UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET
KATEDRA ZA ELEKTRONIKU

Implementacija direktne forme FIR filtra u VHDL-u

Student:
Ivanović Aleksandar
broj indeksa: 9923

Niš, 16.mart 2009.

Sadražaj:

Uvod ……. **3**

1.1 Digitalno filtriranje…….. **4**
1.2 Konvolucija i filtriranje signala…… **4**
1.3 Digitalni filtri …….. **5**
1.4 Diferencne jednačine…… **5**
1.5 Struktura digitalnih sistema za filtriranje……..…… **6**
1.6 Direktna struktura IIR filtra……..…… **7**
1.7 Direktna struktura FIR filtra……..…… **8**
1.8 Struktura bikvadratnog filtra…… **8**

2.1 Realizacija direktne strukture FIR filtra u VHDL-u ……. **9**
2.2 **VHDL** opis osnovnih komponenata……. **14**
2.3 Registar konstanti…… **14**
2.4 X registar……..…… **15**
2.5 Izlazni registar……..…… **16**
2.6 Množac…… **17**
2.7 Sabirać……. **17**
2.8 Dekoder……... **18**
2.9 Opis entiteta FIR filtra …… **19**
2.10 Opis arhitekture FIR filtra ……. **20**

3.1 VHDL kod test bench-a FIR filtra i prikaz rezultata simulacije**..23**

4.1 Rezultati sinteze i implementacije……**28**

5.1 Literatura…… **33**

Uvod

Pod obradom signala podrazumeva se skup operacija koje se izvršavaju nad signalom. Jedna odnajvažnijih operacija u obradi signala je linearno filtriranje signala, kako zbog toga što nalazi veliku primenu u algoritmima za obradu signala, tako i zbog toga što ima relativno jednostavnu matematičku osnovu, pogodnu za implementaciju. Filtriranje se koristi u obradi audio i video signala, za razdvajanje signala sa različitim frekvencama, kombinovanje više signala u jedan signal, za otklanjanje šuma iz signala, izdvajanje signala u određenom frekventnom opsegu, predviđanje promena signala, podešavanje opsega kanala, otklanjanje eha, itd. Svi uređaji za snimanje i reprodukciju zvuka i slike, bez izuzetka, sadrže filtre.U prvom poglavlju date su osnove diskretnog (digitalnog) filtriranja. Prikazana je operacija konvolucije signala. Matematičke osnove digitalnog filtriranja su date i objašnjene korišćenjem diskretnih diferencnih jednačina. Posebna pažnja posvećena je predstavljanju strukture osnovnih klasa digitalnih filtra.

U ostalim poglavljima naručita pažnja posvećena FIR filtru, odnosno direktnoj strukturu FIR filtra. Njegovom opisu u VHDL-u uz upotrebu ***generic*** naredbe koja pruza u ovom slučaju mogućnost promene dimenzija filtra tj. broja sekcija (celija) filtra i širine podataka, na taj način se se frši izmena karakteristike filtra.
Prikazan je VHDL kod test banch-a i rezultat simulacije, rezultat sinteza i implementacije.

1.1 Digitalno filtriranje

U raznim primenama kao što su radiokomunikacije, posebno softverski radio, zatim digitalizacija pretplatničke petlje, digitalizacija i obrada govora, obrada audio signala, obrada slike i mnogim drugim, postavljaju se zahtevi da se realizuje digitalni filtar veoma dobrih karakteristika sa minimalnim zahtevima u pogledu hardvera.

Digitalni filtar se može implementirati na više načina: korišćenjem procesora signala opšte namene, programabilnih logičkih kola (FPGA), *custom* ili *semicustom* integrisanih kola (ASIC), izradom namenskih tipova (VLSI), ili hardverski korišćenjem digitalnih kola i mikrokontrolera. Svaka od ovih implementacija je na svoj način specifična, ali se uvek mora imati na umu da se i koeficijenti i signali implementiraju sa konačnim brojem bita čime se u manjoj ili većoj meri degradira teorijski izračunata karakteristika.

1.2 Konvolucija i filtriranje signala

Pod obradom signala podrazumeva se preslikavanje jednog ili više ulaznih signala u izlazni signal. Preslikavanje signala može biti opisano operatorom Ψ, koji transformiše signal $x(n)$ u signal $y(n)$ na sledeći način:

$$y(n) = \Psi\{x(n)\}. \qquad (1.1)$$

Postoje dva tipa preslikavanja Ψ. Prvi tip je *vremenski nezavisno* preslikavanje, kod koga vrednost signala $y(n)$ zavisi jedino od trenutne vrednosti signala $x(n)$. Drugi tip preslikavanja je *vremenski zavisno* preslikavanje. Vremenski zavisno preslikavanje je tip preslikavanja kod koga trenutna vrednost signala $y(n)$ zavisi, kako od trenutne vrednosti signala $x(n)$, tako i od vrednosti koje je signal $y(n)$ imao u prošlosti. Vremenski zavisno preslikavanje ima veliku primenu u digitalnoj obradi signala.

Kod vremenski zavisnih sistema može se javiti nestabilnost. Da bi vremenski zavistan sistem bio stabilan treba da zadovolji dva uslova: uslov linearnosti i uslov neosetljivost na fazu, tj. na pomeraj signala $x(n)$ u vremenu.

Neka su $x1(n)$ i $x2(n)$ dva nezavisna ulazna signala i neka važi $y1(n)=\Psi\{x1(n)\}$ i $y2(n)=\Psi\{x2(n)\}$. Ukoliko se operator Ψ primeni na linearnu kombinaciju signala $c1x1(n)+c2x2(n)$, gde su $c1$ i $c2$ nezavisni koeficijenti, i važi princip superpozicije:

$$y(n)=\Psi\{c1x_1(n)+c2x_2(n)\}=c1\Psi\{x_1(n)\}+c2\Psi\{x_2(n)\}=y_1(n)+y_2(n), \quad (1.2)$$

tada je operator Ψ linearan.

Neosetljivost operatora Ψ na fazu ulaznog signala se matematički definiše na sledeći način:

$$y(n-n_0)=\Psi\{x(n-n_0)\}. \tag{1.3}$$

Drugim rečima, ukoliko signal $x(n)$ kasni $n0$ vremenskih jedinica, tada se signal $y(n)$ razlikuje u odnosu na signal $y(n)$, koji se dobija kada se na ulaz dovede signal bez kašnjenja, jedino u faznom pomeraju.

Konvolucija je linearna operacija, neosetljiva na fazu signala, koja preslikava dva signala, $x(n)$ i $h(n)$, u jedan signal:

$$y(n)=\Psi\{h(n),x(n)\}. \tag{1.4}$$

Konvolucija se definiše na sledeći način:

$$y(n)=\sum_{k=-\infty}^{\infty}h(k)x(n-k). \tag{1.5}$$

Signal $x(n)$ se naziva ulazni signal, a signal $h(n)$ filtrirajući signal. Filtrirajući signal je u većini praktičnih slučajeva periodičan, što u diskternom domenu predstavlja konačan niz vrednosti, koje se nazivaju koeficijenti filtra.
Suma u izrazu (1.5) je beskonačna, tako da je praktična implementacija ovog izraza nemoguća.
Međutim, u većini praktičnih slučajeva se koristi aproksimacija izraza (1.5), tj. beskonačna suma u izrazu (1.5) se svodi na konačnu.

1.3 Digitalni filtri

Redukcijom sume u izrazu (1.5) na konačni opseg vrednosti, dolazi se do aproksimacije operacije konvolucije koju je moguće implementirati. Aproksimaciju operacije konvolucije implementiraju digitalni filtri. Digitalni filtri se opisuju diferencnim jednačinama. Ove jednačine predstavljaju aproksimaciju konvolucije i pogodne su za manipulacuiju u cilju nalaženja različitih modela filtra.

1.4 Diferencne jednačine

Diferencna jednačina koja daje vezu između izlaznog signala $y(n)$ i ulaznog signala $x(n)$ je:

$$\sum_{k=0}^{N}a_k y(n-k)=\sum_{k=0}^{M}b_k x(n-k), \tag{1.6}$$

gde su {*ak*} i {*bk*} koeficijenti. Moguće je pokazati da jednakost (1.6) zadovoljava uslov linearnosti i uslov neosetljivosti na pomeraj faze.
U cilju jasnijeg predstavljanja, jednakost (1.6) može biti napisana u sledećem obliku:

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k). \qquad (1.7)$$

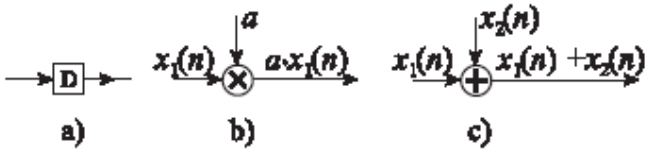
Iz izraza (2.7) sledi da je izlazni signal *y* u trenutku *n* linearna kombinacija *N* prethodnih vrednosti izlaznog signala i *M*+1 vrednosti ulaznog signala.
Veoma važan oblik jednačine (1.7) je specijalni slučaj koji se javlja za *N*=0. U ovom slučaju na izlazni signal utiče jedino ulazni signal, a ne i istorija izlaznog signala. Izraz (1.7) za slučaj *N*=0 ima sledeći oblik:

$$y(n) = \sum_{k=0}^{M} b_k x(n-k). \qquad (1.8)$$

Diferencna jednačina (1.8) predstavlja matematički opis digitalnog filtra sa konačnim impulsnim odzivom (*Finite Impulse Response* – FIR). Filtri kod kojih je *N*>0 se nazivaju filtri sa beskonačnim impulsnim odzivom (*Infinite Impulse Response*– IIR).
Diferencne jednačine (1.7) i (1.8) opisuju ponašanja filtra, međutim, ne postoji jednoznačno preslikavanje ovih jednačina u hardver, jer ne sadrže informaciju o redosledu izvršavanja operacija.
Kako bi se u predstavljanju filtra što više približili hardverskoj implementaciji, uobičajeno je filtre predstavljati strukturom.

1.5 Struktura digitalnih sistema za filtriranje

Struktura filtra je grafička reprezentacija koja daje tačan prikaz redosleda izvršavanja operacija.
Linearni filtri, neosetljivi na fazu signala, sastoje se od elementarnih operacija *kašnjenja, sabiranja*i *množenja*. Grafičke reprezentacije nabrojanih operacija prikazane su na sl. 1.1.



