SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1542

FPGA implementacija digitalnog pretvarača frekvencije bez upotrebe množila

Ivan Laznibat

Zagreb, lipanj 2017.

Sadržaj

Popis slika

Popis tablica

Uvod

Sustav za digitalnu konverziju frekvencije uzorkovanja (eng. Digital down converter, DDC) je sastavni blok svakog programski izvedenog radio prijemnika. Na ulazu, često se nalaze brzi AD pretvornici koji digitaliziraju pojas koji je ograničen analognim filtrima (RF ili IF). U praksi, traženi signal, najčešće predstavlja samo uski dio tog pojasa koji se izdvaja filtarskim lancem visoke selektivnosti (kaskada decimacijskih i selekcijskih filtara). U današnjim radio prijemnicima koriste se niski decimacijski faktori, što znači da njegovi filtri rade s visokim frekvencijama uzorkovanja. Kao posljedica toga od filtara se zahtjeva učinkovita implementacija.

Cilj diplomskog rada je istražiti mogućnost sklopovske izvedbe cjelokupnog filtarskog lanca sustava za digitalnu konverziju frekvencije uzorkovanja bez upotrebe množila. Korištenjem implementacije bez množila značajno se smanjuje složenost sustava i povećava brzina rada. Nakon AD pretvorbe, digitalizirani signal se pomoću numerički kontroliranog oscilatora spušta u područje nižih frekvencija kako bi DDC mogao lakše izdvojiti traženi uski pojas. DDC dodatno spušta frekvenciju traženog pojasa, nakon čega smanjuje i frekvenciju uzorkovanja. Tipični DDC sustav sastoji se od tri filtra spojenih u kaskadu. To su CIC (eng. *Cascaded Integrator Comb*) decimator, CIC kompenzator i filtar kanala. Oni i njihova poboljšanja će biti detaljnije opisani u nastavku.

1. Digitalna pretvorba frekvencije uzorkovanja bez upotrebe množila

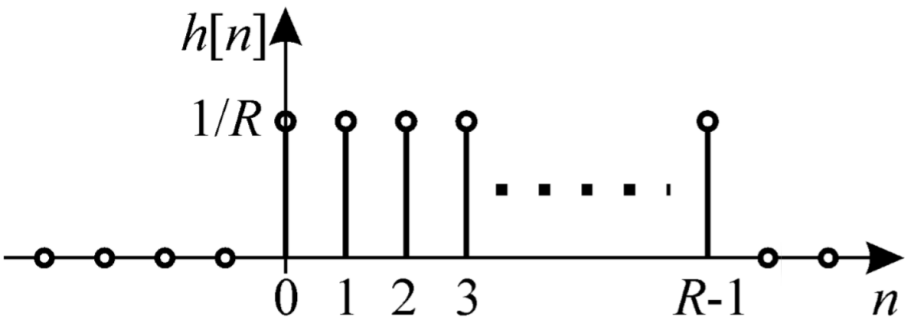
1.1. Decimacijski filtri visokog stupnja decimacije

1.1.1. CIC filtri

CIC filtri su digitalni filtri visokog stupnja decimacije čija implementacije ne sadrži množila, a često se koriste kao decimatori ili interpolatori u digitalnim pretvaračima frekvencije uzorkovanja. CIC filtre je razvio Eugene Hogenauer [1] sredinom 80-ih godina 20. stoljeća, ali njihova daljnja poboljšanja i primjene još su uvijek od velikog interesa, naročito u digitalnim radio prijamnicima.

Impulsni odziv (slika 1., [2]) CIC-a dobiva se usrednjavanjem R uzoraka:

$$h(n) = \frac{1}{R}\{\delta(n) + \delta(n-1) + \ldots + \delta(n-R+1)\} \tag{1}$$



Slika 1. Impulsni odziv CIC filtra

Primjenom z transformacije na impulsni odziv (1) dobiva se:

$$H(z) = \frac{1}{R}\left(1 + z^{-1} + \cdots + z^{-(R-1)}\right) = \frac{1}{R}\sum_{k=0}^{R-1} z^{-k} \tag{2}$$

Izraz za sumu prvih N članova geometrijskog niza:

$$S_N = a_1 \sum_{k=0}^{N-1} q^k = a_1 \frac{1-q^N}{1-q} \tag{3}$$

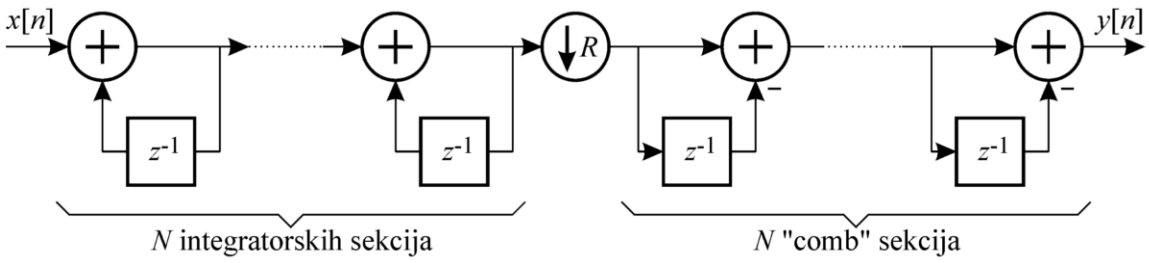
Uvrštavanjem (2) u (3) dobivamo konačan izraz za prijenosnu karakteristiku CIC filtra prvog reda:

$$S_R = H(z) = \frac{1}{R} \frac{1-z^{-R}}{1-z^{-1}} \tag{4}$$

Prijenosna karakteristika CIC filtra N-tog reda je:

$$H_{CIC}(z) = \left(\frac{1}{R} \frac{1-z^{-R}}{1-z^{-1}} \right)^N \tag{5}$$

gdje N označava red filtra, a R faktor decimacije. Struktura CIC decimacijskog filtra prikazana je na slici 2. [2]. Sa slike je vidljivo da je struktura CIC decimacijskog filtra rekurzivna FIR struktura koju čine kaskada digitalnih integratora i derivatora prvog reda. Između njih se nalazi sklop za izbacivanje uzoraka (eng. *downsampler*).



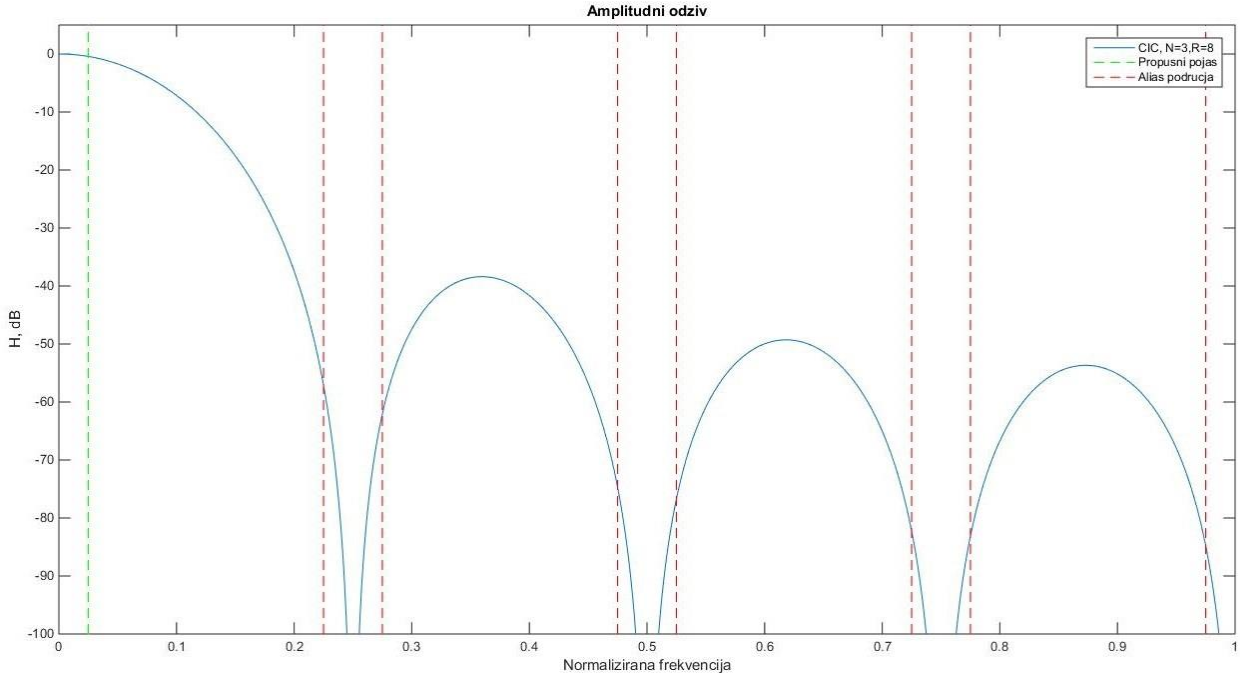
Slika 2. Struktura CIC filtra N-tog reda

Prijelazom iz z domene u frekvencijsku domenu, nakon kratkog matematičkog računa dolazi se do izraza za amplitudno frekvencijski odziv CIC filtra N-tog reda (slika 3.). Ona je oblika:

$$H_{CIC}(\omega) = \left(\frac{1}{R} \frac{sin\left(\frac{\omega R}{2}\right)}{sin\left(\frac{\omega}{2}\right)} \right)^{N} \tag{6}$$

Fazna karakteristika CIC filtra je oblika

$$\Theta(\omega) = -N \frac{R+1}{2} \omega = -D\omega \tag{7}$$



Slika 3. Amplitudna karakteristika CIC filtra s N=3 i R=8

Do aliasa u području propuštanja dolazi zbog komponenata oko nultočaka:

- R je paran:

$$\Omega = \begin{cases} \frac{2\pi n}{R} - \omega_p \leq \omega \leq \frac{2\pi n}{R} + \omega_p, & n = 1, \dots, \frac{R}{2} - 1 \\ \pi - \omega_p \leq \omega \leq \pi \end{cases} \tag{8}$$

- R je neparan:

$$\Omega = \frac{2\pi n}{R} - \omega_p \leq \omega \leq \frac{2\pi n}{R} + \omega_p, \qquad n = 1, \dots, \frac{R-1}{2} \tag{9}$$

U nastavku teksta dani su ključni nedostaci CIC filtra i načini njegova poboljšanja:

- Amplitudna karakteristika ima veliki propad unutar područja propuštanja. Ovaj propad se može smanjiti dodatnim filtrom koji se spaja na izlaz CIC decimatora, a naziva se CIC kompenzator.
- U područje propuštanja se preslikavaju aliasi s viših frekvencija. Maksimalni aliasing je na frekvenciji:

$$\omega_a = \frac{2\pi}{R} - \omega_c \tag{10}$$

Kako bismo osigurali visoko gušenje aliasa, potrebno je koristi visoki red filtra N, tj. velik broj integratora i derivatora (složenija i skuplja sklopovska implementacija). Za smanjivanje reda filtra često se koristi tehnika polinomijalnog izoštravanja CIC amplitude (eng. *sharpened CIC*, *SCIC*).

1.1.2. SCIC filtri

Polinomijalno izoštravanje (eng. *sharpening*) amplitude CIC filtra je tehnika kojom se osigurava bolja selektivnost filtra. Glavna prednost: niži red izoštrenog CIC filtra je usporediv s CIC filtrom višeg reda, pa time dolazi do uštede u sklopovlju i potrošnji energije. Izoštravanje CIC filtra [3] se provodi prema izrazu za polinom M-tog stupnja:

$$f(x) = \sum_{m=0}^{M} a_m x^m \tag{11}$$

Varijabla x se supstituira s amplitudnom karakteristikom CIC filtra, tj. $x = H_{SCIC}(\omega)$, te tako dobivamo izraz za amplitudu SCIC filtra oblika [5]:

$$H_{SCIC}(\omega) = \sum_{m=0}^{M} a_m H_{CIC}^m(\omega) \tag{12}$$

gdje a_m označava koeficijent, a M stupanj polinoma izoštravanja.

