry Pi device for I^{2}S, you need to use a device tree overlay.

Usually, device tree and device tree overlay settings are enabled through entries in the `config.txt` file:

Code
```
sudo nano /boot/firmware/config.txt
```

You can configure the particular device using device tree overlays:

Code
```
# Configures any generic, passive I#super("2")S DAC sound card
dtoverlay=i2s-dac

# Configures a generic I#super("2")S DAC sound card that acts as a clock master
dtoverlay=i2s-master-dac
```

If your device isn't a simple one, you may need to add a device tree overlay for a specific sound card, such as:

Code
```
dtoverlay=hifiberry-dacplus
```

List of available I^{2}S device tree overlays

At the time of writing, the following device tree overlays are available for specific sound cards:

Table 6.
I^{2}S device tree overlays

Overlay name	Description
akkordion-iqdacplus	Digital Dreamtime Akkordion Music Player (based on the OEM IQAudIO DAC+ or DAC Zero module)
allo-boss-dac-pcm512x-audio	Allo Boss DAC audio card
allo-boss2-dac-audio	Allo Boss2 DAC audio card
allo-katana-dac-audio	Allo Katana DAC audio card
allo-piano-dac-pcm512x-audio	Allo Piano DAC (2.0/2.1) audio card (note: initial support is for Piano 2.0 channel audio ONLY, ie. stereo; subwoofer outputs on the Piano 2.1 are not currently supported)
allo-piano-dac-plus-pcm512x-audio	Allo Piano DAC+ (2.1) audio card
applepi-dac	Orchard Audio ApplePi-DAC audio card
chipdip-dac	Chip Dip audio card
dacberry400	DacBerry400 sound card
hifiberry-dac	HiFiBerry DAC audio card
hifiberry-dac8x	HiFiBerry DAC8x audio card (only on Raspberry Pi 5); this driver also detects a stacked HiFiBerry ADC8x and activates the capture capabilities (note: for standalone use of an ADC8x, activate the ADC8x module)
hifiberry-dacplus	HiFiBerry DAC+ audio card
hifiberry-dacplus-pro	HiFiBerry DAC+ PRO audio card (with on-board clocks)
hifiberry-dacplus-std	HiFiBerry DAC+ standard audio card (no on-board clocks)
hifiberry-dacplusadc	HiFiBerry DAC+ ADC audio card
hifiberry-dacplusadcpro	HiFiBerry DAC+ ADC PRO audio card
hifiberry-dacplusdsp	HiFiBerry DAC+ DSP audio card
hifiberry-dacplushd	HiFiBerry DAC+ HD audio card

Overlay name	Description
iqaudio-dac	IQaudio DAC audio card
iqaudio-dacplus	IQaudio DAC+ audio card
justboom-dac	JustBoom DAC HAT, Amp HAT, DAC Zero, and Amp Zero audio cards
mbed-dac	mbed audio codec and DAC (TLV320AIC23B)
pifi-dac-hd	PiFi DAC HD audio HAT
pifi-dac-zero	PiFi DAC Zero audio board
rpi-dacplus	Raspberry Pi DAC+ audio HAT
rpi-dacpro	Raspberry Pi DAC Pro audio HAT
rra-digidac1-wm8741-audio	Red Rocks Audio DigiDAC1 sound card

Some of these devices will take parameters to the `dtoverlay` command to enable customisation of the driver. Lists of these parameters can be found by typing the following on the command line:

Code
```
dtoverlay -h <name of overlay>
```

Device tree in more detail

The device tree hierarchy, which describes the arrangement of devices in great detail, is complicated. This section will attempt to explain how the sections supporting I²S audio fit together.

A device tree needs to define what hardware is present, what driver is needed to support that hardware, and what facilities are needed by that driver. For example, Raspberry Pi 5 has an on-board I²S controller with DesignWare IP from Synopsys, so it needs a DesignWare driver. In addition, the driver's location in the CPU memory map, its connections to the GPIO pins, its requirements for a clock, and its support for DMA all need to be defined in the device tree.

device tree files are part of the Linux kernel source tree. The Raspberry Pi OS 6.12 kernel source can be found here. device tree definitions for 64-bit devices can be found here.

