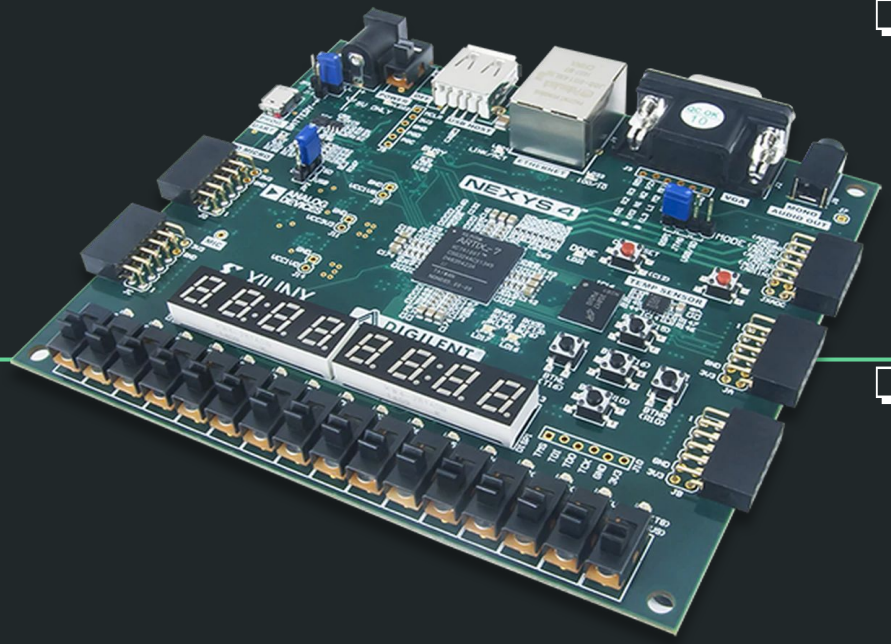
Digital Audio Effects Unit

4710: Computer Hardware Design

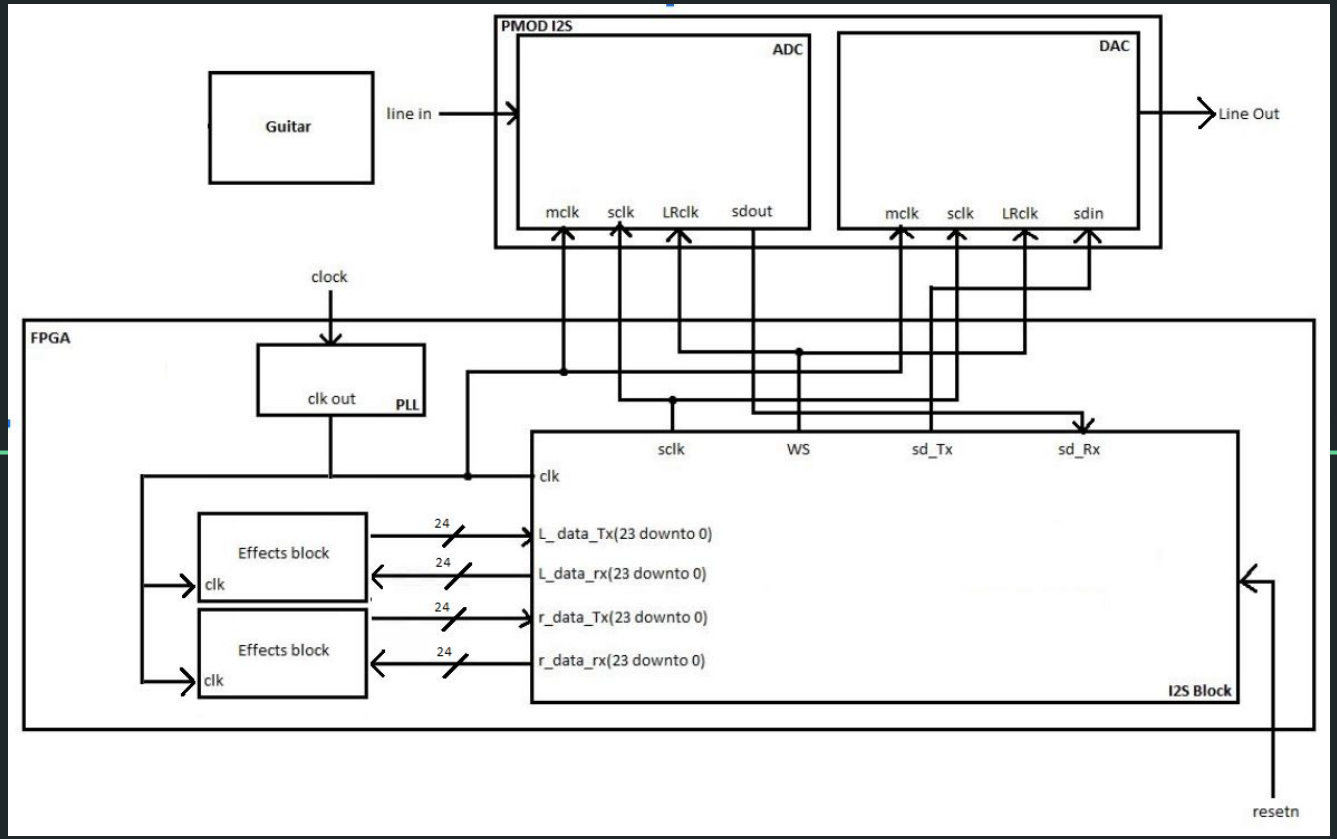
Group Members:
Alain Sfeir
Jessica Odish
Michael Bowers
Madison Cornett