Kako bi se postigla maksimalna brzina takta potrebno je nakon svakog stupnja integratora dodati registar (kašnjenje za 1 uzorak). Prijenosna funkcija tako modificiranog CIC filtra ima oblik

$$H(z) = \frac{1}{R^N}\left(z^{-1}\frac{1-z^{-R}}{1-z^{-1}}\right)^N = \frac{1}{R^N} I^N(z) C^N(z^R) \tag{13}$$

gdje su

$$I(z) = \frac{z^{-1}}{1-z^{-1}} \tag{14}$$

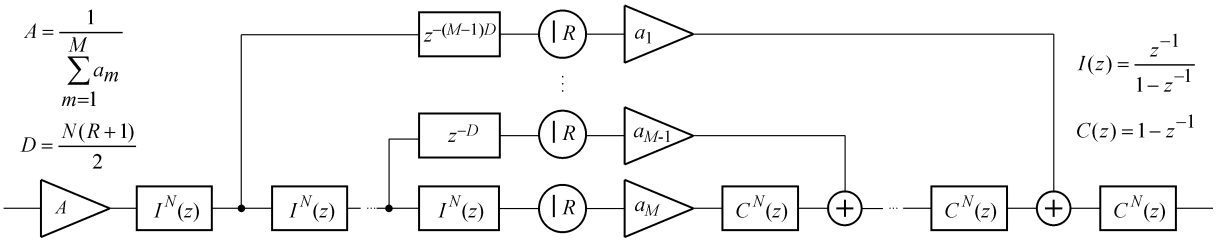
$$C(z) = 1 - z^{-1} \tag{15}$$

Očito je da izraz (14) predstavlja integratorsku sekciju, a izraz (15) „comb“ sekciju polinomijalno izoštrenog CIC filtra.

Često se za smanjenje složenosti SCIC filtra postavlja $a_0 = 0$. Naime ovaj koeficijent ima najmanji utjecaj na izoštravanje CIC karakteristike. Uz pretpostavku $a_0 = 0$ nakon kratkog matematičkog računa dobiva se konačan oblik prijenosne funkcije SCIC filtra

$$H_{SCIC}(z) = \sum_{m=1}^{M} a_m H_{CIC}^m(z)\, z^{-(M-m)D} \tag{16}$$

Slika 4. [7] prikazuje strukturu SCIC filtra prema (17) formuli. D (6) mora biti cjelobrojan, što znači da umnožak *N(R-1)* mora biti paran broj.



Slika 4. Struktura SCIC filtra

Postoje dva pristupa u dizajnu SCIC filtara. Prvi pristup modulira područje propuštanja i područja aliasa [4], [5]. Drugi pristup modulira isključivo područja aliasa. On je pogodan za dobivanje visokih iznosa gušenja aliasa [6], [7]. Međutim, glavni nedostatak je značajan propad amplitude u području propuštanja. U ovom radu za smanjenje ovih propada koriste se SCIC kompenzator.

1.1.2.1 Minimax dizajn SCIC filtra

Minimax kriterij temelji se na minimizaciji najveću apsolutne pogreške dane izrazom

$$\varepsilon(\boldsymbol{a}) = max_{\omega \epsilon \Omega} w(\omega) |H_{SCIC}(\omega, \boldsymbol{a}) - H_d(\omega)| \tag{17}$$

gdje su

- *w(ω)* težinska funkcija
- H_d željena amplitudna karakteristika
- $\boldsymbol{a} = [a_1 \ a_2 \ \dots \ a_M]^T$ vektor koji sadrži koeficijente polinoma za izoštravanje

Pri izračunu a, minimax dizajn uzima u obzir samo alias područja [7]. U tim područjima željena amplituda iznosi *Hd(ω) = 0*. Uz pretpostavku jedinične težinske funkcije, tj. *w(ω) = 1*, pogreška (18) poprima oblik

$$\varepsilon(\boldsymbol{a}) = max_{\omega\epsilon\Omega}|H_{SCIC}(\omega,\boldsymbol{a})| \tag{18}$$

Potrebno je uočiti kako funkcija ne uzima u obzir pojačanje filtra. Stoga je nužno funkciju pogreške normirati prema:

$$\varepsilon(\boldsymbol{a}) = max_{\omega\epsilon\Omega}\left|\frac{H_{SCIC}(\omega,\boldsymbol{a})}{H_{SCIC}(0,\boldsymbol{a})}\right| \tag{19}$$

Uvrštavanjem ω = 0 u (12) dobivamo:

$$H_{SCIC}(0,\boldsymbol{a}) = \sum_{m=1}^{M} a_m \tag{20}$$

Supstitucijom (12) i (21) u (20) dobivamo oblik konačan oblik apsolutne funkcije pogreške:

$$\varepsilon(\boldsymbol{a}) = max_{\omega\epsilon\Omega}\left|\frac{\sum_{m=1}^{M} a_m H_{CIC}^{m}(\omega)}{\sum_{m=1}^{M} a_m}\right| \tag{21}$$

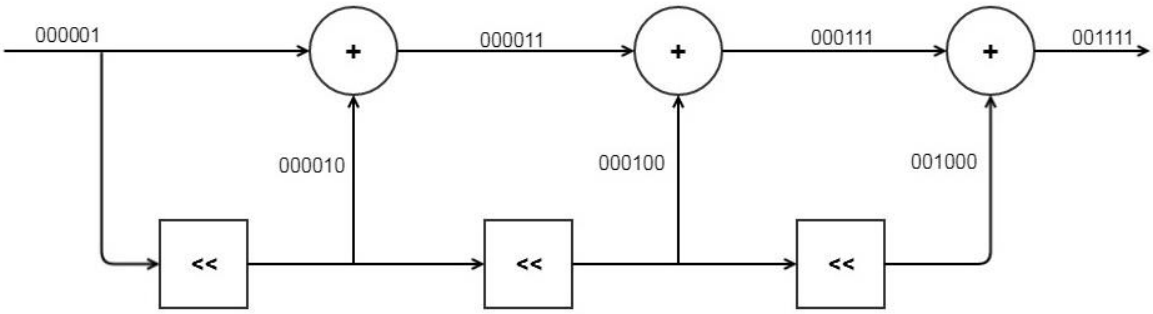
Optimalni koeficijenti se dobivaju rješavanjem sljedećeg minimizacijskog problema:

$$\hat{\mathrm{a}} = \arg min[\varepsilon(\boldsymbol{a})] \tag{22}$$

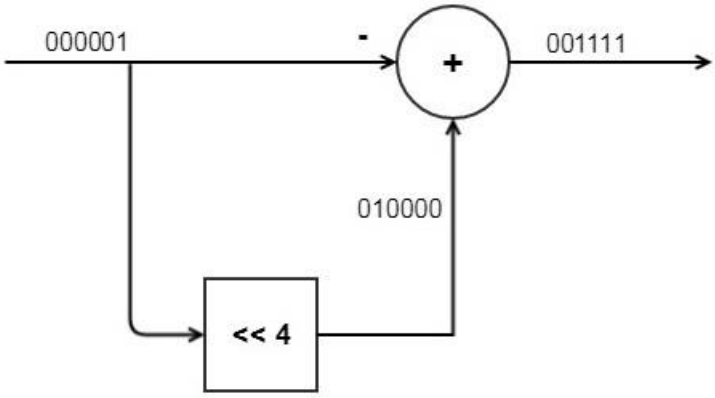
1.1.2.2 SPT koeficijenti

Važno je uočiti da ako koeficijenti polinoma a poprimaju realne vrijednosti tada se SCIC filtar realizira pomoću množila. Korištenje množila u strukturi značajno povećava složenost filtra. Izbjegavanje množila u strukturi može se postići upotrebom koeficijenata polinoma koji su izraženi kao sume potencije broja 2 [7] (eng. *Sum of Powers of Two, SPT*). Time se množila zamjenjuju posmacima i zbrajalima što rezultira jednostavnijom strukturom. Pojedinu potenciju broja dva u sumi nazivamo term. Daljnje poboljšanje u implementaciji ovisi o broju zbrajala u sklopovskoj implementaciji, odnosno o broju termova pojedinih kvantiziranih koeficijenata. Primjenom CSD (eng. *canonic signed digit*) reprezentacije minimizira se broj termova, što sklopovlje čini efikasnijim. Za razliku od 2. komplementa

koji koristi binarni sustav, CSD se reprezentira ternarnim sustavom (0, +1, -1). Sljedeći jednostavni primjer ilustrira razliku u realizaciji broja 15 s rezolucijom od 6 bitova:



Slika 5. Reprezentacija s dvojnim komplementom



Slika 6. CSD reprezentacija

Sa slika 5. i 6. vidi se kako je za CSD reprezentaciju datog primjera potrebno samo jedno zbrajalo, dok su za 2. komplement potrebna čak 3 zbrajala.

U ovom radu je korištena Matlab funkcija [7], koja pronalazi koeficijente polinomijalnog izoštravanja CIC filtra u SPT obliku koristeći opisani minimax kriterij. Deklaracija funkcije je oblika

```
function [a,K]=scic_spt_minimax01(N,R,M,wp,P)
```

gdje su N red filtra, R faktor decimacije, M red izoštrenog polinoma, ω_p granična frekvencija propuštanja, a P maksimalan broj bitova u reprezentaciji koeficijenta koji su različiti od 0. P se zadaje po svakom koeficijentu polinoma.

1.2. Kompenzatori za decimacijske filtre

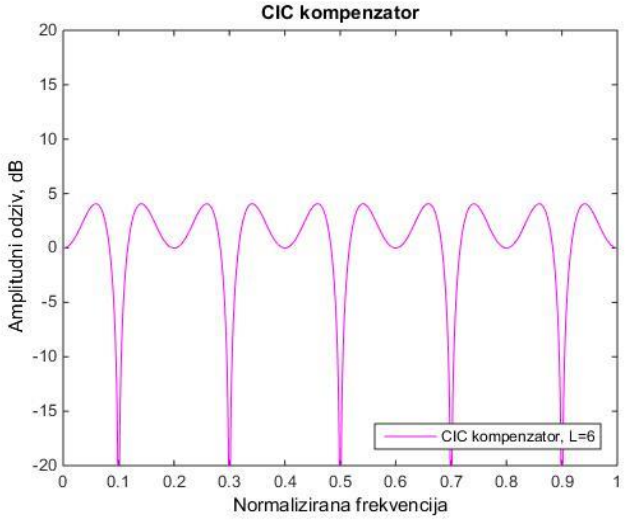
1.2.1. CIC kompenzatori

Kod mnogih primjena propad u području propuštanja CIC decimatora je prevelik i jedno od mogućih rješenja je u kaskadu dodati filtar koji ima inverznu karakteristiku od CIC decimatora. U idealnom slučaju formula je inverz CIC amplitudne karakteristike s nižom frekvencijom uzorkovanja [8], [9]:

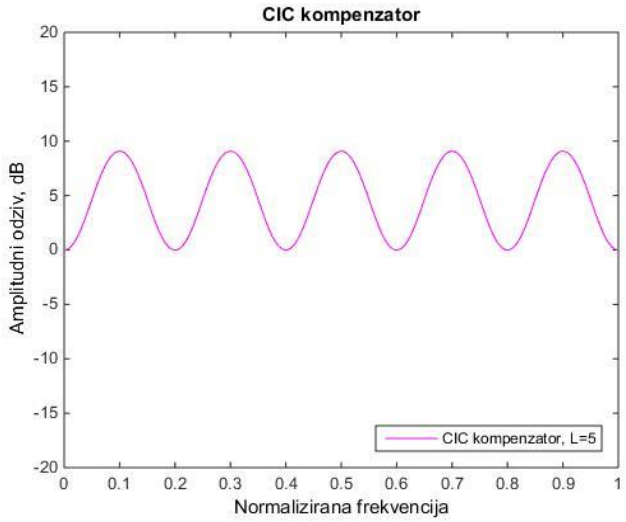
$$H_C(\omega) = \frac{1}{H_{CIC}\left(\frac{\omega}{R}\right)} = \left(R\frac{sin\left(\frac{\omega}{2R}\right)}{sin\left(\frac{\omega}{2}\right)}\right)^N \tag{23}$$

Idealnu karakteristiku nije moguće postići u realnom okruženju, već se ona aproksimira FIR filtrom.

Simetrični FIR filtri s parnim brojem koeficijenata [8] (slika 7.) imaju nulu na z = -1, stoga dolazi do velikog amplitudnog propada na |ω| = π. Kako bi se to izbjeglo, implementira se FIR filtar s neparnim brojem koeficijenata (slika 8.).