Raspberry Pi 5, Raspberry Pi Compute Module 5, Raspberry Pi 500, Raspberry Pi 500+

Listing 1.
Extract from https://github.com/raspberrypi/linux/blob/rpi-6.12.y/arch/arm64/boot/dts/broadcom/rp1.dtsi

```
rp1_i2s0: i2s@a0000 {
    reg = <0xc0 0x400a0000  0x0 0x1000>;
    compatible = "snps,designware-i2s";
    // Providing an interrupt disables DMA
    // interrupts = <RP1_INT_I2S0 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&rp1_clocks RP1_CLK_I2S>;
    clock-names = "i2sclk";
    #sound-dai-cells = <0>;
    dmas = <&rp1_dma RP1_DMA_I2S0_TX>,<&rp1_dma RP1_DMA_I2S0_RX>;
    dma-names = "tx", "rx";
    dma-maxburst = <4>;
    status = "disabled";
};
```

Listing 1 shows the definition of I²S peripheral instance 0 on RP1. There are two further similar entries that define instance 1 and instance 2 — the only real difference is the memory map entry.

> **Warning**
>
> Instance 2 is not accessible on any Raspberry Pi Ltd devices.

Let's explain some of the contents:

Table 7.
Device tree properties

Property	Description
reg	Defines the area of the memory map where the device resides
compatible	Matches a specific hardware device to the correct driver in the operating system
clocks	Points to the clock that drives the peripheral
clock-names	Provides human-readable names for the clocks defined in the 'clocks' property
dmas	Lists the DMAs to be used
dma-names	Provides human-readable names for the DMAs defined in the 'dmas' property

We can see that the required hardware driver is called `designware-i2s` , which matches up with the `dwc-i2s.c` file in the Linux kernel tree here. You can compare the compatible string in this file with the string found in the device tree file and see that they match.

Also in the `rp1.dts` file are two entries that define the GPIO pins used by the driver:

Listing 2.
Extract from https://github.com/raspberrypi/linux/blob/rpi-6.12.y/arch/arm64/boot/dts/broadcom/rp1.dtsi

```
rp1_i2s0_18_21: rp1_i2s0_18_21 {
    function = "i2s0";
    pins = "gpio18", "gpio19", "gpio20", "gpio21";
    bias-disable;
};

rp1_i2s1_18_21: rp1_i2s1_18_21 {
    function = "i2s1";
    pins = "gpio18", "gpio19", "gpio20", "gpio21";
    bias-disable;
};
```

From Listing 2, we see that `i2s0` and `i2s1` both use the same set of GPIO pins: 18 to 21. Also note that this only caters for the first bidirectional data lane on RP1.

Table 8.
Selected PinCtrl device tree entries

Property	Description
function	Defines the intended purpose or role of a hardware component
pins	A key-value pair used to configure the multiplexing and configuration of hardware pins for a device; in this case, rather than defining parameters, there's a name reference in a board definition file, for example: https://github.com/raspberrypi/linux/blob/rpi-6.12.y/arch/arm64/boot/dts/broadcom/bcm2712-rpi-5-b.dts
bias-disable	A boolean property used to disable any internal pull-up or pull-down resistors

We now need to look at another device tree file — one that defines the hardware used in BCM2712 SoC–based devices (e.g. Raspberry Pi 5, Raspberry Pi Compute Module 5, Raspberry Pi 500, Raspberry Pi 500+).

Listing 3.
Extract from https://github.com/raspberrypi/linux/blob/rpi-6.12.y/arch/arm64/boot/dts/broadcom/bcm2712-rpi.dtsi

```
&i2s_clk_producer {
  pinctrl-names = "default";
```

```
  pinctrl-0 = <&rp1_i2s0_18_21>;
};

&i2s_clk_consumer {
  pinctrl-names = "default";
  pinctrl-0 = <&rp1_i2s1_18_21>;
};
```

Listing 3 defines two properties that in turn define, via reference, the GPIO pins for I²S clock producers and clock consumers. Note that the names are preceded by an ampersand (&); this signals that this is a reference to a pre-existing node, so the data is merged with the original definition. For the original definition, see the next section.

