University of Toronto

Electrical and Computer Engineering

ECE 532 – Final Report

Digital Pedalboard

Márcio Moura Bridon Júnior

David

Toronto, Canada

April, 2015

1. Overview

a. Motivation

Pedalboards are widely used by guitar players. Through the 80's and 90's, where many effects were being developed, all those equipment were analogs systems. However, nowadays with the increasing development of the digital world, we are able to reproduce almost any effect with a high fidelity. Thus, companies can deliver a product that is very flexible in regards to the usability and modification of the sound, and also the digital pedalboards are much cheaper compared to a few analog pedals. Furthermore, our motivation related with the execution of a digital pedalboard is related with the curiosity that we both had in executing a hardware design to perform audio processing. As DSP's are the most common unit to process audio signal, we would like to see if a custom hardware system would deliver a high fidelity sound and would be as fast as the DSP can be.

b. Goals

The main goals were related with the capability that the hardware system of a FPGA be as fast as a DSP processor. Nowadays, digital pedalboards are systems that allow musicians to choose almost any emulation of an analog system. Thus, the main goal is to create a chain of effects that contains a high quality of sound, and also a high flexibility to allow the user to change as many parameters as possible to reproduce with high fidelity the system that is being emulated.

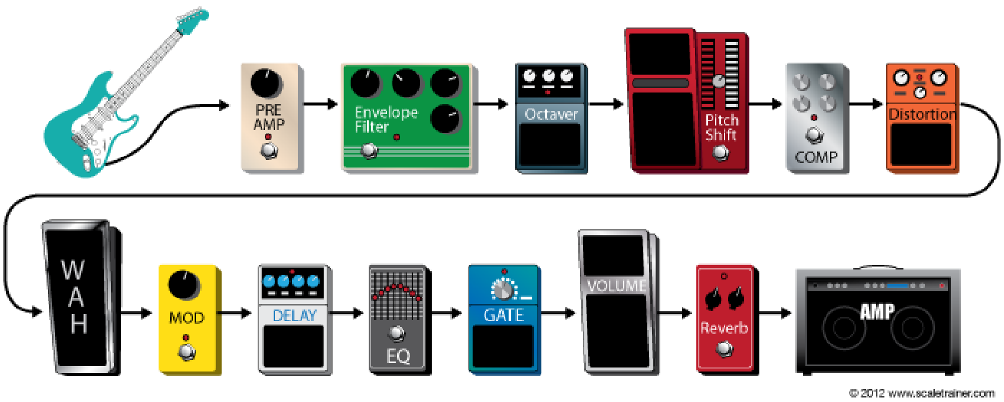


Figure 1. Chain of effects. Most of the pedals are analog

Figure 2. Chain of digital effects in a digital pedalboard

The system that we had in mind would act as a real digital pedalboard. We would have a user interface, in which the user would change the parameters of the effects and also to choose which effect the user would like to use. In addition, we also wanted to make an implementation that would allow the user to make recordings of the audio signal, thus the user would be able to play on the top of a base sound created. Related with effects, the aim was to produce a Delay, an Echoe, a Chorus, a Flanger and a Reverb. Thus, those were the features we wanted to add into our system.

c. List of IPs

Name of IP	**Created by**	**Feature**
XADC	Xilinx	The XADC IP is an analog to digital converter system that is hardwired onto the FPGA.
DDS Compiler	Xilinx	The DDS Compiler is an IP that generates sin/cos functions, as well as the phase. The core has a high flexibility in regards to the frequency of the wave and also the width of the outputs.
Circular Buffer	Márcio	The module instantiates a BRAM that has multiple read pointers and one write pointer. The block drives two effects to its output. One of the effect is an Echoe and the other is a Delay. The IP has some flexibility, in a way that it is possible to have from 1 to 5 read pointers, and also the distance among each pointer (including the write pointer)
Chorus Effect	Márcio	The module instantiates a small BRAM with the objective to generate a circular buffer, with a start delay of 50ms among the read and the write pointer. The delay varies +10 ms and -10ms, and it is determined by a LFO (low frequency oscillator), which is a sine wave with a frequency of 1 Hz. Thus, the block delivers to its output a delayed signal that is in a range of 40 to 50 milliseconds related with the input.