Slika 7. CIC kompenzator s parnim brojem koeficijenat



Slika 8. CIC kompenzator s neparnim brojem koeficijenata

1.2.2. SCIC kompenzatori

Slično razmatranje vrijedi i za SCIC kompenzatore. Kako bi se ispravila amplitudna karakteristika u području propuštanja potrebno je u kaskadu dodati filtar koji ima inverznu karakteristiku u odnosu na SCIC filtar na sniženoj frekvenciji uzorkovanja. Vrijedi:

$$H_{SC}(\omega) = \frac{1}{H_{SCIC}\left(\frac{\omega}{R}\right)} = \frac{1}{\sum_{m=1}^{M} a_m H_{CIC}^{m}\left(\frac{\omega}{R}\right)} = \frac{1}{\sum_{m=1}^{M} a_m \frac{1}{R}\frac{sin\left(\frac{\omega}{2}\right)}{sin\left(\frac{\omega}{2R}\right)}} \tag{24}$$

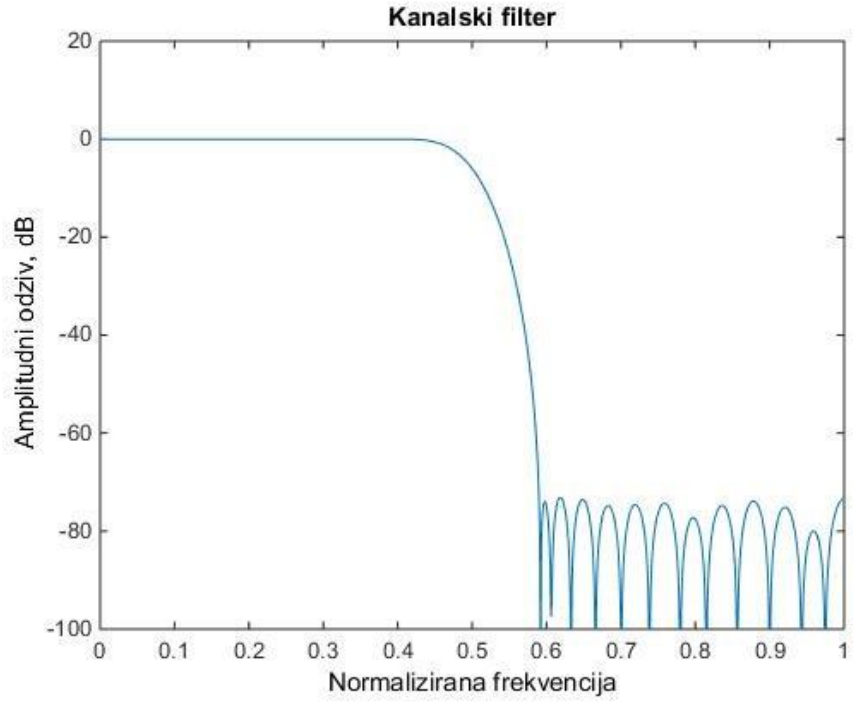
U radu je korištena funkcija koja pronalazi FIR koeficijente u SPT obliku [10] koji najbolje aproksimiraju idealnu karakteristiku SCIC kompenzatora. Deklaracija funkcije je oblika

```
function h=sciccompmf(N,R,a,L,CompType)
```

Argumenti funkcije su red izoštrenog filtra *N*, faktor decimacije *R*, koeficijenti izoštravanja *a*, broj koeficijenata izoštrenog CIC kompenzatora *L*. Argumentom *CompType* korisnik može odabrati želi li računanje FIR koeficijenata kao realnih brojeve ili kao suma potencija broja 2.

1.3. Kanalski filtar

Treća komponenta u kaskadi je filtar kanala. Njegova zadaća je osigurati dodatno gušenje izvan područja propuštanja. U Matlab modelu korišteni su FIR filtri iz rada [11]. Koeficijenti su tipa SPT, čime se osigurava optimizacija sustava prilikom sklopovske implementacije. Slika 9. je model amplitudne karakteristike kanalskog filtra 47. reda, granične frekvencije 0.4 π.



Slika 9. Amplitudna karakteristika kanalskog filtra

1.4. Amplitudna karakteristika DDC sustava

U ovom poglavlju slijedi analiza amplitudne karakteristike cijelog sustava napravljena u Matlabu. Glavne karakteristike koje se promatraju:

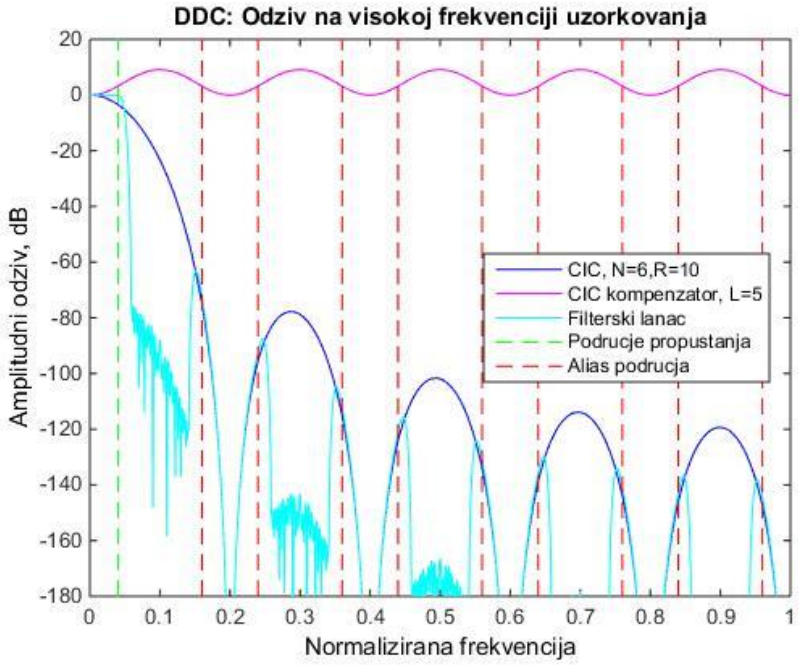
- Valovitost u području propuštanja (eng. *passband ripple*) je vrijednost između maksimalne i minimalne vrijednosti u području propuštanja
- Minimalno gušenje u alias područjima (eng. *minimum attenuation in folding bands*)

Cilj je postići što manju valovitost unutar područja propuštanja, te što veće gušenje unutar alias područja jer nakon decimacije, vrijednosti signala unutar tih područja, preslikavaju se kao smetnja u područje propuštanja (dolazi do aliasinga)

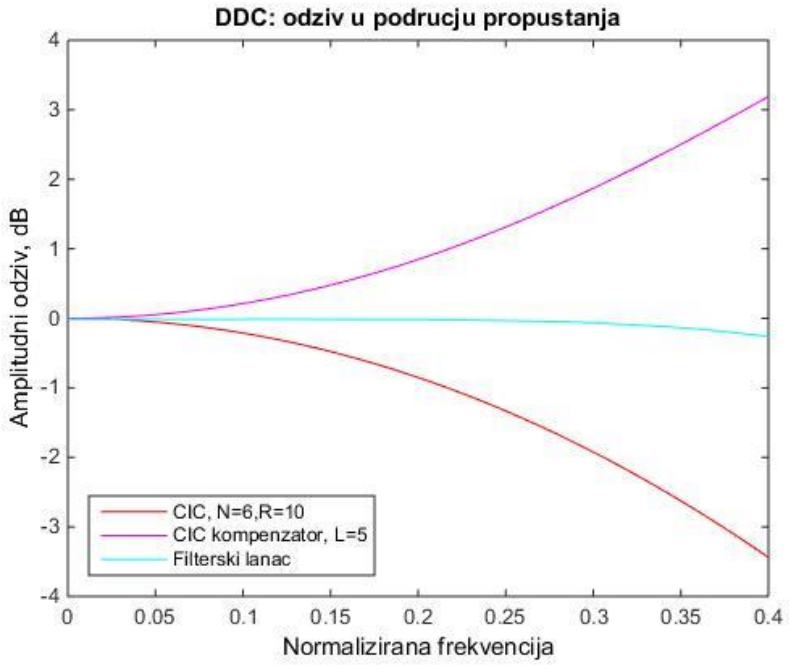
1.4.1. DDC sustav sa CIC decimatorom

Parametri sustava za digitalnu konverziju frekvencije uzorkovanja s graničnom frekvencijom područja propuštanja 0.4π s obzirom na izlaznu (nisku) frekvenciju uzorkovanja su:

- CIC decimator 6. reda, faktor decimacije R=10
- CIC kompenzator 5. reda sa SPT koeficijentima [8]
- Filtar kanala granične frekvencije 0.4π sa SPT koeficijentima



Slika 10. Amplitudna karakteristika DDC-a na visokoj frekvenciji uzorkovanja



Slika 11. Amplitudna karakteristika DDC-a u području propuštanja

Rezultat simulacije u Matlabu (slika 10., slika 11.) :

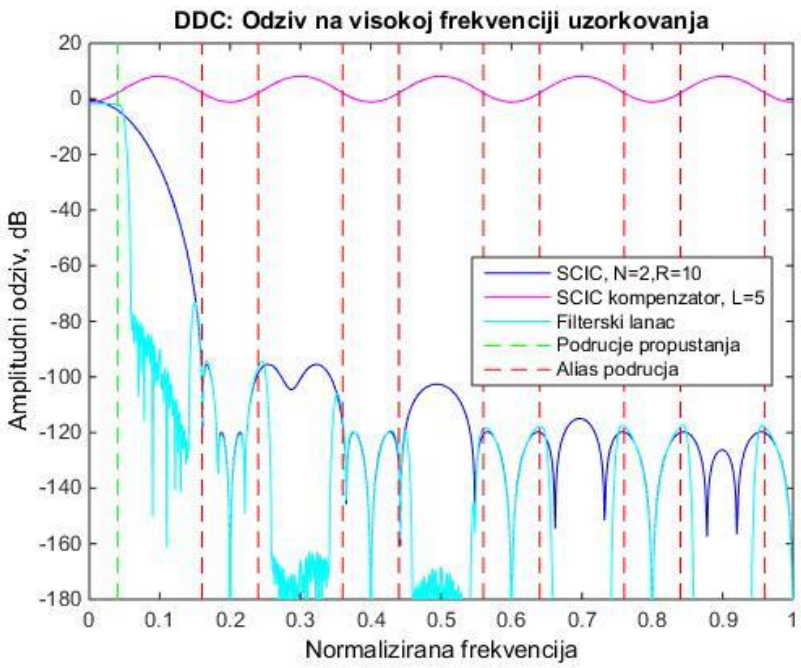
- Valovitost u području propuštanja prije kompenzacije iznosi 3.44 dB
- Valovitost u području propuštanja nakon CIC kompenzacije iznosi 0.25 dB
- Minimalno gušenje unutar alias područja je 75.17 dB

Dodavanjem CIC kompenzatora dolazi do značajnog smanjenja valovitosti u području propuštanja.

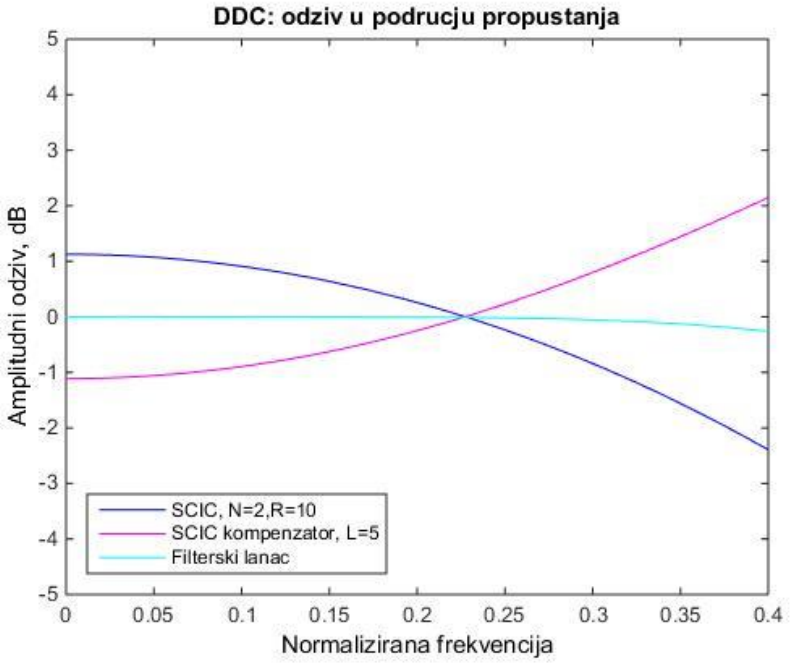
1.4.2. Primjer DDC sustava sa SCIC filtrom

Sljedeći primjer radi na istom području propuštanja (do 0.4π na niskoj frekvenciji uzoraka), ali s izoštrenim CIC filtrom. Cilj je usporediti dobivene rezultate s prethodnim primjerom. Parametri stoga moraju također biti usporedivi s prethodnim primjerom:

- Izoštreni CIC decimator 2. reda, polinom 3. stupnja , faktor decimacije R=10
- SCIC kompenzator 5. reda sa SPT koeficijentima [10]
- Filtar kanala granične frekvencije 0.4π sa SPT koeficijentima



