FPGA kitarski efekt

Projekt pri predmetu DIVS

Aljaž Zadravec,

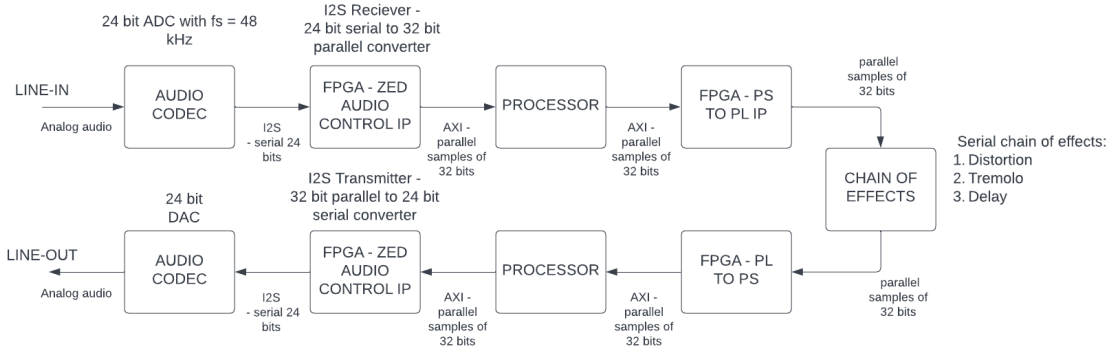
641 801 80

1 UVOD

Cilj projekta je bil narediti kitarski efekt, ki vključuje efekte distortion, delay in tremolo. Možnost izbire efekta in nekaj nastavitev parametrov efekta je možnih s pomočjo stikal in tipk, ki so doseglijve na plošči Zedboard. Procesor in FPGA na Zedboardu se nahajata na čipu, Xilinxova Zynq-7000 družina, ki temelji na SoC arhitekturi (Procesorski del in FPGA sta na istem čipu), XC7Z020-CLG484. Naloga je razdeljena na dva dela in sicer:

1. FPGA del, ki skrbi za obdelavo signala in I2S komunikacijo.
2. Procesor, ki skrbi za nastavitve Audio Codeca ADAU1761, in pošiljanje in prejemanje signala iz PL dela.

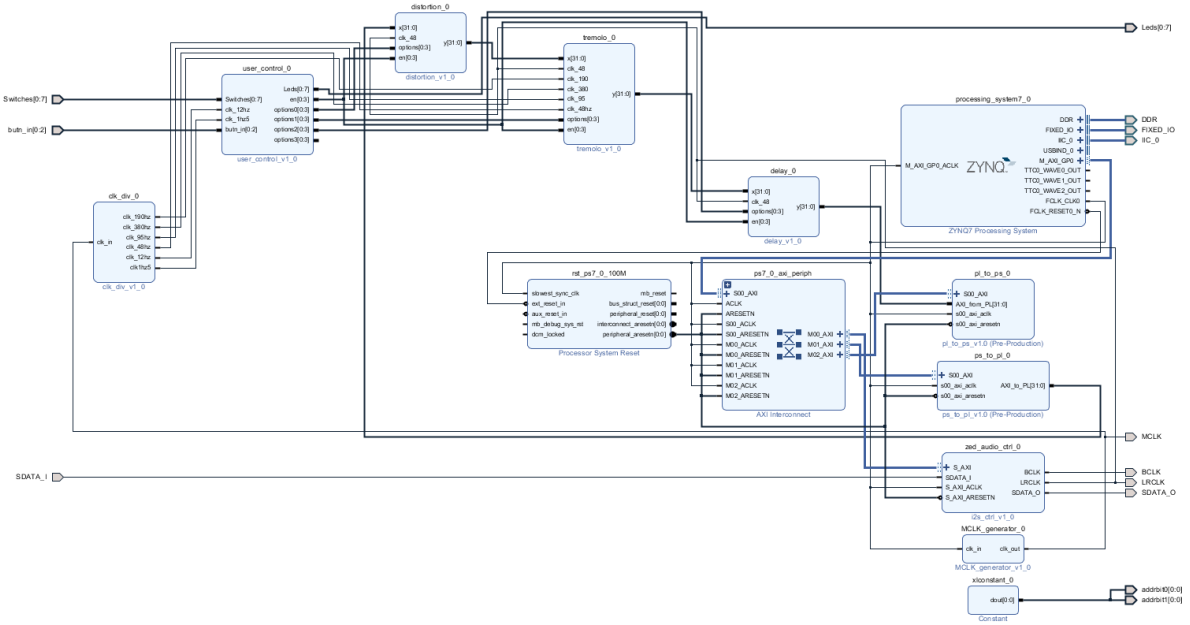
Zajem signala iz kitare se dela s pomočjo 24-bitnega ADC converterja, ki je vgrajen v Audio codec-u Analog Devices ADAU1761. Signal se vzorči s frekvenco 48 kHz. Potem se signal iz audio codec-a prenese preko I2S vodila do I2S recieverja, ki je realiziran na FPGA-ju (AXI4 LITE IP blok zed_audio_ctrl). V I2S reciever-ju se serijski 24 bitni podatki paralelizirajo v 32 bitne podatke, ki so primerni za AXI vodilo in se prek le tega prenesejo do procesorja. Procesor sprejme podatke in jih po AXI vodilu pošlje do FPGA-ja, kjer je IP blok ps_to_pl, ki podatke sprejme in jih pošlje naprej na serijsko verigo efektov. Po vrsti si sledijo v zaporedju distortion, tremolo in delay. Potem ko se signal v verigi efektov obdela se pošlje nazaj do procesorja s pomočjo IP bloka pl_to_ps po AXI vodilu, kjer procesor pošlje signal preko AXI na IP blok zed_audio_ctrl, ki podatke serializira v in preko I2S vodila pošlje na audio codec. Audio codec ima 24 bit DAC, ki serializiane podatke pretvori v analogni signal, ki gre naprej na ojačevalec.