Project Description

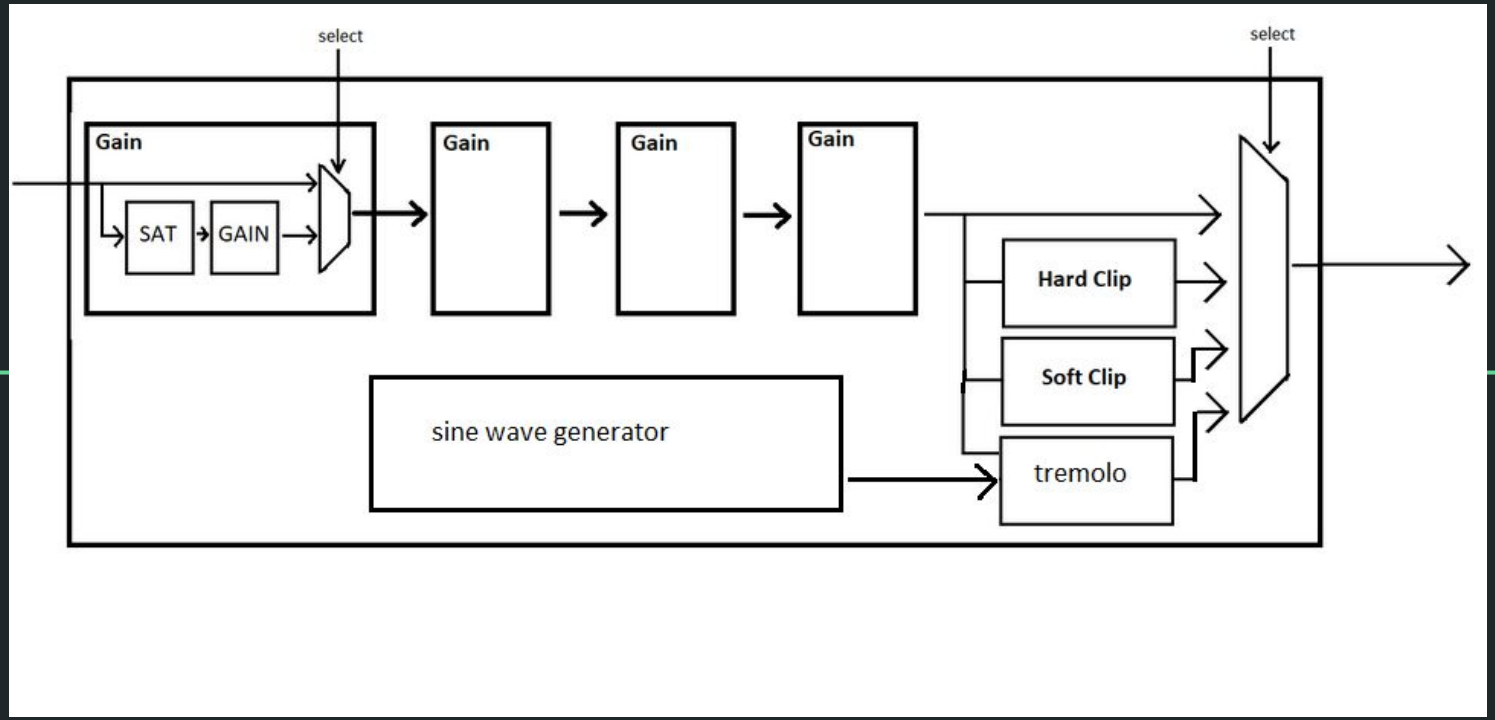


- The main purpose of this project is to create a digital audio effect unit that alters the incoming signal of a musical instrument via signal processing
- The project is implemented on a NEXYS-4 Artix-7 FPGA Trainer as well as utilizing an I2S serial communication protocol to receive and transmit audio data

Block Diagram



Effects Block