Slika 12. Amplitudna karakteristika DDC-a na visokoj frekvenciji uzorkovanja



Slika 13. Amplitudna karakteristika DDC-a u području propuštanja

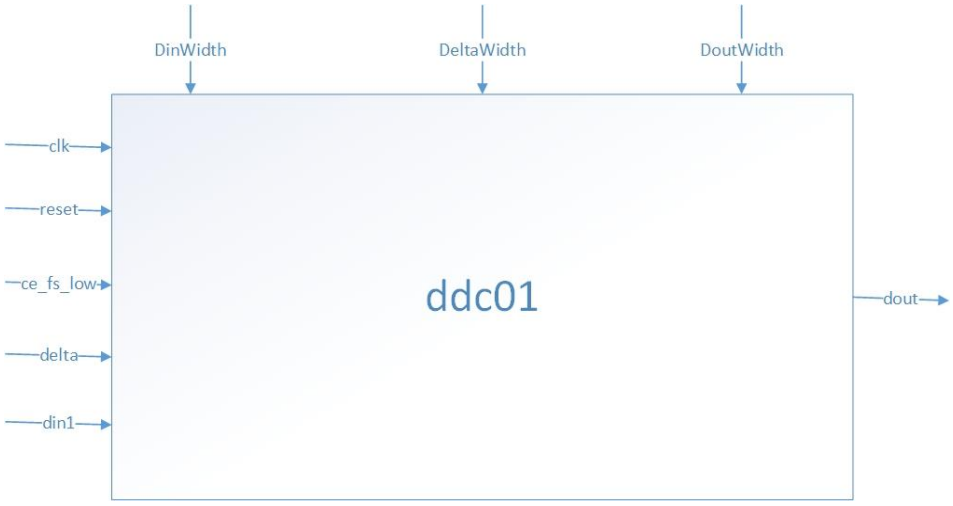
Dobiveni rezultati u Matlabu (slika 12., slika 13.):

- Valovitost u području propuštanja prije kompenzacije iznosi 3.52 dB
- Valovitost u području propuštanja nakon CIC kompenzacije iznosi 0.262 dB
- Minimalno gušenje unutar alias područja je 94.9 dB

Koristeći tehniku polinomijalnog izoštravanja CIC filtra, došlo je do neznatnog povećanja valovitosti u području propuštanja, dok je minimalno gušenje unutar alias područja povećano za skoro 20 dB.

2. RTL model digitalnog pretvarača frekvencije

RTL model DDC-a sadrži polinomijalno izoštreni CIC filtar prema slici 14. Nakon SCIC-a [12] slijedi u kaskadi spojen RTL model SCIC kompenzatora i kanalskog filtra. Ulazni podatak se množi sa sinusnim signalom koji generira numerički upravljan oscilator kako bi se signal spustio u osnovno područje (eng. *baseband*). Koeficijenti polinomijalnog izoštravanja kao i koeficijenti SCIC kompenzatora i kanalskog filtra su u SPT obliku kako bi se izbjegla implementacija množila u sustavu. Za realizaciju SPT koeficijenata korištena je CSD reprezentacija znamenki. RTL model i odgovarajuća ispitna okruženja pisana su u VHDL jeziku za opis sklopovlja.



Slika 14. Entitet DDC-a

Entitet RTL modela digitalne pretvorbe frekvencije uzorkovanja zove se ddc01 i prikazan je na slici 14. Sučelje je opisano u 1. i 2. tablici.

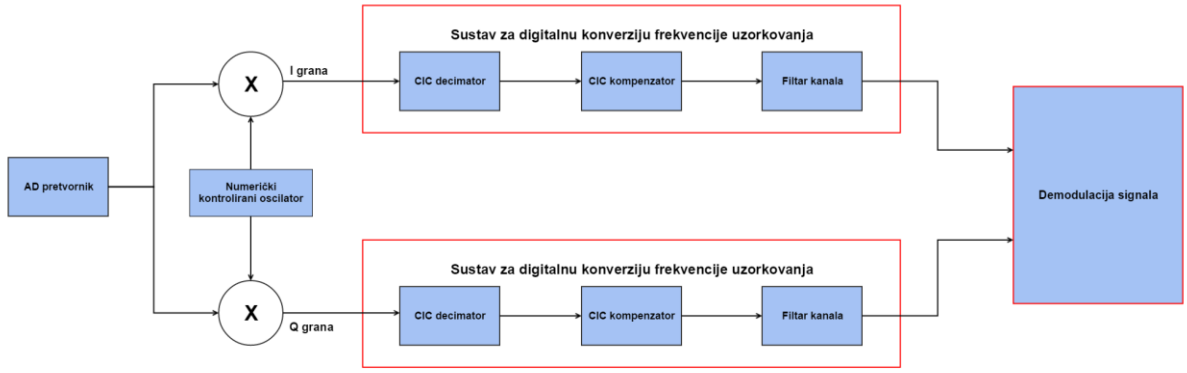
Tablica 1. Generičke konstante DDC-a

Generičke konstante	**Opis**
DinWidth	Duljina ulaznog podatka
DeltaWidth	Broj bitova signala delta
DoutWidth	Duljina izlaznog podatka

Tablica 2. Priključci DDC-a

Priključci entiteta	Opis
clk	Signal takta, odgovara visokoj frekvenciji uzorkovanja
reset	Sinkroni reset
ce_fs_low	Signal koji upravljanja radom SCIC filtra, SCIC kompenzatora i kanalskog filtra. Njegova frekvencija odgovara niskoj frekvenciji uzoraka
delta	Fazni ikrement, određuje frekveniciju sinusnog signala
din1	Ulazni podaci
dout	Izlazni podaci

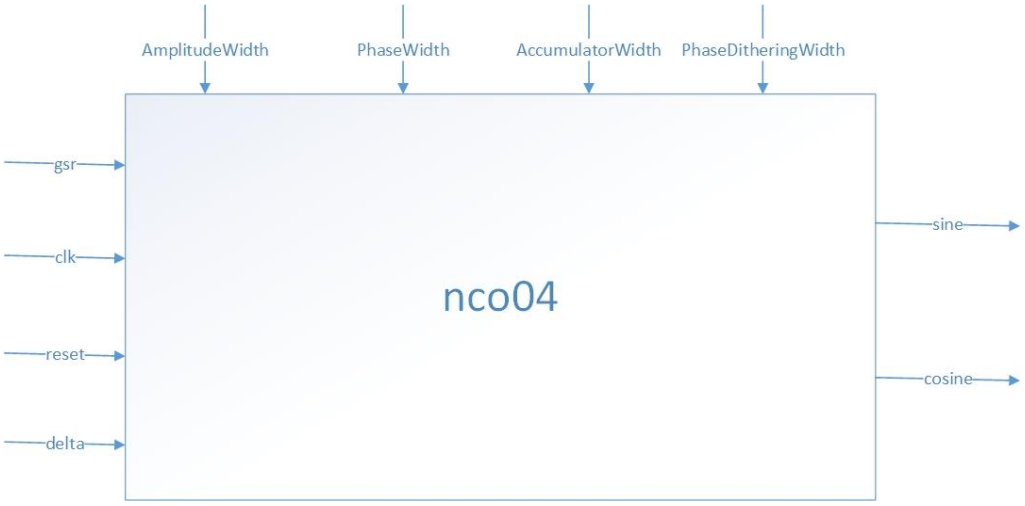
Strukturni opis sklopa za digitalnu pretvorbu frekvencije uzorkovanja sadrži komponenate koje su opisane u nastavku. Na slici 15. je primjer blok sheme unutar koje se nalazi sustav za digitalnu pretvorbu frekvencije uzorkovanja.



Slika 15. Blokovska shema digitalnog prijamnik

2.1. Numerički upravljan oscilator

Entitet RTL modela numerički upravljanog oscilatora (eng. *numerically controlled oscillator*, NCO) zove se nco04 [13] i prikazan je na 16. slici. Zadatak NCO-a je generiranje sinusnog signala na izlazu iz komponente, prema zadanoj karakteristici. Karakteristika sinusnog određuju ulazni parametri koji su opisani u 3. tablici.



Slika 16. Entitet numerički upravljanog oscilatora

Tablica 3. Priključci numerički upravljanog oscilatora

Generičke konstante	**Opis**
AmplitudeWidth	širina amplitude signala sine i cosine
PhaseWidth	širina faze signala sine i cosine
AccumulatorWidth	širina signala delta
PhaseDitheringWidth	širina faznog kvantizacijskog šuma koji sudjeluje u uklanjanju parazitne fazne modulacije
Priključci entiteta	**Opis**
gsr	asinkroni reset
clk	signal takta
reset	sinkroni reset
delta	fazni inkrement, određuje frekvenciju signala
sine	sinusni valni oblik
cosine	kosinusni valni oblik

2.2. SCIC decimator

2.2.1. Entitet SCIC decimatora

RTL model polinomijalno izoštrenog CIC filtra predstavlja implementaciju sa slike 4. Entitet SCIC decimatora zove se scic_decimator01 i prikazan je na 17. slici. Sučelje je opisano u tablicama 4. i 5.

Slika 17. Entitet SCIC-a

Tablica 4. Generičke konstante SCIC-a

Generičke konstante	Opis
N	Red CIC filtra
R	Faktor decimacije
M	Red polinoma
DinWidth	Duljina ulaznog podatka
DoutWidth	Duljina izlaznog podatka
RegWidth	Širina registara integratora i comb sekcija

Tablica 5. Priključci SCIC-a

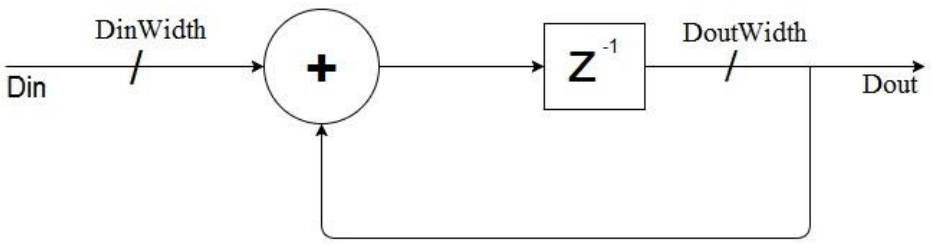
Priključci entiteta	Opis
gsr	asinkroni reset
clk	signal takta
reset	sinkroni reset
ce_fs_low	signal čija frekvencija odgovara niskoj frekvenciji uzorkovanja
din	podaci na ulazu u SCIC
dout	podaci na izlazu iz SCIC-a

2.2.2. Strukturni opis SCIC decimatora

Unutar RTL modela polinomijalno izoštrenog CIC filtra nalaze se sljedeće komponente: integrator, blok za kašnjenje, izbacivač uzoraka, množilo s konstantom, zbrajalo, comb sekcija i izlazni registar. U nastavku slijedi opis značajnijih dijelova [12].

- **Integrator**

Blok shema integratora prikazana je na slici 18. prema formuli (14). Kako bi se osigurala maksimalna frekvencija takta, signal se vodi s izlaza registra, a ne direktno iz zbrajala. Broj integratorskih sekcija ovisi o redu filtra N. Entitet RTL modela integratora zove se integrator01.



Slika 18. Blok shema integratora s registrom za kašnjenje

- **Blok za kašnjenje**

RTL model bloka za kašnjenje izveden je kao kaskada više registara. Iznos kašnjenja se definira podešavanjem generičke konstante *DELAY*. Entitet RTL modela bloka za kašnjenje zove se delay_block01.

- **Izbacivač uzoraka**

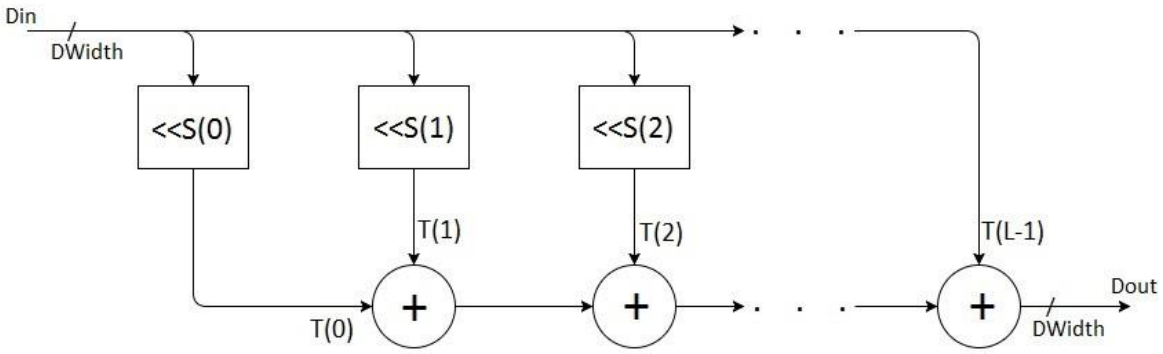
Izbacivač uzoraka (eng. *downsampler*) izveden je kao registar s omogućenim upisom. Broj izbacivača uzoraka unutar SCIC-a jednak je stupnju polinoma *M*. On je programiran tako da propusta samo signale na rastući brid *clk* signala te kad je ulazni signal *ce_fs_low* u visokom stanju. Signal *ce_fs_low* odgovara niskoj frekvenciji uzorkovanja i njegovo trajanje je odgovara jednoj periodi signala takta. Izbacivač uzoraka ne mijenja duljinu ulaznog podatka.