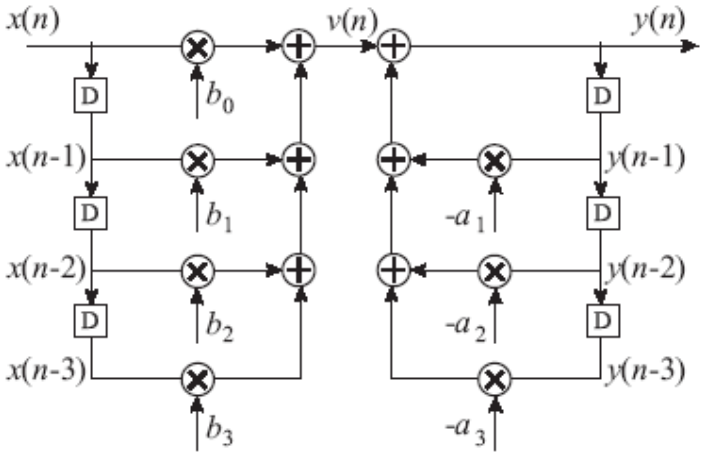
Slika 2.1. Grafičke reprezentacije elementarnih operacija filtra. a) Kašnjenje, b) množenje konstantom, c) sabiranje signala

U cilju ilustracije grafičkog predstavljanja strukture filtra, u daljem tekstu su date strukture IIR i FIR klase filtra, dobijene direktnim preslikavanjem jednačina u strukturu.

1.6 Direktna struktura IIR filtra

Na sl. (2.2) je prikazana direktna struktura IIR filtra, dobijena iz izraza (1.7). Izlazni signal IIR filtra, na osnovu izraza (2.7), zavisi od dve sume. U okviru jedne od ovih suma se nalaze vrednosti ulaznog signala u trenucima n,n-1,...,n-M, dok u drugoj sumi figurišu vrednosti izlaznog signala u trenucima n-1,n-2,...,n-N, što predstavlja povratnu spregu IIR filtra.

IIR filter se može implementirati tako što se na jednoj strani formira težinska suma vrednosti ulaznog signala u trenucima n, n-1, n-2,...,n-M, a na drugoj težinska suma vrednosti izlaznog signala u trenucima n-1, n-2,...,n-N. Na sl. 1.2 je prikazana struktura IIR filtra za N=3 i M=3.



Slika 1.2 Direktna struktura IIR filtra

Signal koji se filtrira se dovodi u lanac kašnjenja, koji ima ulogu da ostatku kola prosledi vrednosti ulaznog signala $x(n-1)$, $x(n-2)$ i $x(n-3)$. Vrednost ulaznog signala x u trenutku n, $x(n)$, i vrednosti koje je ulazni signala imao u trenucima n-1, n-2 i n-3, a koje su sačuvane u lancu kašnjenja, se množe koeficijentima $b0$, $b1$, $b2$ i $b3$. Nakon množenja, dobijeni parcijalni proizvodi se sabiraju, što formira sumu proizvoda ulaznog signala i koeficijenata izraza (1.7).

Vrednost izlaznog signala se, nakon određivanja vrednosti u trenutku n, smešta u lanac kašnjenja izlaznog signala na način prikazan na sl.(1.2), kako bi se vrednost sačuvala u narednih N taktnih intervala.

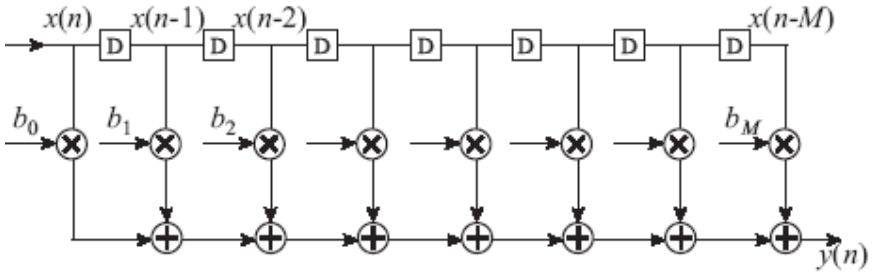
Na isti način na koji se formira suma proizvoda ulaznog signala i koeficijenata, formira se i suma parcijalnih proizvoda izlaznog signala i odgovarajućih koeficijenata. Ove dve sume se sabiraju, čime se dobija vrednost izlaznog signala $y(n)$ slika(1.2), u skladu sa izrazom (1.7).

IIR filter, predstavljen strukturom na slici (1.2), moguće je implementirati, jer je način

predstavljanja takav da sadrži sve potrebne informacije neophodne za implementaciju: redosled izvršavanja operacija, kašnjenja između operacija i veze.

1.7 Direktna struktura FIR filtra

Na slici (1.3) je prikazana direktna struktura FIR filtra, gde reč "direktna" opisuje da je struktura formirana direktno na osnovu jednačine (1.8), bez dodatnih transformacija izraza. FIR filter je, na osnovu izraza (1.8), digitalni sistem bez povratne sprege. Strukturu filtra sa slici(1.3) čini lanac kašnjenja, množači i sabirači.



Slika1.3. Direktna struktura FIR filtra

Signal koji se filtrira, $x(n)$, se dovodi na ulaz lanca kašnjenja. Vrednost izlaznog signala y u trenutku n se dobija množenjem sadržaja lanca kašnjenja, $x(n-k)$, k=0,1,2,...,M, odgovarajućim koeficijentima, u skladu sa izrazom (1.8). Nakon množenja parcijalni rezultati se sabiraju, čime se na izlazim linijama dobija vrednost izlaznog signala $y(n)$.

FIR filter, čija je struktura prikazana na slici (1.3), je pogodan za implementaciju zbog svoje regularne strukture. Regularna struktura filtra se ogleda u tome da se struktura može implementirati nizom funkcionalnih jedinica (FJ) koje rade sinhrono (svaka FJ se sastoji od jednog množača i jednog sabirača, slika (1.3), ima regularne veze i eksploatiše protočnost .

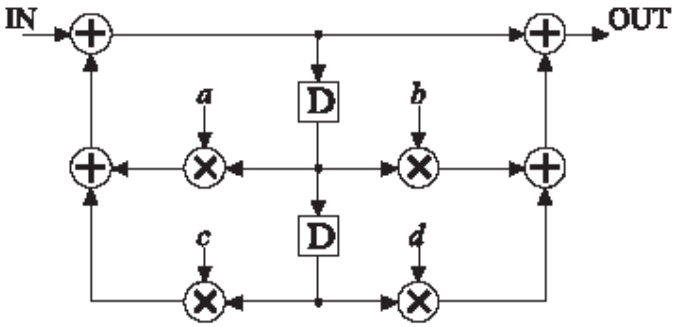
1.8 Struktura bikvadratnog filtra

Za razliku od FIR filtra, IIR filtri imaju bolje prenosne karakteristike. Međutim, kako izlazni signal zavisi, ne samo od ulaznog signala, već i od istorije izlaznog signala (povratna sprega), IIR filtri mogu relativno lako postati nestabilni. Kako bi se izbegla mogućnost da IIR filter dođe u nestabilno stanje, projektuju se filtri *malog reda*, odnosno filtri sa malim brojem koeficijenata. IIR filtri *višeg reda* se projektuju kao kaskadna veza nekoliko filtra nižeg reda .

Na slici (1.4) je prikazan IIR filter drugog reda, koji se u literaturi sreće kao *bikvadratni filter*. Izraz "bikvadratni" je uobičajeno ime za digitalne filtre čija prenosna funkcija ima dva pola, tj. dve nule. Prenosna funkcija filtra sa slici(1.4) je:

$$G(z) = g\frac{1 + az^{-1} + cz^{-2}}{1 + bz^{-1} + dz^{-2}}.$$

Bikvadratni filter sa slika (1.4) se zbog svoje jednostavne strukture koristi u literaturi u cilju ilustracije tehnike savijanja.



Slika 1.4. Struktura bikvadratnog filtra

2.1 Realizacija direktne strukture FIR filtra u VHDL-u

Digitalni filter uzima povorku digitalnog signala na ulazu i generiše na izlazu ulazni signal određenih karakteristika. Slika (2.1) prikazuje blok diagram popularnog digitalnog filtra znanog kao FIR filter. X i Y si N-bitne širine svaki, u našem slučaju su to svaki 12 bita širine.

Slika 2.1. blok diagram FIR filtra

Primer: razmotrimo sledeći primer povorke digitalnog signala temperaturnih vrednosti koje dovodimo na X ulaz iz motora automobila, tj. iz senzora koji se nalazi u motoru automobila svake sekunde: 180,181,240,180,181. Temperatura od 240 stepeni najverovatnije nije ispravno merenje, temperatura motora automobila ne može da

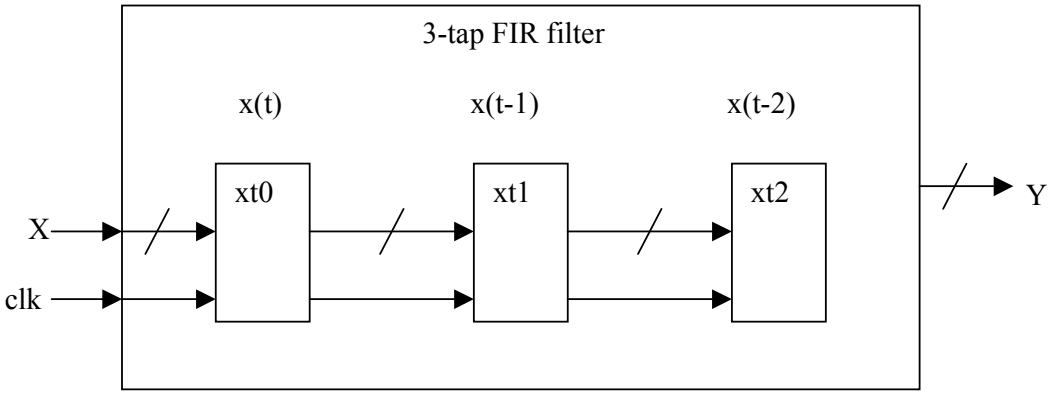
poraste 60 stepeni za jednu sekundu. Digitalni filter uklanja takve vrednosti "šum" iz ulazne povorke signala generišući na izlazu povorku: 180,180,*181*,180,181…
Za FIR filter uobičajna skraćenica sastavljena od slova "F" "I" "R" je skraćeno od "Finite Impulse Response"- (konacan impulsni odziv), je generalno najpopularniji disajn digitalnog filtra može biti korišćen u više varijanti radi postizanja ciljeva.

Slika (2.1) prikazuje blok diagram FIR filtra. Osnovna ideja FIR filtra je vrlo jednostavna: prikazati izlaz kao rezultat koji se dobija višestrukim množenjem ulaznih vrednosti sa konstantama i njihovim sabiranjem, trenutna vrednost ulaznog signala se množi sa odgovarajućom konstantom i taj rezultat se sabira sa proizvodom predhdne vrednosti ulaznog signala i odgovarajuće konstante, i taj rezultat se sabira sa proizvodom ranijom vrednošću ulaznog signala sa konstantom itd… Na taj način se na izlazu filtra dobija prosečna vrednost ulaznog signala. Sve ovo je izraženo određenom matematićkom relacijom:

$$Y(t) = c0 \times x(t) + c1 \times x(t-1) + c2 \times x(t-2).$$

Za FIR filter sa gore navedenom relacijom poznat je kao 3-sekcijski filter. Stvarni filtri imaju više od deset sekcija, mi ovde razmatramo filter sa tri sekcije zbog ilustracije. Dizajn filtra omogućava filtriranje prostim izborom konstanti filtra.

