2/25/2021

Aizaz Ali Larik

ME-Electrical Engineering (Electronic system design)

**Design & Implementation of microphone and ADC Port**

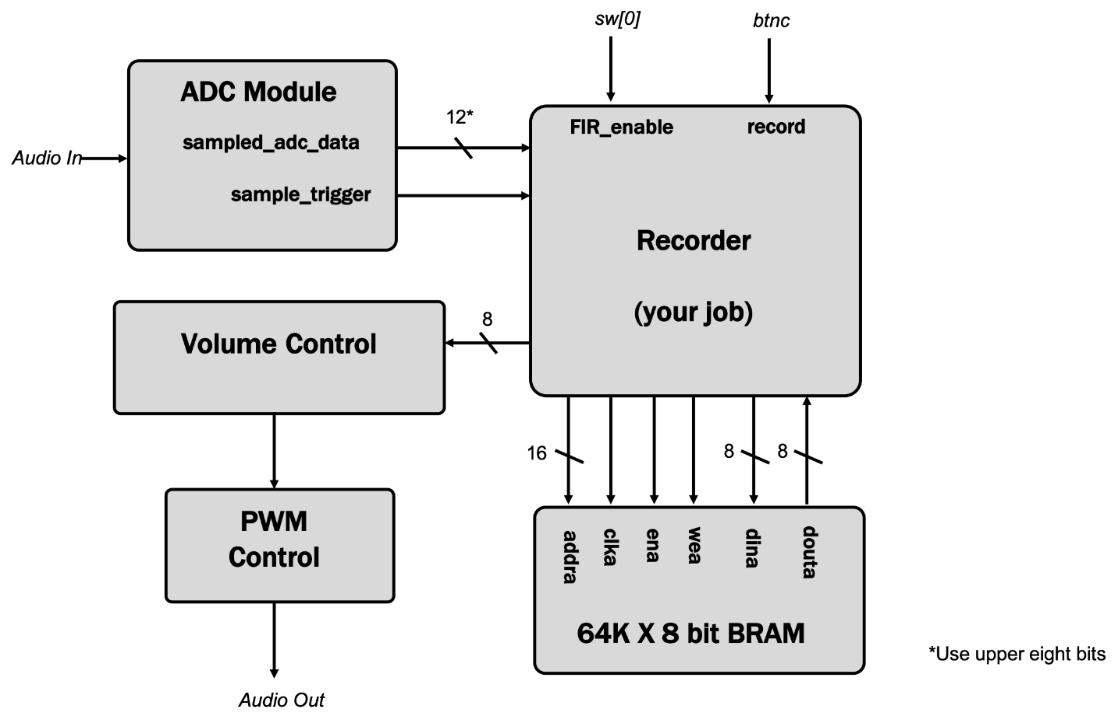
**Design and Implement hardware to receive audio samples, store, process, and send to audio out using NEXYS 4 DDR.**

**Introduction**

The objectives of this project are:

* Set up and read signals from an external microphone using the built-in Analog to Digital Converter that comes with most Series 7 Xilinx chips
* Build a voice recorder that records and plays back signed 8-bit digital audio samples.
* Implement a low-pass FIR filter module that can be used as an anti-aliasing and reconstruction filter.

We have created instances of two IP modules (XADC Wizard and Block Memory Generator) that are used in this project. We have developed the recorder module which will interface with data coming from the Analog-to-Digital Converter, store data in and/or read data from Block RAM (Random Access Memory), and then send an output signal to a PWM-audio generator.

[](http://web.mit.edu/6.111/volume2/www/f2019/handouts/labs/lab5_19a/resources/system_overall.png)

**Audio Input and the XADC Wizard**

The Artix 7 FPGA on our Nexys 4 board is equipped with an input between the analog world and the primarily digital world of the FPGA. We have used the XADC wizard in order to break out this functionality. We created an instance of the XADC interfacer. The code to interface to it is in the *top\_level*module. We added a new IP instance of XADC with following specifications:

* DRP Mode (This sets up the ADC as a “Dynamic Reconfigurable Port”)
* Single Channel
* Continuous Mode
* DCLK is 100 MHz
* Single Channel pick: VAUX3P and VAUX3N (This specifies which external inputs are being sampled (AD3P and AD3N on the Nexsys board)

**Block RAMs (BRAMs)**

We have used the FPGA's block RAMs (BRAMs) to build the memory for saved audio samples. A good size (i.e., one that fits in the FPGA we have) for the memory is 64K locations of 8 bits. To increase the recoding time, we have down-sample the 48 kHz incoming data to 6 kHz, i.e., only store every eighth sample.

