SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 424

SKLOPOVSKI IZVEDEN DIGITALNI GENERATOR SINUSNOG SIGNALA S AMPLITUDNOM, FAZNOM I FREKVENCIJSKOM MODULACIJOM

ALEN MEŠIĆ

ZAGREB, veljača 2012.

Posebna zahvala mentoru prof. Davoru Petrinoviću i asistentu Marku Butorcu na vodstvu i pomoći.

Također veliko hvala Marku Brezoviću na darežljivosti.

SADRŽAJ

1. Uvod

Od samih početaka razvoja elektroničkih sklopova, isprva analognih, a kasnije i digitalnih, postojala je potreba za ispitivanjem odziva tih sklopova. U tu svrhu bilo je potrebno imati signal unaprijed poznatih karakteristika, te su najčešće korišteni laboratorijski izvori koji su davali precizne istosmjerne signale, ili imali mogućnost biranja pravokutnog, trokutastog ili sinusnog valnog oblika.

Napretkom tehnologije razvijen je analogni sklop signalni generator s mogućnošću amplitudne, fazne, frekvencijske i impulsne modulacije u svrhu ispitivanja odziva sklopova na kompleksnije signale. Osnova sklopa bio je generator sinusnog signala, obično LC oscilator, koji se naknadno modulirao po želji.

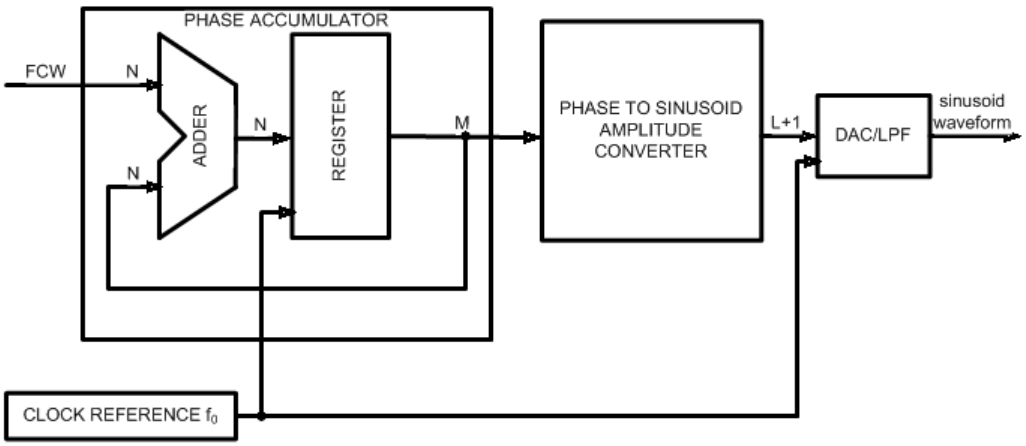
Napretkom digitalnih tehnologija razvijen je direktni digitalni sintetizator (DDS) visoke točnosti koji na svom izlazu daje sinusni signal minimalnog odstupanja od idealnog, te kao takav omogućio izradu digitalnog sklopa za generiranje ispitnih signala.

Sklop realiziran u okviru ovog diplomskog rada upravo je temeljen na takovom DDS-u, te omogućuje amplitudnu, faznu i frekvencijsku modulaciju generiranog signala. Razvijen je pomoću programskog paketa Xilinx ISE Design Suite 12.2, te implementiran na razvojnom sustavu Spartan-3E Starter Kit Board proizvođača "Digilent, Inc.".

2. Direktna Digitalna Sinteza

Direktna Digitalna Sinteza (DDS) je elektronička metoda za realizaciju digitalnih signala proizvoljnih valnih oblika, obično sinusnog, iz zadane frekvencije.

Temeljnu arhitekturu Direktne Digitalne Frekvencijske sinteze predstavili su Tierney, Rader i Goldet 1971. Sastoji se od faznog akumulatora i PSAC (*phase to sinusoid amplitude converter*) pretvornika. Pojednostavljena blok shema prikazana je slikom 1 (Šipić, 2009.).



Slika 1 Pojednostavljeni prikaz DDS arhitekture

Sustav ima 2 ulaza: referentni takt i kontrolnu faznu riječ (FCW). Fazni akumulator daje vrijednost nove fazne riječi u svakom periodu takta. Nakon odrenenog broja perioda takta, ovisno o duljini fazne riječi (broju bitova), dolazi do preljeva i periodičnog ponavljanja vrijednosti fazne riječi.

Izlaz faznog akumulatora predstavlja kut u intervalu [0, 2π>. Frekvencijska rezolucija sintetizatora i izlazna frekvencija dane su izrazom (1) i (2).

$$\Delta f = \frac{f_0}{2^N} \quad (1)$$

$$f_{out} = = FCW \cdot \Delta f \quad (2)$$

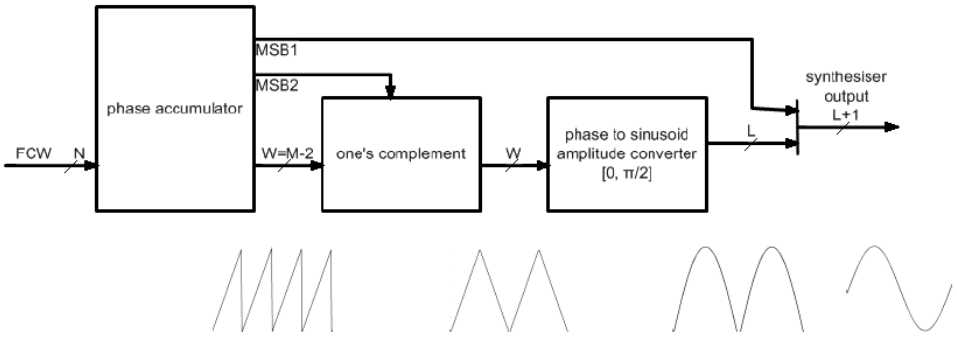
f_0 je frekvencija referentnog takta, a N rezolucija faznog akumulatora (širina fazne riječi). Očito je da frekvencijska rezolucija sintetizatora može biti proizvoljno mala uz dovoljno velik N. U praksi se bira kompromis između širine fazne riječi, složenosti sustava, željene frekvencije takta i kašnjenja izlaznog signala.

PSAC ima zadatak da estimira i na izlaz sustava daje vrijednost amplitude sinusne funkcije za svaki novi kut. Najčešće se implementira digitalno, a ako postoji potreba za analognim izlazom iz sustava, izlaz PSAC-a se dovodi na digitalno-analogni pretvornik i niskopropusni filtar.

Najjednostavnija izvedba PSAC-a je pomoću ROM prozivne tablice. Veza između veličine ROM-a i broja bitova fazne riječi je eksponencijalna i postaje nedozvoljivo velika čak i za sustave umjerenih performansi, pa se širina ulaza u PSAC reducira na M najznačajnijih bitova.

Na taj način se složenost PSAC-a smanjuje bez potrebe za povećanjem frekvencijske rezolucije. Negativna strana ove redukcije je pojavljivanje šuma u izlaznom spektru koji je posljedica periodičke amplitudne pogreške izlazne sinusoide (4).

Složenost PSAC-a se može učinkovito smanjiti i korištenjem svojstava simetrije sinusnog signala. Dva najznačajnija bita fazne riječi određuju u kojem se kvadrantu nalazi određeni kut. Jednostavnim transformacijama se čitava sinusoida može rekonstruirati iz izračunatih vrijednosti amplituda samo za prvi kvadrant. Blok shema koja prikazuje ovakav pristup prikazana je na slici 2 (Šipić, 2009).



Slika 2 DDS sa reduciranom fazom i primjenom kvadratne simetrije

Ulaz u PSAC je W-bitna frakcija x, a izlaz je sinusoida f(x) (izrazi (3) i (4)). f(x) je prikazana s N bitova na izlazu i pozitivna je.

$$x \in \frac{\{1,2,3\ldots.2^W-1\}}{2^W}, W = M - 2 \quad (3)$$

$$f(x) = A \cdot \sin\left(\frac{\pi \cdot x}{2}\right) + \varepsilon(x), 0 \leq x < 1 \quad (4)$$

ε(x) je pogreška izlaza PSAC-a u odnosu na idealnu sinusoidu, predstavljena beskonačnom preciznošću. Ova pogreška je nastala akumulacijom više pogreški različitog karaktera kao što su pogreška amplitudne kvantizacije i pogreške koje unosi algoritam.

Izlazni frekvencijski spektar ima dvije komponente: čistu sinusoidu (jednu frekvenciju) amplitude A i šum koji je posljedica amplitudne pogreške. Omjer ovih dviju komponenata izlaznog spektra naziva se SFDR (*spurious free dynamic range*) i kritični je parametar za ocjenu performansi sustava.

Implementacija funkcije u PSAC-u za prvi kvadrant jedinične kružnice predmet je intenzivnog istraživanja. Razlog tome je jednostavna implementacija ROM prozivnom tablicom, koja je pak nepraktična i skupa ukoliko je potreban veliki kapacitet za pohranu podataka.

Jedan primjer implementacije je polinomijalnom aproksimacijom. Ideja je aproksimirati dijelove sinusnog valnog oblika polinomima određenog stupnja. Sinusnu funkciju moguće je raspisati prema izrazu (5).

$$\sin\left(\frac{\pi x}{2}\right) \cong \begin{cases} \sum_{i=0}^{r} c_{0i}(x - x_0)^i & x_0 \le x < x_1 \quad (x_0 = 0) \\ \sum_{i=0}^{r} c_{1i}(x - x_1)^i & x_1 \le x < x_2 \\ \vdots & \\ \sum_{i=0}^{r} c_{ki}(x - x_k)^i & x_k \le x < x_{k+1} \\ \vdots & \\ \sum_{i=0}^{r} c_{(s-1)i}(x - x_{s-1})^i & x_{s-1} \le x < x_s \end{cases} \tag{5}$$

x je kut predstavljen kao frakcija iz intervala [0,1)

r je stupanj polinomijalne aproksimacije

s je broj po dijelovima glatkih polinomijalnih segmenata

c_{ki} su koeficijenti polinoma

x_k je donja granica k-tog segmenta

Izraz (5) iskorišten je za razne implementacije PSAC-a. Razlike izvedbi mogu se podijeliti na četiri osnovna kvalifikatora:

- stupanj r, $r \geq 1$, polinomijalne aproksimacije
- broj s, $s \geq 1$, po dijelovima glatkih polinomijalnih segmenata
- granice segmenata x_k
- metoda kojom se računaju koeficijenti polinoma c_{ki}

DDS korišten u ovom sklopu koristi implementaciju PSAC-a na temelju aproksimacije polinomom trećeg stupnja. Metoda se može opisati kao aproksimaciju sinusne funkcije po dijelovima polinomom uz B-Spline interpolaciju sinusoide, i to jedne periode signala [0, 2π> podijeljene na proizvoljan broj uniformnih polinomijalnih segmenata. Realizacija DDS-a mora osigurati proizvoljan odabir točaka unaprijed konstruirane po dijelovima

kubične funkcije. To se rješava korištenjem Farrow strukture koja se temelji na Hornerovom algoritmu računanja vrijednosti polinoma u točki.

Idealna sinusna funkcija $y_s(\varphi) = \sin(\varphi)$, gdje je φ zadana faza, aproksimira se po dijelovima polinomijalnom funkcijom $y_r(\varphi)$ koja ima N uniformnih segmenata unutar sva četiri kvadranta $y_s(\varphi)$. Ako se radi o kubičnom polinomu, segment polinoma između faznih granica $\frac{n\cdot 2\pi}{N}$ i $\frac{(n+1)\cdot 2\pi}{N}$ računa se prema Hornerovom pravilu:

$$w_n(\Delta x) = \big((d[n]\cdot \Delta x + c[n])\cdot \Delta x + b[n]\big)\cdot \Delta x + a[n] \qquad (6)$$

Argument Δx kubičnog polinoma je jednak nuli u desnoj točki segmenta, a jedinici u lijevoj točki, $0 \le x \le 1$. Kontinuiranost faze aproksimirane sinusne funkcije dobiva se spajanjem pojedinih kubičnih segmenata prema izrazu

$$y_r(\varphi) = \left\{ w_n(\Delta x) \mid \Delta x = N\frac{\varphi}{2\pi} - n, \varphi \in \left[\frac{n\cdot 2\pi}{N}, \frac{(n+1)\cdot 2\pi}{N}\right\rangle, \forall n \in [0, N-1] \right\} \qquad (7)$$

Koeficijenti se biraju tako da se minimizira maksimalna pogreška aproksimacije:

$$e(\varphi) = y_s(\varphi) - y_r(\varphi), \qquad \varphi \in [0,\ 2\pi] \qquad (8)$$

Interpolacija temeljena na B-spline-u, koja se ovdje primjenjuje, maksimizira glatkoću po dijelovima polinomijalnog modela. Koeficijenti $d[n]$, $c[n]$, $b[n]$ i $a[n]$ mogu se izračunati filtriranjem diskretnog sinusnog signala $y_d[n]$ nekauzalnim sustavom definiranim transfer-matricom.

B-spline interpolator rekonstruira signal $y_s(\varphi)$ s kontinuiranom fazom iz uzoraka $y_d[n]$ oponašajući pritom idealni „sinc“ interpolator. Rekonstruirani signal$y_r(\varphi)$ sadrži skalirani željeni signal $G\cdot y_s(\varphi)$uz beskonačan broj neželjenih harmonika čije magnitude opadaju s faktorom $\approx iN^{-4}$.

Analiza pokazuje da je interpolacija kubičnim spline-om vrlo učinkovita, jer je pogreška reducirana faktorom $\approx N^{-4}$, gdje je N broj interpoliranih segmenata. Za primjer 8-bitne tablice gdje je N=256, RMS

pogreška približno iznosi $\bar{e} = 2^{-31}$. Ovo je aproksimacija najvećeg mogućeg reda izvediva po dijelovima kubičnom interpolacijom. Dobiveni aproksimirani signal $y_r(\varphi)$ ima kontinuirane derivacije prvog i drugog reda na svim prijelazima segmenata.

Za konkretnu primjenu, aproksimirani signal $y_r(\varphi)$ može se proizvoljno podijeliti na željeni broj segmenata, što znači da je moguće proizvoljno odabrati diskretne fazne vrijednosti koje određuju granice segmenata. Da bi se olakšala izvedba DDS-a, željena frekvencija diskretne sinusoide ω_s i njena početna faza φ_0, prikazuju se frakcijama.

$$\omega_s = \frac{k}{M \cdot N} \cdot 2\pi \ , \qquad \varphi_0 = \frac{k_0}{M \cdot N} \cdot 2\pi \ , \qquad 0 < k < \frac{M \cdot N}{2} \tag{9}$$

M određuje frakcionalnu rezoluciju faze. Konkretna vrijednost trenutne faze je $\varphi[m] = \frac{2\pi}{M \cdot N}(k \cdot m + k_0)_{mod(M \cdot N)}$ i može se rastaviti na cjelobrojni i frakcionalni dio $\varphi[m] = \frac{2\pi}{N}(n[m] + \Delta x[m])$. Željeni uzorak $y_r(\varphi[m])$ dobiva se zamjenom $\Delta x[m]$ u izrazima (6) i (7) kubičnim koeficijentima, a $n[m]$ određuje unutar kojeg segmenta se vrši interpolacija. Originalni polinom $w_n(\Delta x)$ je transformiran u novi $w'_n(\Delta x') = \big((d'[n] \cdot \Delta x' + c'[n]) \cdot \Delta x' + b'[n]\big) \cdot \Delta x' + a'[n]$ prije kvantizacije koeficijenata sa $\Delta x' = 2\Delta x - 1$. Transformirani koeficijenti imaju reduciranu magnitudu što omogućava kompresiju koeficijenata koji će se smještati u ROM.