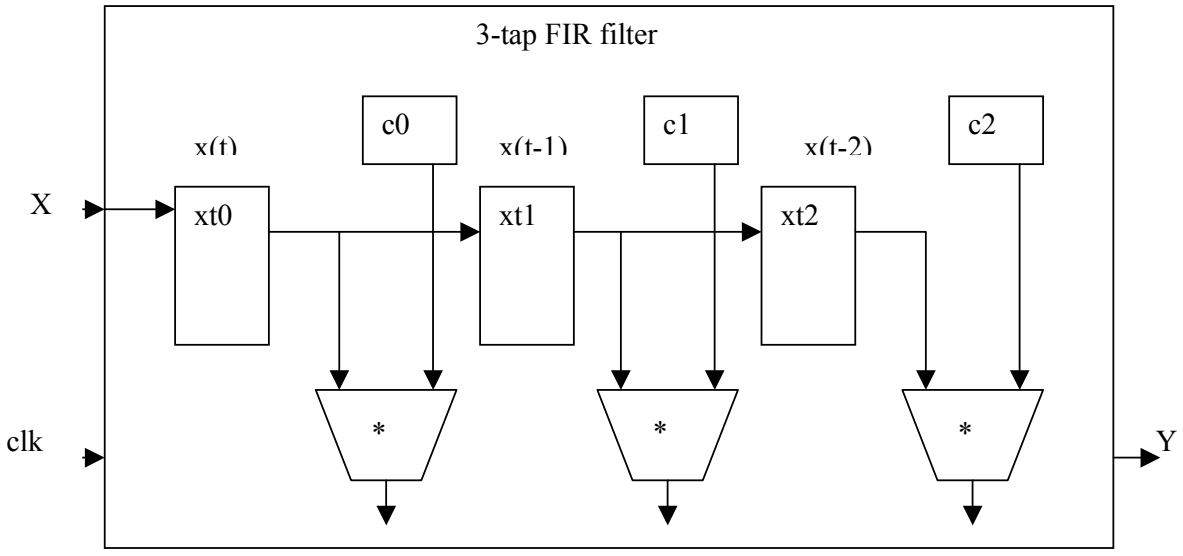
Mi ćemo dizajnirati kolo implementacijom FIR filtra. Prvi korak u RTL dizajniranju je dizajniranje puta podataka, potreban nam je registar za svaku promenjivu: x(t), x(t-1), x(t-2). Na svaku promenu klok signala imamo pomeranje x(t-1) u x(t-2) , x(t) u x(t-1) i prihvatanje trenutne vrednost ulaznog signala. Registri za prihvatanje signala prikazani su na saledećoj slici (2.2).



Slika 2.2. početak gradnje, registri x(t), x(t-1), x(t-2)

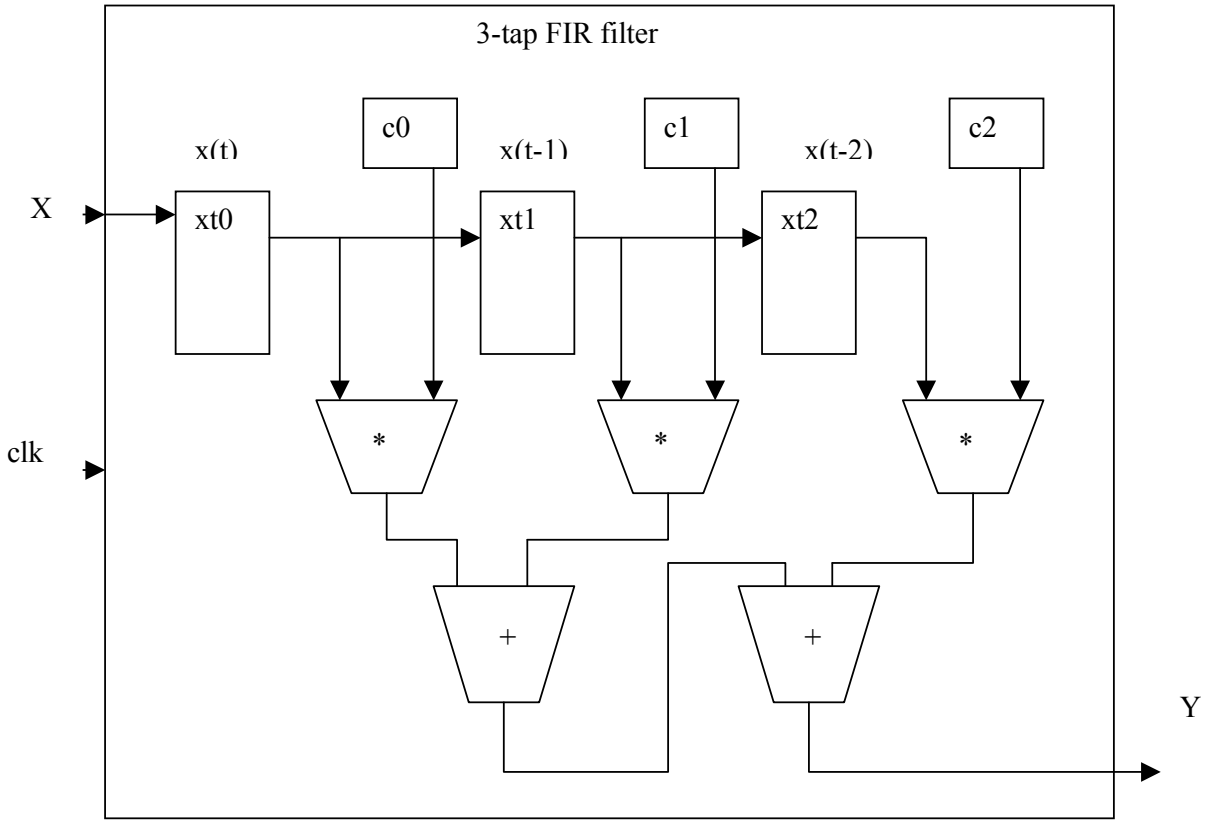
Uputstvo za pomeranje signala nadesno u svakom klok ciklusu izgleda ovako, registar xt0 drži sadašnju vrednost ulaznog signala, xt1 drzi predhodnu vrednost ulaznog signala, i xt2 drži vrednost ulaznog signala koji je bio u xt1 registru.

Sada nam treba još po jedan registar za svaku sekciju za prihvatanje konstanti c0,c1,c2. Treba nam za svaku sekciju jos i po jedan množač koji množi konstantu sa sa promeljivom. Izgled strukture prikazan je na slici (2.3):



Slika 2.3 spajanje množača i registra konstanti

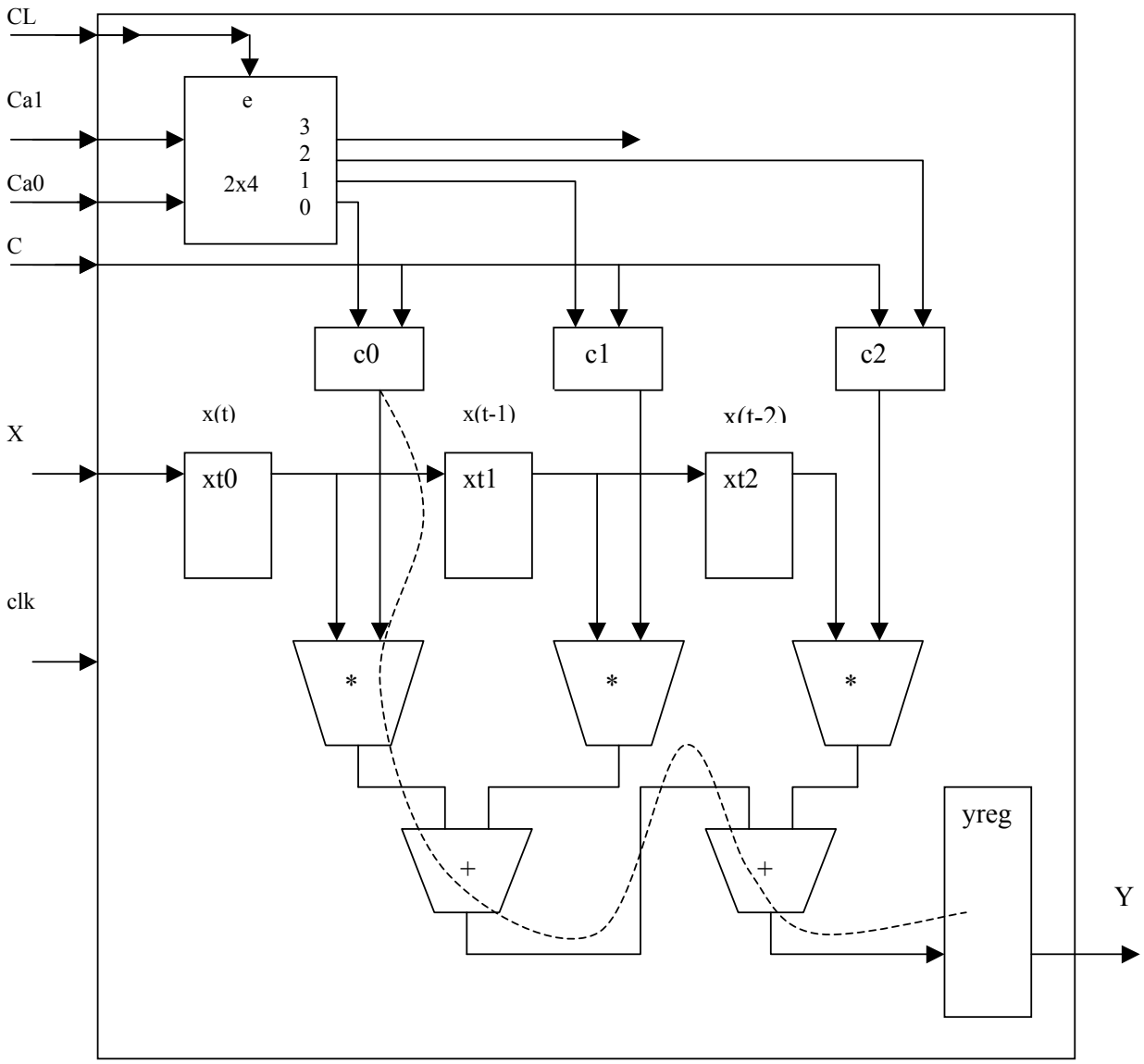
Izlazni signal Y je suma svih etapa. Mi prema tome možemo dodati odgovarajuće sabirače i izračunatu sumu povezati na izlaz Y, to je prikazano na slici(2.4).



