УНИВЕРЗИТЕТ
У НОВОМ САДУ

ФАКУЛТЕТ
ТЕХНИЧКИХ НАУКА

Трг Доситеја Обрадовића 6, 21000 Нови Сад, Југославија
Деканат: 021 350-413; 021 450-810; Централа: 021 350-122
Рачуноводство: 021 58-220; Студентска служба: 021 350-763
Телефакс: 021 58-133; e-mail: ftndean@uns.ns.ac.yu

Projekat iz predmeta Projektovanje složenih digitalnih sistema

Implementacija Phaser efekta u FPGA tehnologiji

Student: Popov Nikola ee42/2016

Mentor:Đorđe Mišeljić

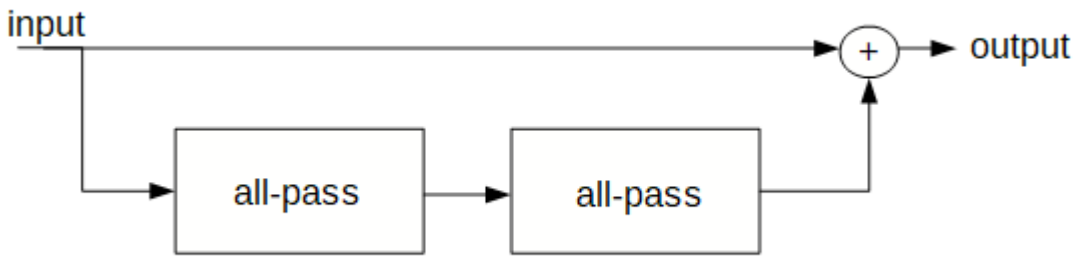
Sadržaj

Uvod

Ovo je dokumentacija implementacije phaser zvučnog efekta u FPGA tehnologiji. Phaser je efekat koji se ponaša kao "šetajući" notch filter, tj. periodično blokira frekvencije iz datog opsega. Phaser koji se implementira je jednokanalni (mono) stalnih depth, speed i level atributa. Phaser jedino radi sa zvučnim signalima koji su odabirani frekvencijom 44100Hz

Matlab simulacija

U matlabu je generisan phaser uz pomoć dva allpass filtera, blok dijagram je na slici 1



Slika 1: Blok dijagram phaser-a

Dva all-pass filtera istih koeficijenata će promeniti fazu za 180 stepeni na frekvenciji određenoj koeficijentima filtara. Kada se izvorni signal i signal propušten kroz all-pass granu saberu dobijamo destruktivnu interferenciju na frekvencijama gde se faza razlikuje za 180, u spektralnoj karakteristici ovo izgleda kao notch filter. Da bi dobili šetajući filter modulišemo koeficijente filtera uz testerasti signal određene periode. U ovoj implementaciji je odabran modulator takav da "notch" filter isključuje frekvencije od 200Hz do 2000Hz periodom od 1 Hz. Dodatno pošto nema slabljenja u all-pass grani imamo najjaču snagu efekta. Ove osobine efekta su ekperimetnalno u simulaciji dobijene tako da odgovaraju zahtevima uha (ne mogu biti formalizovane).
Kada je nađen zvuk koji odgovara, uz pomoć simulacije, koeficijente filtera čuvamo u excel tabeli i njih ćemo kasnije ubaciti u statičnu memoriju od FPGA da bi izbegli zahtevne proračune koeficijenata filtera preko tan funkcije, množenja i deljenja.

rom_coef

C algoritam

Kod iz matlaba pretvoren u C kod. Za osnovu algoritma je uzeta diskretna prenosna funkcija all-pass filtera: y[n] = a*x[n] + x[n-1] - a * y[n-1]. Preko nje je razvijen ceo algoritam kao na slici 2. Da bi ustedeli na memoriji koristimo samo prvih 22050 koeficijenata. Zbog toga imamo brojač **k** koji čuva vrednost trenutnog koeficijenta, a promenjiva up govori u kojem smeru treba da idu koeficijenti. 22050 koeficijenata odgovara pola sekunde, a kako nam je perioda promene koeficijenata 1s pokriveni su svi koeficijenti. Promenjiva valid će biti korisna kasnije da označi ulaz koji je valjan. Kada se uklone while petlje algoritam izgleda identično samo što postoje dve goto naredbe jedna bezuslovna goto idle druga if on = 1 goto load else goto idle

