

HiFiDUINO

Lot of Value, Little Money

- [Home](#)
- [Musiland USB](#)
- [INTRO](#)
- [Free High-Res](#)
- [CODE](#)
- [Apple Remote](#)
- [Sabre32](#)
- [OTHER](#)
- [B-II](#)
-

Type text to search here...

[Home](#) > [Embedded Computing](#) > Raspberry Pi B+ Digital Audio

Raspberry Pi B+ Digital Audio

November 13, 2014 [Leave a comment](#) [Go to comments](#)

I2S AUDIO

Although the Raspberry Pi has built-in analog audio output, the interest here is in digital audio output in particular I2S output signals for direct connection to digital to analog converters. I explored a bit the digital audio capabilities of the Raspberry Pi a while ago [[link](#)]. Here is an update with more accurate information.

The digital audio capabilities of the Raspberry Pi B+ have not changed from previous versions. The I2S audio is supported by the Broadcom BCM2835 [[link](#)] peripheral SoC chip. The datasheet shows that the PCM audio interface consist of 4 signals, notice that there is no Master Clock signal:

- PCM_CLK – bit clock.
- PCM_FS – frame sync signal. Frames can be up to 32 bit wide
- PCM_DIN – serial data input.
- PCM_DOUT – serial data output.

In addition, for more advanced configurations, the device can be configured as master or slave: “the direction of the PCM_CLK and PCM_FS signals can be individually selected, allowing the interface to act as a master or slave device”. In normal operation, it is configured as a master device.

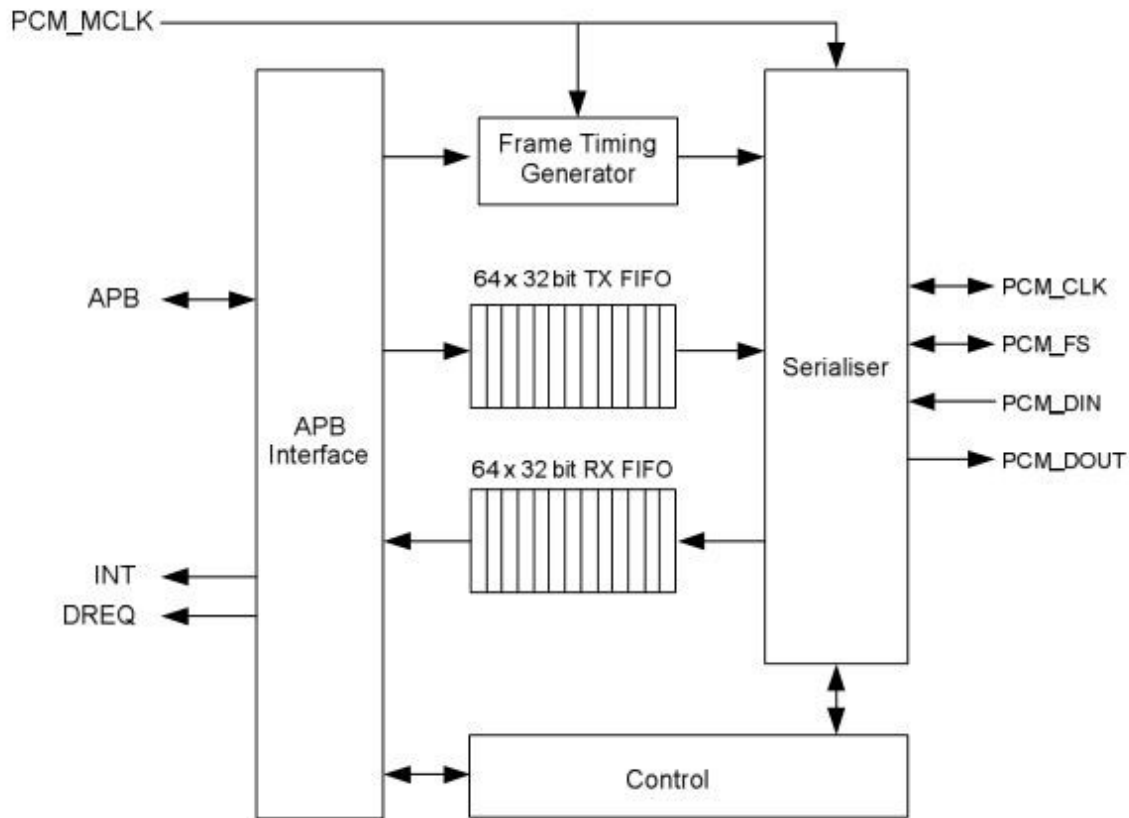


Figure 8-2 PCM Audio Interface Block Diagram

In the Raspberry Pi B+, The I2S output are assigned to the following pins:

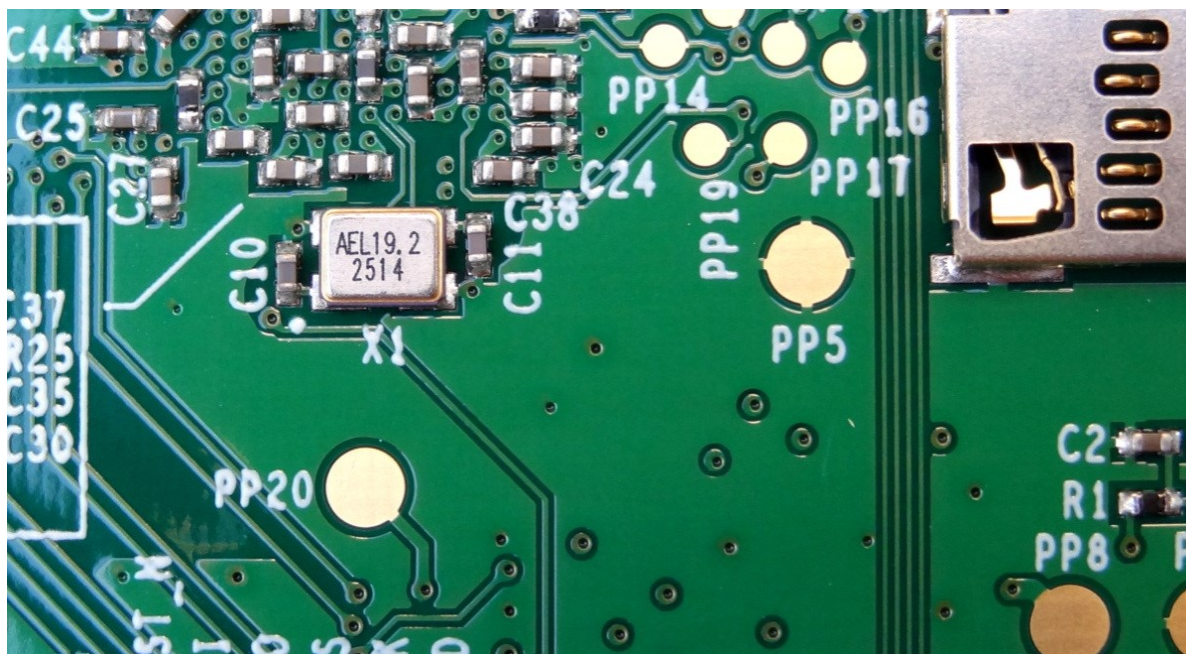
Raspberry Pi B+ J8 Header			
Pin#	NAME		NAME Pin#
01	3.3v DC Power		DC Power 5v 02
03	GPI002 (SDA1 , I2C)		DC Power 5v 04
05	GPI003 (SCL1 , I2C)		Ground 06
07	GPI004 (GPIO_GCLK)		(TXD0) GPI014 08
09	Ground		(RXD0) GPI015 10
11	GPI017 (GPIO_GEN0)		BITCLOCK GPI018 12
13	GPI027 (GPIO_GEN2)		Ground 14
15	GPI022 (GPIO_GEN3)		(GPIO_GEN4) GPI023 16
17	3.3v DC Power		(GPIO_GEN5) GPI024 18
19	GPI010 (SPI_MOSI)		Ground 20
21	GPI009 (SPI_MISO)		(GPIO_GEN6) GPI025 22
23	GPI011 (SPI_CLK)		(SPI_CE0_N) GPI008 24
25	Ground		(SPI_CE1_N) GPI007 26
27	ID_SD (I2C ID EEPROM)		(I2C ID EEPROM) ID_SC 28
29	GPI005		Ground 30
31	GPI006		GPI012 32
33	GPI013		Ground 34
35	GPI019 LRCLOCK		GPI016 36
37	GPI026		DATA IN GPI020 38
39	Ground		DATA OUT GPI021 40

Rev. 1.1
16/07/2014

<http://www.element14.com>

EXTERNAL CLOCK

The audio frequencies (the PCM _MCLK) are supposedly generated by the use of a discrete on-board 19.2 MHz crystal. Unlike the BeagleBone Black, where there are facilities (pins) to feed an external master clock.



The frequency that is generated at any of the I/O pins, say the bit clock, is obtained by dividing the source clock (19.2 MHz oscillator) by configuring a clock division register with an integer part and a fractional part as indicated by the datasheet excerpt shown below:



BCM2835 ARM Peripherals

6.3 General Purpose GPIO Clocks

The General Purpose clocks can be output to GPIO pins. They run from the peripherals clock sources and use clock generators with noise-shaping MASH dividers. These allow the GPIO clocks to be used to drive audio devices.