Slika 2.3 dodavanje sabirača i povezivanje sume Y na izlaz

Mi imamo kompletno srce FIR filtra i putanju podataka. Nama ostaje metod kako da upisujemo konstante u registre c0,c1 i c2. Hajde da kreiramo još jedan ulaz i liniju

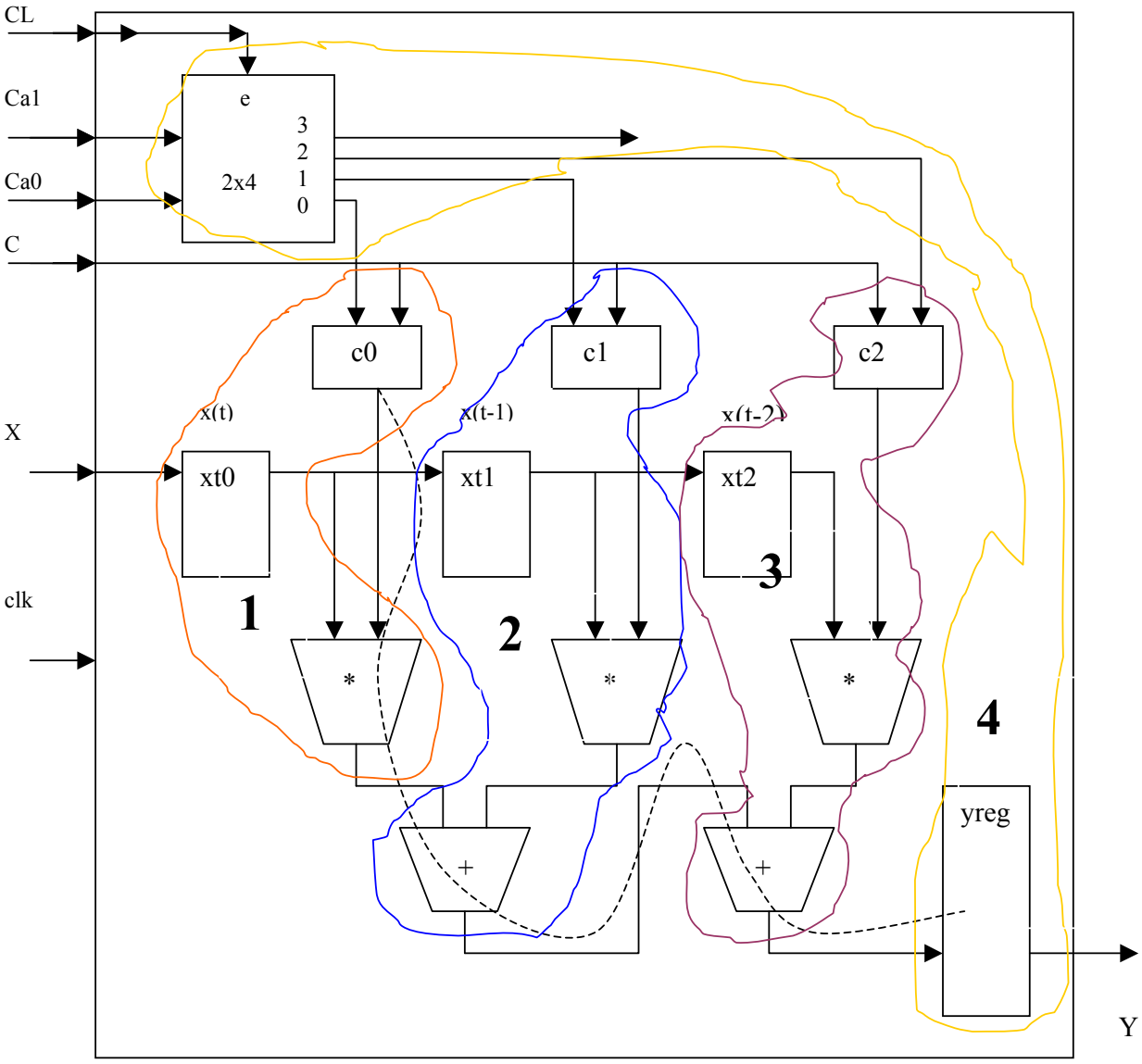
upisa CL, i dvobitne adrese Ca1 i Ca0, koje služe za upis konstanti u svaki registar ponaosob. U kombinaciji Ca0Ca1=00 selektuje registar C0 i može da se upiše u njega, kombinacija 01 selektuje C1 i 10 selektuje C2, a kada je 11 tada nema upisa . Za realizaciju nam je u ovom primeru potreban dekoder tipa 2x4, gde su Ca1 i Ca0 ulazi dekodera a CL je je ulazni signal dozvole, upis u ragistre se vrši na rastućoj ivici klok signala i kada je CL=1, slika (2.4) :



Slika 2.4

Da bi filter bio konfigurabilan moramo ga podeliti na vise sekcija u ovom našem primeru na tri sekcije. Prvu sekciju cine ćelije: registar c0, registar xt0 i mnozač. Drugu sekciju čine ćelije: registar c1, registar xt1, množač i sabirač. Treču sekciju čine ćelije: registar c2, registar xt2, množač i sabirač. Četvrtu sekciju čine ćelije

dekoder i Y registar. Uz korišćenje naredbe ***generic***, za generičku sekcuju, da bi filter bio konfigurabilan koristimo treću sekciju slika(2.5):



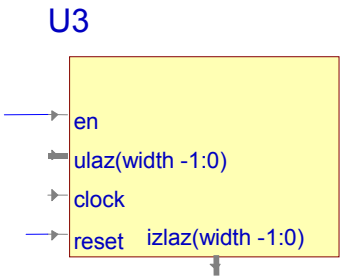
Slika 2.5

2.2 **VHDL** opis osnovnih komponenata

U ovom poglavlju su prikazani VHDL opisi svih komponenata koji čine FIR filter. Uz svaki opis objašnjena je funkcija posmatrene komponente u sistemu i data blok šema komponente sa naznačenim signalima.

2.3 Registar konstanti

Registar konstanti klasični registar sa paralenim ulazom paralelnim izlzom i ima **enable** signal koji dozvoljava upis informacija u registar.
Ovaj registar ima ulaz i izlaz promenjive širine **(width - 1 downto 0)**, port za taktovanje **(clock)**, port za dozvolu upisa **(en)**, i port za resetovanje **(reset)**.
Upis se vrši na uzlaznu ivicu takta. Blok šema registra konstanti je data na slici(2.6):

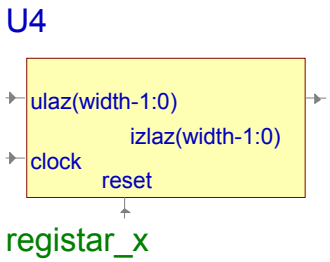


slika 2.6

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity registar_c is
generic(width : integer:=8);
        port(
                ulaz : in STD_LOGIC_VECTOR(width - 1 downto 0);
                clock : in STD_LOGIC;
                reset : in STD_LOGIC;
                en : in STD_LOGIC;
                izlaz : out STD_LOGIC_VECTOR(width - 1 downto 0)
            );
end registar_c;

architecture Behavioral of registar_c is

begin

        proc:process(clock, reset)
        begin
                if reset = '1' then
                        izlaz <= (others => '0');
                else
                        if rising_edge(clock) and en = '1' then
                                izlaz <= ulaz ;
                        end if;
                end if;
        end process;

end Behavioral;
```

2.4 Registar promenljivih(X registar)

Ovaj registar nema kontrolni signal, na svaku rastuču ivicu upisuje informaciju, sa ulaza prenosi je na izlaz.
Ovaj registar sadrži ulaz i izlaz promenjive širine **(width - 1 downto 0)**, port za taktovanje **(clock)** i port za resetovanje **(reset)**. Upis se vrši na uzlaznu ivicu takta. Blok šema registra promenljivih data je na slici (2.7):

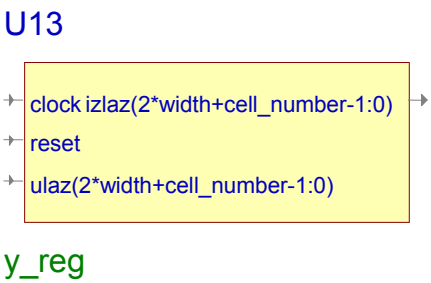


Slika 2.7

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity registar_x is
        generic(width : integer:=8);
         port(
                ulaz : in STD_LOGIC_VECTOR(width - 1 downto 0);
                clock : in STD_LOGIC;
                reset : in STD_LOGIC;
                izlaz : out STD_LOGIC_VECTOR(width - 1 downto 0)
            );
end registar_x;

architecture Behavioral of registar_x is

begin

        proc:process(clock, reset)
        begin
                if reset = '1' then
                        izlaz <= (others => '0');
                else
                        if rising_edge(clock) then
                                izlaz <= ulaz ;
                        end if;
                end if;
        end process;

end Behavioral;
```

2.5 Izlazni registar(Y registar)

Ovaj registar se ne razlikuje od predhodnog, stim sto mu je obim ulaznih i izlaznih podataka samim tim veći sobzirom da se na njemu dovodi rezultat obrade.
Ovaj registar sadrži ulaz i izlaz promenjive širine **(2*width - 1 downto 0)**, port za taktovanje **(clock)** i port za resetovanje **(reset)**. Upis se vrši na opadajućoj ivici takta.
Blok šema izlaznog registra data je na slici(2.8):

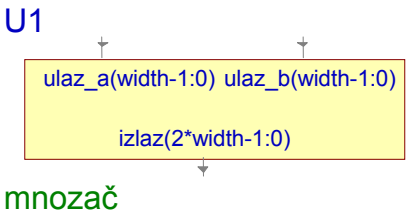


Slika 2.8

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Y_reg is
generic(width : integer:= 8;
            cell_number : integer:= 5);
          port(
          ulaz : in STD_LOGIC_VECTOR(2*width + cell_number - 1 downto 0);
          clock : in STD_LOGIC;
          reset : in STD_LOGIC;
          izlaz : out STD_LOGIC_VECTOR(2*width + cell_number - 1 downto 0)
             );
end Y_reg;

architecture Behavioral of Y_reg is

begin

        proc:process(clock, reset)
        begin
                if reset = '1' then
                        izlaz <= (others => '0');
                else
                        if falling_edge(clock) then
                                izlaz <= ulaz ;
                        end if;
                end if;
        end process;


end Behavioral;
```

2.6 Množač

Postoje tri tipa množenja: serijski, serijsko-paralelni i paralelni. Serijski mnozač nismo primenili zbog množenja u velikom broju taktnih intervala, postojanja kombinacione i kontrolne logike sa strane. Serijsko-paralelni množač nismo primenili bez obzira na manji broj taktnih intervala, zbog složenije logike za upravljanje. Ovde smo primenili paralelni množač iako je kontrolna logika ogromna ona je jednostavna. Serijski, serisko-paralelni množači su sekvencijalna a paralelni kombinaciona logička kola i nije mu potreban taktni interval.
Množač ima dva ulaza promenljive širine **(width-1:0)**, izlaz širine **(2*width-1:0)**. Blok šema množača šrikazana je na slici (2.9):



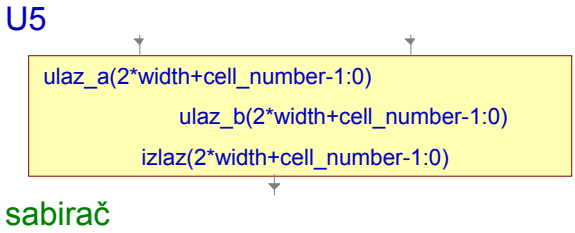
slika 2.9

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity mnozac is
          generic(width : integer:=8);
           port(
      ulaz_a : in STD_LOGIC_VECTOR(width - 1 downto 0);
      ulaz_b : in STD_LOGIC_VECTOR(width - 1 downto 0);
      izlaz : out STD_LOGIC_VECTOR(2*width - 1 downto 0)
               );
end mnozac;

architecture Behavioral of mnozac is

begin

    izlaz <= ulaz_a * ulaz_b ;

end Behavioral;
```