Dodatno pojednostavnjenje izvedbe PSAC-a koje se redovito primjenjuje je korištenje svojstava simetrije i antisimetrije sinusne funkcije. Na taj način moguće je realizirati PSAC samo za izračun aproksimacije amplituda za kutove prvog kvadranta i iz tih vrijednosti rekonstruirati cijelu periodu funkcije. Ako su a, b, c i d koeficijenti polinoma za neki segment prvog kvadranta sinusne funkcije, koeficijenti ostalih kvadranta dobiju se iz izraza danih u tablici:

Tablica 1 Međusobna veza koeficijenata po kvadrantima

1. KVADRANT	2. KVADRANT	3. KVADRANT	4. KVADRANT
a	a	-a	-a
b	-b	-b	b
c	c	-c	-c
d	-d	-d	d

Zapis u tablici 2 ne predstavlja stvarne predznake koeficijenata, već odnos koeficijenata 2., 3. i 4. kvadranta u odnosu na već izračunate koeficijente prvog kvadranta.

O kojem kvadrantu se ustvari radi lako je vidjeti iz MSB i NSB fazne riječi koju daje akumulator. Prikaz stvarnih predznaka koeficijenata i sadržaj MSB i NSB bitova za svaki kvadrant daje sljedeća tablica:

Tablica 2 Stvarni predznak koeficijenata i sadržaj MSB i NSB bitova po kvadrantima

1. KVADRANT		2. KVADRANT		3. KVADRANT		4. KVADRANT	
MSB=0	NSB=0	MSB=0	NSB=1	MSB=1	NSB=0	MSB=1	NSB=1
a > 0		a > 0		a < 0		a < 0	
b > 0		b < 0		b < 0		b > 0	
c < 0		c < 0		c > 0		c > 0	
d < 0		d > 0		d > 0		d < 0	

Svi koeficijenti pohranjeni u ROM tablici su pozitivni, dodjeljivanje predznaka izvedeno je jednostavnim operacijama komplementiranja ovisno o kvadrantu za koji se koeficijenti računaju. Preciznije rečeno, u ROM su pohranjene frakcije koeficijenata, jer se oni ionako nalaze u rasponu $[0, 1\rangle$. Nakon dohvata koeficijenata iz ROM-a dodaje se MSB=0 svakom koeficijentu, i ovisno o MSB i NSB fazne riječi generirane akumulatorom, se provodi dvojno komplementiranje određenih koeficijenata, prema tablici 2.

3. Modulacije

3.1. Amplitudna modulacija

Uzmemo li modulirajući signal $U_m(t)$ takav da vrijedi $F[u_m(t)] = U_m(j\omega) = \int_{-\infty}^{\infty} u_m(t)e^{-j\omega t}dt = 0$ za $|\omega| > \omega_M$ i nosilac oblika $u_0(t) = U_0\cos(\omega_0 t)$, klasičnu amplitudnu modulaciju definiramo kao postupak kojim se amplituda nosioca mijenja oko srednje vrijednosti U_0 pri čemu je promjena proporcionalna modulacijskom signalu:

$$u_{AM}(t) = [U_0 + K_a u_m(t)]\cos(\omega_0 t) \tag{10}$$

$$u_{AM}(t) = \left[1 + \frac{K_a u_m(t)}{U_0}\right]\cos(\omega_0 t) \tag{11}$$

Ovaj izraz je najveći za $u_m(t) = |u_m(t)|_{max}$

Indeks modulacije:

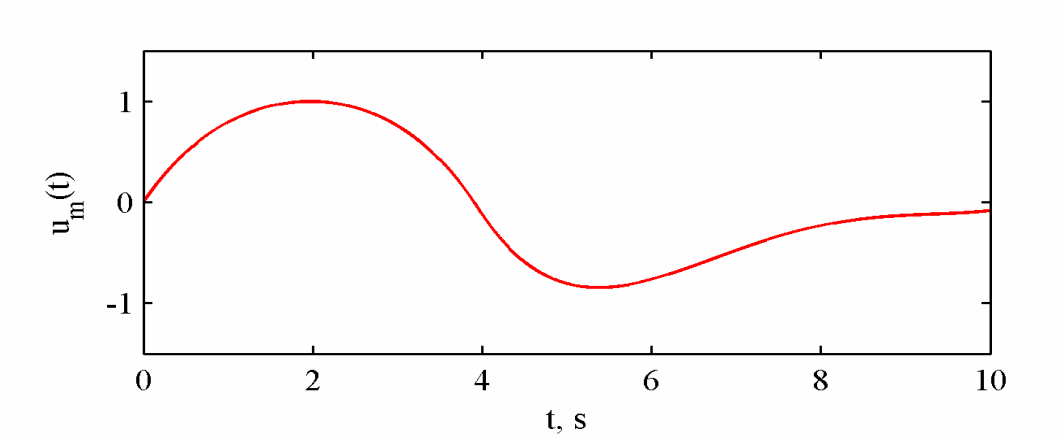
$$\frac{K_a|u_m(t)|_{max}}{U_0} = \frac{\Delta U_0}{U_0} = m_{AM} \tag{12}$$

Za $\Delta U_0 = 1$ i $|u_m(t)|_{max} = 1$, $m_{AM} = K_a$

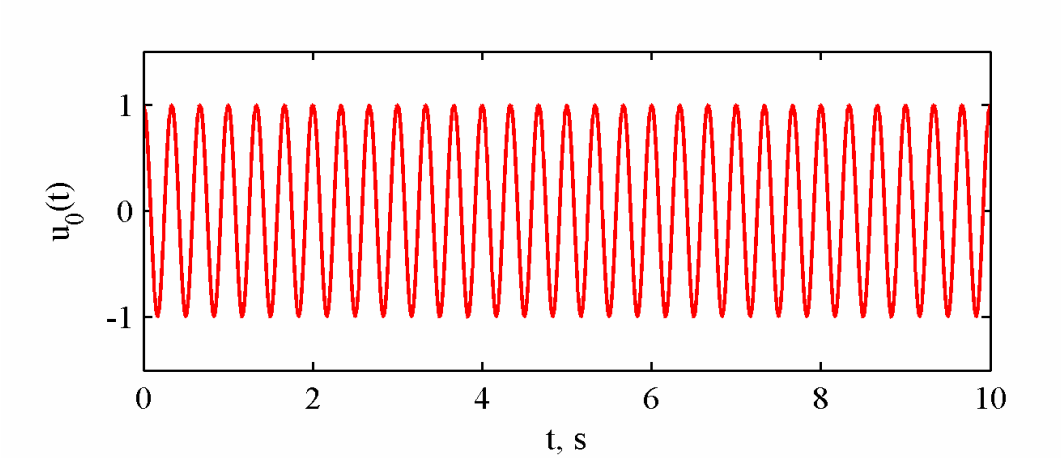
Kod amplitudne modulacije s potisnutim nosiocem amplituda nosioca mijenja oko 0, a ne oko srednje vrijednosti U_0

$$u_{AM}(t) = K_a u_m(t) \cdot U_0\cos(\omega_0 t) \tag{13}$$

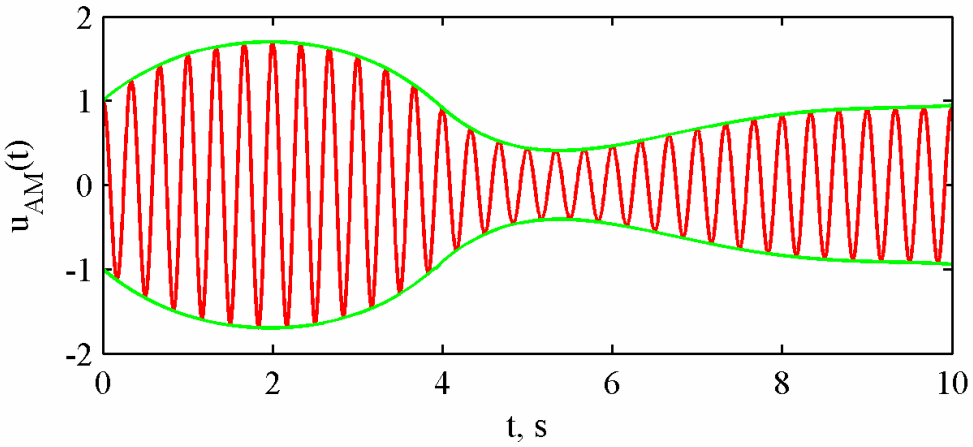
Primjeri AM za $U_0 = 1$, $|u_m(t)|_{max} = 1$, $K_a = 0.7$ i $K_a = 1.7$ za klasičnu AM, $K_a = 1$ za AM s potisnutim nosiocem prikazani su sljedećim slikama (Vučić, 2010.)



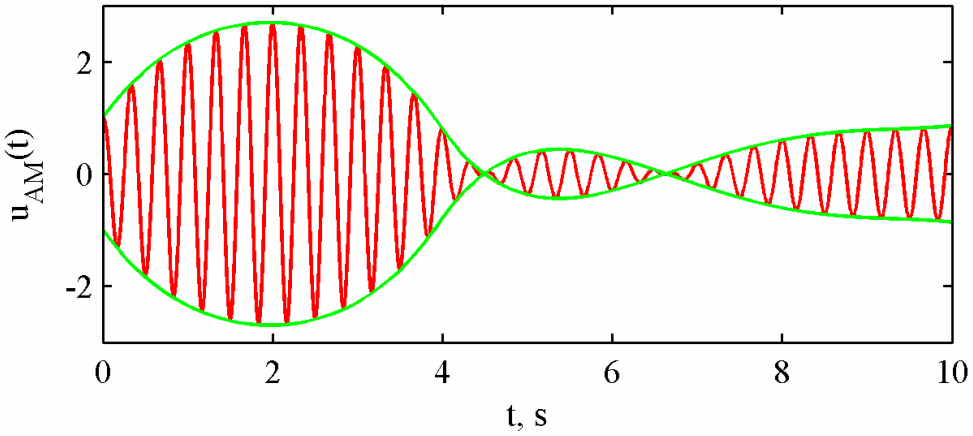
Slika 3 Modulacijski signal, $u_m(t) = f(t)$



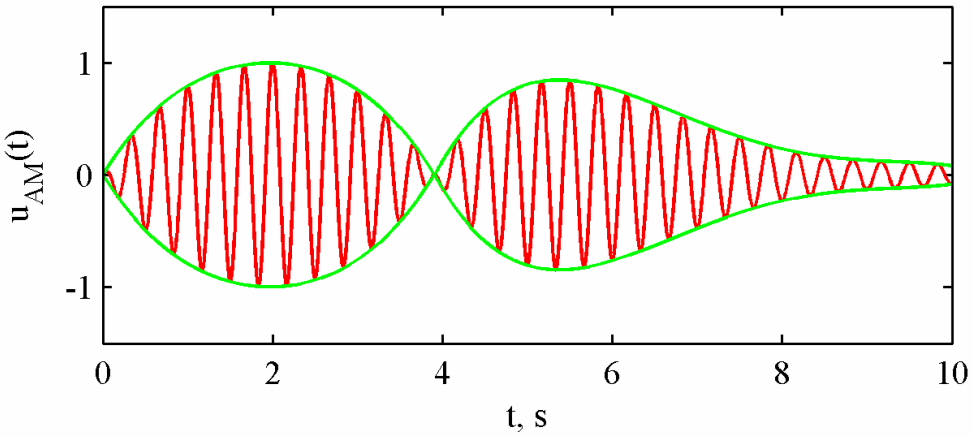
Slika 4 Nosilac, $u_0(t) = U_0 \cos(\omega_0 t)$



Slika 5 Klasična AM, $u_{AM}(t) = [U_0 + K_a u_m(t)]\cos(\omega_0 t)$, $K_a = 0.7$



Slika 6 Klasična AM, $u_{AM}(t) = [U_0 + K_a u_m(t)]\cos(\omega_0 t)$, $K_a = 1.7$



Slika 7 AM s potisnutim nosiocem, $u_{AM}(t) = K_a u_m(t)\, U_0 \cos(\omega_0 t)$

3.2. Fazna i frekvencijska modulacija

FAZNA MODULACIJA

Trenutna promjena faze proporcionalna je modulacijskom signalu $\delta\varphi_i(t) = K_p u_m(t)$, trenutna faza iznosi $\varphi_i(t) = \omega_o t + K_p u_m(t)$, pa modulirani signal ima oblik $u_{PM}(t) = U_0 \cos\left[\omega_o t + K_p u_m(t)\right]$.

Ako se uzme da modulacijski signal ima oblik $u_m(t) = U_m \cos(\omega_m t)$ modulirani signal može se zapisati kao $u_{PM}(t) = U_0 \cos\left[\omega_o t + K_p U_m \cos(\omega_m t)\right]$. Budući da je $|\cos(\omega_m t)|_{max} = 1$, dobivamo izraz za maksimalnu devijaciju faze:

$$\Delta\varphi = K_p U_m \tag{14}$$

Trenutna frekvencija $\omega_i(t) = \frac{d\varphi_i(t)}{dt} = \omega_0 + K_\omega U_m \sin(\omega_m t)$ ima najveći iznos pri $\omega_m t = -\pi/2$, odnosno kad $u_m(t)$ najbrže raste, a najmanji pri $\omega_m t = +\pi/2$, odnosno kad $u_m(t)$ najbrže pada.

FREKVENCIJSKA MODULACIJA

Trenutna promjena frekvencije proporcionalna je modulacijskom signalu $\delta\omega_i(t) = K_\omega u_m(t)$, trenutna frekvencija iznosi $\omega_i(t) = \omega_o + K_\omega u_m(t)$. Integracijom se dobije trenutna faza $\varphi_i(t) = \omega_o t + \int_0^t K_\omega u_m(t)dt$, pa modulirani signal ima oblik $u_{PM}(t) = U_0 \cos\left[\omega_o t + \int_0^t K_\omega u_m(t)dt\right]$.

