

Audio Codec Driver Implementation for Xilinx Zedboard Zynq 7000

by Viktoria Biliouri

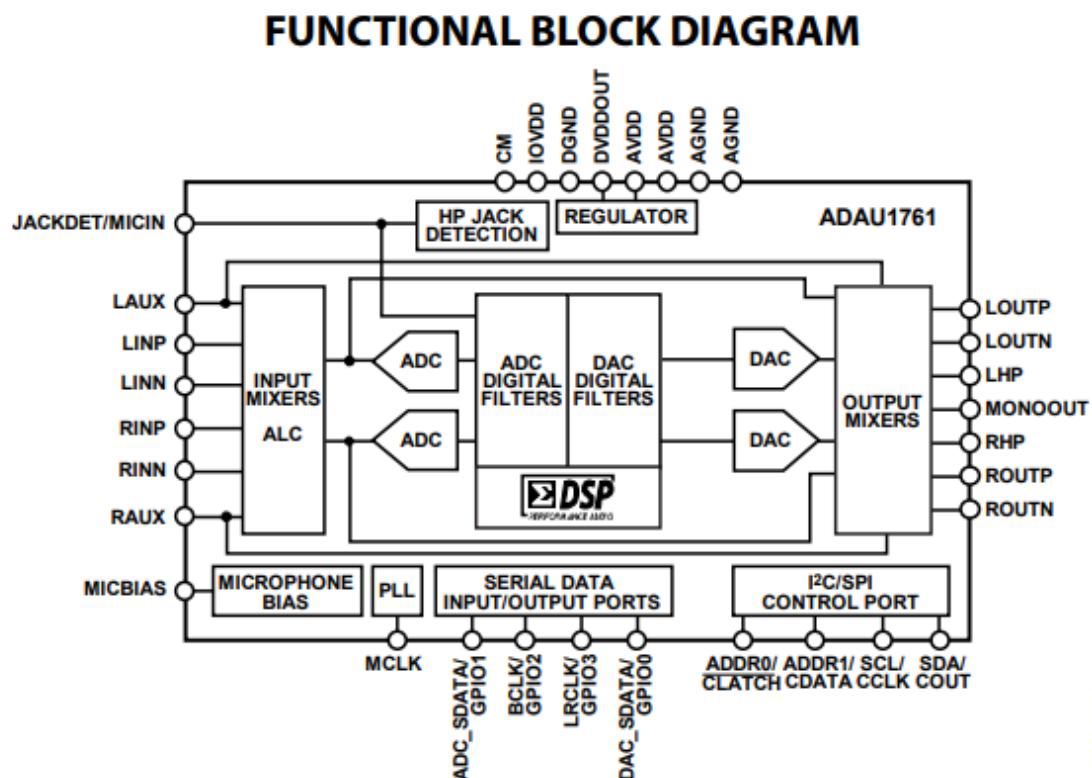
SYNOPSIS

This project contains the implementation of an audio driver for ADAU1761 audio codec, using SPI and I2S protocols for the transaction of audio data, inserted in the FPGA from an SD card. In particular, the audio (.wav) files from the SD Card are inserted to the ARM Processor of the Zedboard, their data are stored in the main system memory, and by DMA (Direct Memory Access) protocol, they are transferred to the audio output (headphones), after having been sampled. The hardware platform used was Xilinx ZedBoard™, and the development platform, was Vivado Design Suite 2017.2. More details will be analysed below.

The project was implemented within the framework of the curriculum of the department of Electrical and Computer Engineering, University of Thessaly, Greece, under the guidance of Professor Nikolaos Bellas.

ADAU1761

First of all, the audio codec that ZedBoard™ contains is ADAU1761 by Analog Devices. The ADAU1761 is a low power, stereo audio codec with integrated digital audio processing that supports stereo 48 kHz record and playback at 14 mW from a 1.8 V analog supply. The stereo audio ADCs and DACs support sample rates from 8 kHz to 96 kHz as well as a digital volume control. This audio codec includes among others, a line-in input, a line-out output, a microphone input and a headphone output, that I have used in this project.

