Audio core for Altera DE-Series Boards

For Quartus Prime 15.1

1 Core Overview

The Audio core interacts with the Audio CODEC (enCOder/DECoder) on the Altera DE-series boards and provides an interface for audio input and output.

2 Functional Description

The Audio core facilitates the transfer of audio data with the Audio CODEC chip on the Altera DE-series boards. Data is transferred serially between the FPGA and the CODEC. However, data written/read to/from the Audio core is transfered in a parallel manner. The Audio core automatically serializes/deserializes the data.

The Audio core contains four FIFOs to buffer the In and Out audio data, both having the right and left audio channels. Each FIFO can store up to 128 32-bit words. To guarantee that the left and right audio output 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 can be connected to a system using one of two different modes: Memory-Mapped or Streaming. Figure 1 shows a block diagram of the Audio core with a memory-mapped interface. This mode allows access to the core's FIFOs through memory-mapped registers. This is suitable when connecting the core to a processor, such as the Nios II processor, using the Qsys System Integration tool. Figure 2 shows a block diagram of the Audio core with a streaming interface. This mode allows direct access to the core's FIFOs. This is suitable when connecting to custom hardware using the Qsys System Integration tool or as a standalone component using the IP Catalog.

The Audio core requires certain clock frequencies based on the sample rate of the audio. The University Program's IP core, *Audio Clock for DE-series Boards*, can be use to provide those required clock frequencies. The Audio core also requires that the audio chip be initialized with some default values. The University Program's IO core, *Audio and Video Config*, provides the functionality required to initialize the audio chip. Refer to the specific documentation for those cores for more information.

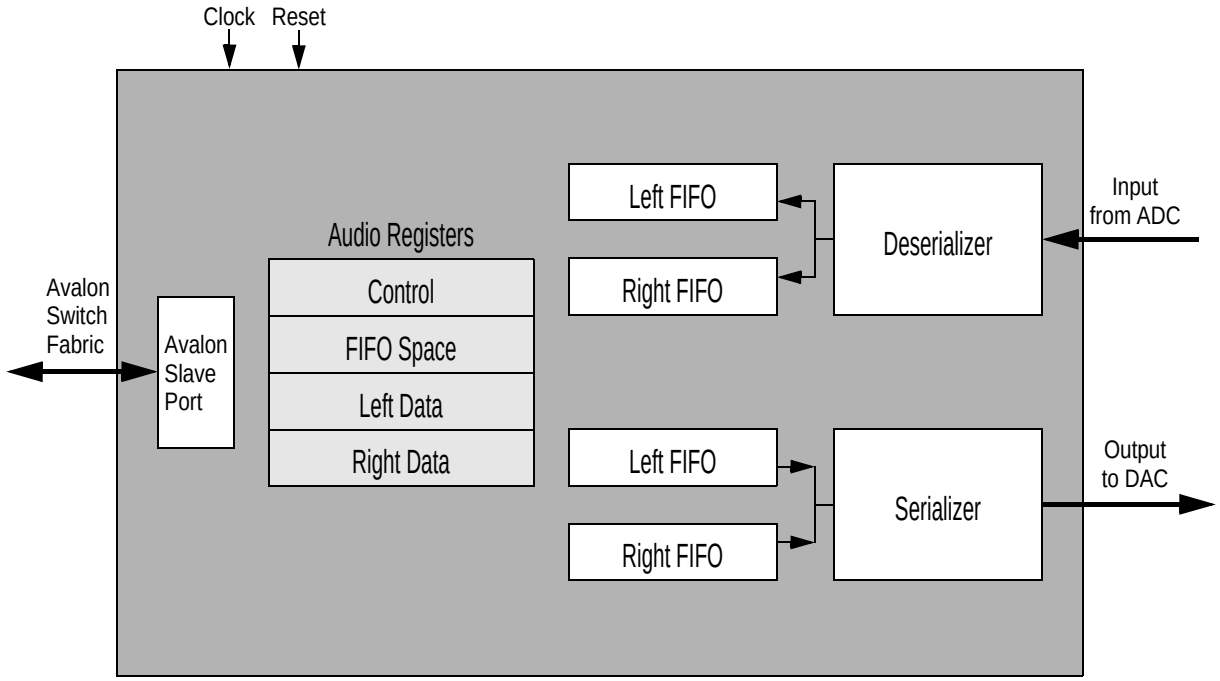

Figure 1. Block diagram for Audio core with Memory-Mapped Interface

3 Instantiating the Core

The Audio core can be instantiated in a system using Qsys or as a standalone component from the IP Catalog within the Quartus II software. Designers use the Audio core's **Configuration wizard** to specify the desired features. In the configuration wizard, the user can choose the interface type, whether it be memory-mapped or streaming. Also, the user can select whether to included only one or both of the audio in and/or audio out channels. In addition, the Data Width per Channel can be specified. Data widths of 16, 20, 24, and 32 bits are supported. Remember to export the external_interface connection to connect the core with the audio CODEC chip. It is recommended to set the Avalon Type to Memory-Mapped when connecting to a processor, otherwise set it to Streaming.