Experimental Setup

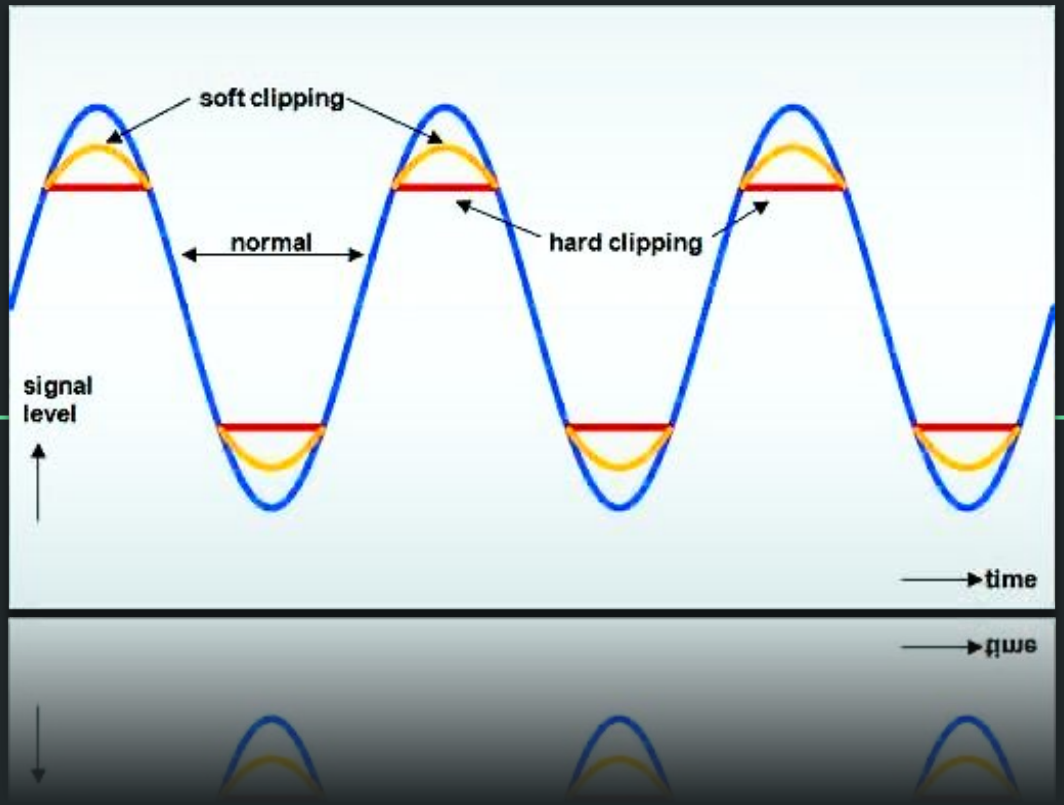
- Used Vivado 2017.2 for development of code
- The output parameters for the design were determined from the oscilloscope to allow debugging the code if the results were not as expected.
- A majority of the testing was done by running tests as well as listening closely to the result to see if the sound released was the desirable output.

