Audio Tutorial

This tutorial explains how to use the audio inputs and outputs on the DE1_SoC. This includes discussion of the code package we provide for controlling these ports. These code files have been adapted from example code developed by Altera.

The Audio Hardware

The DE1_SoC has three audio jacks: a line out, a line in, and a microphone jack. In this tutorial, only the line out and microphone in are used. The jacks interface with a Wolfson WM8731 audio CODEC (coder / decoder) chip. This chip has ADCs (analog to digital converters) and DACs (digital to analog converters) connected to the analog jacks and connects to the FPGA via a digital interface. The Wolfson chip has many configuration options, but only a small subset of its functionality has been implemented in the code that comes with this tutorial (you very likely don't need any other functionality). It may be useful to look at the datasheets for the DE1_SoC and Wolfson chip to get a more complete picture of the hardware.

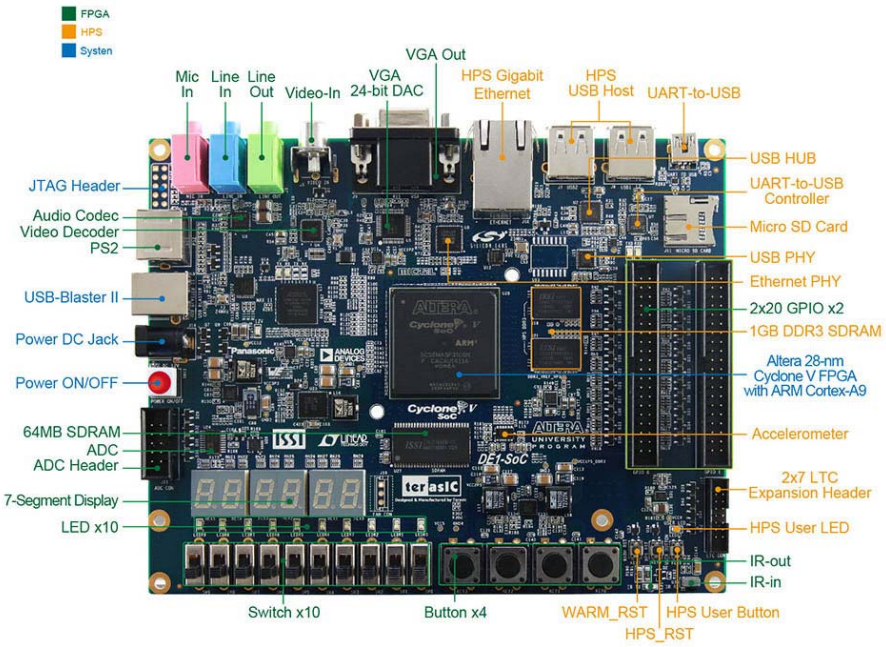

Figure 2: Photo of DE1-SoC Showing Audio Jacks

From DE1-SoC Specifications Page

http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=836&PartNo=3

Interface with User's Design

The job of the code files that come with this tutorial is to abstract away the complexity of configuring the Wolfson chip and interfacing with it. The user is presented with a straightforward module that allows reading incoming audio samples from the microphone jack

and writing outgoing audio samples to the line out jack. Audio samples are expressed using PCM (pulse code modulation). In PCM, the instantaneous level of an analog signal is sampled at regular time intervals and these levels are quantized, or expressed digitally. Figure 2 gives an example of sampling and quantizing a signal. In this case, there are 26 samples, each being expressible using four bit, signed, two's complement numbers.

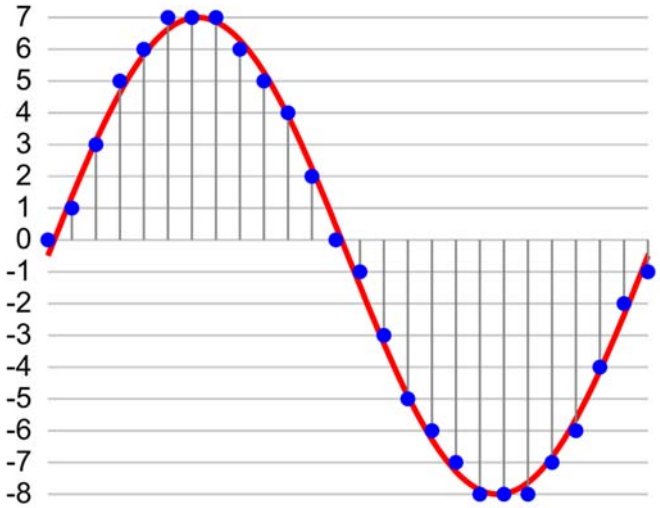

Figure 2: An Example of Pulse Code Modulation

Adapted from Wikimedia commons file PCM.svg
Shared using Creative Commons CC BY-SA 3.0 license