Phaser Algorithm

```
while(1)
{
        while(on == 1)
        {
                input = input_in; /*load next input value*/
                a = mod[k]; /*load next coefficient*/
                valid = 0; /*output value not stable anymore*/

                /*calculate next coefficient value*/
                if(up == 1)
                {
                        if(k == 22048)
                        {
                                up = 0; /*change sawtooth direction*/
                        }
                        else
                        {
                                up = 1; /*continue still more coeff*/
                        }
                        k++;
                }
                else
                {
                        if(k == 1)
                        {
                                up = 1; /*change sawtooth direction*/
                        }
                        else
                        {
                                up = 0; /*continue still more coeff*/
                        }
                        k--;
                }

                /*Main calculation part, two series all-pass filters*/

                MidVal = a*input + PrevInVal - a*PrevMidVal;/*First all-pass*/
                output = a*MidVal + PrevMidVal - a*PrevOutVal;/*Second all-pass*/

                /*Add input and all-pass branch*/
                out = output + input;
                valid = 1;
        }
}
```

Slika 2: Phaser algoritam

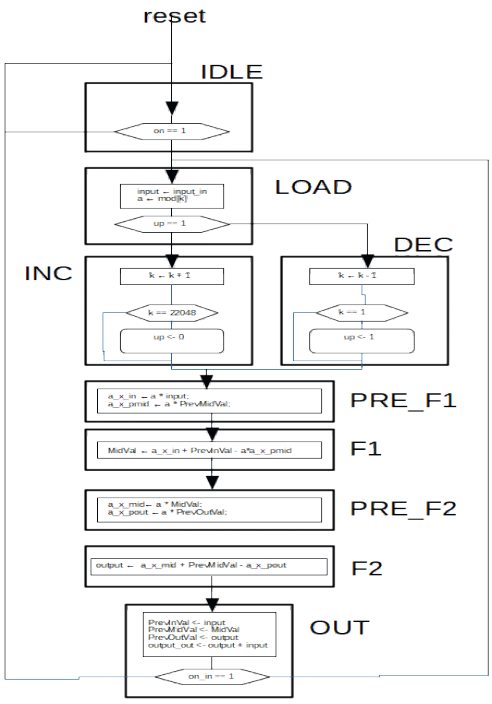
Interfejs dizajna

Ulaz u sistem je signal input_in od 16 bit-a, izlaz je output_out od 16 bit-a. Ovo su glavni ulaz i izlaz za podatke pored njih imamo pomoćne signale on_in koji dozvoljava rad i out_valid koji govori da su izlazni podaci validni. Takođe postoje standardni clk i reset signali. Rad sa modulom kreće tako što se dovede neki signal na input_in i onda se on_in setuje i dozvoljava rad modulu. On_in resetujemo posle 2 rastuće ivice kloka da ne bi stalno nastavljao sistem. Kada je out_valid signal setovan možemo preuzeti podatke i odatle se nastavlja ciklus.

Za kodovanje podataka je izabrano 16bit-a u notaciji komplementa dvojke tako da MSB označava znak podaka. Osim toga decimalni zarez je uzet da bude posle sledećeg bita tako da imamo jedan bit za znak jedan bit za broj (u ovom slučaju samo jedinicu) i 14bit-a za cifre iza decimalnog zareza. Ovakvo mapiranje u decimalnim brojevima označava sve brojeve (-2,2) sa 4 decimalne cifre. Opseg je odabran zato što se koeficijenti i podaci kreću u opsegu (-1,1), a i 16bit-a je format u kojem se muzika čuva na cd-u pa je očekivano da je kvalitet zvuka zadovoljavajuć. Za koeficijente jedino važi opseg (-1,0), a ako je potrebno treba skalirati muzičke podatke.