Ako se uzme da modulacijski signal ima oblik $u_m(t) = U_m \cos(\omega_m t)$ modulirani signal može se zapisati kao $u_{PM}(t) = U_0 \cos\left[\omega_o t + \int_0^t K_\omega U_m \cos(\omega_m t)\, dt\right]$, što nakon integracije daje $u_{PM}(t) = U_0 \cos\left[\omega_o t + \frac{K_\omega U_m}{\omega_m} \sin(\omega_m t)\right]$. Budući da je $|\sin(\omega_m t)|_{max} = 1$, dobivamo izraz za maksimalnu devijaciju faze:

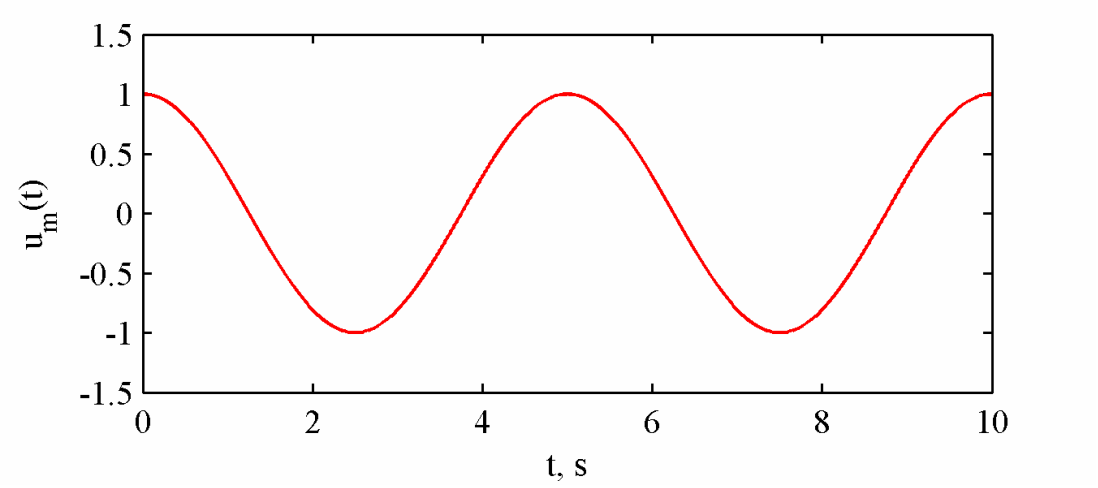
$$\Delta\varphi = \frac{K_\omega U_m}{\omega_m} \tag{15}$$

Iz trenutne kutne frekvencije $\omega_i(t) = \frac{d\varphi_i(t)}{dt} = \omega_0 + K_\omega U_m \cos(\omega_m t)$ uz $|\cos(\omega_m t)|_{max} = 1$ dobije se maksimalna devijacija frekvencije:

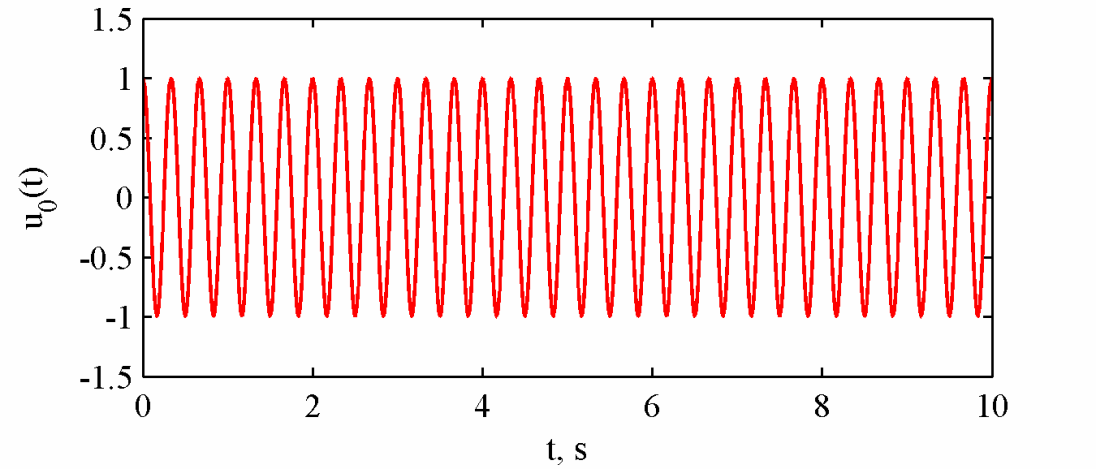
$$\Delta\omega = K_\omega U_m \tag{16}$$

Također je vidljivo da je trenutna frekvencija najveća za $\omega_m t = 0$, odnosno kad $u_m(t)$ ima maksimum, a najmanja za $\omega_m t = \pi$, odnosno kad $u_m(t)$ ima minimum.

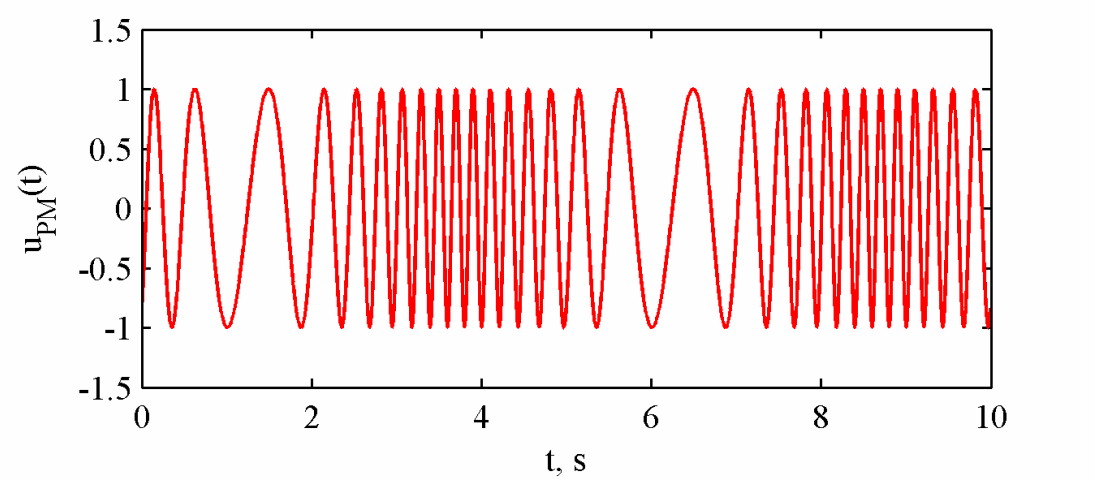
Primjer PM i FM prikazan je sljedećim slikama (Vučić, 2010.)



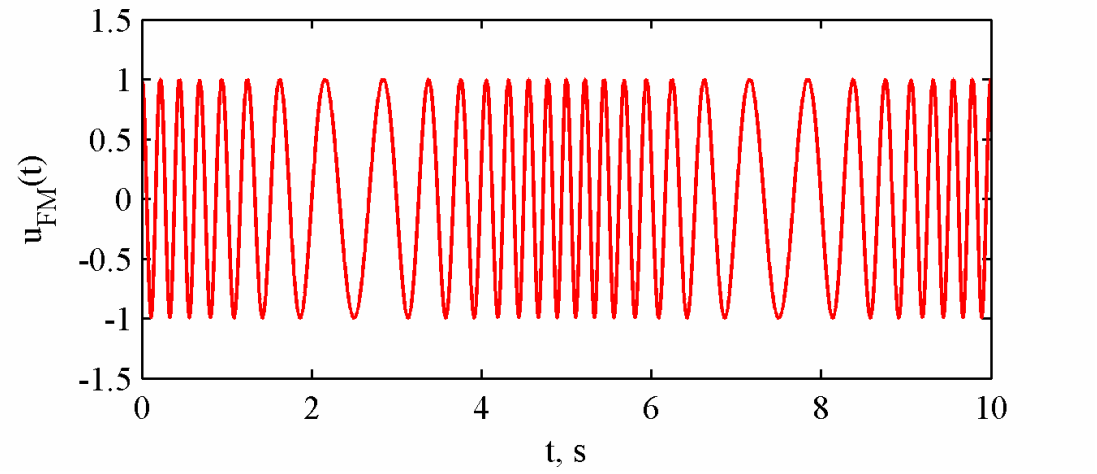
Slika 8 Modulacijski signal $u_m(t) = U_m \cos(\omega_m t)$, $U_m = 1, f_m = 0.2Hz$



Slika 9 Nosilac $u_0(t) = U_0 \cos(\omega_0 t)$, $U_0 = 1, f_0 = 3Hz$



Slika 10 Fazno modulirani signal, $K_p = 10$



Slika 11 Frekvencijski modulirani signal, $K_\omega = 10$

K_p i K_ω su odabrani mnogo veći no u praksi radi bolje ilustracije

4. Realizacija sklopa

4.1. Xilinx ISE Design Suite 12.2

Sklop je projektiran programskim paketom Xilinx ISE Design Suite 12.2. Projektiranje započinje zadavanjem karakteristika sklopa nakon kreiranja projekta. Odabrane karakteristike:

- Familija uređaja: Spartan
- Tip sklopa: xc3s500e
- Kućište: FG320
- Razred brzine: 5

Postoje tri različita načina projektiranja, strukturni opis, slijedni opis te opis ravnopravnim jednadžbama. U projektiranju ovog sklopa korišten je strukturni opis za dizajn glavnog modula, a za podmodule kombinacija svih triju načina, ovisno o potrebama. Modul je dio sklopa koji obavlja određenu funkciju za vrijeme jedne periode referentnog takta.

Nakon kreiranja referentnog modela koji opisuje funkciju pojedine komponente kreira se i ispitno okruženje za svaku komponentu u kojem se definira stanje na ulaznim priključcima u određenim vremenskim trenucima.

Pomoću ispitnog modela moguće je pokrenuti *iSim* simulator koji simulira zadano stanje na ulaznim priključcima i stanje izlaznih priključaka ovisno o zadanim parametrima radnog okruženja. Moguće je pomoću ovog programskog paketa postaviti željena vremenska i prostorna ograničenja na komponentu. Potrebno je razviti fizički ostvariv model da bi se sklop mogao implementirati. Konačno se vrši sinteza i implementacija sustava a korisnik može kao rezultat vidjeti preglednik (*Floor Planner*) koji pokazuje unutrašnji raspored ćelija FPGA sklopa.

4.2. Spartan-3E Starter Kit Board

FPGA sklop xc3s500e pripada familiji Spartan-3E. Ova familija proizašla je iz familije Spartan-3 pri čemu je napredak ostvaren kroz poboljšanje svojstava ulazno-izlaznih blokova. Korišten FPGA sklop sadrži 500 000 logičkih vrata. Pritom je korisniku na raspolaganju veći broj konfigurabilnih logičkih blokova, memorija, množila, sklopova za upravljanje taktom i ulazno-izlaznih blokova. Kućište sklopa i razred brzine vezani su uz tehnologiju izrade sklopa. Razred brzine određuje vremena propagacije kroz pojedine komponente FPGA sklopa. Kod novijih FPGA sklopova vrijedi što je razred brzine veći to je sklop brži.

Slika 12 Razvojni sustav Spartan-3E Starter Kit Board

Slikom 12 (Xilinx, 2011.) prikazana je korištena razvojna pločica. Na pločici nalaze se tipke, preklopke i rotacijski enkoder koji služe za upravljanje sklopom, čijim pritiskom dolazi do istitravanja kontakata koje se mora programski riješiti kako ne bi dolazilo do lažnih višestrukih očitanja pritiska tipke. Takt se sustavu može dovesti izvana ili iz ugrađenog oscilatora SG-8002JF frekvencije 50MHz. Kod ovog oscilatora trajanje visoke razine u odnosu na period je 40% i radi na naponu napajanja od 3.3V.

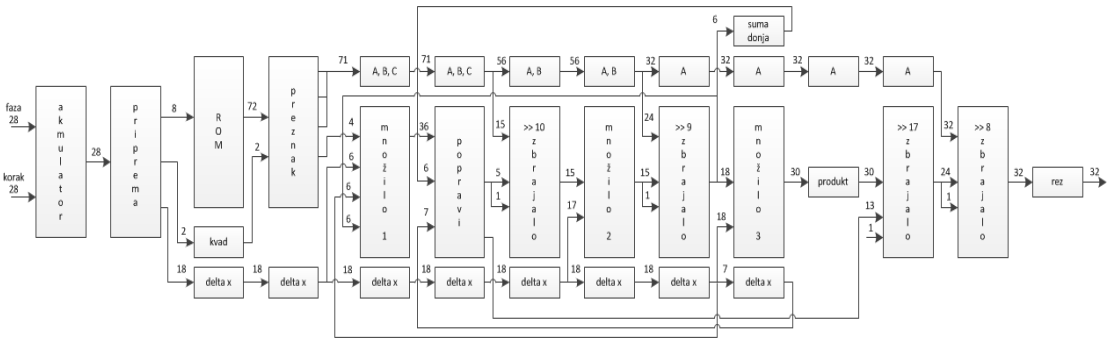
Programiranje FPGA sklopa izvršeno je JTAG/USB sučeljem, te iMPACT programskom podrškom. Ostvarenom USB komunikacijom osobnog računala i razvojnog sustava, iMPACT može ostvariti direktnu vezu s FPGA sklopom. Nakon spajanja USB kabela, te dovođenjem napajanja na štampanu pločicu, operacijski sustav automatski prepoznaje razvojni sustav.

4.3. Moduli sklopa

U ovom poglavlju dan je kratak opis funkcionalnosti sklopa kroz module koji ga čine. Preglednosti radi deklaracija entiteta, popis priključaka kao i detaljna funkcionalnost pojedinog modula opisana strojnim jezikom nalazi se u privitku rada.

DDS modul

DDS modul korišten u ovom radu razvio je Marko Brezović u Xilinx ISE Design Suite 10.1 za Spartan 3E sklopove. Modul ima protočnu strukturu i teoretsku brzinu rada veću od 120 MHz. Blok shema modula prikazana je slikom 13 (Brezović, 2010.).



Slika 13 Blok shema DDS modula

Na slici nisu napisana imena signala već samo njihove širine, a što pojedini moduli radi objašnjeno je u nastavku. Veći pravokutnici su složeni pod module, dok su manji registri i na njima piše vrijednost koja kasni. Može se vidjeti da je registar iskorišten 11 puta, množila tri i zbrajala četiri puta.

Na slici nisu ucrtani signal takta i signal za postavljanje sustava u početno stanje (eng. *reset*) da bi slika bila preglednija. Također, sliku bi se moglo podijeliti u dva dijela. Prvi dio generira fazu i čita koeficijente koje zatim pretvara iz broja bez predznaka u broj s predznakom te drugi dio koji radi evaluaciju polinoma po Hornerovom pravilu.

