

1 Core Overview

The Audio Core interacts with the Audio CODEC (enCOder/DECOder) on the Altera DE2/DE1 Boards and provides an interface for audio input and output.

2 Functional Description

The Audio Core can support two modes, Audio Input and Audio Output, simultaneously. Figure 1 shows a block diagram for the Audio Core. To guarantee that the left and right audio channels are synchronized, data will not play until both channels are received. If only one channel is to be played, the other channel must have zeros written to it. The Audio Core contains four FIFOs for the In and Out audio data, both having the right and left audio channels. Each FIFO can store up to 255 32-bit words.

The Audio Core requires certain clock frequencies based on the sample rate of the audio. It also requires that audio chip to be initialized with some default values. Some other University Program IP Cores provides these functionalities and user should refer to Section 3 for details.

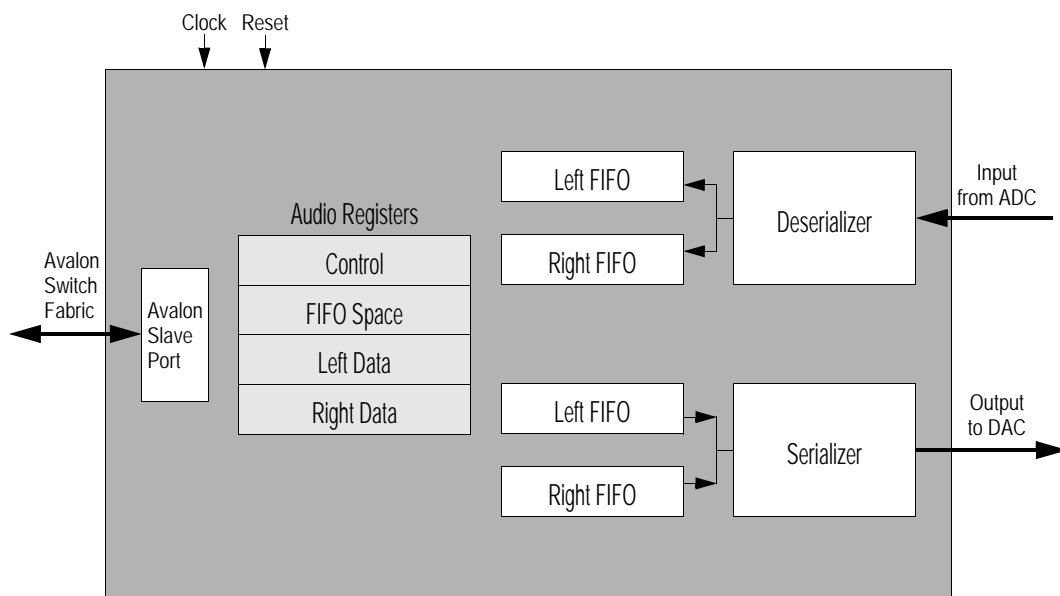




Figure 1. Block diagram for Audio Core


3 Instantiating the Core in SOPC Builder

Designers use the Audio Core's **Configuration wizard** in the SOPC Builder to specify the desired features.

In the configuration tab, the user can choose the mode of the Audio Core by selecting Audio Out and/or Audio In. In addition, the Data Width per Channel can be specified. Data widths of 16, 20, 24, and 32 bits are supported.

 The user **must** also instantiate the **Development Board External Interface** core and choose the proper audio clock setting for the Audio Core. See [Wolfson WM8731 audio CODEC Datasheet](#) for details on the relationship between sampling rate and clock frequency.

 Altera recommends you to instantiate the Audio Core with the standard or fast versions of the Altera Nios® II processor, so that a program running on the processor can keep up with the generation of audio data. If the program runs too slowly, the audio may not be clear. If the audio is not clear, it may be useful to select a lower sampling rate in the audio chip.

 The user can also choose to use the **Audio and Video Config** core to specify the initialization of the Audio Core and configure the audio chip on DE2/BE1 boards during run-time.

4 Software Programming Model

4.1 Register Map

Device drivers control and communicate with the Audio Core through four 32-bit registers. By writing or reading these registers, data can be fetched from the Analog-Digital Converter (ADC) or sent to the Digital-Analog Converter (DAC). Table 1 shows the format of the registers.

Table 1. Audio Core register map												
Offset in bytes	Register Name	R/W	Bit Description									
			31...24	23...16	15...10	9	8	7...4	3	2	1	0
0	control	RW	(1)			WI	RI	(1)	CW	CR	WE	RE
4	fifospace	R	WS LC	WS RC	RA LC			RA RC				
8	leftdata	RW (2)	Left Data									
12	rightdata	RW (2)	Right Data									

Notes on Table 1:

(1) Reserved. Read values are undefined. Write zero.

(2) Only reads incoming audio data and writes outgoing audio data.