The fractional divider operates by periodically **dropping source clock pulses**, therefore the output frequency will periodically switch between:

$$\frac{\text{source_frequency}}{\text{DIVI}} \quad \& \quad \frac{\text{source_frequency}}{\text{DIVI} + 1}$$

Jitter is therefore reduced by increasing the source clock frequency. In applications where jitter is a concern, the fastest available clock source should be used.

The General Purpose clocks have MASH noise-shaping dividers which push this fractional divider jitter out of the audio band.

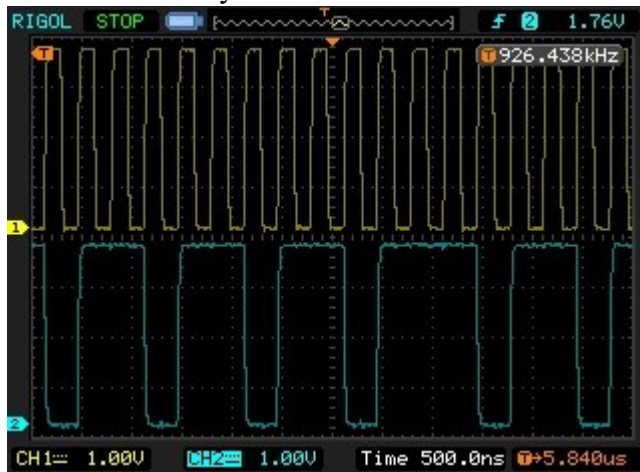
MASH noise-shaping is incorporated to push the fractional divider jitter out of the audio band if required. The MASH can be programmed for 1, 2 or 3-stage filtering. MASH filter, the frequency is spread around the requested frequency and the user must ensure that the module is not exposed to frequencies higher than 25MHz. Also, the MASH filter imposes a low limit on the range of DIVI.

MASH	min DIVI	min output freq	average output freq	max output freq
0 (int divide)	1	$\text{source} / (\text{DIVI})$	$\text{source} / (\text{DIVI})$	$\text{source} / (\text{DIVI})$
1	2	$\text{source} / (\text{DIVI})$	$\text{source} / (\text{DIVI} + \text{DIVF} / 1024)$	$\text{source} / (\text{DIVI} + 1)$
2	3	$\text{source} / (\text{DIVI} - 1)$	$\text{source} / (\text{DIVI} + \text{DIVF} / 1024)$	$\text{source} / (\text{DIVI} + 2)$
3	5	$\text{source} / (\text{DIVI} - 3)$	$\text{source} / (\text{DIVI} + \text{DIVF} / 1024)$	$\text{source} / (\text{DIVI} + 4)$

The Datasheet indicates that the clocks are generated by “noise-shaping MASH dividers” which are fractional dividers. It also says that “The fractional dividers operates by periodically dropping source clock pulses”. I believe this post has an example on how this

is actually implemented [\[link\]](#).

The way that a 3.25x clock divider is implemented is by dividing by 3x for some periods and 4x for other periods, with the average being 3.25x. In this case the repeating pattern will be (3, 3, 3, 4). That is shown in the following scope capture. Note that the first three periods are divided by 3 and then the next is divided by 4.



The way this is implemented in the device is to divide by the smaller divider and then extend the high pulse width by one clock cycle periodically.

We can find the integer divider and fractional divider based on MASH 1 (see above) and determine what is the maximum and minimum output frequency:

- Source clock: 19,200,000 Hz
- Sample rate: 44,100 Hz; bit clock (64fs)=2,822,400 Hz
- Actual divisor: 6.8027. Integer part=6
- Fractional divisor: $=0.8027 \times 1024 = 822$ (round off)
- Maximum frequency: $19,200,000 / 6 = 3,200,000$ Hz (50 KHz sample rate)
- Minimum frequency: $19,200,000 / 7 = 2,742,857$ Hz (42.9 KHz sample rate)
- Average frequency: $19,200,000 / (6 + (822 / 1024)) = 2,822,394$ Hz (44.1 KHz sample rate)

Well, aiming at 44.1KHz sample rate frequency and getting a frequency variation from 42.9 KHz to 50KHz, this can't really work for digital audio. Clearly there has to be a better way to generate these clock frequencies.

GENERATION OF DIGITAL AUDIO FREQUENCY CLOCKS

Much of the credit for enabling I2S output in the RPi (and the proper generation of clock frequencies) is due to the discussion in the Raspberry Pi forums [\[link\]](#) and work of Florian Meier in his master thesis "Low-Latency Audio over IP on Embedded Systems" [\[link\]](#) who subsequently developed the basic "ALSA SoC I2S Audio Layer for Broadcom BCM2708 SoC" audio kernel driver [\[link\]](#)

There it is explained that in order to get good clocks, one has to use integer division but with a 19.2 MHz clock, it is impossible to arrive at 32fs or 64fs bitclocks (e.g. $64 \times 48\text{KHz} = 3.072$ MHz). Therefore other internal clocks must be used.