☞ Altera recommends also instantiating the ***Audio and Video Config*** core. This core automatically configures some required settings of the audio CODEC chip on the DE2/DE1 boards. Refer to the ***Audio and Video Config*** documentation for more information on properly intializing the audio codec.

☞ The user ***must*** also instantiate the ***Audio Clock for DE-series Boards*** core and choose the proper audio clock setting for the Audio core. See Wolfson WM8731 audio CODEC Datasheet in the "Audio Data Sampling Rates" section on page 37 for details on the relationship between sampling rate and clock frequency. Note that the Audio and Video Config core provides settings for these values.

☞ When using the Memory-Mapped Avalon Type in Qsys, Altera recommends that the Audio core be used 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 economic version of the processor is used, then the program may run too

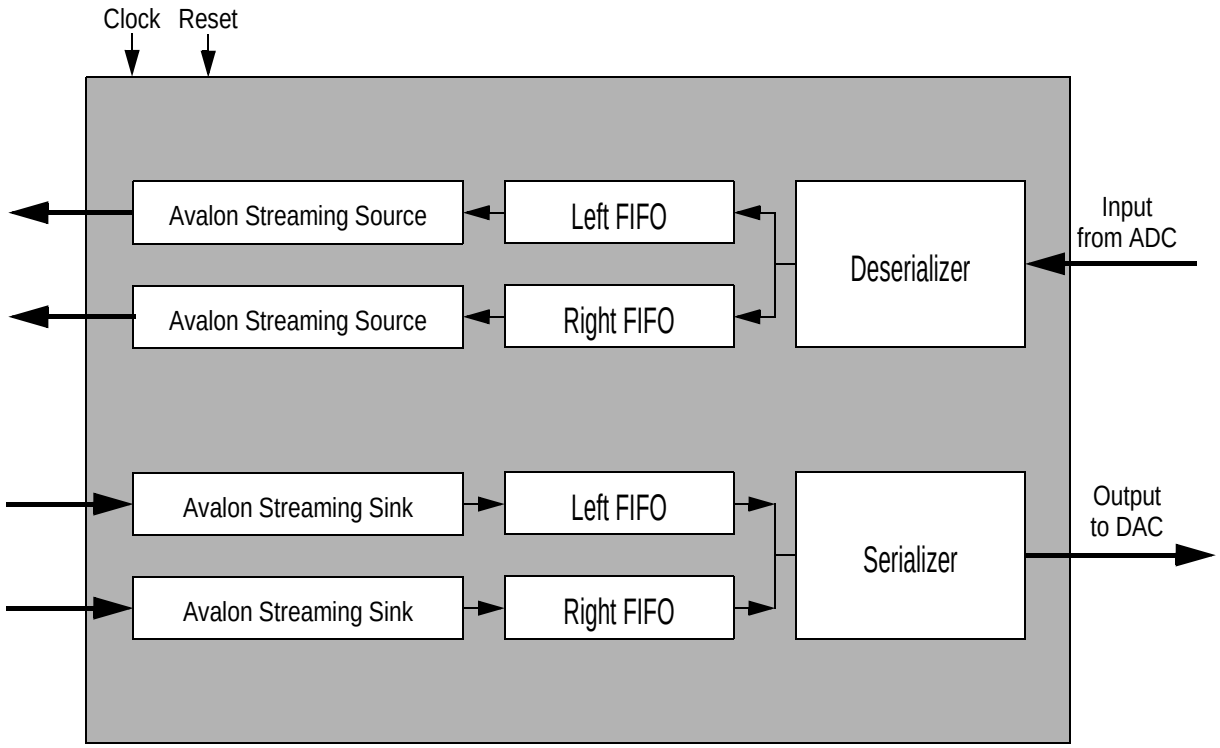

Figure 2. Block diagram for Audio core with Streaming Interface

slowly, and the audio may not be clear. In such, cases, it may be possible to improve the audio clarity by selecting a lower sampling rate in the audio chip.