- **Množilo s konstantom**

Kako bi se izbjegla upotreba množila unutar sustava, implementiran je model u kojem je množenje realizirano korištenjem zbrajala i logičkih posmaka ulijevo. Logički posmak nije izveden kao zasebna komponenta unutar bloka za množenje, već se za njegovu realizaciju koristi drukčije ožičenje na sabirnici. Izvedba množila prikazana je na 19. slici. Izlazni podatak D_{out} je:

$$D_{out} = a_k D_{in} \tag{25}$$

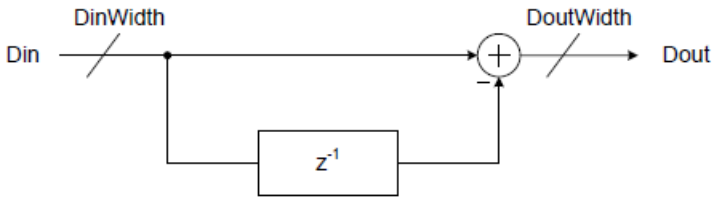
Gdje D_{in} predstavlja ulazni podatak, a a_k cjelobrojni koeficijent. Ulazni i izlazni podatak su iste duljine, a koeficijent a_k je zapisan u kanonskom (eng. *canonic signed digit*, CSD) obliku. U modelu množila, svaki CSD zapis koeficijenta je opisan vektorima *S* i *T*. Vektor *S* sadrži pozicije znamenki te tako određuje za koliko se mjesta obavlja posmak, a vektor *T* određuje njihov predznak. Entitet RTL modela množila s konstantom zove se multiplier_ak01.



Slika 19. Blok shema integratora s registrom za kašnjenje

- **Comb sekcija**

Comb sekcija sklopa (slika 20.) radi na sniženoj frekvenciji uzorkovanja (na rastući brid takta i kad je ulazni signal *cl_fs_low* u stanju 1). Izlazni podatak D_{out} mora biti manji ili iste duljine kao ulazni podatak D_{in}. Entitet RTL modela zove se comb04.



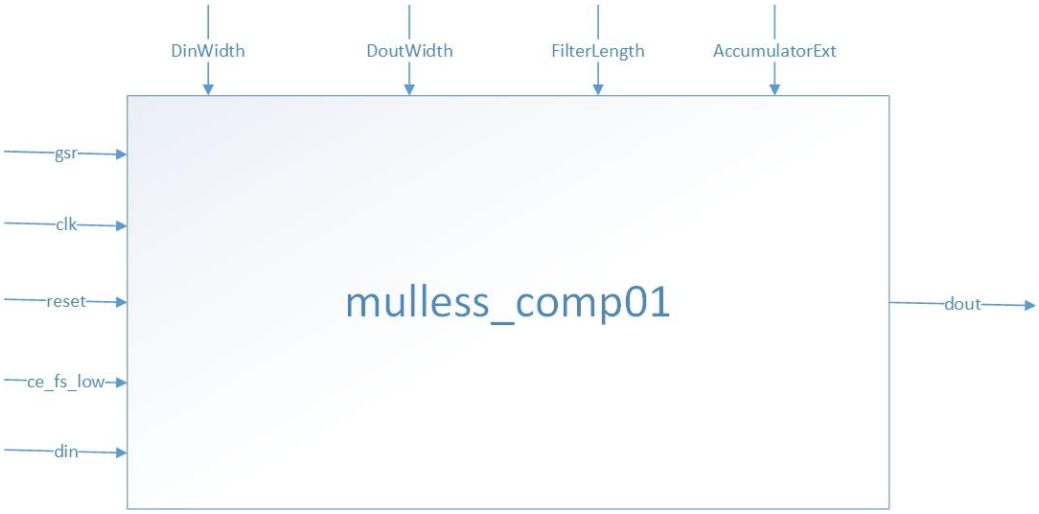
Slika 20. Blok shema comb sekcije

2.3. FIR filtar

2.3.1. Entitet FIR filtra

SCIC kompenzator i kanalski filtar su implementirani kao FIR filtri, te stoga slijedi općeniti opis FIR filtra, a implementacija je gotovo ista (razlika je u funkcijama koje računaju s koeficijentima FIR filtra).

Entiteti FIR filtara zove se mulless_comp01 (slika 21.) i mulless_fir01. Njihovo sučelje je identično te je opisano u 6. tablici.



Slika 21. Entitet SCIC kompenzatora

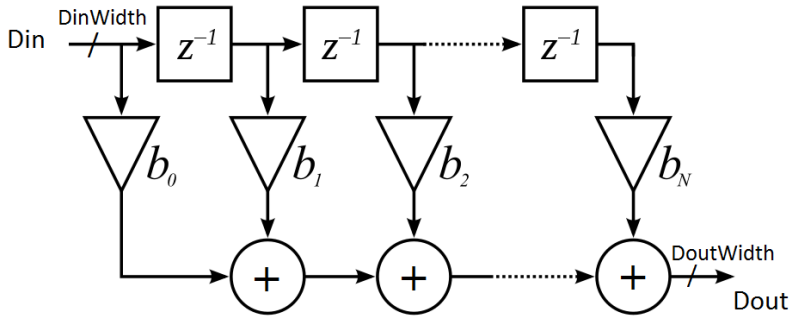
Tablica 6. Priključci FIR filtra

Generičke konstante	**Opis**
DinWidth	duljina ulaznog podatka
DoutWidth	duljina izlaznog podatka
N	red FIR filtra
AccumulatorExt	broj bitova akumulatorske ekstenzije
Priključci entiteta	**Opis**
gsr	asinkroni reset
clk	signal takta
reset	sinkroni reset
ce_fs_low	signal čija frekvencija odgovara niskoj frekvenciji uzorkovanja
din	ulazni podatak
dout	izlazni podatak

2.3.2. Strukturni opis FIR filtra

RTL model FIR filtra predstavlja implementaciju sa slike 22. prema formuli:

$$y(n) = b_0 x(n) + b_1 x(n-1) + \cdots + b_N x(n-N) = \sum_{i=0}^{N} b_i\, x(n-i) \tag{26}$$



Slika 22. Blokovska shema FIR filtra

Unutar RTL modela FIR filtra nalaze se sljedeće komponente: blok za kašnjenje, množilo s konstantom, zbrajalo i izlazni registar. Ovisno o redu filtra *N*, unutar sustava je generiran odgovarajući broj komponenti koje su međusobno spojene. Slično kao i kod polinomijalno izoštrenog CIC filtra, koeficijenti b_i su realizirani u SPT formatu, kako bi se izbjegla sklopovska implementacija množila. Zbog uzastopnog zbrajanja mogućnost preljeva se sprječava proširenjem ulaznog podatka *Din* za iznos prema formuli:

$$regWidth = Dinwidth + accExt \tag{27}$$

gdje *Dinwidth* predstavlja širinu ulaznog podatka, a akumulatorsko proširenje se računa prema sljedećoj formuli:

$$accExt = log_2\left(\sum_{i=0}^{N} |h_i|\right) \tag{28}$$

Gdje su h_i koeficijenti impulsnog odziva. Izlazni registar skraćuje podatke na širinu *DoutWidth* te ih propušta na rastući brid takta kad je signal *ce_fs_low* u visokom stanju.

2.4. Konfiguracija DDC-a

Korisnik definira parametre sustava unutar paketa za inicijalizaciju ddc_package01. Sustav za digitalnu pretvorbu frekvencije ima varijabilne sljedeće vrijednosti:

- Parametri za namještanje NCO komponente
- Parametri za namještanje SCIC komponente
- Parametri za namještanje SCIC kompenzatora
- Parametri za namještanje kanalskog filtra

Nakon što je decimator konfiguriran može se pokrenuti njegova sinteza.

2.5. Resursi u FPGA

U 7. i 8. tablici su uneseni podaci o zauzetosti FPGA resursa nakon sinteze ddc01 sklopa. Unutar ddc01 je implementirano ugrađeno množilo DSP48E1 koje množi ulazni podatak sa sinusnim signalom generiranim u nco04 te tako transponira signal u osnovno područje (eng. *baseband*). Može se primijetiti kako kanalski filtar zauzima najviše resursa što se može objasniti velikim brojem koeficijenata ($N = 46$).

Tablica 7. FPGA resursi za cijeli sklop

Resurs	**Iskorišteno**	**Dostupno**	**Zauzetost %**
LUT	2125	41000	5.18
Bistabil	1414	82000	1.72
BRAM	9	135	6.67
DSP	1	240	0.42

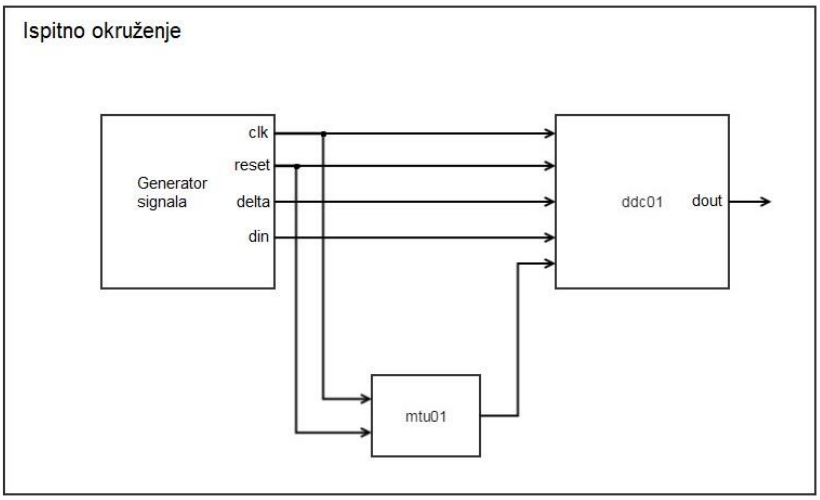
Tablica 8. Zauzetost resursa po komponentama unutar ddc01

Komponenta	**NCO**	**iq_mixer**	**SCIC**	**Kanalski filter**
LUT	103	46	581	1368
Bistabil	96	14	568	752
BRAM	9	0	0	0

3. Verifikacija digitalnog pretvarača frekvencije

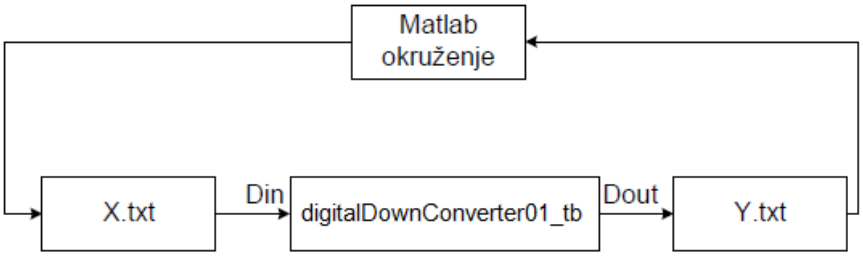
3.1. Simulacija

Za verifikaciju modela ddc01 koristi se Matlab skripta ddc_fpga01 koja kao pobudu generira uzdignuti kosinusni puls (eng. *raised-cosine pulse*). Skripta generira uzorke pobude i upisuje ih u tekstualnu datoteku pobuda.txt. U Vivadu je napisano ispitno okruženje digitalDownConverter_tb.vhd (slika 23.) unutar kojeg se nalazi komponenta mtu01 (generira signal niske frekvencije *ce_fs_low*) i model ddc01 koji se ispituje.



Slika 23. Ispitno okruženje digitalDownConverter01_tb

Nakon provedene simulacije u Vivadu, simulator generira tekstualnu datoteku izlaz_ddc.txt s uzorcima odziva. Ta datoteka se učitava u istu Matlab skriptu ddc_fpga01 gdje se analizira rezultat. Na slici 24. nalazi se blok shema verifikacijskog modela.



.