The audio driver given in the code files with this tutorial configures the Wolfson chip to use PCM samples in the following format:

- Sampling Frequency: 48kHz
- Resolution: 24 bits
- Representation: Signed, two's complement

A 50MHz clock connected to the audio driver provides the timing for all signals that go to the user's design. The user must provide two 24 bit signals (the left and right channels of stereo sound) of output data to the DAC for speakers, and the user may accept two 24 bit signals (the left and right channels of stereo sound) of input from the ADC. An "advance" signal generated by the audio driver indicates when a sample of input audio data is available, and when the next sample of output audio ought to be generated. This signal will periodically go high for one clock cycle, about 48,000 times per second (the sampling frequency), and valid data for that sample must be available on the DAC signal lines at that time. This also indicates the only clock cycle for which the data on the ADC signal lines is known to be valid.

Thus, whenever the advance signal is true, the user should do both of the following:

1. Provide output data to the speakers that is steady and correct by the clock edge. Note that it may be easiest to always be providing output data, and only go on to the next value once you see the advance signal is true at a clock edge.

2. Accept input data from the inputs and capture it into a register or other clocked logic on the clock edge when the advance signal is true. Thus, you can use the advance signal as an enable signal on an enabled register.

Conceptually, the advance signal indicates to the user's design when they are finished with the current sample and should advance to the next. All other inputs and outputs to the audio driver are used to control the Wolfson chip and read or write data and should be connected directly to top level pins. Here is a full list of connections:

Values you must provide:

- CLOCK_50 – 50MHz clock
- reset – active high reset: 1 resets the audio system, 0 means run normally.
- dac_left – 24 bit signed audio samples to the left DAC channel
- dac_right – 24 bit signed audio samples to the right DAC channel

Values provided to you to use:

- adc_left – 24 bit signed audio samples from the left ADC channel
- adc_right – 24 bit signed audio samples from the right ADC channel
- advance – periodically high for one clock cycle to indicate when all DAC/ADC data valid

Signals to or from the external Wolfson chip (hook 'em up and ignore):

- FPGA_I2C_SCLK – top level pin I^2C clock
- FPGA_I2C_SDAT – top level pin I^2C data
- AUD_XCK – system clock for Wolfson chip
- AUD_DACLRCK – left/right channel DAC clock
- AUD_ADCLRCK – left/right channel ADC clock
- AUD_BCLK – bit clock for serialized data
- AUD_ADCDAT – serialized ADC data
- AUD_DACDAT – serialized DAC data

Using Quartus II

The easiest way to use the audio driver is to add files from the code package to an existing project in Quartus. Add all .v, .sv, .qip, and .sip files from the starterkit/ directory to the project using the "Add/Remove Files in Project…" option under Quartus's "Project" menu.

Using Modelsim

Some extra effort is needed to simulate designs with audio. A module used to emulate the functionality of the Wolfson chip, wolfson.v, has been included with the code package. This file also contains a short example test bench. The intention of the test bench is to instantiate two modules, one being the top level design, and the other being the Wolfson chip emulation. The relevant connections shared by these modules ought to be wired together, as shown in the example test bench. A wolfson.do file has also been included for use with Modelsim. This file will need some modification.

- Line 2: Set the COMPILE_LIBS flag to 1 for your first simulation. The compiled libraries will remain for subsequent simulations so you may set the flag back to 0 for simulating in the future as this will significantly reduce the time it takes for the simulation to complete.
- Line 4: You may need to change the path on this line depending on your version of Quartus or its install directory. This path points to simulation models provided by Altera.
- Line 52: Replace example.sv with the file name of your design and add more vlog lines for any other files in your design.

Code Package

The code package that comes with this tutorial contains:

- wolfson.v – for emulating the Wolfson chip in simulation
- wolfson.do – a .do file for simulating the design in Modelsim
- starterkit – this directory contains the audio driver implementation
- starterkit/audio_driver.sv – the top level audio driver module

Tutorial developed by Professor Scott Hauck and Kyle Gagner
Code for audio driver adapted from tutorials supplied by Altera
5/18/15