XADC Control	Márcio	The XADC convert samples in a 1MSPS rate in the continuous mode. Thus, we needed to implement a logic that would be able to control the XADC to convert data in a rate of 48KHz.
uBlaze	Xilinx	A simple processor with 64KB of instruction and data memory.
MIG7	Xilinx	A 128MB DDR2 memory that operates on a 200MHz clock. It uses an AXI_LITE bus.
FIFO generator	Xilinx	Sustains stable data transfers between 100MHz and 200MHz clock regions.
TFT controller	Xilinx	A controller that generated v_sync, h_sync, RGB colors from video memory and TFT clock.
Clock Wizard	Xilinx	Generates multiple clock signals.
GPIO	Xilinx	Reads and writes to board peripherals such as buttons, LEDs and switches.
UART	Xilinx	Establishes communication between FPGA board and SDK
PWM module	David	Converts 12 bit values into a stream of PWM samples
VGA color correction module	David	Truncates 6 bit color from TFT_RGB to 4 bits for output to board pins
Effects/Parameters Module	David	Takes registers written by SDK and outputs them as wires.

2. Outcome

The project was not entirely successful because we didn't integrated both User Interface implemented and the hardware effects in time. Although both parts were complete, we could not demonstrate both working together. The results are shown in the table below:

Feature:	**Proposed:**	**Result:**	**Details:**
Effects	The effects proposed were: Filter, Flanger and Reverb.	The effects generated were Delay, Echoe and Chorus.	The effects could be tested only at the last week of the project due previous problems to bypass the guitar signal correctly. We have decided to develop the Delay, Echoe and Chorus effect during the development of the project. Also, the Chorus effect is slightly similar to the Flanger effect.
Loop	The loop feature was proposed to make the system able to record samples of music into DDR and read it repeatedly.	We didn't work with this feature.	

Input Circuitry	The input circuitry was proposed to buffer the input signal and make it not go above 1.8V, which would prevent any electrical problem with the board from happening.	We didn't use any circuitry to control the voltage of the guitar's input.	The input circuitry was a huge concern, because it was filtering the signal of the guitar, although it was a high-pass circuit used in audio applications.
Audio Output	Originally, the DAC on the board was to be used with an n-bit digital audio sound level as input.	This was not feasible since the DAC only accepts PWM or PDM. Thus, a PWM module was developed which takes in 16-bits (higher 12-bits used) values and outputs corresponding PWM signals (2^{12} cycles per value).	Design of this block was accomplished successfully.

The future steps for this project is to build a chain of effects. For instance, you would be able to use two effects at the same time, and also to implement the recording functionality, and also to add more audio effects.

3. Project Schedule

The table below shows a summary of the weekly report, which contains the main goals reached and also some details of what we have experienced through that week, in which could be something that could be struggling the team or some solution for a problem.

Week	Proposed:	Result:	Details:
1	Implement the AD block to route the signal inside the FPGA	We implemented the XADC and the FSM to control the XADC.	We had to seek help with the TAs to understand the usage of the IP, thus we could set the correct settings.
2	Build the input circuitry and the connection among the XADC and the DAC.	We built the input circuitry and we were testing it to see if it was not filtering my input signal. Also, we built a FIR Filter that later on we didn't used. We tried to bypass the signal of the guitar but we were with the input circuitry problem	Márcio had lots of problems with Vivado. All the designs that he was doing was giving a clock_wiz problem and this was happening because of the system language of the computer's OS. Thus, we lost some time to