4 Software Programming Model

4.1 Register Map

Software can control and communicate with the Audio core through four 32-bit registers when using the Memory-Mapped Avalon Type. By writing or reading these registers, data can be fetched from the CODEC's 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 have space available for more 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 (i.e., the amount of empty space) 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. When all of the outgoing and incoming FIFOs are empty, the fifospace register will hold WSLC = WSRC = 128, and RALC = RARC = 0.

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. The interrupt is cleared when the FIFO becomes less than 75% full. Also, it produces a write interrupt when either of the write FIFOs have available space of 75% or more. The interrupt is cleared when the FIFO becomes less than 75% empty. The Audio core generates an interrupt when either of these individual interrupt conditions are pending and enabled.

4.3 Device Driver for the Nios II Processor

For use with the Nios II processor, the Audio core is packaged with C-language functions accessible through the hardware abstraction layer (HAL). These functions implement basic operations for the Audio core.

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

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

An example of C code that uses the Audio core is given at the end of this section.

4.3.1 alt_up_audio_open_dev

Prototype: `alt_up_audio_dev* alt_up_audio_open_dev(const char *name)`
Include: `<altera_up_avalon_audio.h>`
Parameters: `name` – the audio component name in Qsys.
Returns: The corresponding device structure, or NULL if the device is not found
Description: Opens the audio device specified by *name* (default "/dev/audio/").

4.3.2 alt_up_audio_enable_read_interrupt

Prototype: `void alt_up_audio_enable_read_interrupt(alt_up_audio_dev *audio)`
Include: `<altera_up_avalon_audio.h>`
Parameters: `audio` – the audio device structure
Returns: nothing
Description: Enable read interrupts for the Audio Core.

4.3.3 alt_up_audio_disable_read_interrupt

Prototype: `void alt_up_audio_disable_read_interrupt(alt_up_audio_dev *audio)`
Include: `<altera_up_avalon_audio.h>`
Parameters: `audio` – the audio device structure
Returns: nothing
Description: Disable read interrupts for the Audio Core.

4.3.4 alt_up_audio_enable_write_interrupt

Prototype: `void alt_up_audio_enable_write_interrupt(alt_up_audio_dev *audio)`
Include: `<altera_up_avalon_audio.h>`
Parameters: `audio` – the audio device structure
Returns: nothing
Description: Enable write interrupts for the Audio Core.

4.3.5 alt_up_audio_disable_write_interrupt

Prototype: `void alt_up_audio_disable_write_interrupt(alt_up_audio_dev *audio)`
Include: `<altera_up_avalon_audio.h>`
Parameters: `audio` – the audio device structure
Returns: nothing
Description: Disable the read interrupts for the Audio Core.

4.3.6 alt_up_audio_read_interrupt_pending

Prototype: `int alt_up_audio_read_interrupt_pending(alt_up_audio_dev *audio)`
Include: `<altera_up_avalon_audio.h>`
Parameters: `audio` – the audio device structure
Returns: 1 if read interrupt is pending, else 0
Description: Check if read interrupt pending for the Audio Core.

4.3.7 alt_up_audio_write_interrupt_pending

Prototype: `int alt_up_audio_write_interrupt_pending(alt_up_audio_dev *audio)`
Include: `<altera_up_avalon_audio.h>`
Parameters: `audio` – the audio device structure
Returns: 1 if write interrupt is pending, else 0
Description: Check if write interrupt pending for the Audio Core.

4.3.8 alt_up_audio_reset_audio_core

Prototype: `void alt_up_audio_reset_audio_core(alt_up_audio_dev *audio)`
Include: `<altera_up_avalon_audio.h>`
Parameters: `audio` – the audio device structure
Returns: nothing
Description: Reset the Audio Core by clearing read and write FIFOs for left and right channels.

4.3.9 alt_up_audio_read_fifo_avail

Prototype: `unsigned int alt_up_audio_read_fifo_avail(alt_up_audio_dev *audio, int channel)`
Include: `<altera_up_avalon_audio.h>`
Parameters: `audio` – the audio device structure
`channel` – left or right channel selection
Returns: number of words available
Description: provides number of words of data available in the incoming FIFO for *channel*

4.3.10 alt_up_audio_record_r

Prototype: `unsigned int alt_up_audio_record_r(alt_up_audio_dev *audio, unsigned int *buf, int len)`
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 len words.
`len` – the number of data in words to read from the input FIFO
Returns: The total number of words read.
Description: Read len words of data from right input FIFO, if the FIFO is above a threshold, and store data to where buf points.