Gain Control

```vhdl
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.numeric_STD.all;

entity gain is
    port(
        clock: in std_logic;
        resetn: in std_logic;
        enable: in std_logic;
        wordin : in std_logic_vector(23 downto 0);
        wordout: out std_logic_vector(23 downto 0));
end gain;

architecture bhv of gain is
    signal input: signed(23 downto 0);
    signal saturated: signed(23 downto 0);
    signal gained: signed(23 downto 0);
    signal word_regged: std_logic_vector(23 downto 0);
begin

    input <= signed(wordin);

    saturate: process (input)
    begin
        if input > x"3FFFFF" then
            saturated <= x"3FFFFF";
        elsif input < x"C00000" then
            saturated <= X"C00000";
        else
            saturated <= input;
        end if;
    end process saturate;
```

```vhdl
    gaining: process (saturated, wordin)
    begin
        if wordin(23) = '0' then
            gained <= saturated(22 downto 0)&'1';
        else
            gained <= saturated(22 downto 0)&'0';
        end if;
    end process gaining;

    reg: process (clock, resetn)
    begin
    if rising_edge(clock) then
      if resetn = '0' then
        word_regged <= (others => '0');
      else
        word_regged <= std_logic_vector(gained);
      end if;
    end if;
    end process reg;



    mux: process(enable, resetn, word_regged, input)
    BEGIN
        case enable is
            when '1' =>     wordout <= word_regged;
            when others =>  wordout <= std_logic_vector(input);
       end case;

    end process;
end bhv;
```

Hard/Soft Clipping



Hard Clip Distortion

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
--use ieee.std_logic_arith.all;...

entity hardclipdistortion is
    port(
        clock: in std_logic;
        resetn: in std_logic;
        wordin : in std_logic_vector(23 downto 0);
        wordout: out std_logic_vector(23 downto 0);
        clipfactor: in std_logic_vector(23 downto 0)
    );
end hardclipdistortion;

architecture bhv of hardclipdistortion is
    signal clip_val : std_logic_vector(23 downto 0);
    signal word_done : std_logic_vector(23 downto 0);
    signal word_regged : std_logic_vector(23 downto 0);
    signal word_in: std_logic_vector(23 downto 0);
begin

    clip_val <= clipfactor;                         -- clipping threshold set from outside of the block
    word_in <= wordin;

    clipper: process (word_in, clip_val)
    begin
        if (word_in(23) = '0') then             -- positive value in 2C
            if (word_in(22 downto 0) > clip_val(22 downto 0)) then
                word_done <= '0'&clip_val(22 downto 0);
            else
                word_done <= word_in;
            end if;
        else                                    -- neg value in 2c
            if (not word_in(22 downto 0) > clip_val(22 downto 0)) then
                word_done <= '1'&(not (clip_val(22 downto 0)));
            else
                word_done <= word_in;
            end if;
        end if;
    end process clipper;


    reg: process (clock, resetn)
    begin
    if rising_edge(clock) then
       if resetn = '0' then
           word_regged <= (others => '0');
       else
           word_regged <= word_done;
       end if;
    end if;
    end process reg;


    wordout <= word_regged;
end bhv;
```

Soft Clip Distortion

```
library IEEE;
use IEEE.std_logic_1164.all;

use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

-- adapted from http://users.os.cf.ac.uk/Dave.Marshall/CM0268/PDF/10_CM0268_Audio_FX.pdf...