**Low Pass Filter**

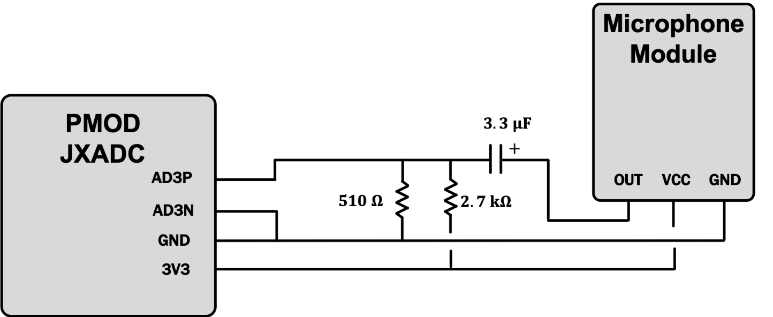
Remembering about Nyquist rates, the original 48 kHz data can represent audio frequencies up to 24 kHz. Down-sampling to 6 kHz yields data that can represent audio frequencies up to 3 kHz. In order to prevent aliasing during the down-sampling process we'll need to remove audio frequencies between 3 kHz and 24 kHz from the data before down-sampling by passing the incoming samples through a low-pass anti-aliasing filter.

The outgoing data stream wants samples every 48 kHz, which we produce by replicating each stored 6 kHz sample eight times. But if we do that we will hear 6 kHz noise (and its overtones) introduced by the replication process. So we'll pass the outgoing samples through a low-pass reconstruction filter to ensure that the 48 kHz output stream only contains audio frequencies up to 3 kHz.

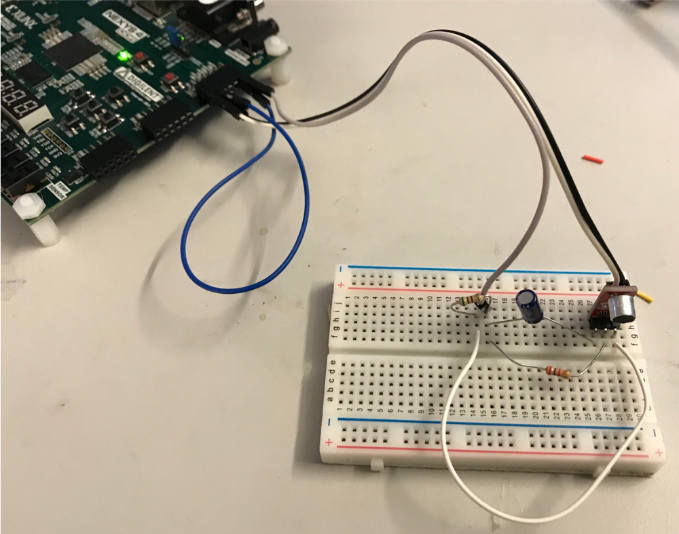
In fact we can use the **same** low-pass filter as both an anti-aliasing filter during recording and a reconstruction filter during playback.

**Hardware Description (ADC)**

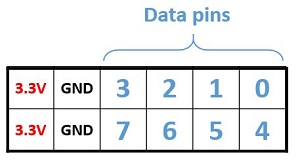
An ADC interface to the AD3P and AD3N pins on the JXADC PMOD port (the upper right PMOD port on the board). This ADC will measure the voltage between the AD3P and AD3N pins. The maximum value this port can measure is +1V and minimum is 0V, using 12 bits to quantize the measurements (12'h000 is the lowest value and 12'hFFF is the highest). We have attached a small microphone module to the ADC input channel using a simple circuit shown below.

[](http://web.mit.edu/6.111/volume2/www/f2019/handouts/labs/lab5_19a/resources/microphone_schematic.png)

In real-life the setup looks like the following:

[](http://web.mit.edu/6.111/volume2/www/f2019/handouts/labs/lab5_19a/resources/mic_wiring.png)

The PMOD connection has the following general pin format, with AD3P and AD3N being on pins 0 and 4, respectively.