2.7 Sabirać

Postoji više hardverskih realizacija sabirača: brzi ***CARRY- LOOKAHEAD ADDITION, CARRY-COMPLETION SENSING ADDITION,*** *CARRY- SAVE ADDITION,* spori *FA array,* vhdl omogućava da se opiše na način dole naveden a koji tip će se izgenerisati ostavljamo alatu za sintezu. Sabirač ima ulaze promenljive širine **(2*width+cell_number-1:0)** i izlaz promenljive širine **(2*width + cell_number-1:0)** slika (2.10):



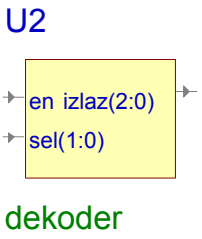

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

slika(2.10)

```
entity sabirac is
        generic(width : integer:=8;
              cell_number : integer:=5);
         port(
 ulaz_a : in STD_LOGIC_VECTOR(2*width + cell_number - 1 downto 0);
 ulaz_b : in STD_LOGIC_VECTOR(2*width + cell_number - 1 downto 0);
 izlaz : out STD_LOGIC_VECTOR(2*width + cell_number - 1 downto 0)
            );
end sabirac;

architecture Behavioral of sabirac is

begin

        izlaz <=  ulaz_a + ulaz_b;

end Behavioral;
```

2.8 Dekoder

Treba nam kolo koje reguliše koja se konstanta upisuje u koji registar, tu funkciju obavlja dekoder. Imamo selekcioni ulaz i kolo dekodera koje nam na izlazu da koji se registar selektuje za upis konstante, ima i port za dozvolu rada **(en)** slika(2.1):




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

Slika 2.11

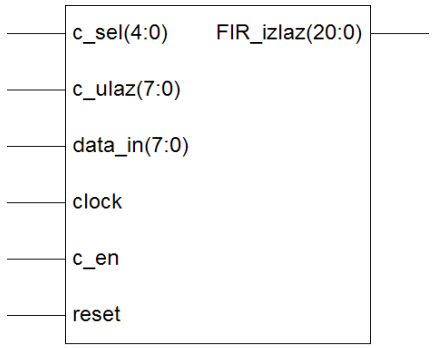
```
entity dekoder is
 generic(cell_number : integer:= 5 );
         port(
                 sel : in STD_LOGIC_VECTOR(cell_number-1 downto 0);
                 en : in STD_LOGIC;
                 izlaz : out STD_LOGIC_VECTOR(cell_number-1 downto 0)
             );
end dekoder;
```

```
architecture Behavioral of dekoder is

begin

        proc:process(sel, en)
        variable j : std_logic_vector(cell_number-1 downto 0);
        begin
                j := (others => '0');
                izlaz <= (others => '0');
                for i in 0 to cell_number-1 loop
                        if j = sel and en = '1' then
                                izlaz(i) <= '1';
                        end if;
                        j := j+1;
                end loop;
```

2.10 Opis entiteta FIR filtra

Na slici (2.12) prikazan je entitet FIR filtra. Spisak spoljnih pinova entiteta FIR filtra:

- **c_sel** : ulazni port kojim se kontroliše upis određene konstante u određeni registar;
- **c_ulaz** : ulazni port na koji se dovode konstante koje želimo da upišemo;
- **data_in**: ulazni port na koji se dovodi ulazni signal;
- **clock**: taktni signal kola;
- **c-en**: ulazni port dozvole za upis konstanti;
- **reset**: reset signal kola;
- **FIR_izlaz**: izlzni port na kome se dobijaju konačni rezultati obrade;



Slika 2.12 entitet FIR filtra

2.10 Opis arhitekture FIR filtra

Kod arhitekture FIR filtra koji međusobno povezuje osnovne komponente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity FIR_filtar is
 generic(width : integer := 8; cell_number : integer :=5); -- generic konstante

          port(
      c_en : in STD_LOGIC;
      clock : in STD_LOGIC;
      reset : in STD_LOGIC;
      c_sel : in STD_LOGIC_VECTOR(cell_number - 1 downto 0);
      c_ulaz : in STD_LOGIC_VECTOR(width - 1 downto 0);
      data_in : in STD_LOGIC_VECTOR(width-1 downto 0);
      FIR_izlaz : out STD_LOGIC_VECTOR(2*width + cell_number - 1 downto 0)
  );
end FIR_filtar;

architecture Behavioral of  FIR_filtar is
-- polja magistrala
type C_reg_type is array (cell_number - 1 downto 0) of STD_LOGIC_VECTOR
(width - 1 downto 0);
type add_out_type is array (cell_number - 1 downto 0) of STD_LOGIC_VECTOR
(2*width + cell_number - 1 downto 0);
type mul_out_type is array (cell_number - 1 downto 0) of STD_LOGIC_VECTOR
(2*width - 1 downto 0);
type data_type  is array (cell_number - 1 downto 0) of STD_LOGIC_VECTOR
(width - 1 downto 0);
-- dodela unutrašnjih signala
signal enable_line : STD_LOGIC_VECTOR (cell_number - 1 downto 0);
signal DATA : data_type;
signal mul_out : mul_out_type;
signal mul_out_pom :add_out_type;
signal add_out : add_out_type;
signal C_reg : C_reg_type;
signal fill: std_logic_vector(cell_number-1 downto 0):=(others => '0');
-- generic petlja
begin

          gen: for i in 0 to cell_number generate
          begin

                    prva: if (i = 0) generate
```

```
 begin
 cell_1:entity work.registar_c generic map(width => width)
 port map(
 ulaz  => c_ulaz,
 clock => clock,
 reset => reset,
 en => enable_line(i),
 izlaz => C_reg(i));

 cell_2:entity work.registar_x generic map(width => width)
 port map(
 ulaz  => data_in,
 clock => clock,
 reset => reset,
 izlaz => DATA(i));

 cell_3:entity work.mnozac generic map(width => width)
 port map(
 ulaz_a => DATA(i),
 ulaz_b => C_reg(i),
 izlaz  => mul_out(i));

 mul_out_pom(i) <= fill & mul_out(i);


end generate prva;

druga: if (i = 1) generate
 begin
 cell_4:entity work.registar_c generic map(width => width)
 port map(
 ulaz  => c_ulaz,
 clock => clock,
 reset => reset,
 en => enable_line(i),
 izlaz => C_reg(i));

  cell_5:entity work.registar_x generic map(width => width)
  port map(
  ulaz  => DATA(i-1),
  clock => clock,
  reset => reset,
  izlaz => DATA(i));

  cell_6:entity work.mnozac generic map(width => width)
  port map(
  ulaz_a => DATA(i),
  ulaz_b => C_reg(i),
  izlaz  => mul_out(i));
```

```
                        mul_out_pom(i) <= fill & mul_out(i);

                        cell_7:entity work.sabirac generic map(width => width,
cell_number => cell_number)
                         port map(
                         ulaz_a => mul_out_pom(i-1),
                         ulaz_b => mul_out_pom(i),
                         izlaz  => add_out(i-1));

                        end generate druga;

                        treca: if (i >= 2) and i < cell_number generate
                        begin
                        cell_8:entity work.registar_c generic map(width => width)
                        port map(
                        ulaz  => c_ulaz,
                        clock => clock,
                        reset => reset,
                        en => enable_line(i),
                        izlaz => C_reg(i));

                        cell_9:entity work.registar_x generic map(width => width)
                        port map(
                        ulaz  => DATA(i-1),
                        clock => clock,
                        reset => reset,
                        izlaz => DATA(i));

                        cell_10:entity work.mnozac generic map(width => width)
                        port map(
                        ulaz_a => DATA(i),
                        ulaz_b => C_reg(i),
                        izlaz  => mul_out(i));

                        mul_out_pom(i) <= fill & mul_out(i);

                        cell_11:entity work.sabirac generic map(width => width,
cell_number => cell_number)
                        port map(
                        ulaz_a => mul_out_pom(i),
                        ulaz_b => add_out(i-2),
                        izlaz  => add_out(i-1));

                        end generate treca;

                        cetvrta: if i = cell_number generate
                        begin

                         cell_12:entity work.Y_reg generic map(width => width,
cell_number => cell_number)
```

```
                        port map(
                        ulaz  => add_out(i-2),
                        clock => clock,
                        reset => reset,
                        izlaz => FIR_izlaz);

                        cell_13: entity work.dekoder generic map(cell_number =>
cell_number)
                        port map(
                        sel => c_sel,
                        en => c_en,
                        izlaz  => enable_line);

                    end generate cetvrta;
            end generate gen;


    end Behavioral;
```

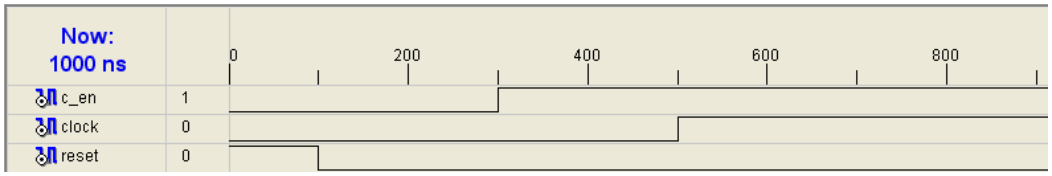
3.1 VHDL kod test bench-a FIR filtra i prikaz rezultata simulacije

VHDL kod test bench-a koji odgovara gore navedenom kodu arhitekture FIR filtra koji međusobno povezuje osnovne komponente izleda ovako:

```
-- deo test-bench-a koji program generiše
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY test_vhd IS
END test_vhd;