Slika 24. Blok shema verifikacijskog modela

Primjer simulacije:

Parametri SCIC decimatora se postavljaju u Matlabu:

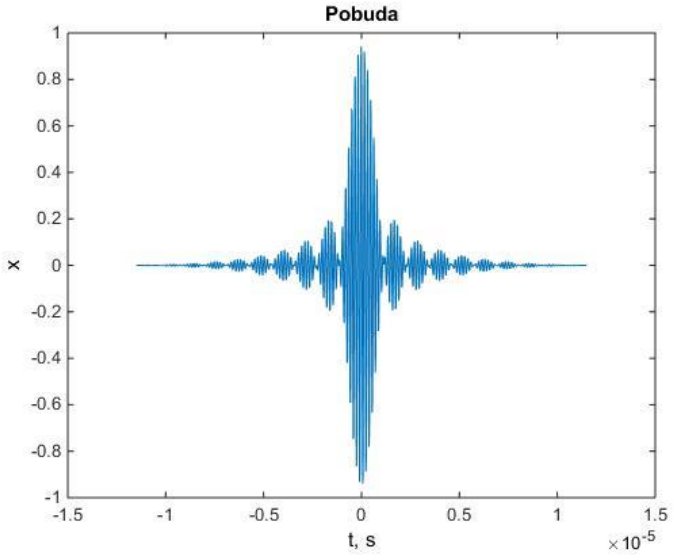
N = 2; R = 10; M = 3; alfa = 1; $B_a = 12$; $B_{in} = 14$; $B_{out} = 16$;

te se poziva funkcija scicdec_regwidth01 koja ispisuje sadržaj koji je potrebno unijeti u ddc_package01. Koeficijentom *alfa* se modificira odrezivanje registara, B_a određuje za koliko se bitova podatak treba pomaknut kako bi bio cjelobrojnog tipa, B_{in} je ulazna širina registra, B_{out} izlazna širina registra.

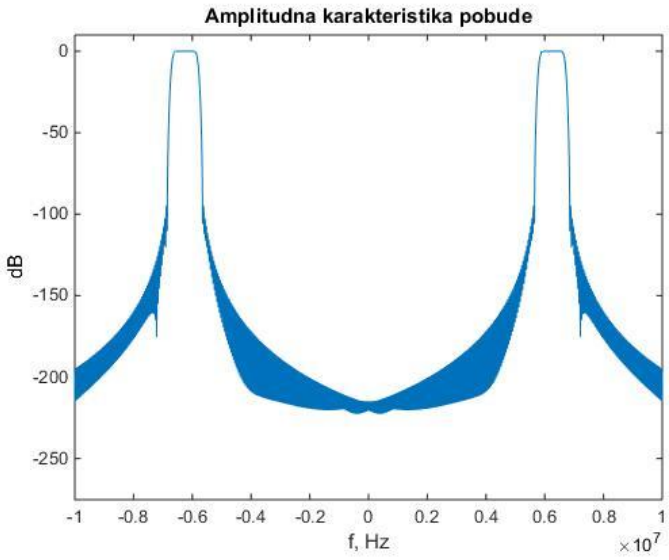
U paket također je potrebno upisati parametre za FIR filtre (SCIC kompenzator, kanalski filter). To su red filtra, koeficijenti u CSD formatu te njihova rezolucija, širina registrara na ulazu,izlazu i unutar filtra.

N = 47; FIR_coeff_res = 11; FIR_DataWidth = 16; FIR_RegWidth = 29;

Za verifikaciju modela, u Matlabu je generirana pobuda izdignutog kosinusnog pulsa granične frekvencije *fc* = 500kHz, frekvencije uzorkovanja f_{sH} = 50 MHz. Pobuda je transponirana množenjem sa sinusnim signalom na frekvenciju f_0 = 6.25 MHz. Karakteristika pobude u vremenskoj i frekvencijskoj domeni prikazana je na slikama 25. i 26.

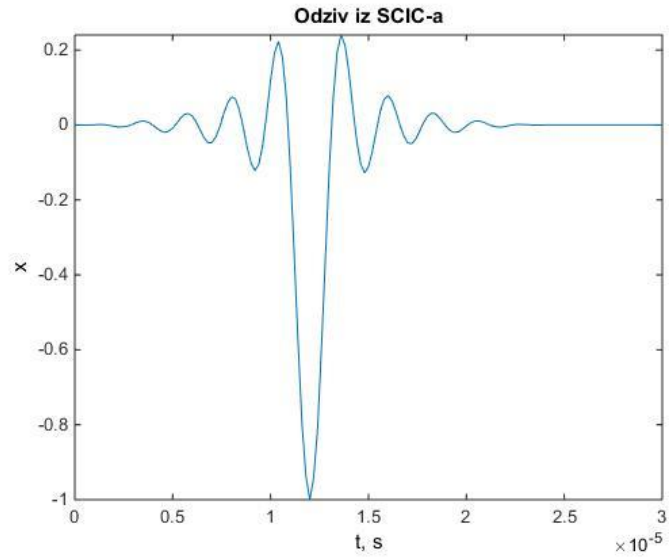


Slika 25. Valni oblik pobude

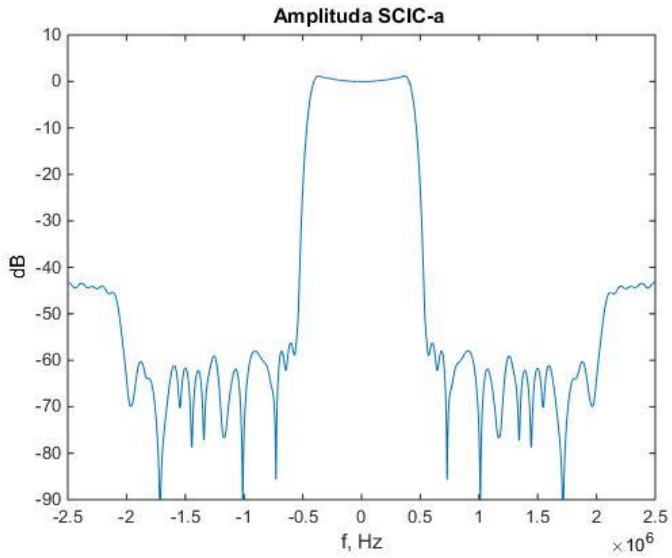


Slika 26. Amplitudni oblik pobude

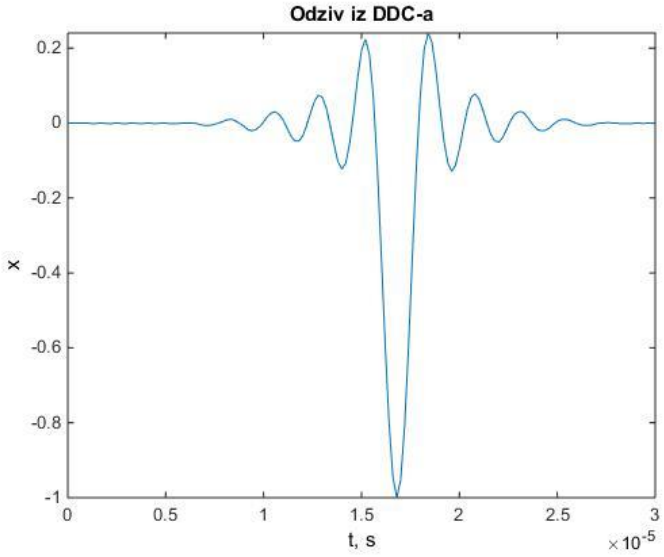
Nakon zadavanja parametara, pokreće se VHDL simulacija (digitalDownConverter01_tb) koja u tekstualnu datoteku ispisuje izlazne podatke u cjelobrojnom obliku iz SCIC-a, SCIC kompenzatora i kanalskog filtra. Pokretanjem Matlab skripta ddc_fpga01 čitaju se ti podaci te se crtaju odzivi iz traženih komponenti. Na slikama 27. i 28. je prikazan odziv SCIC decimatora, dok je na slikama 29. i 30. prikazan izlaz iz kanalskog filtra (izlaz iz DDC-a). Prikazana je vremenska i frekvencijska domena.



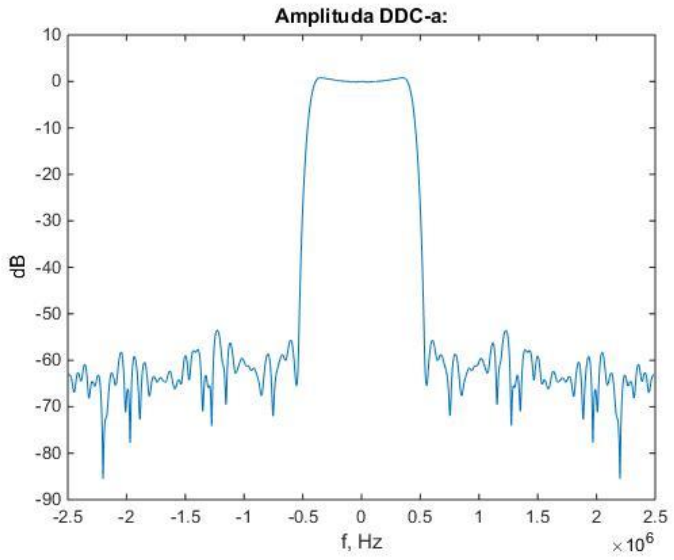
Slika 27. Valni oblik izlaza iz SCIC-a



Slika 28. Amplitudni oblik izlaza iz SCIC-a



Slika 29. Valni oblik izlaza iz DDC-a

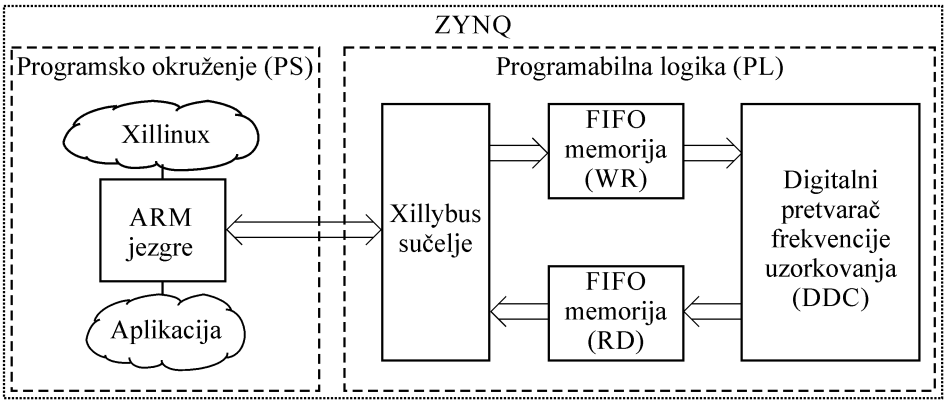


Slika 30. Amplidudni oblik izlaza iz DDC-a

3.2. Implementacija digitalnog pretvarača frekvencije na stvarnom sklopovlju

Zynq je familija SoC (eng. *system on chip*) uređaja koje razvija firma Xilinx (slika 31.). U ovom radu je opisan Zyqn-7000 SoC koji u sebi ima integriran procesorski sustav (2. ARM Cortex- A9 jezgre) i Xilinx programsku logiku u FPGA. Na ZedBoard (razvojna pločica zasnovana na Zynq-7000) je implementiran Xillinux operacijski sustav koji je distribucija Linux verzije za razvojne pločice.

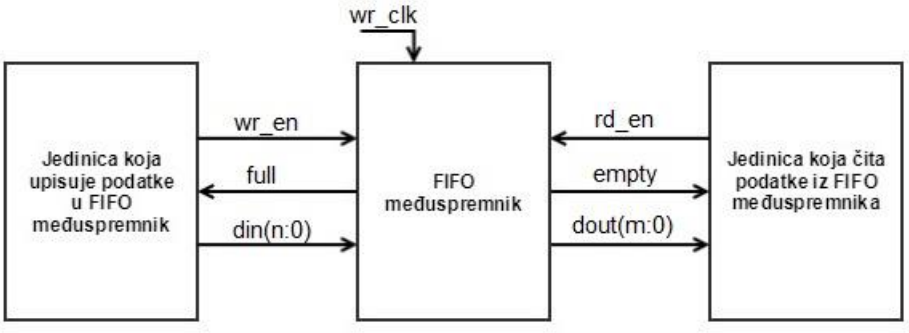
Xillybus je IP jezgra i upravljački program za Xillinux koji omogućuje jednostavnu komunikaciju između FPGA dijela i programskog okruženja. Budući da zahtjevi za komunikaciju između raznih FPGA projekata mnogo variraju, ne postoji jedinstvena konfiuracija prema dijelu koji razvija FPGA dizajner [14].