[](http://web.mit.edu/6.111/volume2/www/f2019/handouts/labs/lab5_19a/resources/pmod.png)

This circuit creates a mid-point voltage of ~0.5V by making a voltage divider with the two resistors, and then adds the AC component of the microphone signal onto that offset. Because the ADC only has a 0-to-1V range this ensures that the bulk of the audio signal is centered and “capturable” by the ADC.

The ADC as set, will sample the microphone approximately 1 million times a second. We have down-sample it to 48 kHz manually using a parameter called SAMPLE\_COUNT in our Verilog.

The audio signal coming in from the microphone varies from 0 to 1.0V with a 0.5 V offset (no sound therefore corresponds to about 0.5V). Consequently a “zero” reading is going to have a value of 12'b1000\_0000\_0000, while a maximum reading will be 12'b1111\_1111\_1111 and a minimum reading will be 12'b0000\_0000\_0000. Our entire audio pipeline deals with signed 2's complement numbers, however. A quick way to convert this singled-sided ADC signal is “offset binary” to 2's complement is to invert the MSB.

**Modular Approach**

We have used modular approach in this project. This section defines the modules used in it and how these are connected to it.

***Sine Generator Module***

The *sine\_generator*module generates sine waves based on the specified PHASE\_INCR parameter . The sine generator represents 2π of phase using an unsigned 32 bit number (phase is mod 2π in the fact that a 32 bit number automatically wraps around as well).

To tune the sine generator's frequency, we specify the amount of normalized phase (0 to 232−1 for 0 to 2π, respectively) to step on each *step\_in*pulse, which is at 48 kHz. The module defaults to 750 Hz, which means PHASE\_INCR starts at 1/64th of the sample rate of 48 kHz (which is why PHASE\_INCR's default is specified that way). If one wanted a different frequency, such as 440Hz, the new phase increment step would be calculated as follows:

***Sine look up Module***

The top 6 bits of the phase generated is then used to “look up” a corresponding amplitude and that is ultimately sent out of module.

***Recorder Module (Sine Wave Generator)***

The starting form of the recorder module is comprised of two sine-wave generators which are selectively routed to the output (using a repurposed *filter\_in* signal to choose between a 440Hz and 750Hz tone.

***Recorder Module (Interfaced with microphone)***

Voice Recorder module will work in two “modes” described below:

* **Record Mode:** Store the stream of incoming samples in memory
* **Playback Mode:** Feed the stored data stream back

***Volume Control Module***

The output of the recorder is fed into a *volume\_control* module which scales (by factors of two) the amplitude of the signal it is provided. You can adjust the volume of the playback using sw [15:13]. When sw [15:13] is 3'b111, volume is maximized and when it is 3'b000 volume is minimized (but not zero).

***PWM Module***

Volume Control then sends its output to the PWM module which converts the signed 8 bits back to 8 bit offset binary, and then generates a PWM signal at 400 kHz used to drive the amplifier.

Low Pass filter consists of the following modules:

***fir31 Module***

It will act as a 31 tap Finite Impulse Response Filter (FIR Filter).

***coeff31 Module***

A combinational module that returns a signed 10-bit filter coefficient given a tap number between 0 and 30.

The filter calculation requires forming the following sum:

When filter is 0, your recorder module should behave as before. When the filter switch is 1 the fir31 module should be used as an anti-aliasing filter during recording and as a reconstruction filter during playback.

Following table shows the connections during various modes of operation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mode** | **Filter** | **Filter Input** | **BRAM Input** | **data\_out (audio)** |
| Record | Off | Don't care | mic\_in | mic\_in |
| Record | On | mic\_in | filter\_output[17:10] | filter\_output[17:10] |
| Playback | Off | Don't care | Don't care | replicated BRAM out |
| Playback | On | zero-expanded BRAM out | Don't care | filter\_output[14:7] |

**References**

<http://web.mit.edu/6.111/volume2/www/f2019/handouts/labs/lab5_19a/>

<https://github.com/Digilent/Nexys-4-DDR-OOB>