entity softclipdistortion is
    port(
        resetn: in std_logic;
        clock: in std_logic;
        word_in : in std_logic_vector(23 downto 0);
        word_out: out std_logic_vector(23 downto 0)
    );
end softclipdistortion;

architecture bhv of softclipdistortion is


signal wordin       : std_logic_vector(23 downto 0);
signal wordtop_dist : std_logic_vector(7 downto 0);
signal word_regged  : std_logic_vector(23 downto 0);

type memory is array (255 downto 0) of std_logic_vector(7 downto 0);
signal RAM: memory := ( x"FE",x"FC",x"FA",x"F8",x"F6",x"F4",x"F2",x"F0",x"EE",x"EC",x"EA",x"E8",x"E6",x"E4",x"E2",x"E0",
                        x"DE",x"DC",x"DA",x"D8",x"D6",x"D4",x"D2",x"D0",x"CE",x"CC",x"CA",x"C8",x"C6",x"C4",x"C2",x"C0",
                        x"BE",x"BC",x"BA",x"B8",x"B6",x"B4",x"B2",x"B0",x"AE",x"AC",x"AA",x"A8",x"A6",x"A4",x"A2",x"A0",
                        x"9E",x"9D",x"9B",x"9A",x"98",x"97",x"95",x"94",x"92",x"91",x"90",x"8F",x"8D",x"8C",x"8B",x"8A",
                        x"89",x"88",x"87",x"87",x"86",x"85",x"84",x"84",x"83",x"83",x"82",x"82",x"81",x"81",x"80",x"80",
                        x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",
                        x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",x"80",
```

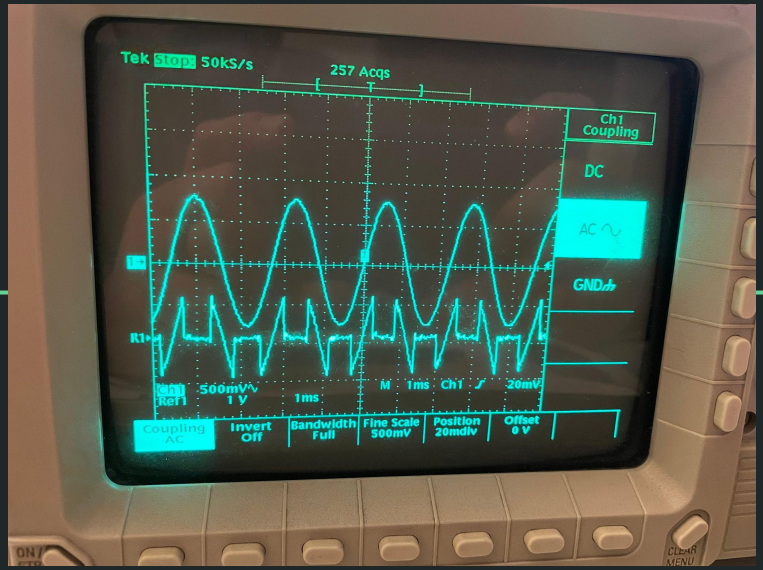
$$f(x) = \begin{cases} 2x & \text{for } 0 \leq x < 1/3 \\ \frac{3-(2-3x)^2}{3} & \text{for } 1/3 \leq x < 2/3 \\ 1 & \text{for } 2/3 \leq x \leq 1 \end{cases}$$

Tremolo

```
wordsgn <= signed(wordin);
signsgn <= signed(sinewavein);
multiplied <= wordsgn*signsgn(23 downto 16);
result <= multiplied (31 downto 8);

reg: process (clock, resetn)
    begin
    if (rising_edge(clock)) then
        if resetn = '0' then
            word_regged <= (others => '0');
        else
            word_regged <= std_logic_vector(result);
        end if;
        end if;
    end process reg;
 wordout <= word_regged;
```

Challenges Faced

- 8 bit processing on 24 bit audio
  - High frequency noise
- LUT creation
- Time Varying Filters vs Modulators vs Non-linear



Demo

https://youtu.be/IMAPpAksYhY