According to this post [\[link\]](#) the clocks sources are:

0 0 Hz Ground

1	19.2 MHz	oscillator
2	0 Hz	testdebug0
3	0 Hz	testdebug1
4	0 Hz	PLLA
5	1000 MHz	PLLC (changes with overclock settings)
6	500 MHz	PLLD
7	216 MHz	HDMI auxiliary
8-15	0 Hz	Ground

The logical choices are the external 19.2MHz and the highest stable frequency clock which is the 500 MHz clock (highest frequency generates a more accurate clock after fractional clock division)

The author presents two solutions:

- Use the 19.2 MHz oscillator with integer division for DACs that do not require a specific ratio of bit clock to frame sync (e.g. 32fs for 16 bit data) as long as there are at most enough bit clock cycles within a frame sync cycle to contain the desired word length
- Use the 500 MHz internal PLL with fractional division for DACs that do require a specific ratio of bit clock to frame synch (e.g 32fs or 64fs)

The first solution says that it is possible to use, say 40fs, to sent 16 bit samples (16bitx2=32bit per frame) because you can transfer all 32 bits in a 40 bit frame. If you can use 40fs for the bitclock, then $40 \times 48\text{KHz} = 1.920\text{ MHz}$ which is $19.2\text{ MHz}/10$. The following excerpt from the thesis explains these two approaches:

There are two solutions for this: The most common is the use of a non integer divider. This clearly can not produce a perfect clock, but produces a clock with a varying frequency that has the desired frequency as average. An early patent presenting this technique is [39]. It uses a structure that is similar to the one found in delta-sigma ADCs. For minimizing the undesirable effect, the BCM2835 uses multi-stage noise shaping (MASH) that shifts much of the noise into higher frequencies where it harms less at the cost of an overall increase of noise [40].

The second solution utilizes the fact, that many audio codecs do not require a specific ratio of bit clock to frame sync as long as there are at most enough bit clock cycles within a frame sync cycle to contain the desired word length, and the bit clock frequency is not too high. The state of the data line in the remaining bits does not care as shown in Fig. 5.3. This leads to the uncommon number of 40 bit clock transitions per frame sync period resulting in a bit clock frequency of $40 \cdot 48\text{ kHz} = 1.92\text{ MHz}$. This can be generated from the 19.2MHz clock by an integer divider of 10. Both solutions are implemented and the latter one is chosen for matching sampling rates (e.g. 48 kHz and 96 kHz).

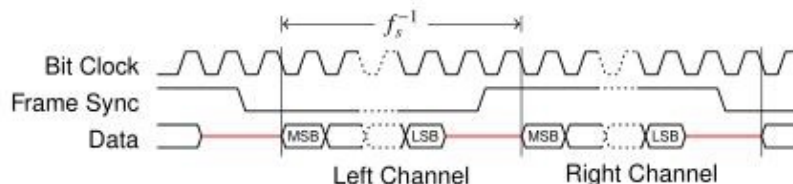


Figure 5.3: Serial audio transmission with word length \neq frame sync period

We notice that integer division of the external 19.2 MHz clock only works for 48KHz and 96 KHz and for DACs that can operate at 40fs (80fs if we want to pass 32×2 bits per frame). The current version of the code is using 50fs and 100fs which also works.

For the 44.1KHz sample rate or for bitclock requiring 32fs or 64fs, then the first solution

with fractional division is used on the 500 MHz PLLD clock

WHY 19.2 MHz CLOCK?

The fact that no clean clocks can be generated in the digital audio frequency range, tells us that this oscillator was not really meant to be used for digital audio. Now why did the designers of RPi use a 19.2 MHz clock?

I have searched extensively and cannot find a “reason” for the 19.2 MHz frequency. If it were digital audio, a more logical selection would have been 24.576 MHz in order to cleanly support 64fs 48KHz sample rate (like the BeagleBone Black).

A better reason is to use this clock for the on-board PWM audio. One can easily generate a 48KHz carrier frequency ($19.2\text{MHz}/40$) and a resolution of 16 bit would require a frequency of approx 64 KHz ($19.2\text{MHz}/30$). In actually, it has been reported that the resolution is in the neighborhood of 11 bit or 2048 levels which can be obtained by dividing 19.2MHz by a factor of 9375.

SLAVING THE RPi

A better solution is to configure RPi as a slave device and the DAC as a master device.