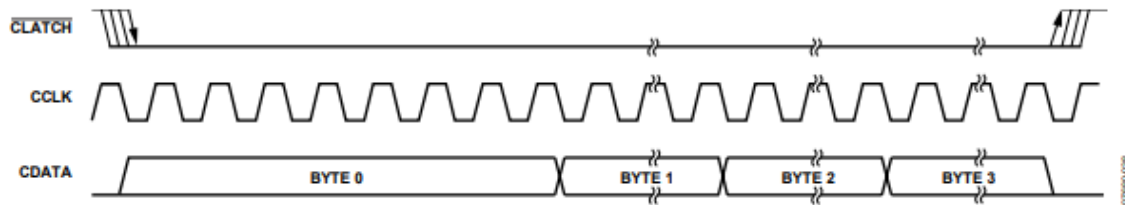


The ADAU1761 can operate in I2C control mode or in SPI control mode. However, in the current implementation, I have used the SPI protocol, for the operation of the audio codec.

SPI - Serial Peripheral Interface Protocol

The **Serial Peripheral Interface (SPI)** is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems. The SPI port uses a 4-wire interface, consisting of the CLATCH, CCLK, CDATA, and COUT signals, and it is always a slave port. The CLATCH signal should go low at the beginning of a transaction and high at the end of a transaction. The CCLK signal latches CDATA on a low-to-high transition. COUT data is shifted out of the ADAU1761 on the

falling edge of CCLK and should be clocked into a receiving device, such as a microcontroller, on the CCLK rising edge. The CDATA signal carries the serial input data, and the COUT signal carries the serial output data. The COUT signal remains three-state until a read operation is requested. This allows other SPI-compatible peripherals to share the same readback line.



SPI Write to ADAU1761

According to the above information about the mode of operation of SPI, I have created an IP, which functions in the same way. To be more specific, the SPI-transmitter IP, takes as inputs, the data that are about to be transferred, “miso” wire, the spi clock “sclk”, and the select signal “ss”, and creates the output data “mosi” wire, according to the steps of a state machine with 4 states (0, 1, 2, 3), sending the data in a synchronized format to the ADAU1761. The modes are defined by the polarity and the clock phase, as shown below:

SPI mode	Clock polarity	Clock phase
0	0	0
1	0	1
2	1	0
3	1	1

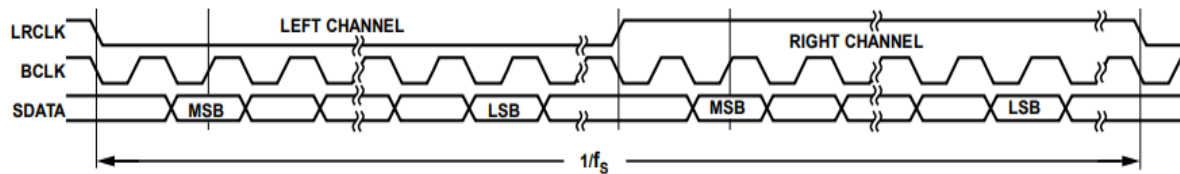
In state 0, the state machine is initialized and it sends the very first bit of data. In state 1, the next bit is sent, even if is a data bit or an address bit. In state 2, we return to state 1 in order to send all the data bits. If the bitcounter reach its highest value, (=width of data), we check if there are other data to be transferred, in state 3, if so, we return to state 1 and if not we return to state 0. The main condition that the SPI Transmitter can receive data is **TVALID=1** and **TREADY=1** during a high-transition of the clock.

The last step for the SPI transmitter, was to add an AXI Stream FIFO (First In First Out). This FIFO is the interface between the SPI Transmitter and the ARM processor. It will provide a memory-mapped interface, which we can talk to, using the C-code.

I2S - Inter-IC Sound Protocol

The next step, was to implement the I2S Transmitter IP. The I2S is an electrical serial bus interface standard used for connecting digital audio devices together. It is used to

communicate PCM audio data between integrated circuits in an electronic device. The I²S bus separates clock and serial data signals, resulting in simpler receivers than those required for asynchronous communications systems that need to recover the clock from the data stream.



The LRCLK signal indicates whether the data are sent to the left channel (0) or to the right channel (1). The BCLK is the bit serial clock and the SDATA the data that are sampled and transmitted. The mclk signal is a modified clock signal, generated by an ip that i constructed, called “myprescaler” and it has 4 times bigger frequency than the bclk.

The sampling frequency is ~40kHz. Every transfer has 32 bits width, the upper 16 bits are the left channel bits, and the rest are the right channel bits.

The I2S protocol ip, is described by a state machine with 5 states (0, 1, 2, 3, 4). Firstly, the new data are stored only when TREADY = 1 and TVALID = 1. The “valid” signal, gets the value 1 only when there are data to be transmitted (when the buffer is not empty). Then, send the first 16 bits, MSB first, and finally the rest 16 bits.