Table 9.
Selected PinCtrl device tree entries

Property	Description
pinctrl-names	Assigns a human-readable name to each pin control state, represented by pinctrl-0, pinctrl-1, and so on
pinctrl-0	Points to a `pinctrl` node (see definition in Listing 2)

The final piece in the puzzle is the original definitions for the I²S.

Listing 4.
Extract from https://github.com/raspberrypi/linux/blob/rpi-6.12.y/arch/arm64/boot/dts/broadcom/bcm2712-rpi.dtsi

```
i2s:  &rp1_i2s0 { };
i2s_clk_producer: &rp1_i2s0 { };
i2s_clk_consumer: &rp1_i2s1 { };
```

This segment declares three things:

1. `i2s_clk_producer` refers to `rp1_i2s0` (see Listing 1)
2. `i2s_clk_consumer` refers to `rp1_i2s1` (see Listing 1)
3. `i2s` refers to `rp1_i2s0` and aliases `i2s_clk_producer` (see Listing 1)

So far, we have defined where the hardware is and which GPIO pins and clocks it needs, and created three nodes that can be referenced to enable it. The final step is to add a device tree overlay that enables the hardware. device tree overlays are device tree fragments that are used on Raspberry Pi devices to inform the system about any hardware attached to it.

Let's take a look at the `i2s-master-dac` we referenced earlier (loaded via `dtoverlay=i2s-master-dac` in `config.txt`). This is a device tree overlay that can be attached to the base device tree definition to add or modify hardware configurations.

Listing 5.
Entire contents of https://github.com/raspberrypi/linux/blob/rpi-6.12.y/arch/arm/boot/dts/overlays/i2s-master-dac-overlay.dts

```
// Definitions for a generic #I2S DAC that acts as the clock master on the bus.
/dts-v1/;
/plugin/;

/ {
  compatible = "brcm,bcm2835";

  fragment@0 {
    target = <&i2s_clk_consumer>;
    __overlay__ {
      status = "okay";
    };
  };

  fragment@1 {
    target-path = "/";
    __overlay__ {
```

```
      codec_bare: codec_bare {
        compatible = "linux,spdif-dit";
        #sound-dai-cells = <0>;
        status = "okay";
      };
    };
  };

  fragment@2 {
    target = <&sound>;
    __overlay__ {
      compatible = "simple-audio-card";
      i2s-controller = <&i2s_clk_consumer>;
      status = "okay";

      simple-audio-card,name = "i2s-master-dac";
      simple-audio-card,format = "i2s";

      simple-audio-card,bitclock-master = <&snd_codec>;
      simple-audio-card,frame-master = <&snd_codec>;

      simple-audio-card,cpu {
        sound-dai = <&i2s_clk_consumer>;
        dai-tdm-slot-num = <2>;
        dai-tdm-slot-width = <32>;
      };

      snd_codec: simple-audio-card,codec {
        sound-dai = <&codec_bare>;
      };
    };
  };
};
```

There is a bit more data to digest here.

Table 10.
Selected device tree properties/entries

Property/entry	Description
compatible	Declares which devices the configuration is compatible with
fragment	A group of changes to a particular area of the device tree; all overlays are made up of one or more fragments
target	Applies the fragment to the device tree node with the given label, e.g. ‘&i2s_clk_consumer’
target-path	Applies the fragment to the device tree node with the given absolute, slash-separated path; use `target-path` only if the node has no label, as paths are likely to be device-specific
codec_bare	Used in situations where the audio codec chip is connected directly to a bus (like an I²C, SPI, or I²S bus), but where a complex, specific audio subsystem driver is not necessary or desired; instead, it uses a simple, generic binding to represent the device
simple-audio-card	A generic device tree binding used to describe simple audio connections between an SoC’s audio interface and a codec; it simplifies sound card configuration by allowing a single generic driver to be used across different boards, with hardware connections and routing defined through a device tree overlay rather than hardcoded driver logic
simple-audio-card,name	Gives the name applied to the audio card