Množitelji u množilima i produkti nisu uvijek jednake širine. Prvi umnožak je koeficijenta D koji 1.3 (frakcioni zapis) i delta x koji je 1.5 (najviših šest bita). Njihov umnožak je 1.8, ali se zaokružuje na 1.4 dodavanjem sljedećeg nižeg bita prethodnom. Nakon toga se 1.4 posmakne u desno za 10 mjesta da bi bio poravnan s koeficijentom C 1.14 s kojim se zbraja (zaokruživanje i posmak se radi u modulu zbrajalo). Nakon zbroja se ponovo radi umnožak te sume 1.14 s delta x 1.16 (najviših 17 bita) te se dobiva produkt 1.30 koji se zaokružuje na 1.14 na isti način kao i prethodni produkt te se posmakne za devet mjesta u desno čime dobivamo 1.23 koji zbrajamo s koeficijentom B 1.23. Taj zbroj množimo s cijelim delta x 1.17. Obzirom da množila mogu maksimalno množiti 1.17x1.17 radi se prošireno množenje na način koji je objašnjen u 3.6.1. Nakon proširenog množenja dobiva se produkt 1.29 koji se kao i ostali zaokružuje na 1.23 i posmakne u desno za osam mjesta te se zbroji s koeficijentom A 1.31 nakon čega dobivamo rezultat tj. izlaz iz modela. Cijeli model je izveden u protočnoj strukturi dubine 14.

Modul je iskorišten tri puta. Po jedan za generiranje modulacijskih signala amplitudne te fazne ili frekvencijske modulacije unutar sklopa, te jedan za generiranje nosioca. Posljednji je neznatno modificiran u podmodulu fazni akumulator gdje se dodatno pribraja delta faze fazne, odnosno frekvencijske modulacije.

Podmoduli DDS modula

Fazni akumulator je prvi modul tj. on se koristi u prvoj fazi protočne strukture. Za ulaze mu se zadaje početna faza i pribroj faze (korak). Oba ulaza su 28 bita široka. Fazni akumulator na svaki rastući brid dodaje pribroj na prošlu vrijednost. Znači na prvi brid dobivamo početnu fazu, pa početnu fazu plus pribroj, zatim ta vrijednost plus pribroj itd. Izlaz iz faznog akumulatora je 28 bita širok te ga dijelimo da dijelove. Dva najviša bita nam služe za određivanje adrese koeficijenata i predznaka tih koeficijenta. Sljedećih osam bitova predstavlja adresu koeficijenta te zadnjih 18 bitova je delta x tj. pozicija uzorka.

Fazni akumulator DDS modula za generaciju signala nosioca se rezlikuje od ostala dva da bi se ispravno mogla implementirati fazna odnosno frekvencijska modulacija. Ima dva dodatna ulaza, dvobitni signal koji određuje vrstu modulacije, faznu ili frekvencijsku, te promjenu koraka faze odnosno frekvencije, ovisno o modulaciji, širine 29 bita, od čega je 28 značajnih bita za promjenu koraka i bit predznaka. U slučaju fazne modulacije promjena koraka dodaje se na akumuliranu vrijednost koraka nosioca čime se mijenja faza, a u slučaju frekvencijske modulacije promjena koraka se dodaje na vrijednost koraka nosioca prije akumulacije čime se mijenja frekvencija. Budući da promjena koraka signal generiran modulom DDS, odnosno signal s predznakom, ovisno o predznaku apsolutna vrijednost promjene koraka se oduzima ili zbraja.

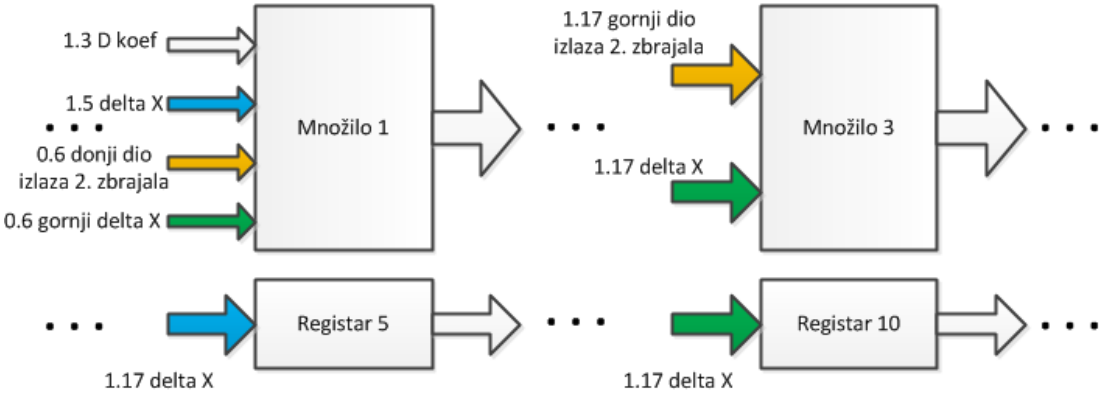
Modul priprema priprema izlaz iz faznog akumulatora koji je ulaz u ovaj modul. Ukoliko su prva dva bita neparna (01 ili 11) tada se sljedećih osam bita komplementira, inače ostaju nepromijenjeni. Donjih 18 bita se iz broja bez predznaka pretvara u broj sa predznakom, komplementiranjem samo najvišeg bita.

Modul ROM je preuzet iz predložaka (*lenguage templates*). Za ROM memoriju iskorištena je jedna dvoportna blok memorija 512x36 bita (RAMB16_S36_S36). U ovom slučaju u memoriju nije moguće ništa pisati te funkcionira kao jednoportna memorija dvostruko veće širine 256x72 bita. Ulaz je osam bitna adresa, a izlaz 72 bitni podatak gdje najviši bit nije iskorišten, obzirom da su širine koeficijenata D 3 bit, C 14 bita, B 23 bita i A 31 bit (ukupno 71 bit). Istim redoslijedom se i nalaze na izlazu, dok su u memoriji zapisani na sljedeći način. Na portu A se nalazi cijeli koeficijent A i najniži bit koeficijenta B, sljedeća četiri bita koeficijenta B nalaze se na paritetnom portu A, a ostatak koeficijenta B i cijeli koeficijent C nalaze se na portu B. Koeficijent D se nalazi na paritetnom portu B. Koeficijenti su u memoriju zapisani bez predznaka.

Modul predznak služi za određivanje predznaka koeficijenata. Predznak koeficijenata se određuje na osnovu dva najviša bita iz faznog akumulatora kao što je prikazano u tablici 2. Koeficijentima se „dodaje“ najviši bit (predznak) te se tada radi određivanje predznaka, što znači da se je širina svih koeficijenata povećala za jedan koeficijent A 32 bita, B 24 bita, C 15 bita i D 4 bita. Ulaz u ovaj modul su koeficijenti bez predznaka i kvadrant dok su izlazi koeficijenti sa predznakom.

Modul mull (množilo) je kao i memorija preuzeta iz predloška (MULT18X18S). Ulazi u množila su dva 18 bitna broja dok je izlaz 36 bitni. Ukoliko se množe manji brojevi (manje širine) tada se oni postave ili na vrh ili na dno s time da treba proširiti predznak do vrha ukoliko se stavlja na dno. Isto tako ovisno gdje su ulazni podaci postavljeni (na dno ili vrh) nalazi se i rezultat (umnožak).

U modelu se koriste, kao što je rečeno, tri množila. Prvo množilo se koristi za prošireno množenje u kombinaciji s trećim množilom gdje bi se trebalo množiti ulazi širi od mogućih (18x24 bita). Dakle, prvo množilo radi prvi umnožak i dio trećeg umnoška iz Hornerove evaluacije polinoma. Na vrhu se nalazi najviših šest bita delta x (uzorka pozicije) i koeficijent D koji se množe dok se na dnu nalazi šest bita delta x (apsolutna vrijednost) i donjih šest bita od drugog množitelja dok se viših 18 bita koristi u trećem množilu. Slika 14 (Brezović, 2011.) prikazuje opisano prošireno množenje.



Slika 14 Prošireno množenje

Na slici su istom bojom označeni podaci koji su jednaki (plava i zelena boja) ili dio nekog drugog podatka (narančasta boja). Registri imaju nazive 5 i 10 iz razloga što se u tim fazama protočne strukture nalaze. Na slici nije prikazano zbrajalo koje zbraja djelomične produkte iz prvog i trećeg množila. Nakon što dobijemo produkte te ih popravimo kao što je opisano u slijedećem podmodulu, radi se zbroj produkta iz trećeg množila 1.29 i produkta iz prvog množila 1.12 koji se posamkne u desno za 17 mjesta da bi mogli odgovarajuće zbroji ta dva produkta te dobili rezultat parcijalnog množenja 1.29.

Modul popravi popravlja rezultate dobivene prvim množilom obzirom da se u njemu radi prošireno množenje. Kako se na vrhu nalazi produkt koeficijenta D i delta x (devet bita širina produkta), a na dnu produkt donjeg dijela iz drugog zbrajala i najviših šest bita (apsolutna vrijednost) delta x (12 bita širina produkta) te u sredini mješoviti produkt ovih vrijednosti taj

mješoviti produkt može utjecati na produkt na koji se nalazi na vrhu. Prva dva bita cijelog izlaza iz prvog množila predstavljaju predznak prvog produkta i iz toga razloga se najznačajniji bit ne gleda. Od 34 do 26 bita se nalazi prvi produkt i na njega može utjecati mješoviti produkt, stoga na 25 bit dodajemo jedinicu da bi „neutralizirali“ moguće djelovanje mješovitoga produkta. Donji produkt treba popraviti na način da ukoliko je delta x bio minus jedan tada se taj produkt ne uzima u obzir već se samo „propušta“ donjih šest bita iz drugog množila i pretvara u negativan broj (simulacija množenja s minus jedan). Ukoliko delta x nije minus jedan tada je donji produkt umnožak dviju vrijednosti bez predznaka, stoga produkt treba pretvoriti u produkt s predznakom u ovisnosti o delta x koji određuje je li produkt pozitivan ili negativan. Ulazi u modul su izlaz iz prvog množila (svih 36 bita), delta x (gornjih sedam bita) i donji dio iz drugog zbrajala (šest bita). Prvi podatak dolazi iz pete faze protočne strukture dok druga dva dolaze iz 10 faze protočne strukture.

Modul generičko zbrajalo radi posmak u desno jednog ulaza (poravnavanje težina) te nakon toga zbrajanje s drugim ulazom koji je jednake širine kao i prvi nakon posmaka i prijenosom koji nije standardni prijenos već služi za zaokruživanje prvog ulaza. Postoje dva generička parametra. Prvi parametar je broj posmaka u desno, a drugi parametar je oznaka širine prvog ulaza i izlaza. Drugi ulaz je umanjen za broj posmaka u desno te se taj podatak posmiče u desno na način da se najviši bit proširi do vrha za broj posmaka koji je zadan prvim generičkim parametrom.

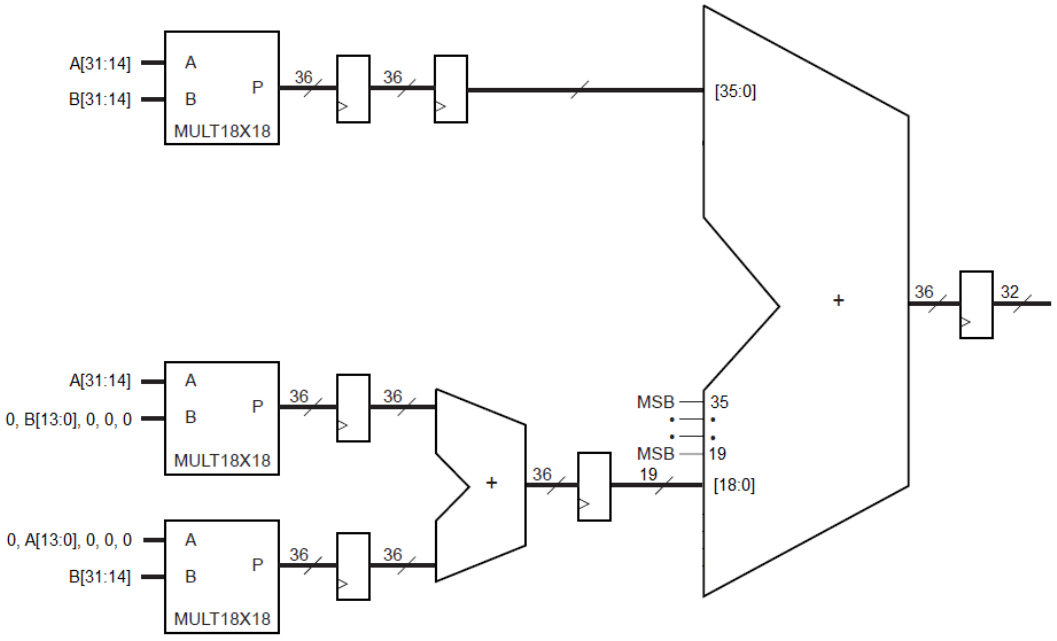
Modul PM_i_FM

U modulu se ovisno o odabranoj faznoj ili frekvencijskoj modulaciji obrađuje modulacijski signal fazne i frekvencijske modulacije. Ulaz u modul su 32-bitni modilacijski signal te signal tipke za odabir modulacije, dok su izlazi promjena koraka širine 29 bita, dvobitni signal za indikaciju vrste modulacije, te dva jednobitna signala za uključivanje signalnih dioda. U slučaju fazne modulacije prosljeđuje se 29 bita modulacijskog signala, a u slučaju frekvencijske prosljeđuje se modulacijski signal skaliran obrnuto proporcionalno frekvenciji modulacijskog signala. Modul omogućuje uključivanje fazne ili frekvencijske modulacije odnosno njihovo isključivanje pritiskom tipke. LED diode ld5 i ld4 indiciraju koja je modulacija upaljena odnosno ugašena.

Modul AM

Amplitudna modulacija implementirana je množenjem signala nosioca i modulacijskog signala amplitudne modulacije. Budući da su oba signala širine 32 bita, a sklop Spartan-3E sadrži samo 18-bitna množila bilo je potrebno pomoću ugrađenih 18-bitnih množila izraditi 32-bitno množilo koje je u modul AM ugrađeno kao podmodul mull32. U modulu se može birati između amplitudne modulacije s potisnutim nosiocem i klasične amplitudne modulacije. Prva je ostvarena samo množenjem nosioca i modulacijskog signala, dok se pri drugoj rezultat spomenutog množenja i signal nosioca dijele s 2 te zbrajaju. Djeljenje je potrebno da bi zbroj ovih dvaju signala ostao u dinamici ±1. U modul je ugrađena i mogućnost gašenja i paljenja klasične amplitudne modulacije odnosno amplitudne modulacije s potisnutim nosiocem pritiskom tipke. LED diode ld7 i ld6 indiciraju da li je jedna od modulacija upaljena ili ugašena.

Mull32 sastoji se od tri 18-bitna množila preuzeta iz predloška (MULT18X18S) i pet generičkih registara za kašnjenje signala. Prvo možilo množi gornjih 18 bita obaju signala, dok druga dva množe gornjih 18 bita jednog signala i donjih 14 bita drugog signala. Dobiveni umnošci trebaju se ispravno poravnati i zbrojiti u konačni rezultat. Umnošci iz druga dva množila se prvo međusobno zbroje, te se gornjih 19 bita zbroja predznačno proširi do 36 bita i zbraja s rezultatom prvog množila. Dobiveni zbroj uvećava se za 1 LSB da bi se izbjegla pogreška do koje dolazi pri množenju vrlo malih negativnih brojeva. MSB zbroja je udvostručeni predznak te se zanemaruje kao i najniža tri bita zbog zaokruživanja na 32 bita. Opisana implementacija množila prikazana je slikom 15.