			understand why this problem was happening.
3	Solve the issues with the Vivado tool	Changes at the input circuitry to avoid filtering, and an Echoe effect which would demand a circular buffer. The effect wasn't tested because we had the XADC problem.	We were having problems with the XADC input, in which we could not specify the constraint of the IP inputs to the pins located at the PMOD.
4	Build a new effect and insert the system onto the board so we could test with the guitar.	Studying how we could generate a sine wave, thus we came with the DDS Compiler IP, and then we could study more the Chorus effect for its implementation.	We couldn't solve the XADC problem. However, we knew the input was hardwired, thus we were trying many options to solve this huge issue.
5	Solve the XADC issue and test the whole system.	We have corrected the XADC problem, thus we could bypass the signal inside the FPGA.	The problem with XADC disappeared when we built the system from scratch.
6 and 7	Simulating the effects and testing the system with the guitar, and also building new effects.	We started integrating the effects onto the system, and also debugging it to see if the effects were working, thus we could adjust the effects the way we wanted.	We had an issue with the effects: they were not working. After debugging the effects, changing them to see if we had another result and try to understand what was happening, we decided to look at the XADC output, and the error was there. The issue was with a *tvalid* signal, which was asserted all the time, triggering the effect unit in a frequency of 20 MHz instead of 48KHz.

Main Task	Problems	Week	Resolved/ successful	Solution
Test audio out	None	1	Yes	N/A
Create PWM module and refine it.	None	2	Yes	N/A
Setup VGA TFT controller as a custom IP.	None	3	Yes	N/A
Setup memory module (MIG 7 series DDR 2 12MB memory)	External clock pin missing from board repository.	4	Yes	Temporary solution was to make our own clock pin. Our permanent solution was provided to us by Charles (TA) who established the clock pin in a new board repository.
Setup VGA TFT controller using SDK.	The MIG7 memory module was not a slave to both uBlaze (write) and TFT controller (read) thus, VGA was not working properly. Also, as soon as the previous problem was fixed, VGA was flickering when SDK was running.	5	No	For the first problem, we had to adjust the axi_interconnect to have two masters: TFT controller and uBlaze. The second problem was left unresolved even after looking at documentation and getting help from professor
Finish the VGA GUI background	The VGA screen is still flickering when SDK is active.	6	Yes	The second problem as described by Charles was resolved by setting the axi_interconnect to maximum performance as opposed to custom.
Finish GUI dynamic foreground and communicate effects/parameters to hardware design modules.	The sampling rate of the buttons and switches used was too high to effectively poll properly.	7	Yes	After an action is triggered, a while loop is instantiated to keep looping until user has finished action.

4. Description of the Blocks

Below we are going to make a more detailed description of what we have developed throughout this project.

a. Input Circuitry:

The configuration of the Operational Amplifier is a voltage follower, so it works as a buffer, and it is also has high pass filter to avoid the input bias. Though, we couldn't get a good result mainly because of the Operational Amplifier, which was a Microchip MCP6002. This OpAmp is not specific for audio purposes, so it was modifying the output offset and it was low-filtering the output.

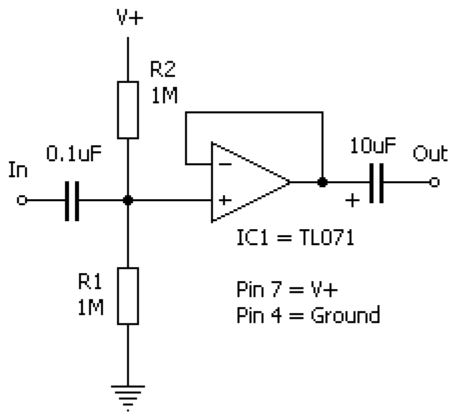


Figure 3. Unity-gain Buffer

b. Circular Buffer for Echo and Delay:

Circular buffer is a basic component to produce certain effects that requires any form of delay. The buffer implemented follows the structure showed by the image below:

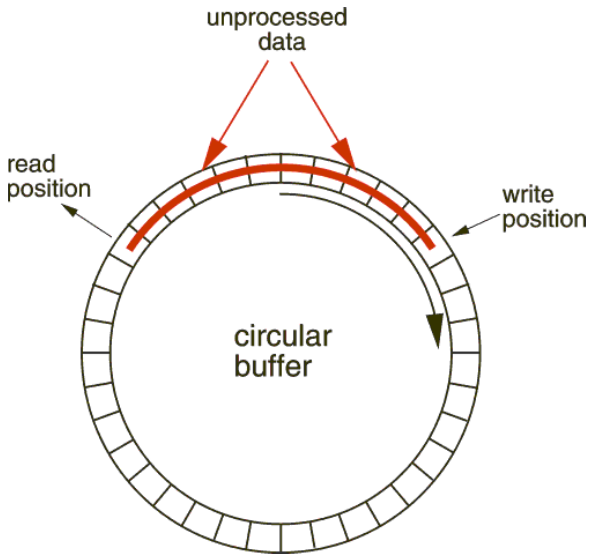


Figure 4. Circular Buffer

The FSM implemented to perform the read and write, and also to set the distance among the pointers, is presented below:

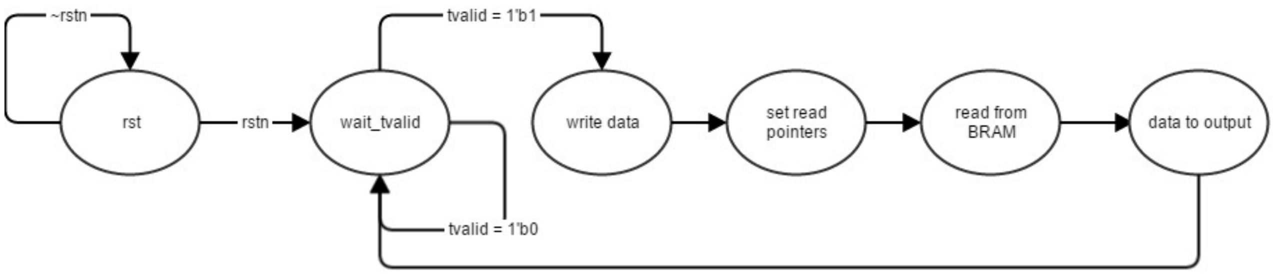


Figure 5. FSM that implements a Circular Buffer

As it is possible to see, the FSM waits for a *tvalid* signal that comes from the XADC to perform the write function, and after the read pointer reads the data from BRAM and sends it to the output. Also, when the data goes to the output, it automatically increments the pointers, and also checks if the pointer are in its maximum value.

Below we provided a Verilog description of this circular buffer, with an implementation of four read pointers, although it is possible to add more pointers easily:

```
always@(posedge clock) begin //FSM
        if(~resetn) ps <= s0;
        else ps <= ns;
end

always@(*) begin
        case (ps)
                s0: ns = s1;                        //If resetn = 1, state to wait for tvalid
                s1: if(s_tvalid) ns = s2; else ns = s1;    //When tvalid = 1, go to state 2
                s2: if(counter_write == 2'b11)ns = s3; else ns = s2; //Write
                 s3: if(counter == 2'b10) ns = s4; else ns = s3; //set_pointer_1
                s4: ns = s5;                    //read_pointer_1
                s5: if(counter == 2'b10) ns = s6; else ns = s5; //set_pointer_2
                s6: ns = s7;                    //read_pointer_2
                s7: ns = s1;                    //data_output
                default: ns = s0;
        endcase
end
```

5. Description of Design Tree

In this case, we have instantiated a BRAM in Verilog instead of using an IP. The code below shows how to implement a dual write/read port BRAM:

```
reg [15:0] ram [55000:0];
//Port 1
always@(posedge clock) begin
        if (wen1) ram[addr1] <= wdata1;
        rdata1 <= ram[addr1];
end

//Port 2
always@(posedge clock) begin
        if (wen2) ram[addr2] <= wdata2;
        rdata2 <= ram[addr2];
end
```