Slika 1: Blok diagram poti signala.

2 FPGA

Za načrtovanje logike v PL delu sem uporabil VHDL. Celotni blokovni diagram je sestavljen iz zynq IP-jev ki se uporabljajo za komunikacijo med ps in pl delom ter zed_audio_ctrl, distortion, tremolo, delay, pl_to_ps, ps_to_pl, MCLK_generator, clk_div in user_control IP-jev, ki bodo opisani v nadaljevanju.

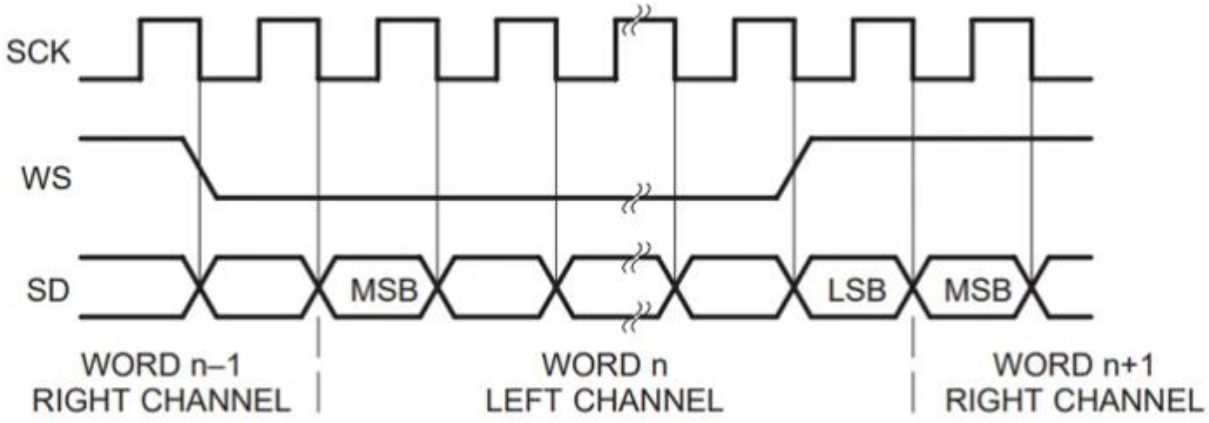


Slika 2: Blokovni diagram IP-jev.

2.1 Zed Audio Control IP Block

Je komponenta, ki je bila priloga knjigi The Zynq Book Tutorials. Omogoča komuikacijo I2S med Zynq čipom in audio codec-om, primerna je kot transmitter in reciever.