Slika 15 Implementacija 32-bitnog množila

Modul attenuate

U modulu se skaliraju modulacijski signali indeksom modulacije. Indeks modulacije radi jednostavnosti izvedbe i prenosivosti (izbjegavanje množila) te da bi se izbjegla premodulacija klasičnog amplitudno moduliranog signala ograničen je na vrijednosti 2^{-n}, $n \in [0, 31]$, te je skaliranje ostvareno aritmetičkim posmakom ulaznog signala za n u desno. Ulazi u modul su dva jednobitna signala koji mijenjaju n i 32-bitni modulacijski signal, a izlaz skalirani 32-bitni modulacijski signal. Modul je iskorišten dva puta, jedan za faznu i frekvencijsku modulaciju, a drugi za amplitudnu modulaciju.

Modul encoder01

Modul obrađuje signale s tipki rotacijskog enkodera te generira dva signala koji indiciraju da je došlo do pomaka rotacijskog enkodera te smijer tog pomaka. Uloga rotacijskog enkodera je promjena indeksa modulacije modulacijskih signala kad je preklopka sw3 u položaju '0', odnosno promjena koraka faze jednog od tri DDS modula kad je sw3 u položaju '1'.

Modul odabir_atenuacije

Budući da postoje dva modulacijska signala, a samo jedan enkoder, ovaj modul omogućuje odabir indeksa modulacije koji se želi mijenjati rotacijskim enkoderom. Implementiran je kao sklopka koja spaja izlaze enkodera na odabrani atenuator. Kad je preklopka sw3 u položaju '1' ovaj modul je isključen.

Modul korak

Modul služi za promjenu iznosa koraka faze za odabrani DDS. U modulu se pomoću tipki može odabrati željeni od tri DDS modula, te bit unutar koraka koji će se mijenjati. Sama promjena vrši se rotacijskim enkoderom. Modul se pali prebacivanjem preklopke sw3 u položaj '1'.

Modul sklopka32

Sklopka32 omogućuje odabir izvora modulacijskog signala, koji može biti DDS modul ili signal doveden izvana. Modul je korišten dva puta, za faznu i frekvencijsku, odnosno amplitudnu modulaciju, te prosljeđuje odabrani signal u modul za faznu i frekvencijsku, odnosno amplitudnu modulaciju. (NAPOMENA: zbog ograničenih resursa signali nisu dovedeni izvana nego su im vrijednosti postavljene na nulu. U slučaju implementacije na platformi s dovoljno resursa potrebno je promjeniti komentare u kodu te dodjeliti priključke signalima)

Modul debouncing

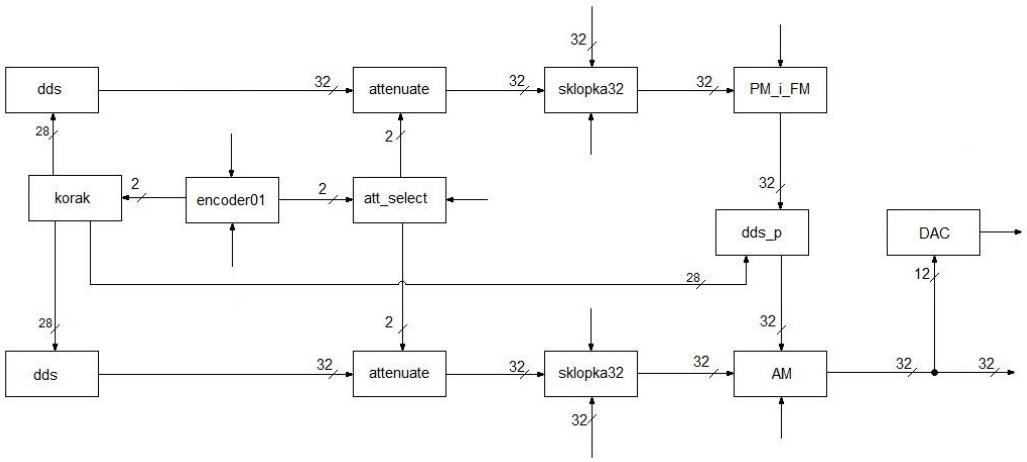
Budući da na razvojnoj pločici nije riješen problem istitravanja kontakata tipki i preklopki, taj problem je bilo potrebno riješiti programski. Problem je riješen jednostavnom pretpostavkom da osoba nije sposobna pri pritisku tipke tipku držati pritisnutom kraće od 5 ms. Svi signali trajanja kraćeg od tog vremena smatraju se istitravanjem i zanemaruju. Najveće istitravanje imaju preklopke sw0-sw3 koje iznosi 2 ms te odabrani period pokriva sva moguća istitravanja.

Modul DAC

Na razvojnoj pločici nalazi se 12-bitni DA pretvornik LTC2624 firme Linear Technology koji je iskorišten da bi se dobila analogna reprezentacija rezultata. Pretvornik na ulazu očekuje vrijednosti bez predznaka pa je u ovom modulu dinamika signala smanjena s $[-1, 1\rangle$ na $[0, 1\rangle$, koja je reprezentirana naponskim razinama $[0, 3.3\rangle V \pm 5\%$. Komunikacija s pretvornikom ostvarena je SPI sabirnicom 24-bitnim protokolom. Budući da je to serijski protokol zabog brzine rada sklopa nije moguće ostvariti pretvorbu svakog izlaznog podatka nego svakog 25., što uz smanjenje širine riječi na 12 bita predstavlja značajan gubitak informacije i rezulucije, no ipak je dostatno za vizualno predočenje rezultata.

Modul signalni_generator

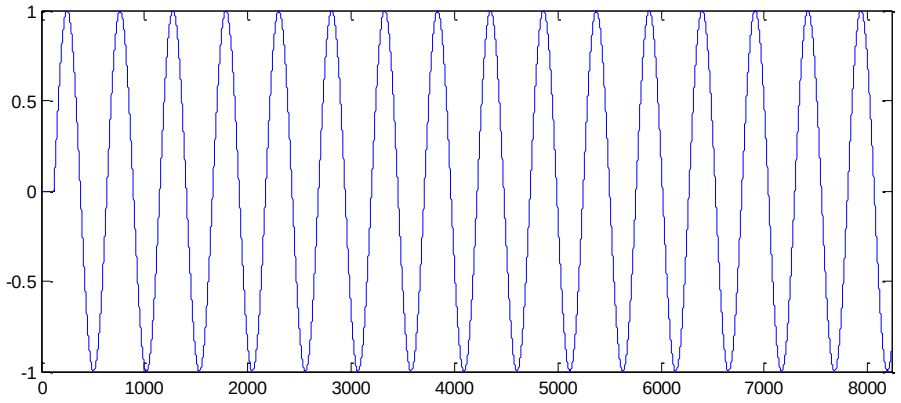
Ovaj modul je glavni modul u kojem su povezani svi ostali moduli. Blokovska shema na slici 16 prikazuje kako su povezani pojedini moduli. Moduli debouncig su izostavljeni radi preglednosti.



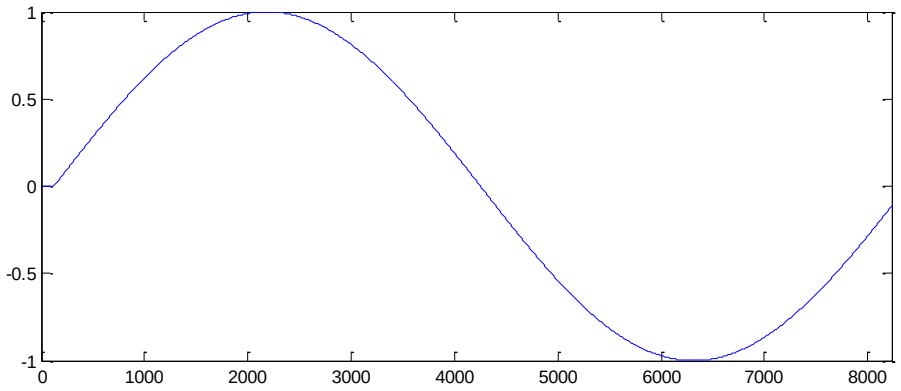
Slika 16 Blokovska shema sklopa

4.4. Simulacija rada sklopa

Simulacija rada sklopa izvedena je u programskom paketu Matlab, verzija 7.5, te Simulinku. Radi jednostavnosti svaka modulacija ispitivana je zasebno. Modeli izrađeni u simulinku kojima je ispitivana ispravnost rada sklopa prikazani su u privitku. Signal nosioca i modulacijski signal korišteni za simulaciju sve četiri modulacije prikazani su slikama 17 i 18.



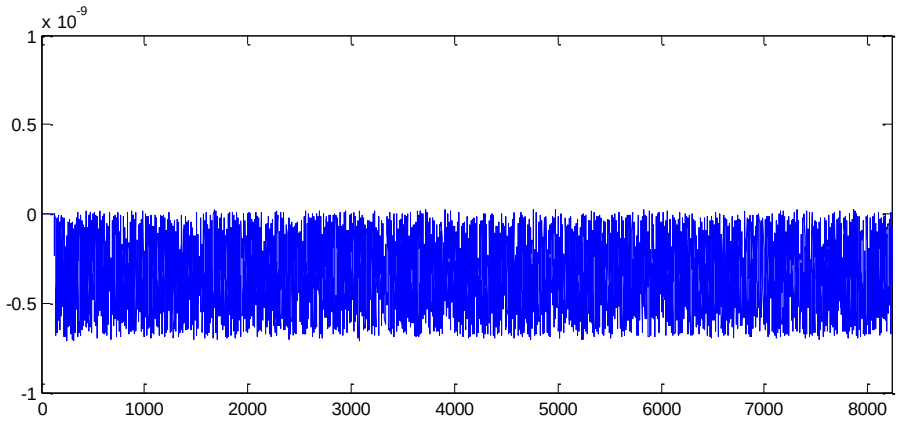
Slika 17 Signal nosioca



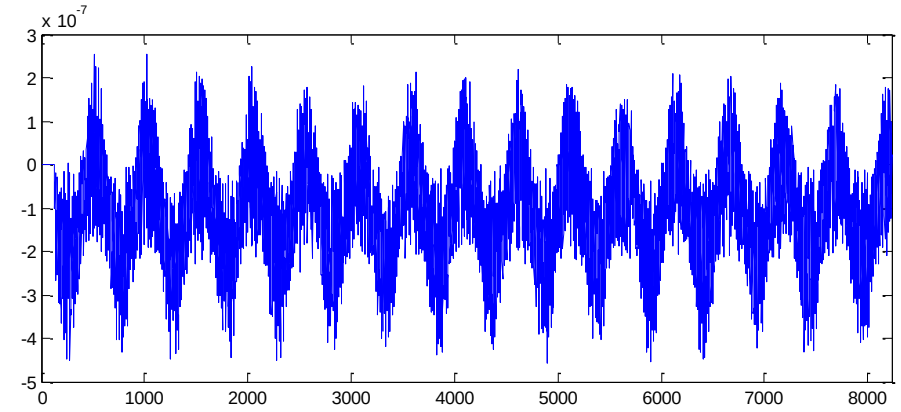
Slika 18 Modulacijski signal

Amplitudne modulacije

Razlike signala klasične amplitudne modulacije i amplitudne modulacije s potisnutim nosiocem dobivenih simulacijama rada samog sklopa u iSim simulatoru i rada referentnog modela u Simulinku prikazani su slikama 19 i 20. Vidljivo je da su pogrješke klasične amplitudne modulacije vrlo male te ne prelaze iznos 1e-9, dok su pogrješke amplitudne modulacije s potisnutim nosiocem nešto veće, no manje od 5e-7.



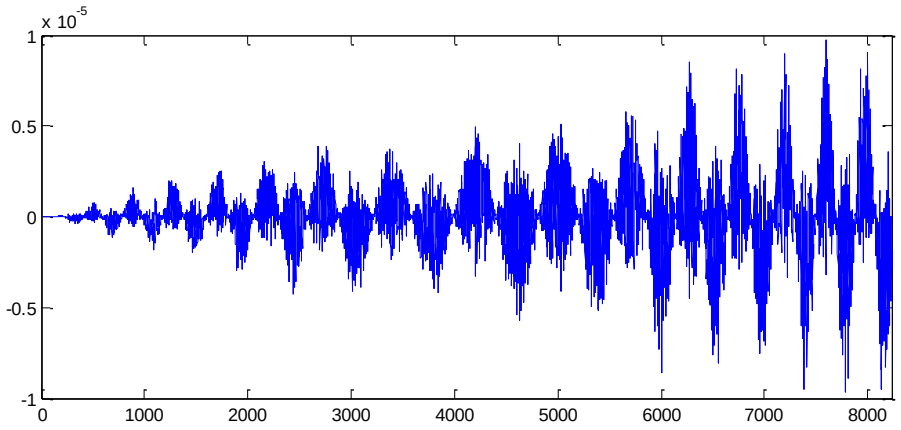
Slika 19 Odstupanje klasične amplitudne modulacije



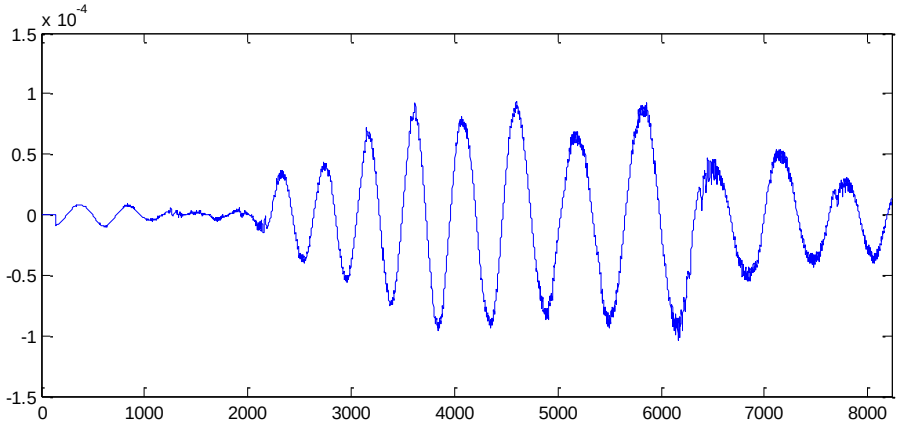
Slika 20 Odstupanje amplitudne modulacije s potisnutim nosiocem

Kutne modulacije

Razlike signala fazne i frekvencijske modulacije dobivenih simulacijama rada samog sklopa u iSim simulatoru i rada referentnog modela u Simulinku prikazani su slikama 21 i 22 na kojima je vidljivo da su odstupanja fazne modulacije relativno mala, ali se akumuliraju te rastu s vremenom, a odstupanja frekvencijske modulacije su razmjerno velikog iznosa, i do 1e-4.