4.3.11 alt_up_audio_record_l

Prototype: `unsigned int alt_up_audio_record_l(alt_up_audio_dev *audio, unsigned int *buf, int len)`
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 len words.
`len` – the number of data in words to read from the input FIFO
Returns: The total number of words read.
Description: Read len words of data from left input FIFO, if the FIFO is above a threshold, and store data to where buf points.

4.3.12 alt_up_audio_write_fifo_space

Prototype: `unsigned int alt_up_audio_write_fifo_space(alt_up_audio_dev *audio, int channel)`
Include: `<altera_up_avalon_audio.h>`
Parameters: `audio` – the audio device structure
`channel` – left or right channel enum
Returns: number of words available
Description: provides the amount of empty space in the outgoing FIFO for *channel*

4.3.13 alt_up_audio_play_r

Prototype: `unsigned int alt_up_audio_play_r(alt_up_audio_dev *audio, unsigned int *buf, int len)`
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 len words.
`len` – the number of data in words to be written into the output FIFO
Returns: The total number of data written.
Description: Write len words of data into right output FIFO, if space available in FIFO is above a threshold.

4.3.14 alt_up_audio_play_l

Prototype: `unsigned int alt_up_audio_play_l(alt_up_audio_dev *audio, unsigned int *buf, int len)`
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 len words.
`len` – the number of data in words to be written into the output FIFO
Returns: The total number of data written.
Description: Write len words of data into left output FIFO, if space available in FIFO is above a threshold.

4.3.15 alt_up_audio_read_fifo

Prototype: `int alt_up_audio_read_fifo(alt_up_audio_dev *audio, unsigned int *buf, int len, int channel)`
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 *len* words.
`len` – the number of data in words to read from each input FIFO
`channel` – left or right channel selection
Returns: The total number of words read.
Description: Read *len* words of data from left input FIFO or right input FIFO, and store data to where *buf* points.

4.3.16 alt_up_audio_write_fifo

Prototype: `int alt_up_audio_write_fifo(alt_up_audio_dev *audio, unsigned int *buf, int len, int channel)`
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 *len* words.
`len` – the number of data in words to be written into each output FIFO
`channel` – left or right channel selector
Returns: The total number of data written.
Description: Write *len* words of data from *buf* to the left or right output FIFOs.

4.3.17 alt_up_audio_read_fifo_head

Prototype: `unsigned int alt_up_audio_read_fifo_head(alt_up_audio_dev *audio, int channel)`
Include: `<altera_up_avalon_audio.h>`
Parameters: `audio` – the audio device structure
`channel` – left or right channel selection
Returns: the word read
Description: Read one data word from left input FIFO or right input FIFO.

4.3.18 alt_up_audio_write_fifo_head

Prototype: `void alt_up_audio_write_fifo_head(alt_up_audio_dev *audio, unsigned int data, int channel)`
Include: `<altera_up_avalon_audio.h>`
Parameters: `audio` – the audio device structure
`data` – the data word to be written
`channel` – left or right channel selector
Returns: nothing
Description: Write one data word to the left or right output FIFOs.

4.3.19 Audio core C Example using Device Drivers

```
#include "altera_up_avalon_audio.h"

int main(void)
{
	alt_up_audio_dev * audio_dev;

	/* used for audio record/playback */
	unsigned int l_buf;
	unsigned int r_buf;

	// open the Audio port
	audio_dev = alt_up_audio_open_dev ("/dev/Audio");
	if ( audio_dev == NULL)
		alt_printf ("Error: could not open audio device \n");
	else
		alt_printf ("Opened audio device \n");

	/* read and echo audio data */
	while(1)
	{
		int fifospace = alt_up_audio_read_fifo_avail (audio_dev, ALT_UP_AUDIO_RIGHT);
		if ( fifospace > 0 ) // check if data is available
		{
			// read audio buffer
			alt_up_audio_read_fifo (audio_dev, &(r_buf), 1, ALT_UP_AUDIO_RIGHT);
			alt_up_audio_read_fifo (audio_dev, &(l_buf), 1, ALT_UP_AUDIO_LEFT);

			// write audio buffer
			alt_up_audio_write_fifo (audio_dev, &(r_buf), 1, ALT_UP_AUDIO_RIGHT);
			alt_up_audio_write_fifo (audio_dev, &(l_buf), 1, ALT_UP_AUDIO_LEFT);
		}
	}
}
```

Figure 3. An example of C with Device Driver Support code that uses Audio Core.