Now that we do have a Circular Buffer controlled by a FSM and also a BRAM inside the module, we can develop the Echo and the Delay effect. The output of the effects are shown below:

```
if(data_output) begin
            case(feedback_number)
              4'd2:
              begin
                 data_out_delay <= in_data + (data_read_1 >> 1);
                 data_out_echoe <= (in_data >> 1) + (data_read_1 >> 1);
              end
              4'd3:
              begin
                 data_out_delay <= in_data + (data_read_1 >> 1) + (data_read_2 >> 2);
                 data_out_echoe <= (in_data >> 1) + (data_read_1 >> 1) + (data_read_2 >> 1);
              end
              4'd4:
              begin
          data_out_delay <= in_data + (data_read_1 >> 1) + (data_read_2 >> 2) + (data_read_3
>> 3);
                 data_out_echoe <= (in_data >> 1) + (data_read_1 >> 1) + (data_read_2 >> 1) +
(data_read_3 >> 1);
              end
              4'd5:
              begin
                 data_out_delay <= in_data + (data_read_1 >> 1) + (data_read_2 >> 2) +
(data_read_3 >> 3) + (data_read_4 >> 4);
          data_out_echoe <= (in_data >> 1) + (data_read_1 >> 1 ) + (data_read_2 >> 1) +
(data_read_3 >> 1) + (data_read_4 >> 1);
              end
              4'd1:
              begin
                 data_out_delay <= in_data ;
                 data_out_echoe <= in_data ;
              end
              default :
              begin
                 data_out_delay <= in_data;
                 data_out_echoe <= in_data;
              end
            endcase
            //drive the sample to FIFO
            m_tvalid <= 1'b1;
            //Increment the addresses
            addr1_read <= addr1_read + 1'b1;
            addr2_read <= addr2_read + 1'b1;
            addr1_write <= addr1_write + 1'b1;
            addr3_read <= addr3_read + 1'b1;
      addr4_read <= addr4_read + 1'b1;
   end
end
```

The code above receives the parameter *feedback_number*, which defines how many feedbacks the block will send to its output (how many read pointers will send data to the output).

c. XADC Controller

As described above, the XADC controller forces the XADC IP to convert data in a 48 KHz rate. The FSM that controls the module is showed below:

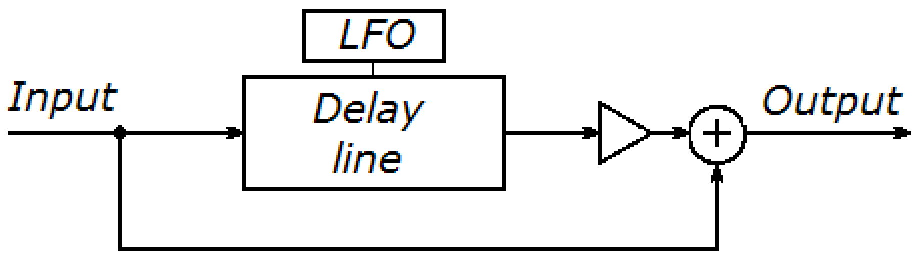
```
always@(posedge clock) begin
   if (~ext_reset_in)  ps <= S0; else ps <= ns;
 end

 always@* begin //combinatinal logic of the ASM Chart
   case (ps)
      S0: ns = S1; //initializes the counter to 0
      S1: if(counter == 12'b100000000000) ns = S3; else ns = S2; //if the counter is 2.260 goes to
next state
      S2: ns = S1; //increment the counter
      S3: ns = S4; //set the convert_start to 1'b1
      S4: if(busy_out) ns = S4; else ns = S0; // waits for the busy_out to drive low
      default: ns = S0;
   endcase
```

The conversion is set when the counter achieves its maximum value, thus the variable *convert_start* goes to 1 for one cycle. After, the FSM waits the *busy_out* output from XADC to be low in order to restart the FSM again from the first cycle. With this code, we can make sure we are feeding the system with a rate of 48 KHz.

d. Chorus Effect

The chorus effect implements a low frequency oscillator (sine wave of 1Hz) to model a circular buffer from a range to the minimum of 40ms to the maximum of 60ms. The same structure used for the delay effects were used to implement the chorus, and the LFO was implemented using the DDS Compiler, which is an Xilinx IP that generates look-up tables for sine and cosine waves.



e. MicroBlaze Processor

The soft-processor known as uBlaze is provided by Xilinx. The one that we used contains 64Kbytes for data and instruction memory, runs in a clock of 100MHz and allow the connection with multiple slave units.