Slika 21 Odstupanje fazne modulacije



Slika 22 Odstupanje frekvencijske modulacije

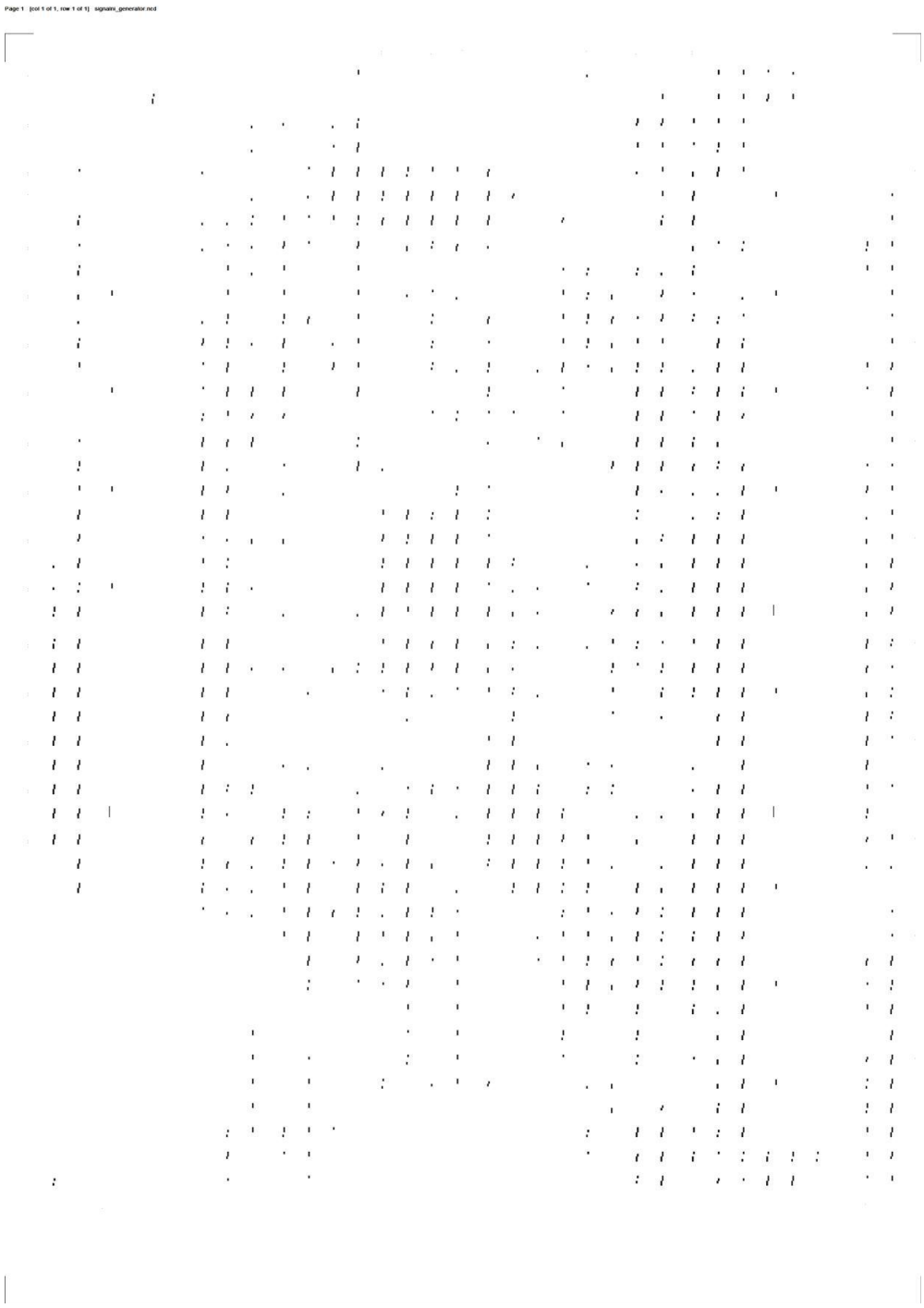
4.5. Implementacija sklopa

Slika 23 daje pregled ukupnih resursa Spartan-3E sklopa i postotak njihova iskorištenja implementiranim dizajnom. Vidi se da je postotkom najkorištenija komponenta množilo, njih čak 12, po tri za svaki DDS modul i dodatna tri za 32-bitno množilo.

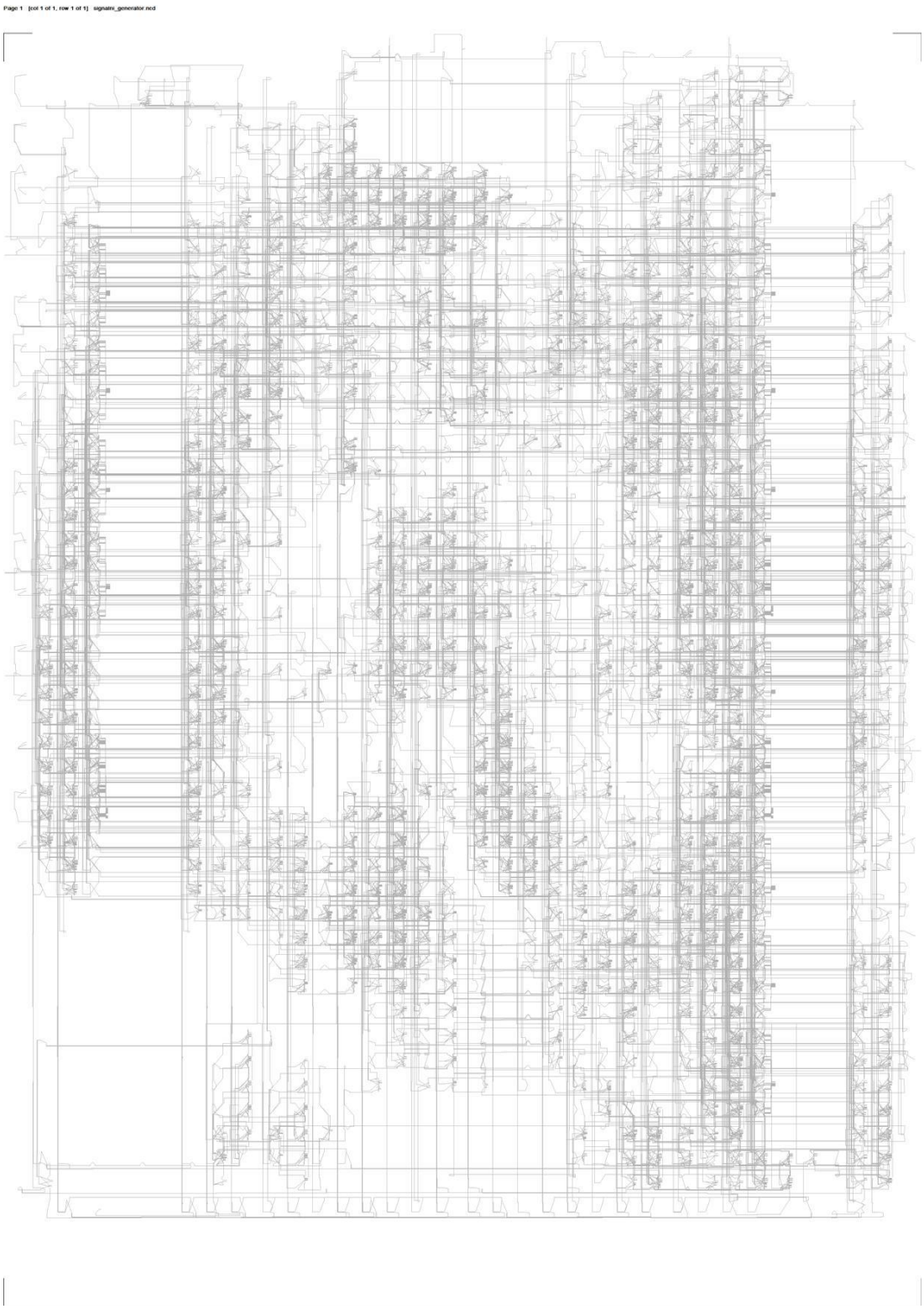
Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	2,455	9,312	26%	
Number of 4 input LUTs	2,206	9,312	23%	
Number of occupied Slices	1,724	4,656	37%	
Number of Slices containing only related logic	1,724	1,724	100%	
Number of Slices containing unrelated logic	0	1,724	0%	
Total Number of 4 input LUTs	2,503	9,312	26%	
Number used as logic	1,987			
Number used as a route-thru	297			
Number used as Shift registers	219			
Number of bonded IOBs	50	232	21%	
Number of RAMB16s	3	20	15%	
Number of BUFGMUXs	1	24	4%	
Number of MULT18X18SIOs	12	20	60%	
Average Fanout of Non-Clock Nets	2.30			

Slika 23 Iskorištenje FPGA sklopa

Slika 24 grafički prikazuje iskorištenje resursa, odnosno razmještaj komponenti, a slika 25 prikazuje vodove iskorištene za spajanje komponenti.



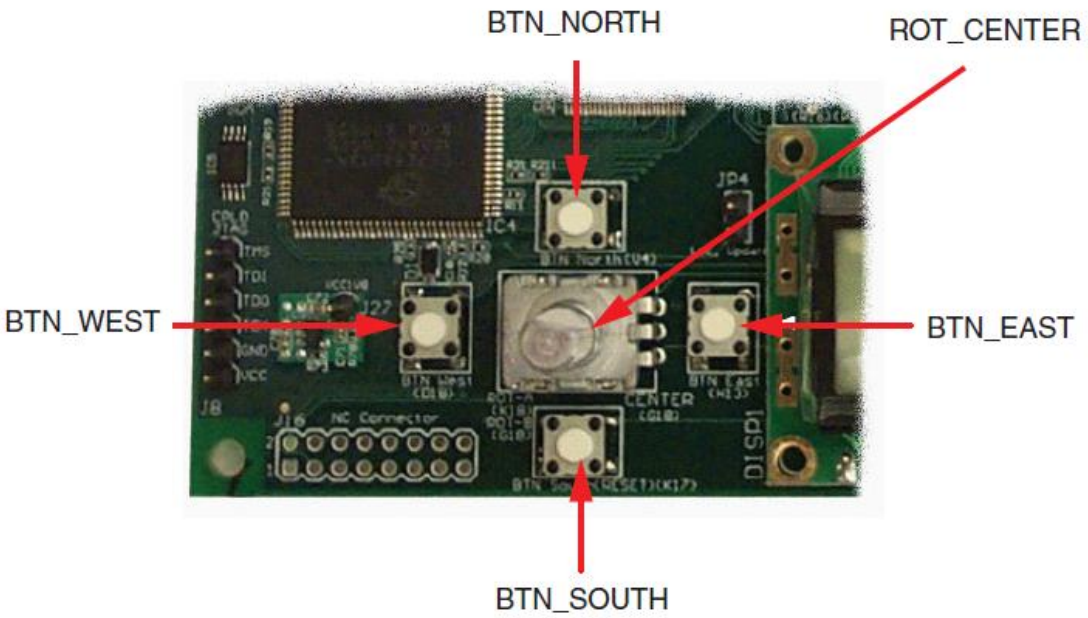
Slika 24 Raspored komponenti na FPGA sklopu



Slika 25 Iskorišteni vodovi FPGA sklopa

4.6. Upute za korištenje

Slike 26 i 27 (Xilinx, 2011) prikazuju preklopke, svijetleće diode, tipke i rotacijski enkoder sklopa. Kontakt tipke ROT_CENTER koja služi za reset sklopa ostvaruje se pritiskom rotacijskog enkodera.



Slika 26 Tipke i rotacijski enkoder

Slika 27 Preklopke i svijetleće diode

Nakon paljenja odnosno reseta sklopa sve početne faze i fazni koraci modulacijskih signala inicijaliziraju na vrijednost $7f00_{16}$, a nosioca na $7ff00_{16}$, dok se ostali signali inicijaliziraju na vrijednost 0_{16}, no vrlo je važno obratiti pozornost na preklopke sw0 – sw3 jer njihova vrijednost ovisi o položaju preklopke i ne mijenja se resetom. Inicijalno su sve modulacije isključene. Amplitudna modulacija se pali i gasi tipkom BTN_WEST, a uključivanje amplitudne modulacije očituje se paljenjem LED diode ld7. Fazna i frekvencijska modulacija se pale i gase tipkom BTN_EAST, a njihovo uključivanje očituje se paljenjem LED diode LD6 za faznu, odnosno LD5 za frekvencijsku modulaciju.

Preklopka sw3 u položaju '0' omogućuje podešavanje atenuacije. Koji signal se atenuira određuje preklopka sw0, u položaju '0' modulacijski signal amplitudne modulacije, a u položaju '1' fazne i frekvencijske modulacije. Iznos atenuacije se mijenja rotacijskim enkoderom. Kad je sw3 u položaju '1' omogućena je promjena koraka DDS-a. Kojim se DDS-om upravlja indicirano je LED diodama, a promjena odabranog DDS-a vrši se pritiskom na tipku BTN_NORTH. Za odabrani DDS nosioca pali se LD0, za DDS fazne i frekvencijske modulacije LD1, a za DDS amplitudne modulacije LD2. Nakon promjene odabranog DDS-a ili reseta sklopa bit kojim se upravlja postavlja se na MSB. Pritiskom na tipku BTN_SOUTH upravljenje se prebacuje na idući bit. Budući da je korak 28-bitni signal, nakon 27 pritisaka tipke upravljanje je pozicionirano na LSB. Idućim pritiskom upravljanje se ponovno vraća na MSB. Potrebno je malo koncentracije i pažnje pri odabiru bita kojim se upravlja budući da nema indikatora koji bit je odabran. Promjena koraka vrši se rotacijom enkodera na način da se za odabrani n-ti bit korak uveća ili umanji za 2^n ovisno o smijeru rotacije enkodera.

Preklopke sw1 i sw2 u položaju '0' propuštaju modulacijske signale modula DDS, dok u položaju '1' sw1 propušta izvana priključeni modulacijski signal za amplitudnu modulaciju, a sw2 za faznu i frekvencijsku modulaciju. Kako je već napomenuto ovi signali nisu zapravo fizički dovedeni zbog ograničenih resursa, nego im je vrijednost trajno postavljena u 0_{16}.

Zaključak

Idealni sinusni signal neizostavan je u mnogim primjenama kao referentni signal s kojim se vrše usporedbe testiranog signala s aspekta frekvencijskih ili vremenskih karakteristika signala. Služi i kao nosilac kod moduliranih signala, a takvim principom se danas realizira gotovo svaki prijenos signala s kojima se susrećemo svaki dan.

Potpuno idealan signal nemoguće je realizirati, no implementacija korištenog DDS sklopa daje sinusni signal vrlo malih odstupanja od idealnog, te kao takva je idealna kao osnova za daljnju obradu.