The last step for the I2S was to add a DMA (Direct Memory Access) in order to transfer the data from the system memory to the I2S IP.

DMA - Direct Memory Access

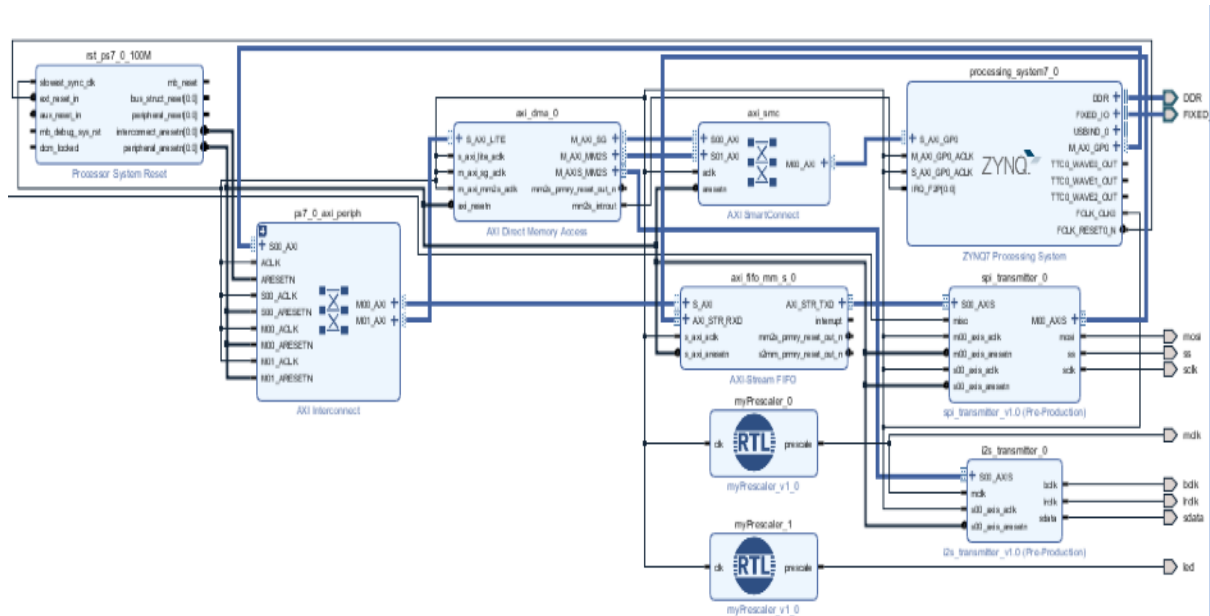
Direct memory access (DMA) is a feature of computer systems that allows certain hardware subsystems to access main system memory (random-access memory), independent of the central processing unit (CPU).

Without DMA, when the CPU is using programmed input-output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an interrupt from the DMA controller (DMAC) when the operation is done.

Thus, I added a DMA component to the block design in order to access the data and transfer them to the I2S component.

MyPrescaler IP

This IP, produces a clock pulse which constitute the mclk input of the I2S. I have also used this ip component, for the production of a led light, which indicates that the Verilog code is functional.




SDK - Software Implementation

After implementing the IP components, generating the bitstream and exporting the hardware, I initialized the DMA component, and the SPI Transmitter, and finally I exported the data from the SD Card. Then the .wav files are translated into digital signals and transmitted to the DMA.

By default the chip uses I2C as an interface. But it can be switched into SPI mode by pulling /CLATCH down three times. This is done by reading from address 0x4000 three times during initialization. The initialization is completed when the input and output ports that we need (for example the headphones) are unmuted and ready to be used.

Finally, I used minicom in order to manage the .wav files, which can be selected by Up and Down buttons on the keyboard, and played by pressing Enter. The volume can also be changed by Left and Right buttons.

```
>  vbiliouri : sudo — Konsole
File Edit View Bookmarks Settings Help
[ VOL = 400% ]
*LOOPER~1.WAV
LOOPER~2.WAV
LOOPER~3.WAV
VOLUME~1.WAV
LOOPER~5.WAV
PIANO.WAV
LOOPER~6.WAV
UP : Previous file      LEFT: Volume -
DOWN : Next file       RIGHT: Volume +
RETURN: Play
SPACE : New SD Card

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7 | VT102 | Offline | ttyACM0
> vbiliouri : sudo
```