Slika 31. Blok shema Zynq sustava

Na toj strani se implementiraju dva FIFO međuspremnika za komunikaciju između Xillybusa i izvedene logike. Parametrima tih međuspremnika kao sto su dubina, frekvencija takta, upravlja FPGA dizajner koji ih prilagođava potrebama svog dizajna. Unutar Xilinx-ovog repozitorija IP jezgri nalazi se generički FIFO međuspremnik naziva FIFO generator v13.1 [15] gdje su obvezni priključci (slika 32.):

- wr_enable: Ako je signal u visokoj razini, omogućeno je upisivanje u FIFO međuspremnika
- full: visoka razina označava da je FIFO popunjen i da ne može primati nove podatke
- din(n:0): ulazni podatak
- rd_en: visoka razina omogućuje čitanje podataka iz FIFO međuspremnika
- empty: visoka razina signala javlja da je FIFO međuspremnik prazan
- dout(m:0): izlazni podatak
- wr_clk: takt upisa u FIFO međuspremnik



Slika 32. Blok shema jednostavnog sustava za komunikaciju s FIFO međuspremnikom

Osim obveznih pinova, FIFO međuspremnik može imati dodatne ulaze/izlaze kao što su:

- almost_full: visoka razina označava da je spremnik skoro popunjen i da treba prestati sa slanjem podataka (ukoliko se čeka da se aktivira full signal, već je došlo do gubitka ulazne informacije).
- almost_empty: visoka razina označava da je spremnik skoro prazan. Njegova prednost je što omogućuje jedinici koja čita iz FIFO međuspremnika da na vrijeme prekine s čitanjem.

4. Zaključak

Sustav za digitalnu konverziju frekvencije uzorkovanja (DDC) predstavlja jedan od ključnih dijelova svakog programski izvedenog radio prijemnika, te je stoga bitno da bude efikasno izveden u sklopovlju. DDC se sastoji od tri filtra spojenih u kaskadu: SCIC filtra, SCIC kompenzatora i kanalskog filtra. Jedan od načina povećanja efikasnosti i brzine rada sklopovlja je dizajniranje sustava bez množila, koji se oslanja isključivo na zbrajala. CIC filtar nema množila, ali zbog loših amplitudnih svojstava (propad u području propuštanja), potrebna su ispravljanja: tehnikom izoštravanja i dodavanjem simetričnog FIR filtra (SCIC kompenzator). Kako bi se izbjegla upotreba množitelja, koeficijenti polinoma izoštravanja i SCIC kompenzatora su implementirani u SPT obliku. U napravljenom Matlab modelu SCIC kompenzator ispravlja propad i valovitost u području propuštanja, a tehnika izoštravanja povećava minimalno gušenje zabranjenih područja. RTL model sustava za digitalnu konverziju frekvencije uzorkovanja razvijen je u VHDL jeziku za opis sklopovlja. Razvijeno je odgovarajuće ispitno okruženje i provedena je verifikacija rada.

5. Literatura

[1] E. B. Hogenauer, “An economical class of digital filters for decimation and interpolation,” IEEE Trans. Acoust., Speech, Signal Process., vol. ASSP-29, no. 2, pp. 155–162, Apr. 1981.

[2] M. Vucic, Obrada signala u komunikacijama, interna skripta, FER-ZESOI, Zagreb, 2009.

[3] J. Kaiser and R. Hamming, “Sharpening the response of a symmetric nonrecursive filter by multiple use of the same filter,” IEEE Trans. Acoust., Speech, Signal Process., vol. 25, no. 5, pp. 415–422, Oct. 1977.

[4] G. Molnar, M. Glavinic Pecotic, and M. Vucic, “Weighted least-squares design of sharpened CIC filters,” in Proc. Int. Conf. MIPRO, vol. MEET, Opatija, Croatia, May 2013, pp. 104–108.

[5] G. Molnar and M. Vucic, “Weighted minimax design of sharpened CIC filters,” in Proc. IEEE Int. Conf. Electron. Circuits Syst., Abu Dhabi, UAE, Dec. 2013, pp. 869–872.

[6] J. O. Coleman, “Chebyshev stopbands for CIC decimation filters and CIC-implemented array tapers in 1D and 2D,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 59, no. 12, pp. 2956–2968, Dec. 2012.

[7] G. Molnar, A. Dudarin, and M. Vucic, “Minimax design of multiplierless sharpened CIC filters based on interval analysis,” in Proc. Int. Conf. MIPRO, vol. MEET, Opatija, Croatia, 2016, pp. 94–98.

[8] G. Molnar and M. Vucic, “Closed-form design of CIC compensators based on maximally flat error criterion,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 58, no. 12, pp. 926–930, Dec. 2011.

[9] A. Fernandez-Vazquez and G. Jovanovic Dolecek, “Maximally flat CIC compensation filter: design and multiplierless implementation,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 59, no. 2, pp. 113–117, Feb. 2012.

[10] G. Molnar, A. Dudarin and M. Vucic, “Design and Multiplierless Realization of Maximally Flat Sharpened-CIC Compensators”, IEEE Trans. on Circuits and Sys. II, Exp. Briefs, vol. PP, Issue 99, pp. 1-1, May 2017.

[11] Kwentus, A. and Willson, A., Jr., “Application of filter sharpening to Cascaded Integrator-Comb Decimation Filters,” IEEE Trans. on Signal Processing, vol.45, No.2, February 1997, pp.457-467.

[12] Ivica Dvoršćak, FPGA implementacija CIC filtra s poboljšanom selektivnošću, Diplomski rad, ZESOI, Fakultet elektrotehnike i računarstva, Zagreb, 2012.

[13] Goran Molnar, FPGA implementacija numerički upravljanog oscilatora, Fakultet elektrotehnike i računarstva, Zagreb, 2008.

[14] www.kernel.org/doc/Documentation/xillybus.txt

[15] www.xilinx.com/support/documentation/ip_documentation/fifo_generator/v13_0/pg057-fifo-generator.pdf

Sažetak

Naslov: FPGA implementacija digitalnog pretvarača frekvencije bez upotrebe množila

U ovom radu opisana je kratka teoretska pozadina o sustavu za digitalnu konverziju frekvencije uzorkovanja (eng. Digital down converter, DDC). Sustav se sastoji od kaskade: polinomijalno izoštrenog CIC decimatora (SCIC), SCIC kompenzatora i filtra kanala. Najčešća primjena DDC sustava je u programski definiranom radiju. Napravljen je model sustava u Matlabu te su uspoređena dva ekvivalentna primjera DDC-a: s i bez polinomijalnog izoštravanja. Napravljen je RTL model u VHDL i odgovarajuće ispitno okruženje gdje je model ispitan.

Ključne riječi: DDC, CIC, SCIC, SCIC kompenzator, kanalski filter, FPGA

Summary

Title: FPGA Implementation of Multiplierless Digital Down Converter

In this paper it is described a short theoretical background of the Digital Down Converter (DDC) system. The system consists of a cascade of: polynomial sharpened CIC decimator (SCIC), SCIC compensator and channel filter. The most common application of DDC is in program-defined radio. A model was developed in Matlab and two equivalent examples of DDC were compared: with and without polynomial sharpening. An RTL model was created in VHDL. Also test environment was created where the model was tested.

Keyword: DDC, CIC, SCIC, SCIC compensator, channel filter, FPGA

Dodatak

dc01.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
library UNISIM;
use UNISIM.VComponents.all;

library work;
use work.ddc_package01.all;


entity ddc01 is
    generic
    (
        -- Wordlength of input data
        DinWidth:        integer := 12;
        -- NCO frequency
        DeltaWidth:      integer := 32;
        -- Wordlength of output data
        DoutWidth:       integer := 16
    );
    port
    (
        clk:         in  std_logic;
        reset:       in  std_logic;
        ce_fs_low:   in  std_logic;
        delta:       in  std_logic_vector(DeltaWidth-1 downto 0);
        din1:        in  std_logic_vector(DinWidth-1 downto 0);
        dout:        out std_logic_vector(DoutWidth-1 downto 0);
    );
end ddc01;


architecture RTL of ddc01 is

-- ============================================================================
--                              Local components
-- ============================================================================
    component nco04
    generic
    (
        -- Amplitude width < 35
        AmplitudeWidth:         positive := NCO_AmplitudeWidth;
        -- Phase width
        PhaseWidth:             positive := NCO_PhaseWidth;
        -- Accumulator width, AccumulatorWidth >> PhaseWidth
        AccumulatorWidth:       positive := NCO_AccumulatorWidth;
        -- Phase dithering width,
        -- PhaseDitheringWidth >= AccumulatorWidth - PhaseWidth
```

```
        PhaseDitheringWidth:    positive := NCO_PhaseDitheringWidth
    );
    port
    (
        gsr:    in  std_logic;
        clk:    in  std_logic;
        reset:  in  std_logic;
        delta:  in  std_logic_vector(AccumulatorWidth-1 downto 0);
        sine:   out std_logic_vector(AmplitudeWidth-1 downto 0);
        cosine: out std_logic_vector(AmplitudeWidth-1 downto 0)
    );
    end component;

   component iq_mixer03 is
    generic(
            -- DWidth <= 18
            Sig_DinWidth : integer := IQ_Sig_DinWidth;
            Osc_DinWidth : integer := IQ_Osc_DinWidth;
            DoutWidth    : integer := IQ_DoutWidth
            );
    port(
         clk    : in  std_logic;
         reset  : in  std_logic;
         gsr    : in  std_logic;
         Din    : in  std_logic_vector(Sig_DinWidth-1 downto 0);
         sine   : in  std_logic_vector(Osc_DinWidth-1 downto 0);
         cosine : in  std_logic_vector(Osc_DinWidth-1 downto 0);
         I      : out std_logic_vector(DoutWidth-1 downto 0);
         Q      : out std_logic_vector(DoutWidth-1 downto 0)
         );
    end component;

    component scic_decimator01
    generic
    (
        -- Order of classic CIC filter
        N:          integer := SCIC_N;
        -- Sample rate conversion factor
        R:          integer := SCIC_R;
        -- Sharpening polynomial order
        M:          integer := SCIC_Poly'right;
        -- Length of the input data Din
        DinWidth:   integer := SCIC_DinWidth;
        -- Length of the output data Dout
        DoutWidth:  integer := SCIC_DoutWidth;
        -- Register widths of sharpened CIC filter
        RegWidth:   integer_vector(1 to 2*SCIC_Poly'right*SCIC_N):=
SCIC_RegWidth
    );
    port
    (
        gsr       : in  std_logic;
        clk       : in  std_logic;
        reset     : in  std_logic;
        ce_fs_low : in  std_logic;
        Din       : in  std_logic_vector (DinWidth-1 downto 0);
        Dout      : out std_logic_vector (DoutWidth-1 downto 0)
    );
    end component;
```

```
    component mulless_comp01 is
    generic
    (
        DinWidth:          integer := 12;
        DoutWidth:         integer := 14;
        N:                 integer := 21
    );
    port
    (
        gsr         : in  std_logic;
        clk         : in  std_logic;
        reset       : in  std_logic;
        ce_fs_low   : in  std_logic;
        din         : in  std_logic_vector(DinWidth-1 downto 0);
        dout        : out std_logic_vector(DoutWidth-1 downto 0)
    );
    end component;

    component mulless_fir01 is
    generic
    (
        DinWidth:          integer := 12;
        DoutWidth:         integer := 15;
        N:                 integer := 21
    );
    port
    (
        gsr         : in  std_logic;
        clk         : in  std_logic;
        reset       : in  std_logic;
        ce_fs_low   : in  std_logic;
        din         : in  std_logic_vector(DinWidth-1 downto 0);
        dout        : out std_logic_vector(DoutWidth-1 downto 0)
    );
    end component;

-- ============================================================================
--                             Local signals
-- ============================================================================

    signal sine:          std_logic_vector(NCO_AmplitudeWidth-1 downto 0);
    signal mix_din1:      std_logic_vector(IQ_Sig_DinWidth-1 downto 0);
    signal mix_sine:      std_logic_vector(IQ_Osc_DinWidth-1 downto 0);
    signal mix_cose:      std_logic_vector(IQ_Osc_DinWidth-1 downto 0);
    signal mix_dout:      std_logic_vector(IQ_DoutWidth-1 downto 0);

    --IQ mixer outputs
    signal I : std_logic_vector(IQ_DoutWidth-1 downto 0);
    signal Q : std_logic_vector(IQ_DoutWidth-1 downto 0);

    signal scic_din:        std_logic_vector(SCIC_DinWidth-1 downto 0);
    signal scic_dout:       std_logic_vector(SCIC_DoutWidth-1 downto 0);
    signal comp_din:        std_logic_vector(COMP_DataWidth-1 downto 0);
    signal comp_dout:       std_logic_vector(COMP_DataWidth-1 downto 0);
    signal channel_dout:    std_logic_vector(FIR_DataWidth-1 downto 0);
    signal dout_internal:   std_logic_vector(FIR_DataWidth-1 downto 0);

begin
```

```
-- ============================================================================
--                    Numerical Controlled Oscillator (NCO)
-- ============================================================================

    nco: component nco04
        generic map
        (
            AmplitudeWidth       =>  NCO_AmplitudeWidth,
            PhaseWidth           =>  NCO_PhaseWidth,
            AccumulatorWidth     =>  NCO_AccumulatorWidth,
            PhaseDitheringWidth  =>  NCO_PhaseDitheringWidth
        )
        port map
        (
            gsr     =>  '0',
            reset   =>  reset,
            clk     =>  clk,
            delta   =>  delta,
            sine    =>  sine,
            cosine  =>  open
        );