I2S je komunikacijski protokol, ki se uporablja za prenašanje digitaliziranega avdio signala. Za komunikacijo so potrebni trije signali, to so SD (serial data), WS (word select) in SCK (serial clock). Podatki se prenašajo po liniji SD, linija WS pa se uporablja za izbiro kanala (stereo prenos signala, pri mono prenosu se podatki prenašajo le ko je WS v visokem ali nizkem stanju, v tem primeru ima frekvenco 48 kHz) in SCK je ura na katero se pri pozitivni fronti zajemajo podatki na liniji SD.



Slika 3: I2S komunikacija.

2.2 MCLK generator in delilnik ure

MCLK generator je komponenta, ki se uporablja za delitev porcesorske ure na polovico, torej iz 100 MHz na 50 MHz. Izhodni signal iz komponente MCLK_generator je vhodna ura za audio codec. Komponenta clk_div pa se uporablja za deljenje ure, ki je izhod MCLK_generatorja na nižje frekvence, to so 380, 190, 95, 48 (uporabljene za tremolo efekt), 12 in 1.5 Hz (Za kontroli blok).

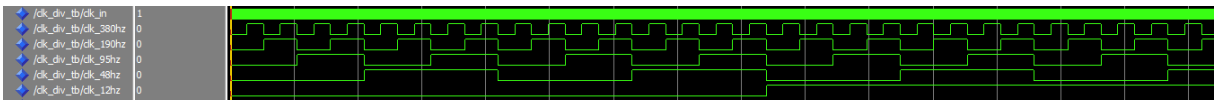
```
divide_by_2: process(clk_in)
begin
    if rising_edge(clk_in) then
        clk_sig <= not clk_sig;
    end if;
end process;
clk_out <= clk_sig;
```

```
count: process(clk_in)
begin
    if rising_edge(clk_in) then
        clk_cntr <= std_logic_vector(unsigned(clk_cntr)+1);
    end if;
end process;

--****** for control block *****
clk_12hz <= clk_cntr(21);
clk1hz5 <= clk_cntr(24);
--*******************************

--****** for tremolo  **********
clk_380hz <= clk_cntr(16);
clk_190hz <= clk_cntr(17);
clk_95hz <= clk_cntr(18);
clk_48hz <= clk_cntr(19);
--*******************************
```

Sliki 4 in 5: Prikazani sta kodi za delitev ure na polovico(levo) in na manjše frekvence (desno).



Slika 6: Simulacija delovanja delilnika ure.

2.3 PS_to_PL in PL_to_PS IP bloka

PS_to_PL:

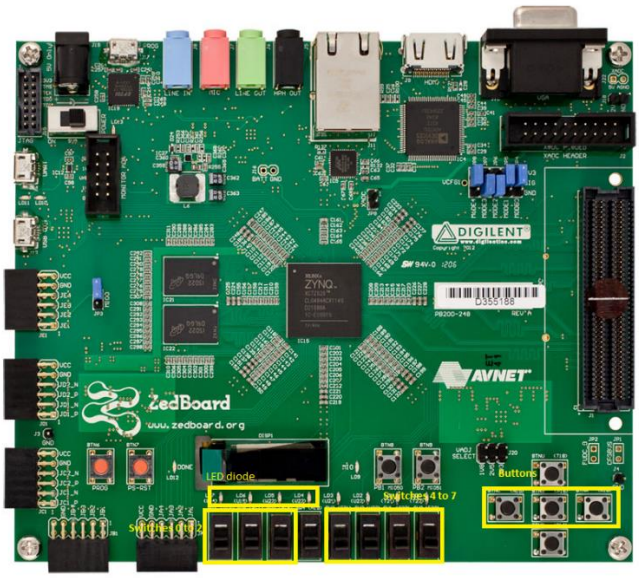
Vsak AXI4-lite blok ima štiri registre. Mi od teh štirih uporabljamo prvega slv_reg0 v katerega zapisujemo podatke iz procesorja in potem naprej iz slv_reg0 na izhod AXI_to_PL izhod.

PL_to_PS:

AXI4-Lite IP-ju sem dodal vhod AXI_from_PL, ki sem ga povezal na signal S_AXI_WDATA. Potem pa sem spremenil naslov (axi_awaddr = ''0000'') za pisanje tako da vedno piše v register slv_reg0 in je pisanje vedno omogočeno (slv_reg_wren <= '1'). Te podatke dobimo iz FPGA dela v procesorski s C funkcijo, ki bere iz registra slv_reg0.