Property/entry	Description
simple-audio-card,format	Specifies the audio data format, such as `i2s` , `ac97` , or `pcm`
simple-audio-card,bitclock-master	Specifies which device (CPU or codec) is the producer of the bit-clock signal in a digital audio interface (DAI) link
simple-audio-card,frame-master	Identifies which device in the audio data link is responsible for generating the frame synchronisation signal
simple-audio-card,cpu	A sub-node specifically describing the SoC's role in the audio connection
simple-audio-card,cpu,sound-dai	Points to the actual CPU DAI node in the device tree; see Listing 4
simple-audio-card,cpu,dai-tdm-slot-num	The total number of time-division multiplexing (TDM) slots used by a DAI
simple-audio-card,cpu,dai-tdm-slot-width	Tells the audio system how many bits are used per audio sample within a TDM slot
snd_codec	A node or a reference to a node that describes an audio encoder or decoder hardware component; the device tree uses properties within this node to provide the operating system with all the static, non-discoverable information it needs to initialise and configure the specific audio codec chip

So, we can see that loading this device tree overlay will update the `i2s_clk_consumer` node with an `okay` status in fragment 0. In fragment 1, the `codec_bare` node also has an `okay` status added, and its compatible string is set to the `S/PDIF` driver, which will handle a lot of the underlying audio functions. Finally, fragment 2 sets up the device as it is presented to Linux, based on the `simple-audio-card` driver. The `i2s-controller` is linked to the `i2s_clk_consumer` we saw in Listing 4, and various parameters required by `simple-audio-card` are defined as per Table 10 above.

Support for extra channels on RP1

At the time of writing, there is one third-party product that takes advantage of all four lanes available on RP1: the HiFiBerry DAC8x. To the Advanced Linux Sound Architecture (ALSA), this appears as a device that can support up to eight channels (e.g. four stereo pairs) and play a single audio file with eight (four stereo) homogenous audio tracks.

The driver for this is the `rpi-simple-soundcard` device; this code was written by Raspberry Pi Ltd to aid with implementing audio drivers, abstracting a lot of the boilerplate implementation details into one file that can be used by a number of different devices. The source for this driver can be found here. Examination of the source will show that it supports a number of different hardware devices, including the DAC8x.

The device tree overlay for this device is shown in Listing 6.

Listing 6.
Extract from https://github.com/raspberrypi/linux/blob/rpi-6.12.y/arch/arm/boot/dts/overlays/hifiberry-dac8x-overlay.dts

```
compatible = "brcm,bcm2712";

fragment@0 {
  target = <&gpio>;
  __overlay__ {
    rp1_i2s0_dac8x: rp1_i2s0_dac8x {
      function = "i2s0";
      pins = "gpio18", "gpio19", "gpio20",
             "gpio21", "gpio22", "gpio23",
             "gpio24", "gpio25", "gpio26",
             "gpio27";
      bias-disable;
      status = "okay";
    };
  };
};

fragment@1 {
```

```
    target = <&i2s_clk_producer>;
    __overlay__ {
      pinctrl-names = "default";
      pinctrl-0 = <&rp1_i2s0_dac8x>;
      status = "okay";
    };
  };

  fragment@2 {
    target-path = "/";
    __overlay__ {
      dummy-codec {
        #sound-dai-cells = <0>;
        compatible = "snd-soc-dummy";
        status = "okay";
      };
    };
  };

  fragment@3 {
    target = <&sound>;
    __overlay__ {
      compatible = "hifiberry,hifiberry-dac8x";
      i2s-controller = <&i2s_clk_producer>;
      hasadc-gpio = <&gpio 5 GPIO_ACTIVE_LOW>;
      status = "okay";
    };
  };
```

This device tree overlay tells the `pinctrl` system to allocate GPIO pins 18–27 to the `i2s0` mux, and sets up the appropriate compatible strings to ensure the correct driver is loaded. Note that the `hifiberry-dac8x` compatible string matches that in the `rpi_simple_soundcard` driver.

In order to use this device, we'll need to send eight-channel data to it. This is covered briefly in the next section.

I^{2}S and ALSA

ALSA stands for Advanced Linux Sound Architecture. It is a software framework that manages sound card drivers within the Linux kernel, handling audio playback, recording, and mixing. It acts as an application programming interface (API) for developers and sits beneath higher-level sound servers like PulseAudio or PipeWire, providing managed access to hardware. It consists of kernel drivers and user-space libraries (such as alsa-lib) for applications to use.