The documentation that talks more about this processor can be accessed through this link: http://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v2_1/pg059-axi-interconnect.pdf

f. Clock Wizard

The clock wizard is a source of clock that feeds the digital circuit, and its input is the crystal clock from the board. At this project we used three outputs of 100 MHz, 200 MHz and 25 MHz.

g. TFT Controller

The TFT Controller reads from an attached memory module the color scheme data of each pixel in the normal order. The input of the IP is a 100MHz clock, a 25MHz clock for horizontal and vertical synchronization, and the output consists in a horizontal and vertical synchronization (of one bit to physical board pins). Other outputs are also the colors red, green and blue color scheme, which is 6 bit wide. More details can be found on: http://www.xilinx.com/support/documentation/ip_documentation/axi_tft/v2_0/pg095-axi-tft.pdf

h. GPIO Inputs

The GPIO were used to allow the communication among the user and the system. We have used two switches for the effect selection, the central press-button to select/deselect, and the buttons to up/down/right/left to flow through the options of the GUI.

i.MIG 7 Memory Controller

The memory controller used 2MB of 128 MB to store a single VGA frame, mainly RGB colors. The module reads and writes unsigned words (32 bits), and it is a slave to both TFT controller and MicroBlaze. The clock input was a 200MHz clock.

j.FIFO Generator

The FIFO Generator facilitated a proper data transfers between 100MHz and 200MHz clock zones, thus synchronizing this data transfer. We used a stream data transfer among the slave and master sides. The input was the data from the multiplexed effect and the output was towards the PWM.

k.UART

The UART establishes communication among the FPGA and the SDK.

l. VGA Color Correction

The module truncates a 6-bit RGB color scheme into a 4-bit color scheme. The input comes from the TFT Controller and the output drives the data to the VGA color pins.

m. PWM Generator

The PWM module translates a 12-bit value into a PWM signal at an interval of 48KHz. The module has as input a clock of 200MHz, and also a 16-bit data packet that comes from the FIFO. In addition, the output is a PWM waveform that goes to the audio jack of the board (mono out).

n. Effects Parameters

This module allows the communication among uBlaze and the effect blocks. It is an AXI4-Lite peripheral that stores two different parameters and the effect that it has been selected. These three registers are connected into the effect modules.

4.1 Communication among Blocks

IP	Slave to			
	uBlaze	TFT Controller	None	Other
uBlaze			Y	
Clock Wizard			Y	
TFT controller	Y			
GPIO	Y	Y		
MIG7 Memory	Y	Y		
FIFO Generator				XADC
UART	Y	Y		
VGA Color Correction	Y	Y		
PWM Generator				FIFO Generator
Effects/Parameters	Y	Y		