2.4 Kontrolni blok

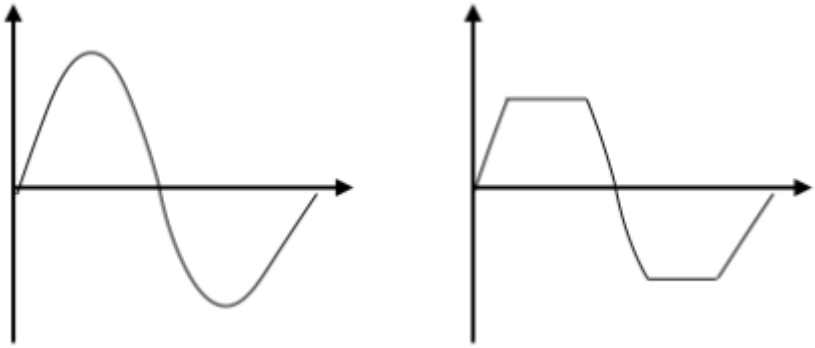
Na sliki 6 so označena stikala, ki so uporabljena za izbiro efektov. Stikala označena na sliki z »Switches 0 to 2« se uporabljajo, da se prižge eden ali več izmed treh efektov, ki so implementirani. Stikala označena z »Switches 4 to 7« se uporabljajo za izbire različnih nastaviteh teh efektov (npr. Močnejši oz. blažji distortion, daljši oz. krajši delay, itd.). Te nastavitve se pa upoštevajo le, če smo izbrali da hočemo izbrati te nastavitve za določen efekt (prve tri LED diode prikazujejo kateremo efektu bomo spremenili nastavitve. Med nastavitvami se premikamo s pomočjo leve in desne tipke na sliki), da se pa te nastavitve dejansko spremenijo pa moramo klikniti sredinsko tipko prikazano na sliki, da zasveti četrta LED dioda. Če se hočemo spet premikati med efekti moramo še enkrat klikniti sredinsko tipko, da se četrta LED dioda ugasne.



Slika 6: Zedboard z označenimi stikali za izbiranje efektov.

2.5 Distortion IP

Je prva komponenta v verigi efektov. Ima štiri različne nastavitve. Od blagega overdrive-a do najmočnejšega. Komponenta deluje tako, da poreže vrh signala in s tem povzroči popačenje signala. Rezanje pri nižji vrednosti povzroči večje popačenje. Na sliki 8 lahko vidimo simulacije VHDL kode, ko vhodni signal »X« preseže vrednost 70000, se na izhodu »Y« pojavi vrednost 70000, če je pa vrednost na vhodu manjša od 70000 pa velja »Y <= X«.



Slika 7: Modulacija signala za pridobitev distorzije.

/distortion_tb/x	0	0	500000	487260	323742	166543	18472	80856	332323	42336	102068	83653	327479	64408
/distortion_tb/y	X	X	70000				18472	70000		42336	70000			64408

Slika 8: Simulacija delovanja distortion IP-ja.

```
if rising_edge(clk_48) then
    if en(0)= '1' then
        if options="1000" then --weak overdrive
            if signed(x(23 downto 0)) >= 70000 then
                y<=std_logic_vector(to_signed(70000,32));
            elsif signed(x(23 downto 0)) <= -70000 then
                y<=std_logic_vector(to_signed(-70000,32));
            else
               y<=x;
            end if;

        elsif options="0100" then --strong overdrive
            if signed(x(23 downto 0)) >= 70000 then
              y<=std_logic_vector(to_signed(90000,32));
            elsif signed(x(23 downto 0)) <= -70000 then
              y<=std_logic_vector(to_signed(-90000,32));
            else
              y<=x;
            end if;

        elsif options="0010" then --overdrive
            if signed(x(23 downto 0)) >= 50000 then
                y  <= std_logic_vector(to_signed(50000,32));
            elsif signed(x(23 downto 0)) <= -50000 then
                y  <= std_logic_vector(to_signed(-50000,32));
            else
                y<=x;
            end if;
```

Slika 9: Primer VHDL kode za distortion IP.