The DAC can provides a much more accurate clock to the RPi by feeding the Bitclock. I don't think is being done by the current crop of DACs (the ones based on the PCM5122) but the capability is there for both in the RPi (“clock slave mode” and “frame synch slave mode”) and the PCM5122 as shown in the following excerpt from the datasheet:

BCKO	BCK Output Enable This bit sets the BCK pin direction to output for I ² S master mode operation. In I ² S master mode the PCM51xx outputs the reference BCK and LRCK, and the external source device provides the DIN according to these clocks. Use Page 0 / Register 32 to program the division factor of the SCK to yield the desired BCK rate (normally 64f _s) Default value: 0 0: BCK is input (I ² S slave mode) 1: BCK is output (I ² S master mode)
LRKO	LRCLK Output Enable This bit sets the LRCK pin direction to output for I ² S master mode operation. In I ² S master mode the PCM51xx outputs the reference BCK and LRCK, and the external source device provides the DIN according to these clocks. Use Page 0 / Register 33 to program the division factor of the BCK to yield 1f _s for LRCK. Default value: 0 0: LRCK is input (I ² S slave mode) 1: LRCK is output (I ² S master mode)

Here is what is required to set the DAC in Master Mode, say for example the PCM5122.

1. RPi detects the sample rate of the clicked-to-play track.
2. RPi has a way to indicate the sample rate (for example using GPIO pins).
3. Microcontroller reads sample rate.
4. Microcontroller programs the appropriate frequency by generating an appropriate master frequency from the PLL and setting the appropriate divider to generate the bitclock.

Other considerations:

- Timing of the different events. For example, wait until the microcontroller programs the DAC to the appropriate clock frequency before starting the data stream (DMA) in the RPi.
- Selection of external clock. For example use a single frequency clock, say a multiple

of 44.1KHz in order to take advantage of integer divider only when dealing with frequencies multiple of 44.1KHz.

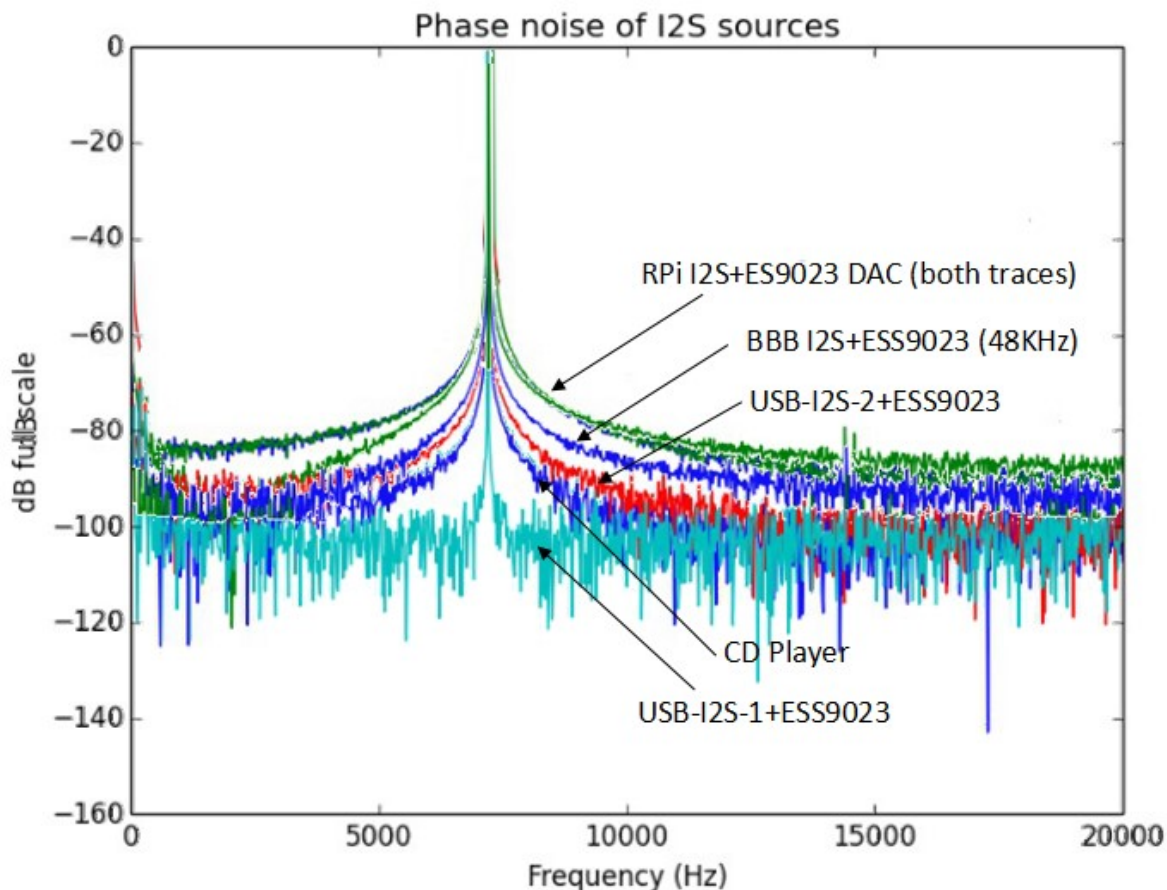
COMPARISON WITH OTHER DEVICES

Here is the previous comparison table with some updated observations (*italics*)