ARCHITECTURE behavior OF test_vhd IS
-- Component Declaration for the Unit Under Test (UUT)
COMPONENT FIR_filtar
PORT(
c_en : IN std_logic;
clock : IN std_logic;
reset : IN std_logic;
c_sel : IN std_logic_vector(4 downto 0);
c_ulaz : IN std_logic_vector(7 downto 0);
data_in : IN std_logic_vector(7 downto 0);
FIR_izlaz : OUT std_logic_vector(20 downto 0)
);
END COMPONENT;
```

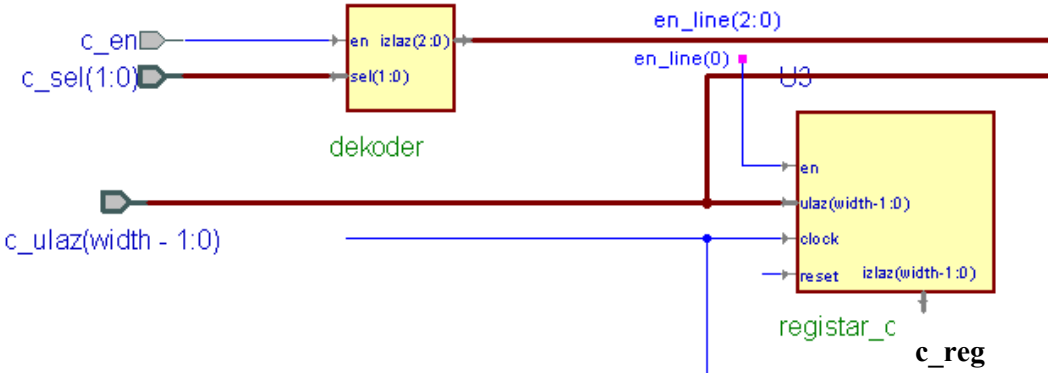
```
--Inputs
SIGNAL c_en :  std_logic := '0';
SIGNAL clock :  std_logic := '0';
SIGNAL reset :  std_logic := '0';
SIGNAL c_sel :  std_logic_vector(4 downto 0) := (others=>'0');
SIGNAL c_ulaz :  std_logic_vector(7 downto 0) := (others=>'0');
SIGNAL data_in :  std_logic_vector(7 downto 0) := (others=>'0');

--Outputs
SIGNAL FIR_izlaz :  std_logic_vector(20 downto 0);

BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: FIR_filtar PORT MAP(
c_en => c_en,
clock => clock,
reset => reset,
c_sel => c_sel,
c_ulaz => c_ulaz,
data_in => data_in,
FIR_izlaz => FIR_izlaz
);
-- deo test-bench-a koji se piše ručno
clock_proc:process(clock)
  begin
    if clock = 'U' then
    clock <= '1';
  else
    clock <= not clock after 500ns;
  end if;
end process;

reset <= '1', '0' after 100ns;
c_en <= '0', '1' after 300ns, '0' after 5300ns;

c_sel   <= "00000", "00001" after 1100ns, "00010" after 2100ns, "00011" after
3100ns, "00100" after 4100ns, "10000" after 5100ns;

c_ulaz  <= "00001100", "00000111" after 1100ns, "00001011" after 2100ns,
"00001111" after 4100ns, "00000000" after 5100ns;

data_in <= "00000000", "00001010" after 7100ns, "00001100" after 8100ns,
"00000111" after 9100ns, "00001011" after 10100ns, "00010000" after 11100ns,
"00000000" after 12100ns;