ASMD

Na slici 3 je ASMD dijagram sistema koji se implementira

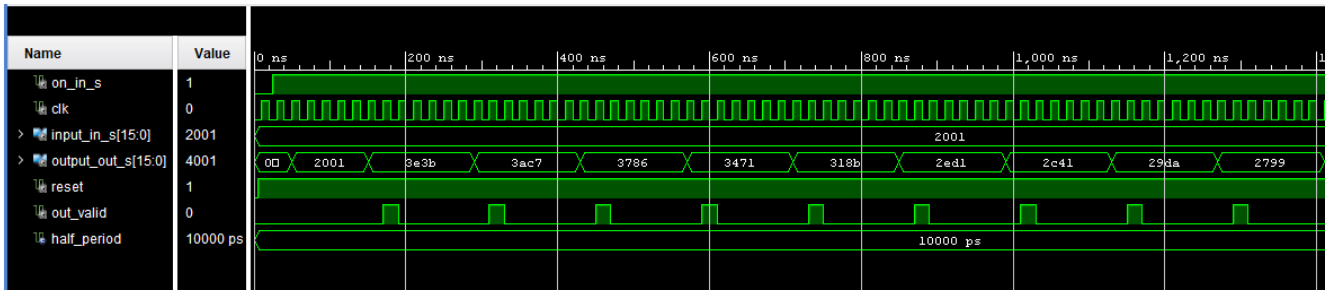


Slika 3: ASMD

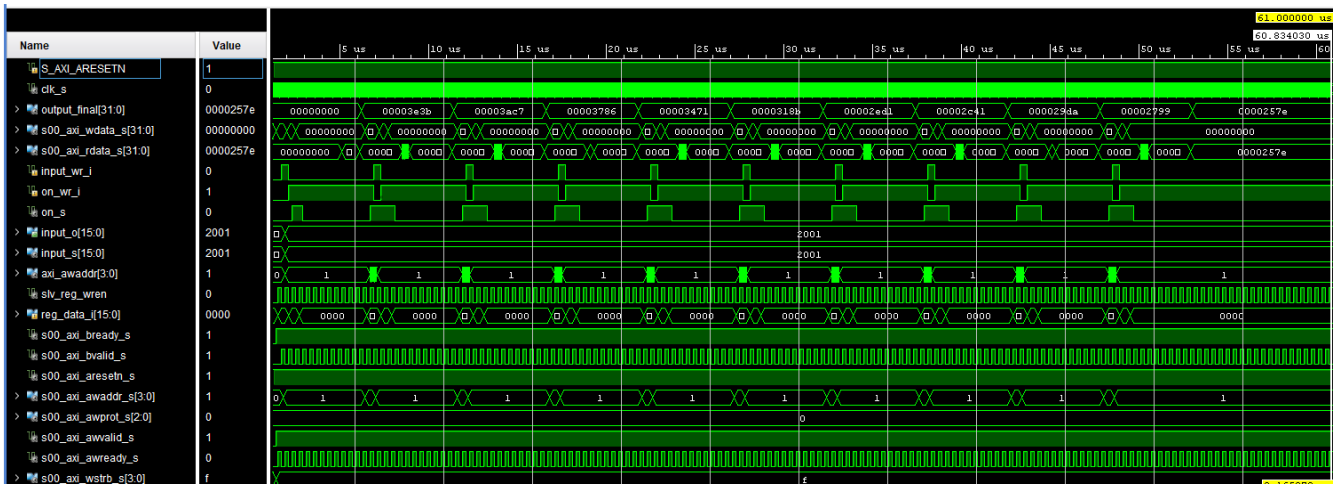
Rezultati simulacije

Sledeće dve slike su odziv sistema na 0.500 konstantan ulaz iz Vivado simulacije. Uzeto je prvih 10 odbiraka. Kada se iz Matlab simulacije vrednosti pretvore u prikaz koji se koristi u sistemu dobijamo:

1	2	3	4	5	6	7	8	9	10
0.5000	0.9723	0.9184	0.8675	0.8196	0.7744	0.7319	0.6920	0.6545	0.6194
2001	3E3B	3AC8	3786	3475	3190	2ED8	2C4A	29E4	27A5



Slika 5: Phaser modul



Slika 6: Phaser upakovan u IP jezgro

Može se videti da počne da se pojavljuje razlika od 0.005 i to možemo prepisati zaokruživanju kod množenja i kod pravljenja koeficijenata. Da bi povećali preciznost trebalo bi povećati broj bita sa kojim se radi. Kako vidimo da je odziv na odskočnu funkciju poprilično sličan onom iz simulacije možemo reći da imamo sistem koji će raditi zadato ali uz šum.

Rezultati sinteze i implementacije

Nakon sinteze smo dobili zauzeće LUT, BRAM i DSP na Zibo Z7-10

LUT - 156

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	156	0	17600	0.89
LUT as Logic	156	0	17600	0.89
LUT as Memory	0	0	6000	0.00
Slice Registers	223	0	35200	0.63
Register as Flip Flop	223	0	35200	0.63
Register as Latch	0	0	35200	0.00
F7 Muxes	0	0	8800	0.00
F8 Muxes	0	0	4400	0.00

BRAM - 16

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	16	0	60	26.67
RAMB36/FIFO*	16	0	60	26.67
RAMB36E1 only	16			
RAMB18	0	0	120	0.00

DSP - 4

Site Type	Used	Fixed	Available	Util%
DSPs	4	0	80	5.00
DSP48E1 only	4			

Maksimalna frekvencija rada nakon sinteze je 202,02MHz

```
| Design Timing Summary
| ---------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints
    -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------
      0.023        0.000                      0                  809        0.133        0.000                      0                  809


All user specified timing constraints are met.


| Clock Summary
| -------------

Clock        Waveform(ns)         Period(ns)      Frequency(MHz)
-----        ------------         ----------      --------------
s00_axi_aclk {0.000 2.475}        4.950           202.020
```

Maksimalni throughput - 27 clocks/sample -