Parameter	Raspberry Pi	BeagleBone Black	Comments
Native I2S support	Yes	Yes	Both platforms can support I2S output, <i>Custom drivers have been developed by the audio community</i>
I2S Sample Rate limitation	Up to 192KHz (because the on-board clock is 19.2MHz)	Only 48KHz family (because the on-board clock is 24.576MHz and integer clock dividers)	BBB supports 48KHz, 96KHz, 192KHz and 384KHz. RPi supports 44.1KHz, 48KHz, 88.2KHz, 96KHz, 176.4KHz and 192KHz (in theory). RPi uses “fractional clock dividers” to generate the 44.1KHz sample rate family <i>as explained above</i>
Support for USB DAC	Yes (LAN9512 chip [link])	Yes (Built-in in the main processor)	USB in the RPi goes through a built-in HUB and it is shared with the LAN controller within the USB/LAN chip. USB in the BBB is natively supported by the main processor; LAN has a separate chip
Support for external, low jitter clocks	<i>Not possible unless you replace the on board oscillator and modify the driver</i>	Yes with custom boards and custom s/w: [link]	The master clock in BBB may be provided externally by disabling the on-board audio-freq clock. The Master clock in the RPi seems internally generated and there is no I/O pin to feed an external master clock
Master clock output	No	Yes (from on-board clock)	The Master clock in BBB is provided by the on-board 24.576MHz and fixed at this frequency and can be directly accessed from the outside. The Master clock of RPi seems internally generated but un-accessible from the outside. Without Master Clock, you can only use DACs that can operate asynchronously without a Master Clock input such as the ESS DACs or DACs that can operate with the master clock = bit clock
Built-in rechargeable battery operation	No	Yes [link] . <i>Maybe</i>	Rechargeable Battery operation in BBB would disable the 5V supply to the USB. Thus for USB operation, where the USB adapter takes the

Built-in Storage	No. <i>But the new model has plenty of USB ports for USB memory sticks</i>	2 GB eMMC Flash	power from USB, BBB must be powered with 5V DC BBB can boot from the internal storage freeing the SD card for music storage. RPi requires that the OS be stored in the SD card (although it may be possible to also store music in the SD card)
Looks	<i>The latest model looks Good</i>	Good	😊
Audio H/W and S/W community support	<i>Large</i>	<i>Small</i>	
Price	\$35	\$55	

Here is a summary of the phase noise measurements from this post [\[link\]](#):



Observations

- The ESS9023 implements a “jitter eliminator” (asynchronous sample rate converter) but cannot eliminate all the jitter
- The clocks on the embedded boards have a lot of jitter. It also makes sense that the BBB has better measurement than the RPi. In the BBB, the 48KHz sample rate frequency is derived by integer division of the external clock. In the RPi, the 44.1KHz sample rate frequency is derived from the 500MHz clock which is derived from the external clock as explained above

- The “lowly” CD player is still a pretty good playback device in terms of jitter
- I would guess (and only a guess since the author does not identify these interfaces) USB-I2S-2 is an XMOS-based device based on how the clocks are generated and that USB-I2S-1 is a device based on an FPGA or CPLD using two external audio frequency clocks (where straight integer division is used)

SUMMARY

The current method of generating the clocks for digital audio in the RPi are far from perfect. The best clocks are obtained by integer division of the external clock and works for 48K and 96K sample rates and only if the DAC can accept 40fs or 80fs. For anything else, the clocks are derived from the 500MHz PLL through fractional division as explained above. It has been reported that the 500MHz clock itself is derived from the 19.2MHz external clock through a clock multiplier.

However imperfect as these clocks might be, there are a good number of DAC boards that have been developed and reported to work well with the RP1. As these products are being developed by audio fanatics, we can expect continuous improvements and enhanced approaches to better clock generation such as external reclocking and slave configurations.

For additional info, you can check the Raspberry Pi I2S discussion thread here: [[link](#)]

Advertisements

Share this:

Related[Raspberry Pi version B+](#)

In "Embedded Computing"

[BBB DAC](#)[DEVELOPMENTS](#)

In "Embedded Computing"

[Low Cost Audiophile Music](#)[Servers](#)

In "Computer (HW and SW)"

[Comments \(13\)](#) [Trackbacks \(3\)](#) [Leave a comment](#) [Trackback](#)

1.