Za potrebe ovog diplomskog rada DDS sklop koristi se ne samo kao generator nosioca moduliranih signala, već i kao generator modulirajućih signala ukoliko oni nisu dovedeni izvana.

Uređaj visoke preciznosti koji generira sinusni signal s mogućnošću amplitudne, fazne, frekvencijske i impulsne modulacije naziva se signalni generator. Ovakav uređaj postoji već više desetaka godina u analognom obliku, gdje je za dobivanje sinusnog signala korišten LC oscilator, a u digitalnom obliku postoji u zadnjih nekoliko godina, te je još u fazi razvoja.

Budući da je ovaj uređaj implementiran u ovom radu zasnovan na DDS visoke preciznosti, omogućuje i generiranje moduliranih signala visoke preciznosti. No kako je kao platforma za implementaciju korištena Spartan-3E Starter Kit Board razvojna pločica, usljed ograničenih resursa sklop se ne može koristi u punoj funkcionalnosti, ponajviše zbog 12-bitnog DA pretvornika niske brzine rada.

Također postoji mogućnost dovođenja digitalnog signala na priključke pločice čime bi se omogućila bolja DA pretvorba, no to zahtjeva dodatni DA pretvornik veće rezolucije i brzine rada. Idealno bi bilo razviti sklop implementiran na razvojnoj pločici s primjerenim DA pretvornikom.

Liteatura

[1] Marko Brezović : „Poboljšanje digitalnog generatora sinusnog signala“, diplomski projekt, 2011.

[2] Davor Petrinović, Marko Brezović : „Spline Based High-Accuracy Piecewise-Polynomial Phase to Sinusoid Amplitude Converters“, 2009.

[3] Antonela Šipić : „DIREKTNA DIGITALNA SINTEZA IZNIMNO VISOKE TOČNOSTI“, diplomski rad, 2009.

[4] Mladen Vučić : „Obrada signala u komunikacijama“, predavanje 8, 2010.

[5] Mladen Vučić, Goran Molnar : „ALATI ZA RAZVOJ DIGITALNIH SUSTAVA“, materijali za predavanja, 2010.

[6] Mladen Vučić, Goran Molnar, Marko Butorac : „ALATI ZA RAZVOJ DIGITALNIH SUSTAVA“, Upute za praktični rad, 2010.

[7] Xilinx : „Using Embedded Multipliers in Spartan-3 FPGAs“, 2003.

[8] Xilinx : „Spartan-3E FPGA Starter Kit Board User Guide“, 2011.

[9] Linear Tehnology: „LTC2624 Datasheet“, 2004.

SKLOPOVSKI IZVEDEN DIGITALNI GENERATOR SINUSNOG SIGNALA S AMPLITUDNOM, FAZNOM I FREKVENCIJSKOM MODULACIJOM

Sažetak

U okviru diplomskog rada razvijen je sklop s mogućnošću klasične amplitudne modulacije, amplitudne modulacije s potisnutim nosiocem, fazne i frekvencijske modulacije. Svim karakteristikam signala moguće je upravljati. Sklop ovih mogućnosti ima primarnu ulogu u generiranju ispitnih signala za druge elektroničke uređaje, bilo za testiranje, dizajn ili popravak, te obično ima naziv funkcijski ili signalni generator. Sklop je temeljen na direktnoj digitalnoj sintezi sinusnog signala visoke preciznosti, te implementiran na Spartan-3E Starter Kit Board razvojnoj pločici i radi na taktu 50 MHz koji daje oscilator ugrađen na pločici, no teoretski može raditi i na taktu do 130 MHz.

Ključne riječi

Direktna digitalna sinteza (DDS), amplituna modulacija, fazna modulacija, frekvencijska modulacija, signalni generator.

HARDWARE IMPLEMENTATION OF DIGITAL SINE GENERATOR WITH AMPLITUDE, PHASE AND FREQUENCY MODULATION

Summary

In the thesis a device with classic amplitude modulation, amplitude modulation with suppressed carrier, phase and frequency modulation capabilities was developed. Every attribute of the signals is adjustable. Such device has primary application as test signal generator for other electronic devices, weather for designing, testing or repairing, and is usually called function or signal generator. The device is based on high-precision direct digital synthesis sine generator and implemented on Spartan-3E Starter Kit Board. It runs on 50 MHz clock provided by onboard oscillator, but is capable of running on up to 130 MHz.

Keywords

Direct digital synthesis (DDS), amplitude modulation, phase modulation, frequency modulation, signal generator.