2.6 Tremolo IP

Je druga komponenta v verigi efektov. Deluje tako, da se avdio signal množi z trikotnim signalom, kar spremeni amplitudo avdio signalu. Frekvence counterjev, s pomočjo katerega generiramo trikotni signal, so 48, 95, 190 in 380 Hz (frekvence izhodnega signala so 0.8, 1.6, 3.2, 6.35 Hz).

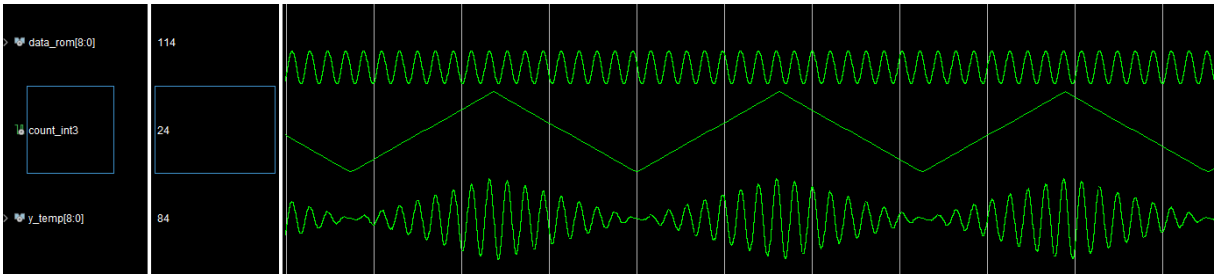
```
process (clk_48)
begin
    if rising_edge(clk_48) then
        if en(1)= '1' then
            if options="1000" then --tremolo frequency -  1.6hz
                temp_vec_64 <= std_logic_vector((signed(x))*count_int);
                y <= "00000000" & std_logic_vector(shift_right(signed(temp_vec_64(23 downto 0)),5));

            elsif options="0100" then --tremolo frequency -  3.2 hz
                temp_vec_64 <= std_logic_vector((signed(x))*count_int2);
                y <= "00000000" & std_logic_vector(shift_right(signed(temp_vec_64(23 downto 0)),5));

            elsif options="0010" then --tremolo frequency - 6.35hz
                temp_vec_64 <= std_logic_vector((signed(x))*count_int3);
                y <= "00000000" & std_logic_vector(shift_right(signed(temp_vec_64(23 downto 0)),5));

            elsif options="0001" then --tremolo frequency - 0.8hz
                temp_vec_64 <= std_logic_vector((signed(x))*count_int4);
                y <= "00000000" & std_logic_vector(shift_right(signed(temp_vec_64(23 downto 0)),5));
            else
                y <= x;
            end if;
        else
            y<=x;
        end if;
    end if;
end process;
```

Slika 10: Primer VHDL kode za Tremolo IP.

