Audio Core for Altera DE2/DE1 Boards

Preliminary

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 be 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.

The Audio Core requires certain clock frequencies based on the sample rate of the audio. Another Altera University Program IP Core named *Development Board External Interface* can create the necessary clock signals automatically. Also, the Audio Core requires that audio chip to be initialized with some default values. Again, another Altera University Program IP Core named *Audio/Video Configuration Core* can set the necessary registers in the audio chip automatically. Refer to those components' documentation on how to properly instantiate and connect them to a system.

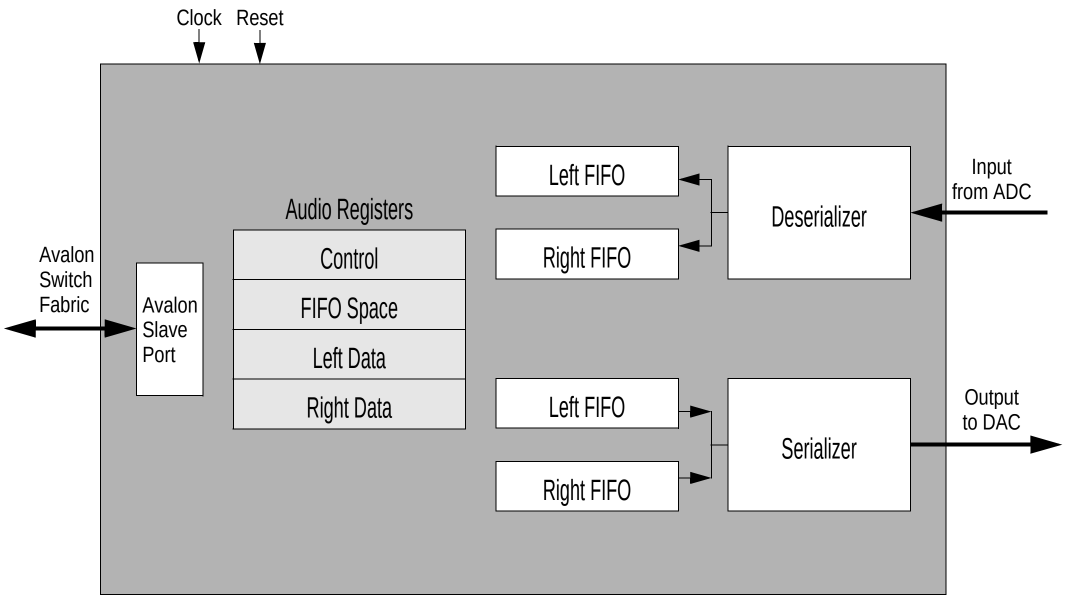

Figure 1. Block Diagram for Audio Core

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. The data widths of 16, 20, 24 and 32 bits are supported.

It is recommended to instantiate the Audio Core with the standard or fast versions of Altera's 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.

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 ADC or sent to the 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/Clear	Description
0	RE	R/W	Interrupt-enable bit for read interrupts. When the RE bit is 1, the Audio Core will generate an IRQ when both the left and right channel FIFOs have data available.
1	WE	R/W	Interrupt-enable bit for write interrupts. When the WE bit is 1, the Audio Core will generate an IRQ when both the left and right channel FIFOs have write space available.
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 spaces available for outgoing data in the left and right channel FIFOs, respectively, while RALC (b_{15-8}) and RARC (b_{7-0}) indicate the amount 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 of the data always starts at 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 of the data always starts at 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 functions accessible through the SOPC software development kit (SDK) libraries. These functions implement basic operations that users need for the Audio Core. When using the Altera Debug Client, these functions are automatically provided for use in a C-language application program, as presented in Section 4.4. To use the functions, the C code must include the statement:

```
#include "alt_up_audio.h"
```

4.4 Audio Core Functions

4.4.1 int alt_up_audio_enable_read_interrupt ()

Enable the read interrupts for Audio Core.

Returns:

0 for success

4.4.2 int alt_up_audio_disable_read_interrupt ()

Disable the read interrupts for Audio Core.

Returns:

0 for success

4.4.3 int alt_up_audio_reset_audio_core ()

Reset the Audio Core by clearing the Incoming and Outgoing FIFOs.

Returns:

0 for success

4.4.4 int alt_up_audio_read_left_channel (alt_u32 $*$ *buf*, unsigned *len*)

Read *len* number of data from the Left Channel Incoming FIFO, and store to where *buf* points.

Parameters:

buf – the pointer to the allocated memory for storing data
len – the number of data to read

Returns:

the total number of data read

Note:

The function will read the FIFO until *len* is reached or the FIFO is empty

4.4.5 int alt_up_audio_read_right_channel (alt_u32 ∗ *buf*, unsigned *len*)

Read *len* number of data from the Right Channel Incoming FIFO, and store to where *buf* points.

Parameters:

buf – the pointer to the allocated memory for storing data
len – the number of data to read

Returns:

the total number of data read

Note:

The function will read the FIFO until *len* is reached or the FIFO is empty

4.4.6 int alt_up_audio_write_left_channel (alt_u32 ∗ *buf*, unsigned *len*)

Write *len* number of data from *buf* to the Left Channel Outgoing FIFO.

Parameters:

buf – the pointer to the data to be written
len – the number of data to be written to the FIFO

Returns:

the total number of data written

Note:

The function will write to the FIFO until *len* is reached or the FIFO is full

4.4.7 int alt_up_audio_write_right_channel (alt_u32 ∗ *buf*, unsigned *len*)

Write *len* number of data from *buf* to the Right Channel Outgoing FIFO.

Parameters:

buf – the pointer to the data to be written
len – the number of data to be written to the FIFO

Returns:

the total number of data written

Note:

The function will write to the FIFO until *len* is reached or the FIFO is full

■