ALSA should automatically detect any I^{2}S sound card drivers that are loaded from the device tree overlays specified in `config.txt` . There are a number of ALSA command line programs, but the first one to look at is `aplay` — this can be used to do the majority of the tasks required.

Open a terminal window and type the following for information about the ALSA system (this example uses a Raspberry Pi 5 with no extra sound devices enabled — only HDMI, which is on by default):

Code
```
$ aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: vc4hdmi0 [vc4-hdmi-0], device 0: MAI PCM i2s-hifi-0 [MAI PCM i2s-hifi-0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: vc4hdmi1 [vc4-hdmi-1], device 0: MAI PCM i2s-hifi-0 [MAI PCM i2s-hifi-0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

Note

Because there are two HDMI ports on Raspberry Pi 5, two sound devices will be listed, even if only one port is connected.

We can now play sound on the default device using the following:

```
$ aplay <path to audio file>
```

Now we can add another sound device (either on the command line, as shown below, or via `config.txt`). In this example, we'll add the DAC8x device described above:

```
$ sudo dtoverlay  hifiberry-dac8x
$ aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: vc4hdmi0 [vc4-hdmi-0], device 0: MAI PCM i2s-hifi-0 [MAI PCM i2s-hifi-0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 1: vc4hdmi1 [vc4-hdmi-1], device 0: MAI PCM i2s-hifi-0 [MAI PCM i2s-hifi-0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 2: sndrpihifiberry [snd_rpi_hifiberry_dac8x], device 0: HifiBerry DAC8x HiFi snd-soc-dummy-dai-0 [HifiBerry
DAC8x HiFi snd-soc-dummy-dai-0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

The extra sound card is now installed and listed. We can output the audio to this using the '-D' option of `aplay` . Note that we must tell the driver we are using all eight channels with `-c 8` :

```
$ aplay -D plughw:2,0 -c 8 <path to audio file>
```

Since the source file is likely either mono or stereo and we are sending it to eight channels, the output will not be particularly useful. To get a correct eight-channel output, the source file will need to contain eight channels of audio information. Tools like `ffmpeg` have functions (merge, join) that can take multiple audio files and combine them into a single multi-channel file. For example:

```
$ ffmpeg -i <infile1> -i <infile2> -i <infile3> -i <infile4> -i <infile5> -i <infile6> -i <infile7> -i <infile8> -
filter-complex join=inputs=8 <outputfile>
```

Note

For `aplay` to output directly to a device, the audio format must be one the driver understands — no conversion will be done in real time to match an appropriate format. For example, the HDMI system only supports the IEC928_SUBFRAME_LE format.

If the previous command complains that the format of the file is incompatible, one way to test it is to input white noise from the on-board random number generator and tell ALSA that it's one of the required formats. For example:

```
$ aplay -D plughw:2,0 -f S16_LE -c 8 /dev/random
```

ALSA is not the easiest system to get to grips with, as it's very low level, which is why libraries like PulseAudio and PipeWire exist. This is not the place to go into great detail about these libraries, but it is worth describing how Raspberry Pi OS is set up to provide audio to the end user.

ALSA is used at the lowest level, and above this is PipeWire — PulseAudio is not installed. PipeWire implements a PulseAudio-compatible API, so any applications using these APIs will continue to operate correctly; however, they are still using the PipeWire implementation.

Conclusion

This white paper has detailed the I^2S peripherals available on Raspberry Pi Ltd's single-board computers. Using the information provided in this document, the user should now be familiar with how to attach I^2S devices to a Raspberry Pi SBC and set up the appropriate software to drive them. Those connecting a new and unsupported device should also be familiar enough with the device tree system to provide their own overlays, though it is well worth reading the device tree documentation for more background information.

You can find a section of the Raspberry Pi Forums dedicated to device tree questions here.

Contact Details for more information

Please contact applications@raspberrypi.com if you have any queries about this whitepaper.

Web: www.raspberrypi.com

Raspberry Pi