[Rick Yarussi](#)

December 30, 2014 at 22:51

[Reply](#)

Can you comment on RPiB+, with the added HiFiBerry Digi board? I am using that to feed the SPDIF input of a DAC. The idea was to avoid USB DAC's entirely... Or am I still better off with BBB?

o [BlgGear](#)

December 31, 2014 at 21:07

[Reply](#)

BBB will only work with 48KHz sample rate (and multiples). You will need to wait for the clock boards from TwispedPear Audio in order to support 44.1KHz sample rate. The Rpi can support 44.1KHz sample rate and the plug-in DAC boards that are available work well with the Rpi. Although the concept of a barebones "PC" feeding a DAC is appealing, I much prefer a computer with a USB interface if listening to music is a priority or a CD player for that matter (especially you can buy used CDs for \$1-\$5).

I think the upcoming clocks boards for BBB, especially the reclocking board will allow the BBB to generate good quality I2S signals. The RPi in its current implementation cannot accept external clocks.



2.

[S Lareau](#)

January 22, 2015 at 22:05

[Reply](#)

Hi, I have came across the MinnowBoard Max mini computer board , price start at 99\$ for the single core, with seems to have decent spec on I2S expansion board (I am not an expert) Intel HDA spec up to 8 channel 192 khz/ 32 bit). I would like to get your opinion on this. It run on Linux and Windows 8.1. Would this overcome the limitation of the Pi? Thanks



BlgGear

January 23, 2015 at 04:58

[Reply](#)

Don't know. But unless they specifically say something about the I2S output and how the clocks are wired, chances is that there is no driver support and some brave s/w guy would have to write the drivers for the I2S output...

3.



[CXMX \(@CXMXdesign\)](#)

March 25, 2016 at 09:07

[Reply](#)

Dear BigGear,

Hope you don't mind a noob question.

Building a Headphone Amp and Dac for the Raspberry Pi

Raspberry Pi > DAC > Amp

It it better to use a Digi + board or I2S input ?

DGI+ coaxial in to a Dual AK4395SEQ <http://www.ebay.com/itm/Assembled-AK4495SEQ-AK4118-NE5534-Double-Parallel-Soft-Control-Board-DOP-DSD-/121709573225>

Or something like the AK4395 I2S <http://www.diyinhk.com/shop/audio-kits/74-768khz32bit-ak4495seq-dac-i2sdsd-input.html>

Can the RPi supply a master clock linked to the sampling frequency ?

Thank you !



BlgGear

March 26, 2016 at 15:25

[Reply](#)

Yes, the RPi can generate a "master clock". However, it is a "poor" implementation and may DAC makers for the RPi do not use it. Please read: <https://hifiduino.wordpress.com/2014/11/13/raspberry-pi-b-digital-audio/> and <https://hifiduino.wordpress.com/2014/11/12/raspberry-pi-version-b/>

You can try any DAC that does not require a master clock input. The ones specially made to fit on top of the board are not bad and much easier to configure.

Regarding the AKM, I don't remember exactly, but according to the datasheet, it needs a master clock input. You can try the master clock from the RPi, but it is not of highest quality.

4.



soundcheck

April 26, 2016 at 10:17

[Reply](#)

Please checkout the HifiBerry DAC+ Pro.

That one slaves the PI with onboard quality clocks!

The 44.1/48x clocks are selected via GPIO.

Your very good writeup might need a little update if I'm not mistaken.



o

BlgGear

July 4, 2016 at 20:24

[Reply](#)

I wasn't aware the PI has external pins where you can feed a clock to the audio subsystem. Can you point me to it? Last I check only the Blackberry Black has a pin where an external clock can be used in the audio subsystem.



■

Graef

July 8, 2016 at 10:47

You can't feed the Pi with a masterclock, but you can feed it with a bitclock. The dac chip on the Hifiberry DAC+ Pro is dividing the bitclock from the on-dac-board masterclocks and is sending it to the RPi.

I'm doing the same with a Twisted Pear Cronus (+Hermes-amanero), but only with a fixed sample frequency (352.4 kHz) where bitclock is 22.5792MHz, so software upsampling before playing audio.

I plan to make an external board (and dedicated linux driver) selecting and dividing the masterclock to the right bitclock, to avoid upsampling.



5.

[Lukasz L](#)

May 5, 2016 at 11:45

[Reply](#)

How would fifo based reclocking in Soekris Dam affect all you state above?



6.

[Geoffrey Reinoehl](#)

September 14, 2016 at 16:06

[Reply](#)

Shaahin Cheyene



7.

[Ralph Echelberger](#)

September 14, 2016 at 18:16

[Reply](#)

opaski plastikowe



8.