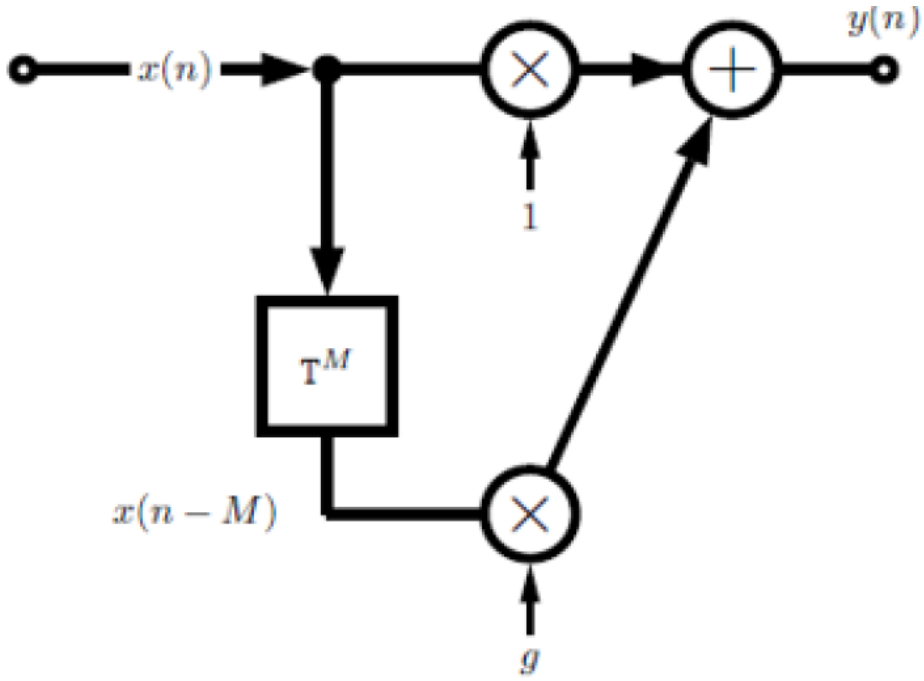


Slika 11: Simulacija delovanja IP komponente za 380 Hz. Vhod je data_rom izhod pa y_temp.

2.7 Delay IP

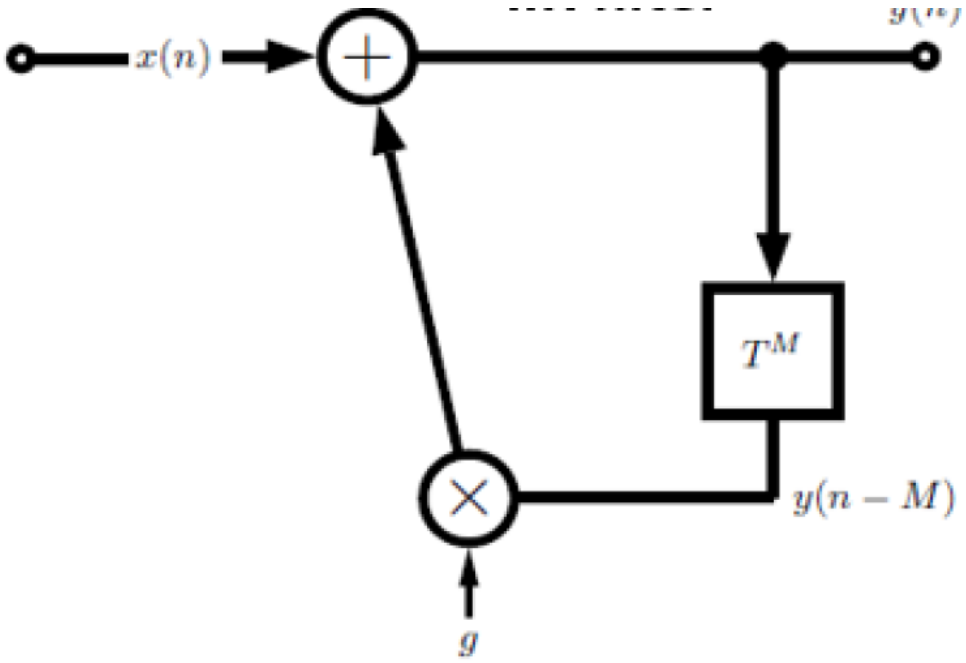
Je tretja komponenta v verigi efektov. Je efekt, ki shranjuje vzorce signala in jih potem zakasnjene prišteje direktnemu signalu. Implementacijo Delay efekta sem implementiral s pomočjo FIR in IIR sita. Za shranjevanje vzorcev je bil uporabljen BRAM. Največja možna zakasnitev je 400 ms.

Uporabil sem FIR filter, ki vhodni signal zadrži za M vzorcev in jih nato atenuira z vrednostjo 0.5 ter jih prišteje vhodnemu signalu x: $y(n) = x(n) + g * x(n - M)$

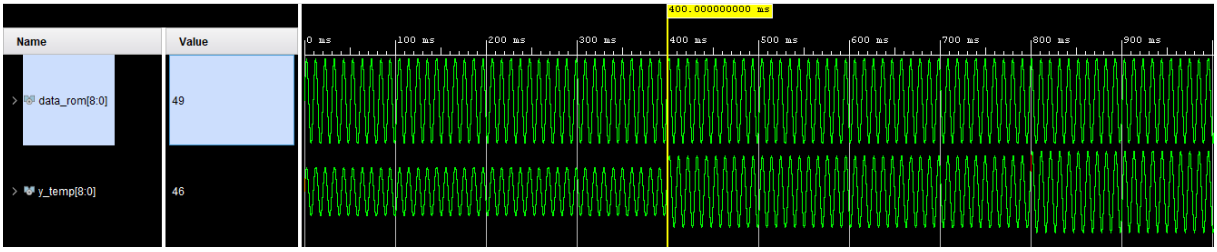


Slika 12: FIR filter.