4.1.1 Control Register

Table 2. Control register bits			
Bit number	Bit name	Read/Write	Description
0	RE	R/W	Interrupt-enable bit for read interrupts. If the RE bit is set to 1 and both the left and right channel read FIFOs contain data, the Audio Core generates an interrupt request (IRQ).
1	WE	R/W	Interrupt-enable bit for write interrupts. If the WE bit is set to 1 and both the left and right channel write FIFOs contain data, the Audio Core generates an interrupt request (IRQ).
2	CR	R/W	Clears the Audio Core's Input FIFOs, when the bit is 1. Clear remains active until specifically set to zero.
3	CW	R/W	Clears the Audio Core's Output FIFOs, when the bit is 1. Clear remains active until specifically set to zero.
8	RI	R	Indicates that a read interrupt is pending.
9	WI	R	Indicates that a write interrupt is pending..

4.1.2 Fifospace Register

The fifospace register fields WSLC (b_{31-24}) and WSRC (b_{23-16}) indicate the number of words available for outgoing data in the left and right channel FIFOs, respectively, while RALC (b_{15-8}) and RARC (b_{7-0}) indicate the number of words of incoming audio data in the left and right channel FIFOs, respectively.

4.1.3 Leftdata Register

The leftdata register is readable only for Audio In and writable only for Audio Out. It stores the data coming from or going to the left channel. The data is always flush right, i.e., the LSB is b_0 of the leftdata register.

4.1.4 Rightdata Register

The rightdata register is readable only for Audio In and writable only for Audio Out. It stores the data coming from or going to the right channel. The data is always flush right, i.e., the LSB is b_0 of the rightdata register.

4.2 Interrupt Behavior

The Audio Core produces a read interrupt when either of the read FIFOs are filled to 75% or more. Also, It produces the write interrupt when either of the write FIFOs have available space of 75% or more. The Audio Core generates an interrupt when either of these individual interrupt conditions are pending and enabled.

4.3 Programming with the Audio Core

The Audio Core is packaged with C-language device drivers accessible through the [hardware abstraction layer \(HAL\)](#). These functions implement basic operations that users need for the Audio Core.

To use the functions, the C code must include the statement:

```
#include "altera_up_avalon_audio.h"
```

4.3.1 alt_up_audio_open_dev

Prototype:	alt_up_audio_dev* alt_up_audio_open_dev(const char *name)
Thread-safe:	No
Commonly called by:	User program
Available from ISR:	—
Include:	<altera_up_avalon_audio.h>
Parameters:	name – the Audio component name in SOPC Builder.
Returns:	The corresponding device structure, or NULL if the device is not found
Description:	Open the Audio device specified by <i>name</i> .

4.3.2 alt_up_audio_enable_read_interrupt

Prototype:	int alt_up_audio_enable_read_interrupt(alt_up_audio_dev *audio)
Thread-safe:	No
Commonly called by:	User program
Available from ISR:	—
Include:	<altera_up_avalon_audio.h>
Parameters:	audio – the audio device structure
Returns:	0 for success
Description:	Enable the read interrupts for the Audio Core.

4.3.3 alt_up_audio_disable_read_interrupt

Prototype:	int alt_up_audio_disable_read_interrupt(alt_up_audio_dev *audio)
Thread-safe:	No
Commonly called by:	User program
Available from ISR:	—
Include:	<altera_up_avalon_audio.h>
Parameters:	audio – the audio device structure
Returns:	0 for success
Description:	Disable the read interrupts for the Audio Core.

4.3.4 alt_up_audio_reset_audio_core

Prototype: `int alt_up_audio_reset_audio_core(alt_up_audio_dev *audio)`

Thread-safe: No

Commonly called by: User program

Available from ISR: —

Include: `<altera_up_avalon_audio.h>`

Parameters: `audio` – the audio device structure

Returns: 0 for success

Description: Reset the Audio Core by clearing the input and output FIFOs for both the left channel and the right channel.

4.3.5 alt_up_audio_read_fifo

Prototype: `int alt_up_audio_read_fifo(alt_up_audio_dev *audio, alt_u32 *buf, unsigned len)`

Thread-safe: No

Commonly called by: User program

Available from ISR: —

Include: `<altera_up_avalon_audio.h>`

Parameters: `audio` – the audio device structure
`buf` – the pointer to the allocated memory for storing audio data. Size of `buf` should be no smaller than $2*len$ words. The first `len` words will store data from the left channel while the second stores data from the right channel.
`len` – the number of data in words to read from each input FIFO

Returns: the total number of words read

Description: Read `len` words of data from left input FIFO and right input FIFO, respectively, and store data to where `buf` points.

Notes: The function will read the FIFO until $2*len$ is reached or the FIFO is empty.

4.3.6 alt_up_audio_write_fifo

Prototype: `int alt_up_audio_write_fifo(alt_up_audio_dev *audio, alt_u32 *buf, unsigned len)`

Thread-safe: No

Commonly called by: User program

Available from ISR: —

Include: `<altera_up_avalon_audio.h>`

Parameters:
audio – the audio device structure
buf – the pointer to the data to be written Size of *buf* should be no smaller than $2*len$ words. The first *len* words will store data for the left channel while the second stores data for the right channel
len – the number of data in words to be written into each output FIFO

Returns: the total number of data written

Description: Write *len* words of data from *buf* to each of the output FIFO (left and right).

Notes: The function will write to the FIFO until $2*len$ is reached or the FIFO is full.