END;
```

Sinteza, implementacija i testiranje se vrši u programskom paketu ***Xilinx ISE 9.1 i***. Ovde imamo standardni **clock_ proces** koji je osetljiv na promene **clock** signala, promena **clock** signala je na svakih 500ns što znači da imamo periodu od 1000ns. **Reset** signal je aktivan odmah po početku simulacije a prestaje da bude aktivan 100ns što je sasvim dovoljno da se svi registri resetuju po početku simulacije, **c_en** signal je aktivan za svo vreme dok se upisuju konstante slika (2.13):



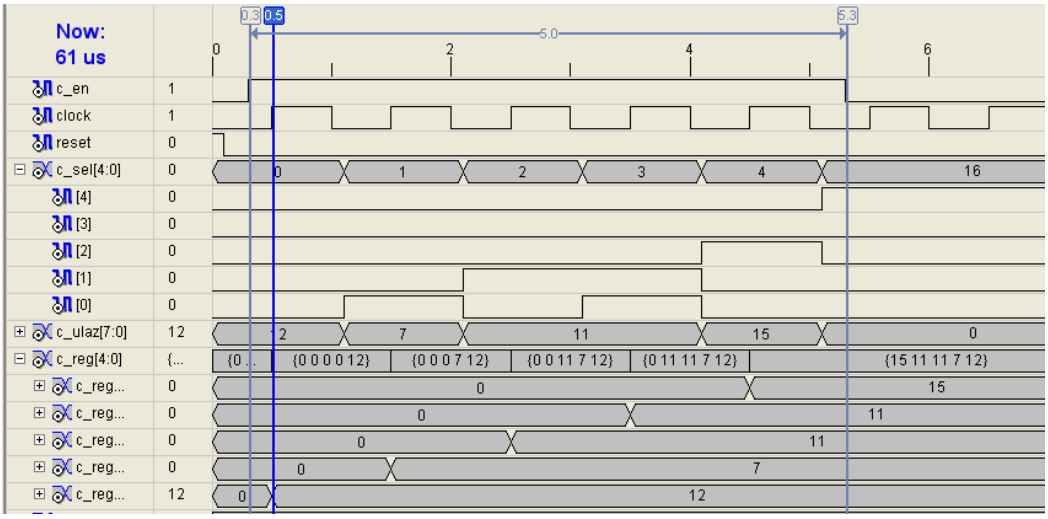
slika 2.13

Selekcija i upis konstanti u registre vrši se pomoću dekodera i registara konstanti, kojih u ovom našem slučaju ima pet, slika (2.14).



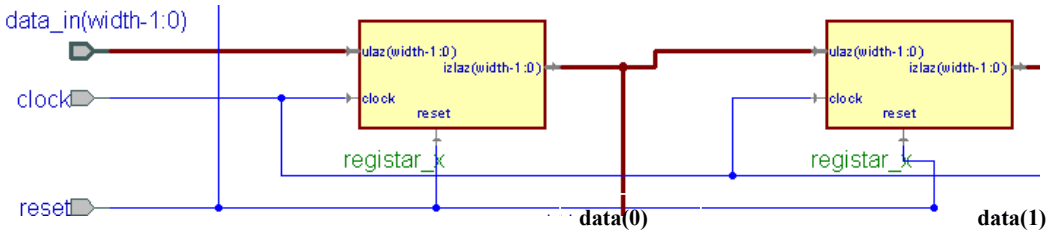
slika 2.14

Na ulaz dekodera se dovodi **c_en** signal koji je aktivan posle 300ns a prestaje da bude aktivan posle 5300ns, i signal **c_sel** koji ima vrednost u intervalu od 300ns do 1100ns jednak svim nulama. Na taj način je signalom **c_en** omogućen upis u registre konstanti a **c_sel** signalom je selektovan prvi registar. Kako signal **c_ulaz** ima vrednost neke konstante u ovom našem primeru 12, na rastuću ivicu taktnog signala(500ns) se vrši upis konstante u selektovani registar, u ovom našem primeru je selektovan prvi registar a konastanta je 12. Upis u svih pet registra konstanti je potpuno indentićan i vrši se na rastuću ivicu taktnog signala sve do trenutka(5300ns) kada signal **c_en** prestaje da bude aktivan slika (2.15).



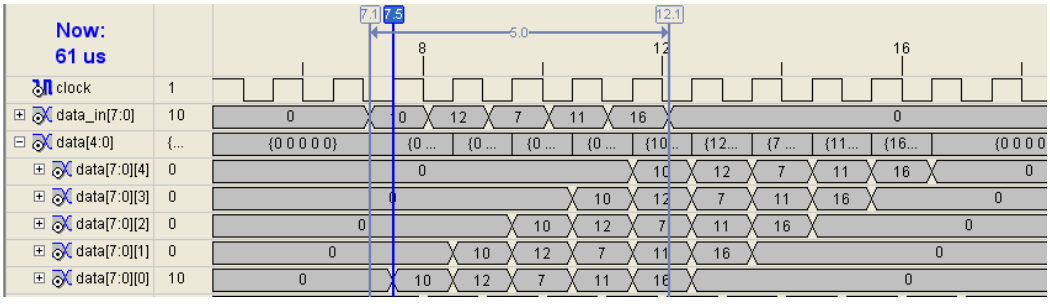
slika 2.15

U trenetku 7100ns počinje da se dovodi ulazni signal, signal na **data_in** ulazu kola slika (2.16).



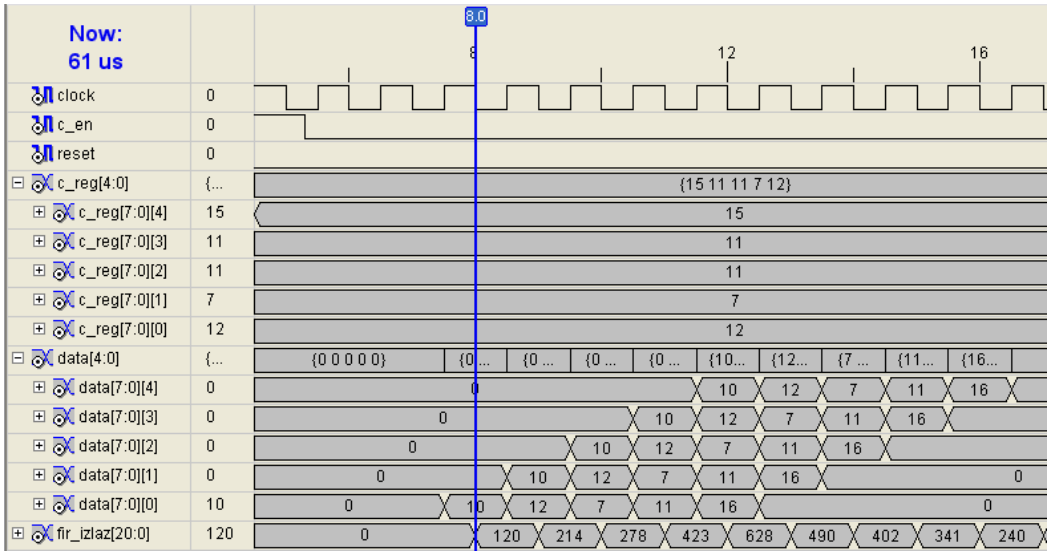
slika2.16

Na rastućoj ivici **clock** signala vrši se upis u prvi registar (**registar_x**) vrednost 10, na sledećoj rastučoj ivici **clock** signala predhodna vrednost ulaznog signala 10 se upisuje u naredni registar (**registar_x**) a nova vrednost ulaznog signala 12 se upisuje u prvi registar (**registar_x**) na taj način se vrši upis i pomeranje svake vrednosti ulaznog signala kroz registre slika (2.17).

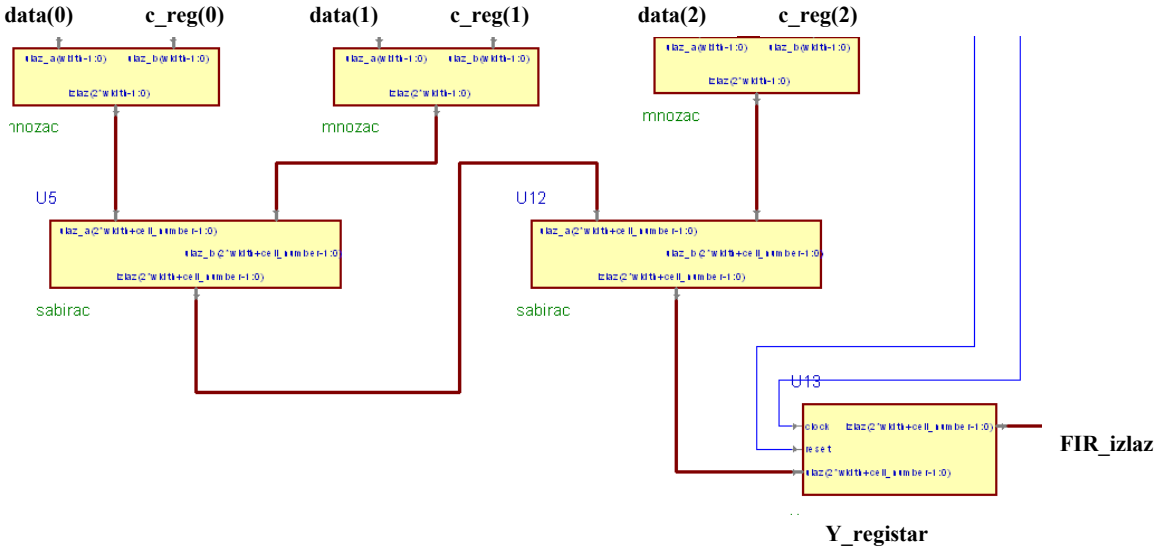


slika 2.17

U trenutku 8000ns slika(2.18), na opadajućoj ivici **clock** signala, na izlazu **y_registra** javlja se rezultat. Kako su izlazi registra konstanti **c_reg(0) =** 12, **c_reg(1)** = 0, **c_reg(2)** = 0, a izlazi registra za prihvatanje ulaznog signala **data_in** : **data(0)** = 10, **data(1)** = 0, **data(2)** = 0 slika (2.19), na izlazu **y_registra** dobija se vrednost 120**.** Zbog promene vrednosti ulaznog signala **data_in**, menja se i vrednost na izlazu **y_registra.** Sabirači i množači su kombinaciona logička kola tako da nad vrednostima se vrše operacije sabiranja i množenja redosledom i putanjom kao na slici (2.19).



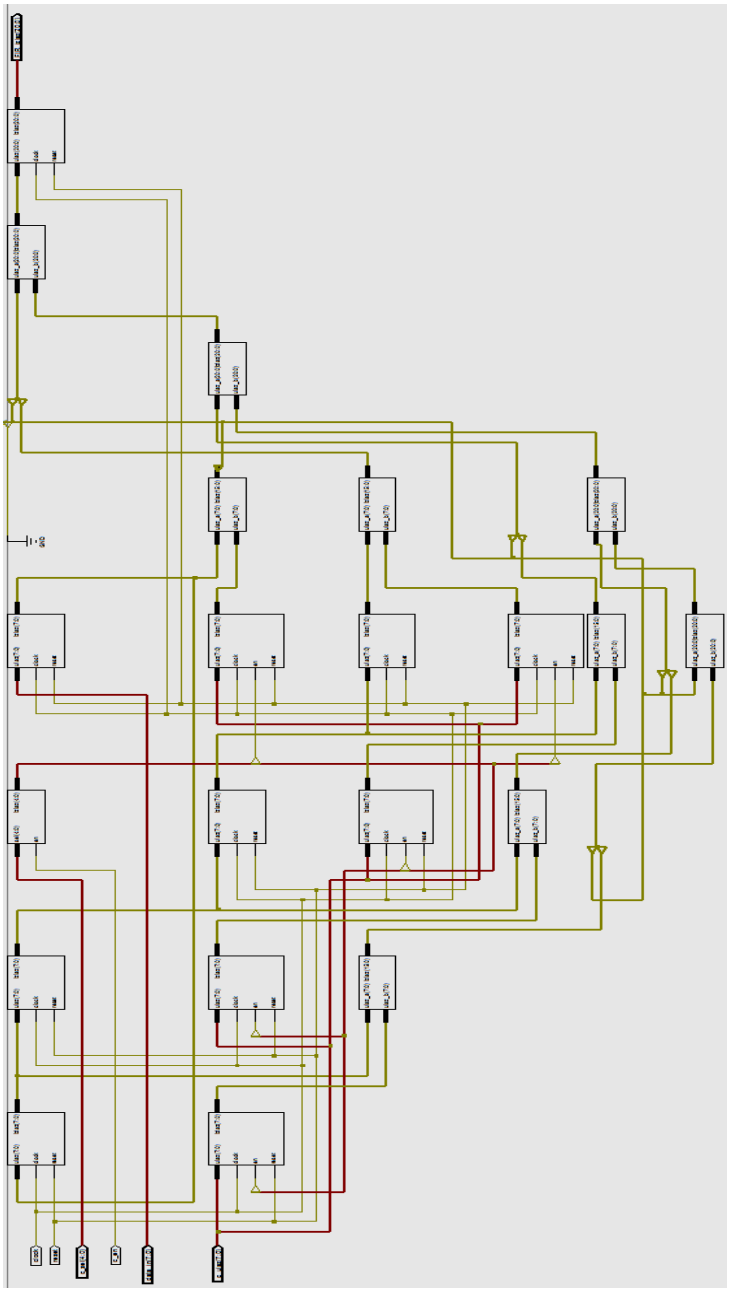
slika2.18



slika 2.19

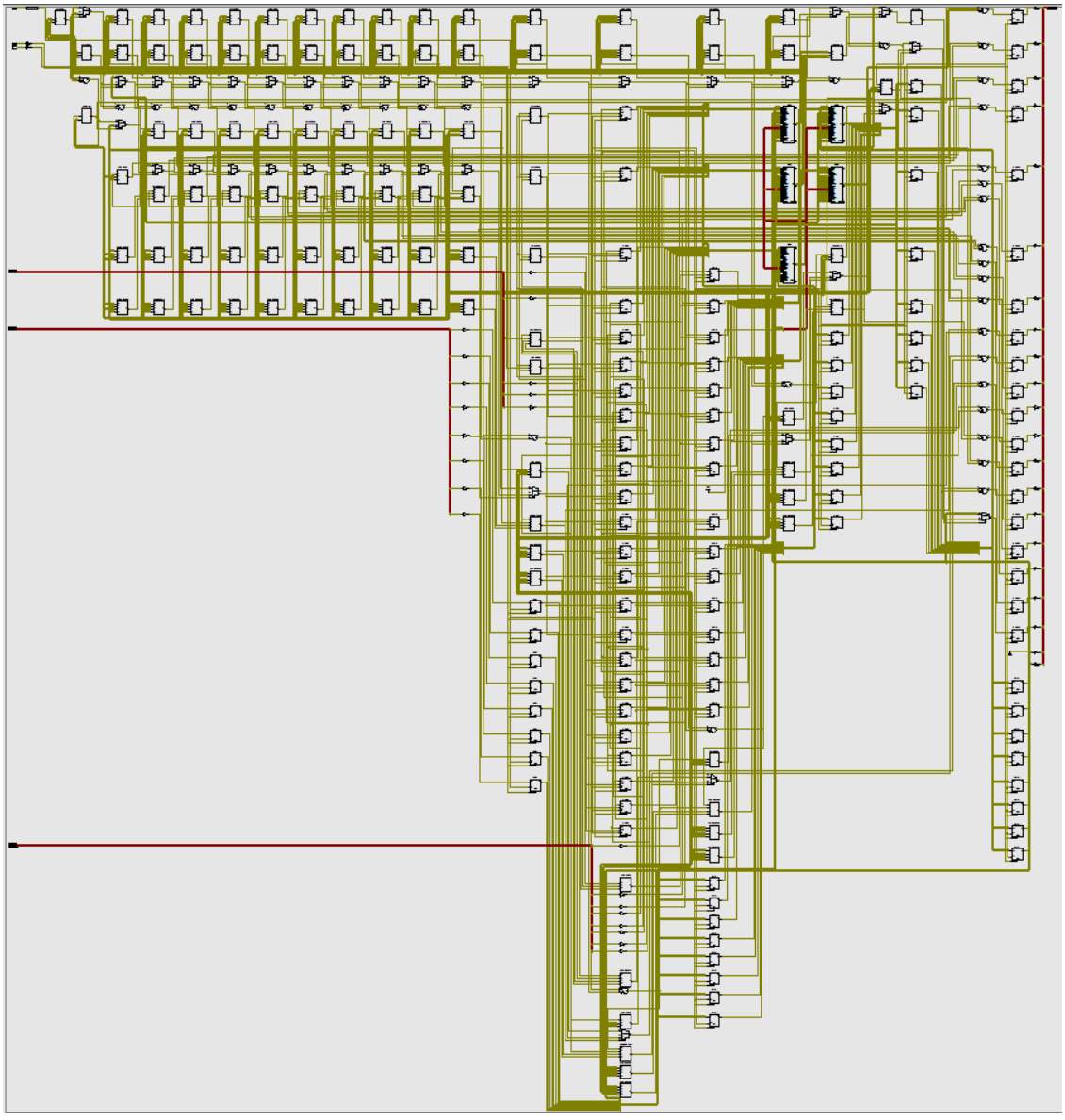
4.1 Rezultati sinteze i implementacije

Postupak sinteze i implementacije urađen je u programskom paketu Xilinx 9.1. U nastavku teksta date su slike koje prikazuju rezultat sinteze i implementacije u pomenutom programskom paketu. Najpre je data slika (2.20) rezultata sinteze na kojoj se jasno uočavaju entiteti upotrebljeni za realizaciju filtra. Sa slike se jasno uočavaju kola sabirača, množača, registara za čuvanje konstanti i X registri za formiranje toka podataka. Poređenjem slike sintetizovanog kola i blok šeme filtra uočava se da je njihova struktura ista.



slika 2.20 RTL nivo

Na sledećoj slici (2.21) je data slika kola koja je rezultat implementacije kola na čipu iz familije Virtex4 XC4VLX15 u pakovanju SF363. Slika koja prikazuje rezultat implementacije data je tako da se na njoj vide sva upotrebljena kola za realizaciju filtra. Ovo je urađeno tako što se u meniju za izbor broja kola postavi parametar unlimited.



slika 2.21 GATE nivo

U daljem tekstu dat je pregled rezultata implementacije. Ovaj pregled generiše se automatski od strane programskog paketa Xilinx. U okviru ovog pregleda nalaze se svi relevantni podaci koji ukazuju na iskorišćene resurse čipa korišćenog za implementaciju kola FIR filtra.

Design Summary

Number of errors: 0
Number of warnings: 0
Logic Utilization:
Number of Slice Flip Flops: 65 out of 12,288 1%
Number of 4 input LUTs: 97 out of 12,288 1%
Logic Distribution:
Number of occupied Slices: 94 out of 6,144 1%
Number of Slices containing only related logic: 94 out of 94 100%
Number of Slices containing unrelated logic: 0 out of 94 0%