-- ============================================================================
--                      In phase Quadrature (IQ) mixer
-- ============================================================================

    iq_mixer: component iq_mixer03
      generic map(
                  Sig_DinWidth   => IQ_Sig_DinWidth,
                  Osc_DinWidth   => IQ_Osc_DinWidth,
                  DoutWidth      => IQ_DoutWidth
                  )
      port map(
               clk     => clk,
               reset   => reset,
               gsr     => '0',
               Din     => din1,
               sine    => sine,
               cosine  => mix_cose,
               I       => open,
               Q       => mix_dout
               );

-- ============================================================================
--                      Sharpened CIC (SCIC) Decimation Filter
-- ============================================================================

    scic: component scic_decimator01
        generic map
        (
            N               =>  SCIC_N,
            R               =>  SCIC_R,
            M               =>  SCIC_Poly'right,
            DinWidth        =>  SCIC_DinWidth,
            DoutWidth       =>  SCIC_DoutWidth,
            RegWidth        =>  SCIC_RegWidth
        )
        port map
        (
```

```
            gsr             =>  '0',
            clk             =>  clk,
            reset           =>  reset,
            ce_fs_low       =>  ce_fs_low,
            din             =>  mix_dout,
            dout            =>  scic_dout
        );

-- ============================================================================
--                          SCIC Compensation FIR Filter
-- ============================================================================

    scic_comp: component mulless_comp01
        generic map
        (
            DinWidth         =>  COMP_DataWidth,
            DoutWidth        =>  COMP_DataWidth,
            N                =>  COMP_Length
        )
        port map
        (
            gsr         =>  '0',
            clk         =>  clk,
            reset       =>  reset,
            ce_fs_low   =>  ce_fs_low,
            din         =>  scic_dout,
            dout        =>  comp_dout
        );

    dout_comp<=comp_dout;
    -- Compensator is allpass filter. Hence, it does not introduce
processing gain.

-- ============================================================================
--                              Channel FIR Filter
-- ============================================================================

    channel_filter: component mulless_fir01
        generic map
        (
            DinWidth        =>  FIR_DataWidth,
            DoutWidth       =>  FIR_DataWidth,
            N               =>  FIR_Length
        )
        port map
        (
            gsr          =>  '0',
            clk          =>  clk,
            reset        =>  reset,
            ce_fs_low    =>  ce_fs_low,
            din          =>  comp_dout,
            dout         =>  channel_dout
        );
```

```
--=============================================================================
--                              Output register
--=============================================================================

process(clk)
begin
    if rising_edge(clk) then
        if reset = '1' then
            dout_internal <= (others =>'0');
        elsif ce_fs_low = '1' then
            dout_internal <= channel_dout;
        end if;
    end if;
end process;

dout <= dout_internal;

end RTL;
```

mulless_comp01.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_signed.all;

library work;
use work.ddc_package01.all;

entity  mulless_comp01 is
    generic
    (
        DinWidth:           integer := 12;
        DoutWidth:          integer := 14;
        N:                  integer := 21
    );
    port
    (
        gsr         : in  std_logic;
        clk         : in  std_logic;
        reset       : in  std_logic;
        ce_fs_low   : in  std_logic;
        din         : in  std_logic_vector(DinWidth-1 downto 0);
        dout        : out std_logic_vector(DoutWidth-1 downto 0)
    );
end mulless_comp01;


architecture RTL of mulless_comp01 is

-- =============================================================================
--                              Local constants
-- =============================================================================

    -- internal arithmetic width
    constant A: integer := COMP_RegWidth;
```

```
constant comp_csd_length : integer_vector (0 to comp_coeff'right) :=
comp_coeff_length;

-- ============================================================================
--                               Local components
-- ============================================================================

    component multiplier_ak01
    generic (
             S      : integer_vector := (0,3,5);
             T      : csd_vector := ('-','+','-');
             DWidth : integer := 14
             );
    port (
          Din  : in  std_logic_vector(DWidth-1 downto 0);
          Dout : out std_logic_vector(DWidth-1 downto 0)
          );
    end component;

    component delay_block01 is
    generic (
             DELAY  : integer := 8;
             DWidth : integer := 10
             );
    port (
          gsr   : in  std_logic;
          clk   : in  std_logic;
          reset : in  std_logic;
          Din   : in  std_logic_vector(DWidth-1 downto 0);
          Dout  : out std_logic_vector(DWidth-1 downto 0)
          );
    end component;


-- ============================================================================
--                               Local signals
-- ============================================================================

    signal din_se:  std_logic_vector(A-1 downto 0);

    type filter_type is array (natural range <>) of std_logic_vector(A-1
downto 0);
    signal del:  filter_type(0 to N-1);   --  N-1 delays
    signal mul:  filter_type(0 to N);     --  N multipliers
    signal add:  filter_type(0 to N-1);   --  N adders

begin

-- ============================================================================
--                           Signed extension of input
-- ============================================================================

    din_se(A-1 downto DinWidth) <= (others => din(DinWidth-1));
    din_se(DinWidth-1 downto 0) <= din;
```

```
-- ============================================================================
--                          Multiple constant multipliers
-- ============================================================================

    MULTIPLIER_0: if comp_csd_length(0) /= 0 generate
        constant S : integer_vector(0 to comp_csd_length(0)-1):=
comp_coeff_positions(0, comp_csd_length(0)-1);
        constant T : csd_vector(0 to comp_csd_length(0)-1):=
comp_coeff_terms(0, comp_csd_length(0)-1);
    begin
        MULTIPLIER_0_0: component multiplier_ak01
            generic map
            (
                S       =>  S,
                T       =>  T,
                DWidth  =>  A
            )
            port map
            (
                Din     =>  din_se,
                Dout    =>  mul(0)
            );
    end generate;

    OTHER_MULTIPLIERS: for i in 1 to N generate
    begin
        MULTIPLIER_I: if comp_csd_length(i) /= 0 generate
            constant S :integer_vector(0 to comp_csd_length(i)-1):=
comp_coeff_positions(i, comp_csd_length(i)-1);
            constant T :csd_vector(0 to comp_csd_length(i)-1)    :=
comp_coeff_terms(i, comp_csd_length(i)-1);
            begin
                MULTIPLIER_I_I: component multiplier_ak01
                    generic map
                    (
                        S       =>  S,
                        T       =>  T,
                        DWidth  =>  A
                    )
                    port map
                    (
                        Din     =>  del(I-1),
                        Dout    =>  mul(I)
                    );
        end generate;
    end generate;

-- ============================================================================
--                            Delay elements and adders
-- ============================================================================

    DELAY_0_0: component delay_block01
        generic map
        (
            DELAY  => 1,
            DWidth => A
        )
        port map
        (
            gsr     =>  gsr,
```

```
            clk     =>  ce_fs_low,
            reset   =>  reset,
            Din     =>  din_se,
            Dout    =>  del(0)
        );


    GENERATE1:for i in 1 to N-1 generate
    begin
    DELAY_I: component delay_block01
        generic map
        (
             DELAY  => 1,
             DWidth => A
        )
        port map
        (
            gsr         =>  gsr,
            clk         =>  ce_fs_low,
            reset       =>  reset,
            Din         =>  del(I-1),
            Dout        =>  del(I)
        );
    end generate;

    add(0) <= to_stdlogicvector(to_bitvector(mul(0))) +
to_stdlogicvector(to_bitvector(mul(1)));

    GENERATE2: for i in 1 to N-1 generate
        add(i) <= to_stdlogicvector(to_bitvector(add(i-1))) +
to_stdlogicvector(to_bitvector(mul(i+1)));
    end generate;

-- ============================================================================
--                                  Output register
--============================================================================

    process (gsr, clk) is
    begin
        if gsr='1' then
            dout <= (others => '0');
        elsif rising_edge(clk) then
            if reset = '1' then
                dout <= (others => '0');
            elsif ce_fs_low = '1' then
                dout <= add(N-1)(A-1 downto A-DoutWidth);
             --   dout <= add(N-1)(DoutWidth-1 downto 0);
            end if;
        end if;
    end process;

end RTL;
```

digitalDownConverter01_tb

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.NUMERIC_STD.all;
use IEEE.std_logic_signed.all;

library std;
use std.textio.all;

library work;
use work.ddc_package01.all;

entity digitalDownConverter01_tb is
end digitalDownConverter01_tb;

architecture test of digitalDownConverter01_tb is

-- ============================================================================
--                     Declaration of constants
-- ============================================================================

   constant DeltaWidth:    integer := NCO_AccumulatorWidth;

-- ============================================================================
--                     Declaration of components
-- ============================================================================

component ddc01 is
    generic
    (
        -- Wordlength of input data
        DinWidth:        integer := 12;
        -- NCO frequency
        DeltaWidth:      integer := 32;
        -- Wordlength of output data
        DoutWidth:       integer := 16
    );
    port
    (
        clk:         in  std_logic;
        reset:       in  std_logic;
        ce_fs_low:   in  std_logic;
        delta:       in  std_logic_vector(DeltaWidth-1 downto 0);
        din1:        in  std_logic_vector(DinWidth-1 downto 0);
        dout:        out std_logic_vector(DoutWidth-1 downto 0);
    );
end component;

component mtu01
    generic
    (
        R: integer:=5
    );
    port
    (
        gsr:        in std_logic;
        clk:        in std_logic;
        reset:      in std_logic;
```

```
        ce_fs_low:  out std_logic
    );
 end component;

-- ============================================================================
--                     Declaration of local signals
-- ============================================================================

    signal gsr:             std_logic;
    signal reset:           std_logic;
    signal clk:             std_logic := '0';
    signal ce_fs_low:       std_logic;
    signal delta:           std_logic_vector(DeltaWidth-1 downto 0);
    signal DataIn:          std_logic_vector(DataInWidth-1 downto 0);
    signal DataOut:         std_logic_vector(DataOutWidth-1 downto 0);

-- ============================================================================
--                     Time specification
-- ============================================================================

    constant R:             integer := 10;
    constant T_gsr:         time :=125ns;
    constant T_clk:         time :=20ns;
    constant T_reset:       time :=335ns;
    constant T_clk_to_q:    time :=3ns;

begin

-- ============================================================================
--                       Signals gsr, clk, reset and delta
-- ============================================================================

gsr <= '1', '0' after T_gsr;
reset <= '1', '0' after T_reset;
clk <= not clk after T_clk/2;
--delta <= x"00008000"; -- f = 6.25MHz
delta <= x"20000000"; -- f = 6.25MHz

-- ============================================================================
--                       Unit under test
-- ============================================================================

uut: ddc01
generic map
(
    DinWidth    => DataInWidth,
    DeltaWidth  => DeltaWidth,
    DoutWidth   => FIR_DataWidth
)
port map
(
    clk => clk,
    reset => reset,
    ce_fs_low=> ce_fs_low,
    delta => delta,
    din1 => DataIn,
    dout => DataOut,
    dout_scic=>dout_scic
);
```

```
-- ============================================================================
--                       Master Timing Unit
-- ============================================================================


mtu: component mtu01
generic map
(
    R => R
)
port map
(
    gsr => gsr,
    clk => clk,
    reset => reset,
    ce_fs_low => ce_fs_low
);



-- ============================================================================
--                       Reading input samples from .txt files
-- ============================================================================


process is
        file FIDX: text open read_mode is
"../all_design/simulation/text_files/pobuda.txt";
    variable X_sample:  integer;
    variable X_trace_line: line;
begin
 -- initial value during reset sequence
    DataIn <= conv_std_logic_vector(0,DataInWidth);

    -- wait until reset sequence is over
    wait until reset='0';

    -- reading samples
    while not(endfile(FIDX)) loop
        wait until rising_edge(clk);
        wait for T_clk_to_q;
        readline(FIDX,X_trace_line);
        read(X_trace_line,X_sample);
        DataIn <= conv_std_logic_vector(X_sample,DataInWidth);
    end loop;
    --wait forewer
    wait;
end process;
```

```
-- ============================================================================
--       Writing output of SCIC samples to out_ddc.txt files
-- ============================================================================

process is
        file FIDY:text open write_mode is
"../all_design/simulation/text_files/out_ddc.txt";
    variable Y_trace_line:line;
begin
    wait until reset='0';
    --while true loop
    for I in 1 to 1000 loop
        wait until ce_fs_low='1';
        wait until rising_edge(clk);
        wait for T_clk_to_q;

write(Y_trace_line,conv_integer(to_stdlogicvector(to_bitvector(DataOut))));
        writeline(FIDY,Y_trace_line);
    end loop;
 --   wait;
end process;

end test;
```