[Jestine Couillard](#)

September 14, 2016 at 18:41

[Reply](#)

Career Guide

1. February 17, 2015 at 11:35

[Malinowy DAC dla I2S na TDA1543 \(lub 1541\) cz.3 | aroundwaves](#)

2. May 12, 2016 at 20:02

[a good writing about raspberry pi + digital audio | My Word Press](#)

3. February 23, 2017 at 21:10

[Raspberry Pi B+ Digital Audio | H i F i D U I N O | Rabung's Blog](#)

Leave a Reply

Enter your comment here...

[AKM Verita 4490EQ DAC Raspberry Pi version B+](#)
[RSS feed](#)












Recent Posts

- [Firmware v 0.99](#)
- [Soekris DAC: Modding VRef](#)
- [R2R DAC Benchmark Filters \(for now\)](#)
- [dam 1021 R2R More Mods...](#)
- [Digital Filters for Soekris R-2R DAC](#)
- [Soekris dam 1021 R-2R DAC ILLUSTRATED GUIDE](#)
- [BBB DAC DEVELOPMENTS](#)
- [dam1021 R-2R DAC MODs](#)
- [Soekris dam1021 Build](#)
- [Raspberry Pi 2](#)
- [The Soekris R-2R DAC: Technical Details](#)
- [INSIDE MARANTZ AV RECEIVER](#)
- [AKM Verita 4490EQ DAC](#)
- [Raspberry Pi B+ Digital Audio](#)
- [Raspberry Pi version B+](#)

Latest Comments



[jimster480](#) on [C-Media CM6631A Firmware ...](#)

-  Alexandr on [Crystek CCHD-957](#)
[Building a compact a...](#) on [Rotary Encoder: H/W, S/W or No...](#)
[ES9018 "DAC-END R" F...](#) on [Buffalo III "Smart...](#)
[ES9018 "DAC-END R" F...](#) on [Interesting details of Weiss I...](#)
-  zosoncsu on [Adding Input Capacitors to the...](#)
-  gm rimaan on [Apple Remote](#)
-  investor adviser on [Soekris dam 1021 R-2R DAC ILLU...](#)
-  great Ebay seller on [Musiland 03US Dragon](#)
-  Anonymous on [Inside FiiO D3](#)
-  Effingham Dispensary on [Inside FiiO D3](#)
-  investment adviser on [Musiland 03US Dragon](#)
-  420 app on [Soekris dam1021 Build](#)
-  weedmaps clinic on [Soekris dam1021 Build](#)
-  Jesse Grillo Jesse G... on [Inside FiiO D3](#)

Arduino

- [Arduino Tutorials](#)
- [ARDUINOOS](#)
- [Code and Life](#)
- [Electronics Lab Blog](#)
- [Ken Shirriff's blog](#)
- [Kerry Wong -Arduino, etc](#)

Theory

- [Choosing and Using Bypass Caps](#)
- [Complete List of Linear Tech Application Notes](#)
- [Decoupling, Bypassing, Layout](#)
- [FIR Filter Tutorial](#)
- [Grounding Data Converters](#)
- [Intro to FIR Digital Filters](#)
- [Intuitive Guide to Principles of Communications](#)
- [Rakon Jitter Whitepaper](#)
- [Reduce EMI, RFI](#)
- [Successful PCB grounding with mixed-signal chips](#)
- [The Scientist and Engineer's Guide to Digital Signal Processing](#)
- [Tutorials and other Resources](#)
- [USB Power](#)

Tools for DIYaudio

- [% to dB Calculator](#)
- [Audio Equipment Database](#)
- [Best Solder](#)
- [Circuit simulation](#)
- [Generate DSD128 and DSD256](#)
- [Hex<->Binary<->Decimal Converter](#)
- [IC Manufacturer's LOGOs](#)
- [Inductor markings](#)
- [Inside Audio Equipment](#)
- [Jitter Calculator](#)
- [LOTs of Calculators](#)
- [Schematics](#)
- [Service Manuals](#)
- [SMD Codebook](#)
- [SMD Marking Code Search](#)
- [SMD Markings](#)
- [Tantalum Capacitor Markings](#)

Understanding Audio & Reviews

- [Artifact Audibility](#)
- [Audio Musings](#)
- [BitPerfect](#)
- [牙尖嘴利音響頻道](#)
- [In Ear Headphone Comparison](#)
- [Low-Jitter USB](#)
- [OpAmp Comparisons](#)
- [Red Spade Audio Blog](#)
- [TC Electronics](#)
- [The Slow Diyer](#)
- [Tweak's Guide to Recording Success](#)
- [USB-SPDIF shootout](#)

Follow HifiDuino via Email

Join 311 other followers

Follow

[Top](#)

[Create a free website or blog at WordPress.com.](#)

u