***See NOTES below for an explanation of the effects of unrelated logic**
Total Number of 4 input LUTs: 98 out of 12,288 1%
Number used as logic: 97
Number used as a route-thru: 1
Number of bonded IOBs: 45 out of 240 18%
Number of BUFG/BUFGCTRLs: 1 out of 32 3%
Number used as BUFGs: 1
Number used as BUFGCTRLs: 0
Number of DSP48s: 5 out of 32 15%

Total equivalent gate count for design: 1,584
Additional JTAG gate count for IOBs: 2,160

U daljem tekstu dat je pregled ukupnog broja komponenti upotrebljenih za realizaciju kola filtra. Ovaj spisak generiše sam program i vidi se da je iz VHDL koda program generisao odgovarajuća kola za realizaciju kola filtra. Na ovaj način smo dobili potvrdu da je opis kola optimalan jer je sam program sintetizovao kola koja su VHDL kodom opisana behavioralno.

HDL Synthesis Report

Macro Statistics
 # Multipliers : 5
8x8-bit multiplier : 5
 # Adders/Subtractors : 4
21-bit adder : 4
 # Registers : 11
21-bit register : 1
8-bit register : 10

*** Advanced HDL Synthesis ***

Advanced HDL Synthesis Report

Macro Statistics
 # Multipliers : 5
8x8-bit multiplier : 5
 # Adders/Subtractors : 4

21-bit adder : 4
 # Registers : 101
Flip-Flops : 101

Vremenski izveštaj kola nam govori o tome da kolo može da ispravno radi na frekvenciji od 66.8MHz, i ovim izveštajem predložena su vremena postavljanja i držanja signala kako bi kolo ispravno radilo.

The AVERAGE CONNECTION DELAY for this design is: 0.948
The MAXIMUM PIN DELAY IS: 2.717
The AVERAGE CONNECTION DELAY on the 10 WORST NETS is: 1.755

Timing Summary:

Speed Grade: -12

Minimum period: 14.967ns (Maximum Frequency: 66.812MHz)
Minimum input arrival time before clock: 2.917ns
Maximum output required time after clock: 3.753ns
Maximum combinational path delay: No path found

U daljem tekstu dat je pregled podataka koji se tiču procenjene potrošnje kola. Ukupna procenjena potrošnja kola je 281 mW.

Power summary: I(mA) P(mW)

Total estimated power consumption: 281

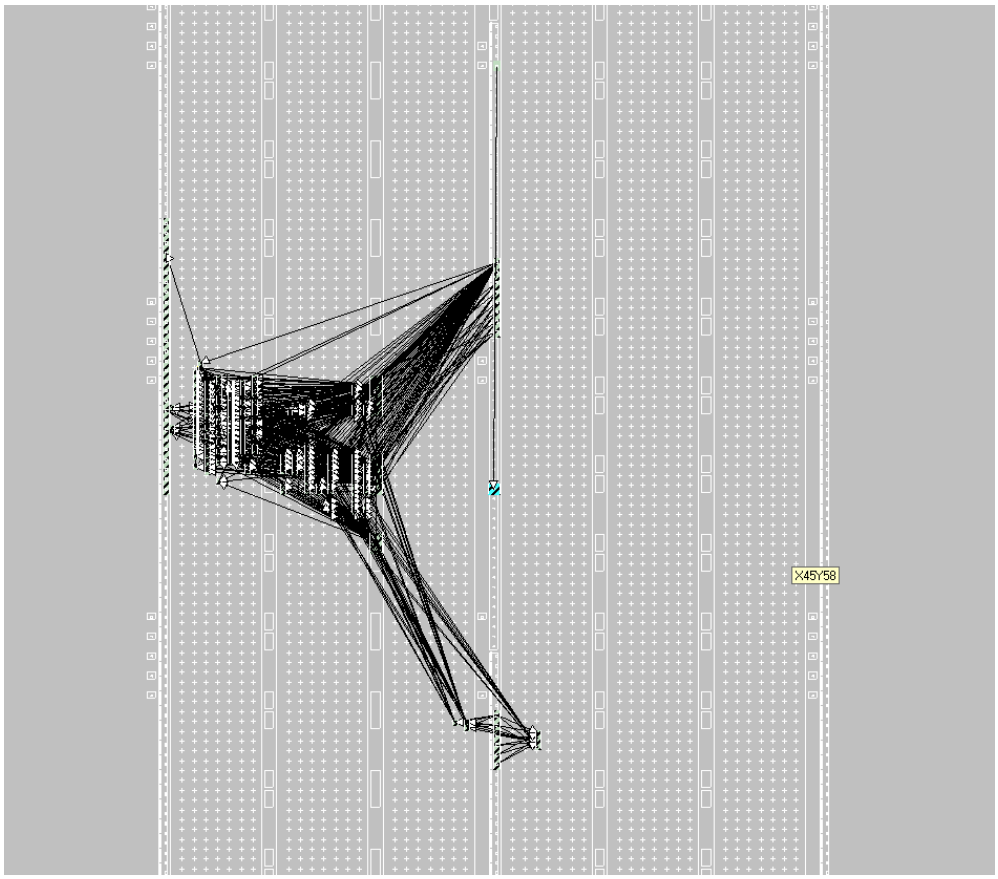
Vccint 1.20V: 50 60
Vccaux 2.50V: 88 221
Vcco25 2.50V: 0 0

Clocks: 0 0
Inputs: 0 0
Logic: 0 0
Outputs:
Vcco25 0 0
Signals: 0 0

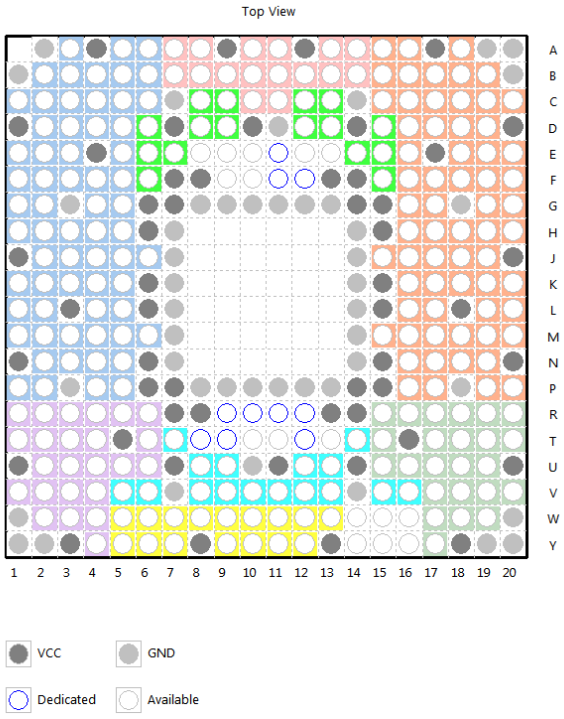
Quiescent Vccint 1.20V: 50 60
Quiescent Vccaux 2.50V: 88 221

Thermal summary:

Estimated junction temperature: 31C
Ambient temp: 25C
Case temp: 31C
Theta J-A range: 21 - 18C/W



slika 2.22 zauzeće kola sa prikazom net-ova



slika 2.23 raspored pinova

5.1 Literatura

[1] R.E. Crocheire, L.R. Rabiner, "*Multirate Digital Signal processing*", Prentice-Hall, 1982.
[2] N.J. Fliege, "*Multirate Digital Signal processing*", John Wiley, 1994.
[3] Lj. D. Mili}, M.D. Lutovac, "Efficient Multirate Filtering", in Gordana Jovanovi}-Dole~ek, "*Multirate Systems: Design and Applications*", Idea Group Publishing, 2002, pp. 108-145.
[4] Y.C. Lim, "Frequency-Response Masking Approach for the Synthesis of Sharp Linear Phase Digital Filters", *IEEE Transactions on Circuits and Systems*, vol. 33, no. 4, pp. 357-364, 1986.
[5] H. Johansson, L. Wanhammar, "High-Speed Recursive Digital Filters Based on the Frequency-Response Masking Approach", *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 47, no.1, pp. 48-61, 2000.
[6] M. Popovi}, "*Digitalna obrada signala*", Nauka, 1997.
[7] Lj. Mili}, Z. Dobrosavljevi}, "*Uvod u digitalnu obradu signala*", Elektrotehni~ki fakultet – Beograd, 1999.
[8] M. D. Lutovac, D. V. To{i}, B. L. Evans, *Filter Design for Signal Processing Using MATLAB and Mathematica*, Upperside River, New Jersey, Prentice Hall, 2000.
[9] S.K. Mitra, "*Digital Signal Processing: A Computer Based Approach*", McGraw Hill, 2001.
[10] The MathWorks, Inc., Filter Design Toolbox, MATLAB Version 6 Release 12.1, Natick, MA 01760-2098, 2001
[11] P.P. Vaidyanathan, T.O. Nguen, "*A Trick for the Design of FIR Half-Band Filters*", *IEEE Transactions on Circuits and Systems,* Vol. 34, 1987., pp. 297-300.
[12] E.B. Hogenauer, "An Economical Class of Digital Filters for Decimation and Interpolation", *IEEE Trans. Acoustics, Speech, and Signal Processing*, Vol. 29, 1981., pp. 155-162.
[13] L. Gaszi, "Explicite Formulas for lattice wave digital filters", *IEEE Trans. Circuits and Systems*, vol. 32, no. 1, pp. 68-88, 1985.
[14] M. D. Lutovac, Lj. D. Mili}, "Lattice Wave Digital Filters with a Reduced Number of Multipliers", *Yugoslav IEEE MIT Chapter Informer*, no. 3, pp. 29-39, 1996.
[15] Lj. D. Mili}, M. D. Lutovac, "Design of Multiplierless Elliptic IIR Filters with a Small Quantization Error", *IEEE Transactions on Signal Processing*, vol. 47, no. 2, pp. 469-479, 1999.