4.2 Block Diagram

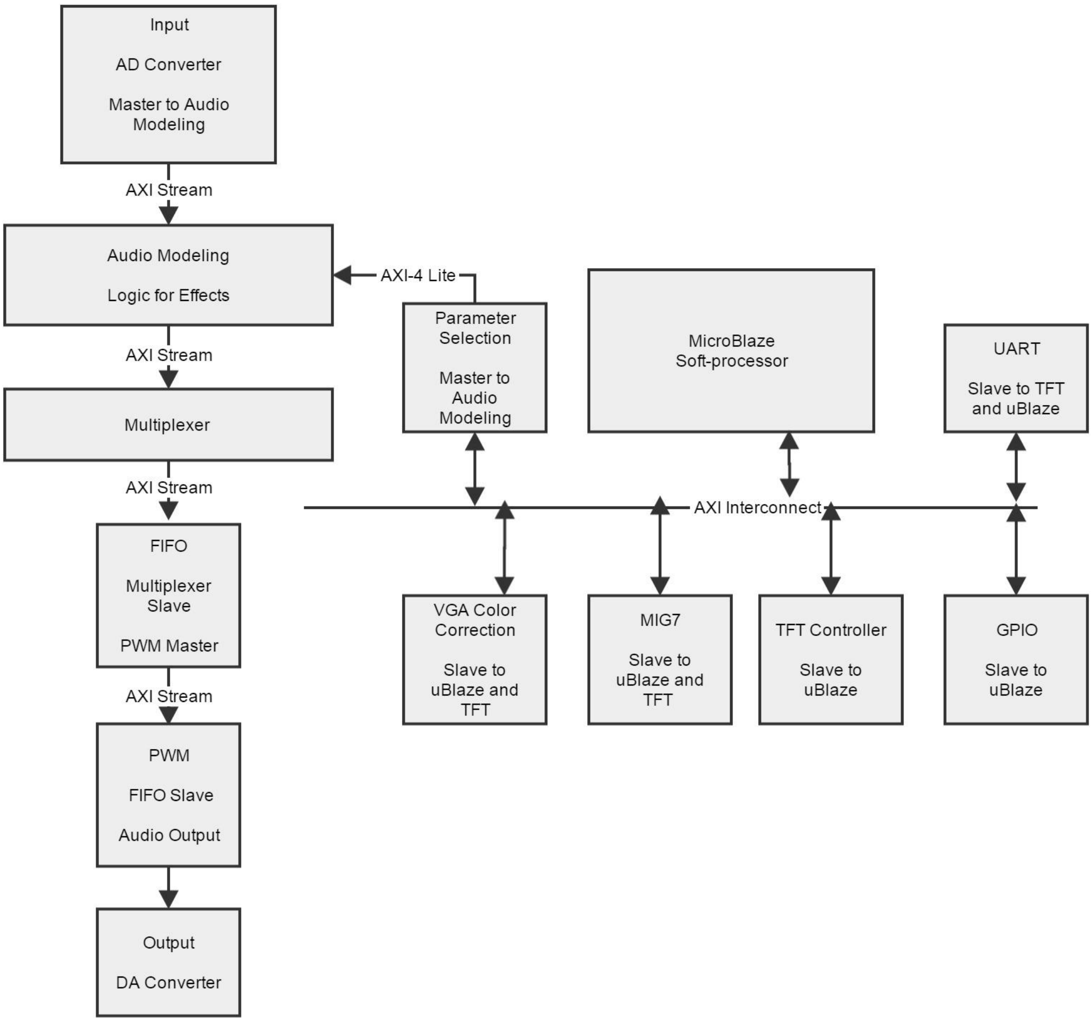


Figure 5. Block Diagram

5. Description of Design Tree

The main files of the program are:

chorus.v: Verilog file that contains the chorus effect;

circular_buffer.v: Verilog file that contains the delay and the echoe effect;

mux_effects.v : Verilog file that contains the multiplexer that chooses which delay it will be delivered to the output.

mux_tvalid.v : Verilog file that drives the tvalid signal of the effect that its being sampled.

xadc_fsm.v: Verilog file that controls the XADC sampling.

helloworld.c: C file that is the main program of the uBlaze.

Letters.c : C file that contains the functions to draw letters.

Letters.h: Header file for all the letter templates

6. Sources:

Figure 1. Image source:

http://globalguitarnetwork.com/wpcontent/uploads/2014/06/Effect-Order.png

Figure 2. Image source:

http://cdn.mos.musicradar.com/images/Product%20News/Tech/Jan12/zoom-g5-main-630-80.jpg

Figure 3. Image source: http://www.muzique.com/lab/buffers.htm

Figure 4. Image source: http://luaview.esi-cit.com/dl_pics/circular_buffer.png

Figure 5. Block Diagram: developed by the author.