Pri IIR filtru pa so sešteti signal x(n) in vsi prejšnji atenuirani z faktorjem 0.5. Enačba takega IIR filtra je sledeča: $y(n) = x(n) + g * y(n - M)$



Slika 13: IIR filter.



Slika 14: Simulacija VHDL kode za delay IP. Vidimo, da je zakasnitev enaka 400 ms.

```
if rising_edge(clk_48) then
    if en(2)= '1' then
        if options="1000" then --iir
            max_delay <= T-1;
            y_temp_s <= signed(x) + signed("00000000" & std_logic_vector(shift_right(signed(data_out2(23 downto 0)),1)));
            data_in <= std_logic_vector(y_temp_s);
            y <= std_logic_vector(y_temp_s);

        elsif options="1100" then  --iir, faster delay
            max_delay <= T/2-1;
            y_temp_s <= signed(x) + signed("00000000" & std_logic_vector(shift_right(signed(data_out2(23 downto 0)),1)));
            data_in <= std_logic_vector(y_temp_s);
            y <= std_logic_vector(y_temp_s);

        elsif options="1110" then --iir, very slight reverb
            max_delay <= T/2-1;
            y_temp_s <= signed(x) + signed("00000000" & std_logic_vector(shift_right(signed(data_out2(23 downto 0)),3)));
            data_in <= std_logic_vector(y_temp_s);
            y <= std_logic_vector(y_temp_s);

        elsif options="0100" then --iir, long delay
            max_delay <= count;
            y_temp_s <= signed(x) + signed("00000000" & std_logic_vector(shift_right(signed(data_out2(23 downto 0)),1)));
            data_in <= std_logic_vector(y_temp_s);
            y <= std_logic_vector(y_temp_s);

        elsif options="0010" then --fir sigle tap, long delay
            max_delay <= count;
            y_temp_s <= signed(x) + signed("00000000" & std_logic_vector(shift_right(signed(data_out2(23 downto 0)),1)));
            data_in <= x;
            y <= std_logic_vector(y_temp_s);
```

Slika 15: Primer VHDL kode za FIR in IIR sita.

```
architecture Behavioral of bram is

type ram_type is array (0 to T-1) of std_logic_vector (31 downto 0);
signal RAM : ram_type := (T-1 downto 0 => x"00000000"); --define and initialize RAM

begin

proc: process (clk)
begin
    if rising_edge(clk) then
        if we = '1' then
            if to_integer(addr1) < T then
                RAM(to_integer(addr1)) <= data_in;
            end if;
        end if;
        if to_integer(addr1) < T then
            data_out1 <= RAM(to_integer(addr1));
        end if;
        if to_integer(addr2) < T then
            data_out2 <= RAM(to_integer(addr2));
        end if;
    end if;
end process;
end Behavioral;
```

Slika 16: Primer VHDL kode za BRAM.

3 PROCESOR

C program, ki se žene na procesorju preko I2C bus-a konfigurira audio codec in potem cel čas skrbi za pretok avdio signalov iz audio codec-a do FPGA-ja in nazaj.

```c
#include "headers.h"

 *                                    main()
int main(void)
{
    //Configure the IIC data structure
    IicConfig(XPAR_XIICPS_0_DEVICE_ID);
    //Configure the Audio Codec's PLL
    AudioPllConfig();

    //Configure all audio registers
    //AudioConfigureJacks();
    LineinLineoutConfig();

    //stream audio
    audio_stream();

    return 1;
}
```

```c
void audio_stream(){
    u32 in_left;
    u32 output;

    while (1){
        // Read audio input from codec
        in_left = Xil_In32(I2S_DATA_RX_L_REG); //left only, mono
        //in_right = Xil_In32(I2S_DATA_RX_R_REG);

        Xil_Out32(XPAR_PS_TO_PL_0_S00_AXI_BASEADDR, in_left);
        output=Xil_In32(XPAR_PL_TO_PS_0_S00_AXI_BASEADDR);

        Xil_Out32(I2S_DATA_TX_L_REG, output);
    }
}
```

Slika 17: Primer C kode.