Privitak

Strojni kod sklopa

```
entity signalni_generator is
   Port ( clk                : in  STD_LOGIC;
          rst                : in  STD_LOGIC;
          AM_on_off          : in  STD_LOGIC; -- tipka za on/off AM
          PM_FM_off          : in  STD_LOGIC; -- tipka za PM/FM/nijedno
          position           : in  STD_LOGIC; -- tipka za odabir bita unutar koraka
          dds_btn            : in  STD_LOGIC; -- tipka za odabir dds-a
          AM_int_ext         : in  STD_LOGIC; -- preklopka za AM vanjskim/unutarnjim sig.
          PM_FM_int_ext      : in  STD_LOGIC; -- preklopka za PM/FM vanjskim/unutarnjim sig.
          ATT                : in  STD_LOGIC; -- preklopka za odabir atenuacije AM ili PM/FM
          on_off             : in  STD_LOGIC; -- preklopka za odabir att ili korak
          rot_a              : in  STD_LOGIC; -- s encodera
          rot_b              : in  STD_LOGIC; -- s encodera
--        faza1       : in  STD_LOGIC_VECTOR (27 downto 0);
--        korak1      : in  STD_LOGIC_VECTOR (27 downto 0);
--        faza2       : in  STD_LOGIC_VECTOR (27 downto 0);
--        korak2      : in  STD_LOGIC_VECTOR (27 downto 0);
--        faza0       : in  STD_LOGIC_VECTOR (27 downto 0);
--        korak0      : in  STD_LOGIC_VECTOR (27 downto 0);
--        P_F_mod     : in  STD_LOGIC_VECTOR (31 downto 0); -- vanjski modulacijski za PM ili FM
--        AM_mod      : in  STD_LOGIC_VECTOR (31 downto 0); -- vanjski modulacijski za AM
          AM_led             : out STD_LOGIC; -- LD7
          AM_SC_led          : out STD_LOGIC; -- LD6
          PM_led             : out STD_LOGIC; -- LD5
          FM_led             : out STD_LOGIC; -- LD4
          led0               : out STD_LOGIC; -- LD0 za dds0-nosilac
          led1               : out STD_LOGIC; -- LD1 za dds1-FMPM
          led2               : out STD_LOGIC; -- LD2 za dds2-AM
          izlaz_sklopa       : out STD_LOGIC_VECTOR (31 downto 0);
          CS                 : out STD_LOGIC;
          CLR                : out STD_LOGIC;
          SCK                : out STD_LOGIC;
          MOSI               : out STD_LOGIC;
          SPI_SS_B           : out STD_LOGIC;
          AMP_CS             : out STD_LOGIC;
          AD_CONV            : out STD_LOGIC;
          SF_CE0             : out STD_LOGIC;
          FPGA_INIT          : out STD_LOGIC
          );

end signalni_generator;


architecture rtl of signalni_generator is

signal faza1       :  STD_LOGIC_VECTOR (27 downto 0) := x"0000000";
signal faza2       :  STD_LOGIC_VECTOR (27 downto 0) := x"0000000";
signal faza0       :  STD_LOGIC_VECTOR (27 downto 0) := x"0000000";
signal P_F_mod     :  STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
signal AM_mod      :  STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
signal AM_out      :  STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
```

```
component debouncing is
                    port (    clk             : in      std_logic;
                              rst             : in      std_logic;
                              ulaz            : in      std_logic;
                              izlaz           : out     std_logic
                             );
end component;

signal AM_on_off_d          : std_logic := '0';
signal PM_FM_off_d          : std_logic := '0';
signal AM_int_ext_d         : std_logic := '0';
signal PM_FM_intext_d       : std_logic := '0';
signal ATT_d                : std_logic := '0';
signal rot_a_d              : std_logic := '0';
signal rot_b_d              : std_logic := '0';
signal on_off_d             : std_logic := '0';
signal position_d           : std_logic := '0';
signal dds_btn_d            : std_logic := '0';


component dds is
   Port ( clk               : in  STD_LOGIC;
          rst               : in  STD_LOGIC;
          faza              : in  STD_LOGIC_VECTOR (27 downto 0);
          korak             : in  STD_LOGIC_VECTOR (27 downto 0);
          rez               : out STD_LOGIC_VECTOR (31 downto 0)
           );
end component;
component dds_p is
   Port ( clk               : in  STD_LOGIC;
          rst               : in  STD_LOGIC;
          modulacija        : in        std_logic_vector (1 downto 0);
          faza              : in  STD_LOGIC_VECTOR (27 downto 0);
          delta_faze        : in  STD_LOGIC_VECTOR (28 downto 0);
          korak             : in  STD_LOGIC_VECTOR (27 downto 0);
          rez               : out STD_LOGIC_VECTOR (31 downto 0)
          );
end component;

signal     dds0, dds1, dds2 : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');


component PM_i_FM is
          Port(     clk               : in  STD_LOGIC;
                    rst               : in  STD_LOGIC;
                    PM_FM_off         : in  STD_LOGIC;  -- odabir vrste modulacije
                    mod_sig           : in  STD_LOGIC_VECTOR (31 downto 0); -- modulacijski signal
                    korak             : in  STD_LOGIC_VECTOR (27 downto 0);
                    PM_led            : out STD_LOGIC;
                    FM_led            : out STD_LOGIC;
                    modulacija_o      : out std_logic_vector (1 downto 0);
                    delta_faze        : out STD_LOGIC_VECTOR (28 downto 0)
                    );
end component;

signal     modulacija : STD_LOGIC_VECTOR (1 downto 0) := (others => '0');
signal     delta_faze : STD_LOGIC_VECTOR (28 downto 0) := (others => '0');
```

```
component AM is
        port(   clk             : in  STD_LOGIC;
                rst             : in  STD_LOGIC;
                am_mod_in       : in  STD_LOGIC; -- AM uključena / isključena
                mod_sig         : in  STD_LOGIC_VECTOR (31 downto 0); -- modulacijski signal
                d_in            : in  STD_LOGIC_VECTOR (31 downto 0);
                AM_led          : out STD_LOGIC;
                AM_SC_led       : out STD_LOGIC;
                d_out           : out STD_LOGIC_VECTOR (31 downto 0)
                 );
end component;


component ATTENUATE is
        port (  clk             : in  STD_LOGIC;
                rst             : in  STD_LOGIC;
                r_event         : in      std_logic;
                r_direction     : in      std_logic;
                d_in            : in      std_logic_vector (31 downto 0);
                d_out           : out     std_logic_vector (31 downto 0)
                );
end component;

signal  AM_ATT, PM_FM_ATT : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');


component att_select is
        port (   clk            : in  STD_LOGIC;
                 rst            : in  STD_LOGIC;
                 tipka          : in      std_logic;
                 on_off         : in      std_logic;
                 r_event        : in      std_logic;
                 r_direction    : in      std_logic;
                 AM_event       : out     std_logic;
                 AM_direction   : out     std_logic;
                 FPM_event      : out     std_logic;
                 FPM_direction  : out     std_logic
                );
end component;

signal  AM_event, AM_direction, FPM_event, FPM_direction : std_logic := '0';


component korak is
        port (  clk             : in  STD_LOGIC;
                rst             : in  STD_LOGIC;
                on_off          : in  STD_LOGIC; -- preklopka za uključivanje
                dds_btn         : in  STD_LOGIC; -- tipka za odabiranje dds-a
                position        : in  STD_LOGIC; -- tipka za odabir pozicije bita u koraku
                r_event         : in      std_logic;
                r_direction     : in      std_logic;
--              korak0          : in  STD_LOGIC_vector (27 downto 0);
--              korak1          : in  STD_LOGIC_vector (27 downto 0);
--              korak2          : in  STD_LOGIC_vector (27 downto 0);
                led0            : out STD_LOGIC;
                led1            : out STD_LOGIC;
                led2            : out STD_LOGIC;
                korak0_out      : out STD_LOGIC_vector (27 downto 0);
                korak1_out      : out STD_LOGIC_vector (27 downto 0);
                korak2_out      : out STD_LOGIC_vector (27 downto 0)
                );
end component;

signal korak0_out, korak1_out, korak2_out : STD_LOGIC_vector (27 downto 0) := (others => '0');
```

```
component sklopka32 is
        port (      clk             : in  STD_LOGIC;
                    rst             : in  STD_LOGIC;
                    int_ext         : in  STD_LOGIC; -- tipka
                    prvi_ulaz       : in        std_logic_vector (31 downto 0); -- unutarnji
                    drugi_ulaz      : in        std_logic_vector (31 downto 0); -- vanjski
                    odabrani        : out       std_logic_vector (31 downto 0)
                  );
end component;

signal      PM_FM_selected, AM_selected : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');


component encoder01 is
  port  (
            clk                     : in std_logic;
            reset                   : in std_logic;

            rot_a                   : in std_logic;
            rot_b                   : in std_logic;

            rotary_event            : out std_logic;
            rotary_direction        : out std_logic
      );
end component;

signal      r_event, r_direction : std_logic := '0';


component DAC is
   Port ( clk       : in  STD_LOGIC;
          rst       : in  STD_LOGIC;
          ulaz      : in        std_logic_vector(11 downto 0);

          CS        : out STD_LOGIC;
          CLR       : out STD_LOGIC;
          SCK       : out STD_LOGIC;
          MOSI      : out  STD_LOGIC;

          SPI_SS_B          : out STD_LOGIC;
          AMP_CS            : out STD_LOGIC;
          AD_CONV           : out STD_LOGIC;
          SF_CE0            : out STD_LOGIC;
          FPGA_INIT         : out STD_LOGIC
          );
end component;

begin

          AMonoff :         component debouncing
                                port map (      clk             =>      clk,
                                                rst             =>      rst,
                                                ulaz            =>      AM_on_off,

                                                izlaz           =>      AM_on_off_d
                                          );

          PMFMoff :         component debouncing
                                port map (      clk             =>      clk,
                                                rst             =>      rst,
                                                ulaz            =>      PM_FM_off,

                                                izlaz           =>      PM_FM_off_d
                                          );
```

```
AMintext_d :    component debouncing
                port map (      clk             =>      clk,
                                rst             =>      rst,
                                ulaz            =>      AM_int_ext,

                                izlaz           =>      AM_int_ext_d
                );

PMFMintext_d :  component debouncing
                port map (      clk             =>      clk,
                                rst             =>      rst,
                                ulaz            =>      PM_FM_int_ext,

                                izlaz           =>      PM_FM_intext_d
                );

ATTd :          component debouncing
                port map (      clk             =>      clk,
                                rst             =>      rst,
                                ulaz            =>      ATT,

                                izlaz           =>      ATT_d
                );

rota :          component debouncing
                port map (      clk             =>      clk,
                                rst             =>      rst,
                                ulaz            =>      rot_a,

                                izlaz           =>      rot_a_d
                                                );

rotb :          component debouncing
                port map (      clk             =>      clk,
                                rst             =>      rst,
                                ulaz            =>      rot_b,

                                izlaz           =>      rot_b_d
                );

onoff :         component debouncing
                port map (      clk             =>      clk,
                                rst             =>      rst,
                                ulaz            =>      on_off,

                                izlaz           =>      on_off_d
                );

pos :           component debouncing
                port map (      clk             =>      clk,
                                rst             =>      rst,
                                ulaz            =>      position,

                                izlaz           =>      position_d
                );

ddsbtn :        component debouncing
                port map (      clk             =>      clk,
                                rst             =>      rst,
                                ulaz            =>      dds_btn,

                                izlaz           =>      dds_btn_d
                );
```

```
FMPM_dds :      component dds
                    port map (      clk             =>      clk,
                                    rst             =>      rst,
                                    faza            =>      faza1,
                                    korak           =>      korak1_out,

                                    rez             =>      dds1
                                );

AM_dds :        component dds
                    port map (      clk             =>      clk,
                                    rst             =>      rst,
                                    faza            =>      faza2,
                                    korak           =>      korak2_out,

                                    rez             =>      dds2
                                );

nosilac_dds :   component dds_p
                    port map (      clk             =>      clk,
                                    rst             =>      rst,
                                    modulacija      =>      modulacija,
                                    faza            =>      faza0,
                                    delta_faze      =>      delta_faze,
                                    korak           =>      korak0_out,

                                    rez             =>      dds0
                                );

FMPM :          component PM_i_FM
                    port map (      clk             =>      clk,
                                    rst             =>      rst,
                                    PM_FM_off       =>      PM_FM_off_d,
                                    mod_sig         =>      PM_FM_selected,
                                    korak           =>      korak1_out,

                                    PM_led          =>      PM_led,
                                    FM_led          =>      FM_led,
                                    modulacija_o    =>      modulacija,
                                    delta_faze      =>      delta_faze
                                );

A_M :           component AM
                    port map(       clk             =>      clk,
                                    rst             =>      rst,
                                    am_mod_in       =>      AM_on_off_d,
                                    mod_sig         =>      AM_selected,
                                    d_in            =>      dds0,

                                    AM_led          =>      AM_led,
                                    AM_SC_led       =>      AM_SC_led,
                                    d_out           =>      AM_out
                                );

AMATT :         component ATTENUATE
                    port map (      clk             =>      clk,
                                    rst             =>      rst,
                                    r_event         =>      AM_event,
                                    r_direction     =>      AM_direction,
                                    d_in            =>      dds2,

                                    d_out           =>      AM_ATT
                                );
```

```
FMPM_ATT :      component ATTENUATE
                    port map (      clk             =>      clk,
                                    rst             =>      rst,
                                    r_event         =>      FPM_event,
                                    r_direction     =>      FPM_direction,
                                    d_in            =>      dds1,

                                    d_out           =>      PM_FM_ATT
                                );

ATT_sel :       component att_select
                    port map (      clk             => clk,
                                    rst             => rst,
                                    tipka           => ATT_d,
                                    on_off          => on_off,
                                    r_event         => r_event,
                                    r_direction     => r_direction,

                                    AM_event        => AM_event,
                                    AM_direction    => AM_direction,
                                    FPM_event       => FPM_event,
                                    FPM_direction   => FPM_direction
                                );

Korak_com :     component korak
                    port map (      clk             => clk,
                                    rst             => rst,
                                    on_off          => on_off_d,
                                    dds_btn         => dds_btn_d,
                                    position        => position_d,
                                    r_event         => r_event,
                                    r_direction     => r_direction,
--                                  korak0          => korak0,
--                                  korak1          => korak1,
--                                  korak2          => korak2,

                                    led0            => led0,
                                    led1            => led1,
                                    led2            => led2,
                                    korak0_out      => korak0_out,
                                    korak1_out      => korak1_out,
                                    korak2_out      => korak2_out
                                );

AMintext :      component sklopka32
                    port map(       clk             => clk,
                                    rst             => rst,
                                    int_ext         => AM_int_ext_d,
                                    prvi_ulaz       => AM_ATT, -- unutarnji
                                    drugi_ulaz      => AM_mod, -- vanjski

                                    odabrani        => AM_selected
                                );

FPMintext :     component sklopka32
                    port map(       clk             => clk,
                                    rst             => rst,
                                    int_ext         => PM_FM_intext_d,
                                    prvi_ulaz       => PM_FM_ATT, -- unutarnji
                                    drugi_ulaz      => P_F_mod, -- vanjski

                                    odabrani        => PM_FM_selected
                                );
```

```
        enkoder :       component encoder01
                                port  map (
                                                clk         => clk,
                                                reset       => rst,

                                                rot_a       => rot_a_d,
                                                rot_b       => rot_b_d,

                                                rotary_event      => r_event,
                                                rotary_direction  => r_direction
                                        );

        DAconv :        component DAC
                                Port map (
                                                clk         => clk,
                                                rst         => rst,
                                                ulaz        => AM_out (31 downto 20),

                                                CS                      => CS,
                                                CLR                     => CLR,
                                                SCK                     => SCK,
                                                MOSI                    => MOSI,

                                                SPI_SS_B        => SPI_SS_B,
                                                AMP_CS                  => AMP_CS,
                                                AD_CONV                 => AD_CONV,
                                                SF_CE0          => SF_CE0,
                                                FPGA_INIT       => FPGA_INIT
                                                                        );

        izlaz_sklopa <= AM_out;

end rtl;


entity debouncing is
                port (  clk             : in    std_logic;
                        rst             : in    std_logic;
                        ulaz            : in    std_logic;
                        izlaz   : out   std_logic
                );
end debouncing;

architecture rtl of debouncing is

signal count1000000 : integer range 0 to 999999;
signal ulaz_sampled: std_logic :='0';

begin

        process(clk,rst) is

        begin
                if rising_edge(clk) then
                        if rst ='1' then
                                count1000000 <= 0;
                                izlaz <= '0';
                        else
                                if count1000000 = 999999 then
                                        count1000000 <= 0;
                                        if ulaz = ulaz_sampled then
                                                izlaz <= ulaz;
```

```
                                        end if;
                                        ulaz_sampled <= ulaz;
                                else
                                        count1000000 <= count1000000 + 1;
                                end if;
                        end if;
                end if;
        end process;


end rtl;



entity dds is
   Port ( clk                   : in  STD_LOGIC;
          rst                   : in  STD_LOGIC;
          faza                  : in  STD_LOGIC_VECTOR (27 downto 0);
          korak                 : in  STD_LOGIC_VECTOR (27 downto 0);
          rez                           : out STD_LOGIC_VECTOR (31 downto 0)
                        );
end dds;

architecture rtl of dds is


---------------------------------------------------------------------------------
-- f-ja vraca apsolutnu vrijednost std_logic_vectora
function abs_std (broj std_logic_vector) return std_logic_vector is

        variable pom    :       std_logic_vector (broj'length-1 downto 0) := (others => '0');

begin

        if (broj(broj'left) = '1') then
                pom := not(broj) + 1;
                return pom;
        else
                return broj;
        end if;

end function;
---------------------------------------------------------------------------------


component akumulator is
   Port ( clk                   : in  STD_LOGIC;
          rst                   : in  STD_LOGIC;
          faza                  : in  STD_LOGIC_VECTOR (27 downto 0);
          korak                 : in  STD_LOGIC_VECTOR (27 downto 0);
          faza_out              : out  STD_LOGIC_VECTOR (27 downto 0)
                        );
end component akumulator;

component priprema is
        Port (  clk                     : in  STD_LOGIC;
                rst                     : in  STD_LOGIC;
                faza                    : in  STD_LOGIC_VECTOR (27 downto 0);
                addr                    : out  STD_LOGIC_VECTOR (7 downto 0);
                kvadrant                : out  STD_LOGIC_VECTOR (1 downto 0);
                deltaX                  : out  STD_LOGIC_VECTOR (17 downto 0)
                        );
end component priprema;
```

```
component ROM is
   Port ( clk        : in  STD_LOGIC;
          rst        : in  STD_LOGIC;
          addr       : in  STD_LOGIC_VECTOR (7 downto 0);
          data       : out  STD_LOGIC_VECTOR (71 downto 0)
                       );
end component ROM;

component predznak is
   Port ( clk                : in  STD_LOGIC;
          rst                : in  STD_LOGIC;
          u_koef             : in  STD_LOGIC_VECTOR (70 downto 0);
          kvadrant           : in  STD_LOGIC_VECTOR (1 downto 0);
          a_koef             : out  STD_LOGIC_VECTOR (31 downto 0);
          b_koef             : out  STD_LOGIC_VECTOR (23 downto 0);
          c_koef             : out  STD_LOGIC_VECTOR (14 downto 0);
          d_koef             : out  STD_LOGIC_VECTOR (3 downto 0)
                               );
end component predznak;

component mull is
   Port ( clk        : in  STD_LOGIC;
          rst        : in  STD_LOGIC;
           a         : in  STD_LOGIC_VECTOR (17 downto 0);
          b          : in  STD_LOGIC_VECTOR (17 downto 0);
          p          : out  STD_LOGIC_VECTOR (35 downto 0)
                       );
end component mull;

component popravi is
   Port ( clk        : in  STD_LOGIC;
          rst        : in  STD_LOGIC;
          data       : in  STD_LOGIC_VECTOR (35 downto 0);     -- izlaz iz prvog mnozila
          delta_x    : in  STD_LOGIC_VECTOR (6 downto 0);      -- 1.6 gornji dio
          s2_low     : in  STD_LOGIC_VECTOR (5 downto 0);      -- 0.6 iz 2. zbrajala; donji dio
          p1_p       : out STD_LOGIC_VECTOR (5 downto 0);      -- gornji produkt D*x + lsb za carry in
          p3_low     : out STD_LOGIC_VECTOR (12 downto 0)      -- donji produkt
                       );
end component popravi;

component shf_n_add_m is
           generic ( N        :          positive;          -- broj posmaka u desno
                     M        :          positive);         -- sirina prvog ulaznog podatak i izlaza
   Port ( clk        : in  STD_LOGIC;
          rst        : in  STD_LOGIC;
          c_in       : in  STD_LOGIC;
          -- prvi desni nakon odbacivanja kod produkta
          a          : in  STD_LOGIC_VECTOR (M-1 downto 0);  -- koeficijent
          b          : in  STD_LOGIC_VECTOR (M-N-1 downto 0);-- produkt iz mnozila koji se zaokruzuje
          sum        : out STD_LOGIC_VECTOR (M-1 downto 0)
                       );
end component shf_n_add_m;


component reg_N is
          generic ( N : positive);
          Port (     clk                : in  STD_LOGIC;
                     rst                : in  STD_LOGIC;
                     d_in               : in  STD_LOGIC_VECTOR (N-1 downto 0);
                     d_out              : out  STD_LOGIC_VECTOR (N-1 downto 0)
                                );
end component reg_N;
```

```
signal    faza_out                :std_logic_vector (27 downto 0) := (others => '0');

signal    addr                    :std_logic_vector (7 downto 0) := (others => '0');
signal    kvadrant, kvadrant_3    :std_logic_vector (1 downto 0) := (others => '0');
signal    dx, dx_3, dx_4, dx_5    :std_logic_vector (17 downto 0) := (others => '0');
signal    dx_6, dx_7, dx_8, dx_9  :std_logic_vector (17 downto 0) := (others => '0');
signal    dx_10                   :std_logic_vector (6 downto 0) := (others => '0');

signal    koef                    :std_logic_vector (71 downto 0) := (others => '0');

signal    koef_abc, koef_abc_5    :std_logic_vector (70 downto 0) := (others => '0');
signal    koef_abc_6              :std_logic_vector (70 downto 0) := (others => '0');
signal    koef_d                  :std_logic_vector (3 downto 0) := (others => '0');

signal    p1, p2, p3              :std_logic_vector (35 downto 0);

signal    p1_p, sum2_low          :std_logic_vector (5 downto 0) := (others => '0');
signal    p3_low                  :std_logic_vector (12 downto 0) := (others => '0');

signal    sum1                    :std_logic_vector (14 downto 0) := (others => '0');
signal    koef_ab_7, koef_ab_8    :std_logic_vector (55 downto 0) := (others => '0');

signal    sum2                    :std_logic_vector (23 downto 0) := (others => '0');
signal    koef_a_9, koef_a_10     :std_logic_vector (31 downto 0) := (others => '0');
signal    koef_a_11, koef_a_12    :std_logic_vector (31 downto 0) := (others => '0');

signal    p3_11, p3_all           :std_logic_vector (29 downto 0) := (others => '0');

signal    sum3                    :std_logic_vector (31 downto 0) := (others => '0');

-----
signal    dx_9abs                 :std_logic_vector (6 downto 0)  := (others => '0');
signal    dx_9msb                 :std_logic_vector (6 downto 0)  := (others => '0');
signal    dx_9neg                 :std_logic_vector (6 downto 0)  := (others => '0');
signal    mull1_dx, mull1_a       :std_logic_vector (17 downto 0) := (others => '0');
signal    mull2_a, mull2_b        :std_logic_vector (17 downto 0) := (others => '0');


begin

--------------------------1. FAZA-----------------------------------
fazni_akumulator  :       component akumulator
                          port map ( clk          => clk,
                                     rst          => rst,
                                     faza         => faza,
                                     korak        => korak,

                                     faza_out     => faza_out
                                    );


--------------------------2. FAZA-----------------------------------

pripremi_addr_dx  :       component priprema
                          port map ( clk          => clk,
                                     rst          => rst,
                                     faza         => faza_out,

                                     addr         => addr,
                                     kvadrant     => kvadrant,
                                     deltaX   => dx
                                    );
```

```
--------------------------3. FAZA----------------------------------

citaj_rom:          component rom
                        port map (          clk     => clk,
                                            rst     => rst,
                                            addr    => addr,

                                            data    => koef
                                    );


reg3_kvad_dx    :       component reg_N
                                generic map (20)
                                port map ( clk                          => clk,
                                           rst                          => rst,
                                           d_in(19 downto 18)           => kvadrant,
                                           d_in(17 downto 0)            => dx,

                                           d_out(19 downto 18)          => kvadrant_3,
                                           d_out(17 downto 0)           => dx_3
                                          );

--------------------------4. FAZA----------------------------------

koef_predznak   :       component predznak
                                port map ( clk      => clk,
                                           rst      => rst,
                                           u_koef   => koef (70 downto 0),
                                           kvadrant         => kvadrant_3,

                                           a_koef   => koef_abc (70 downto 39),
                                           b_koef   => koef_abc (38 downto 15),
                                           c_koef   => koef_abc (14 downto 0),
                                           d_koef   => koef_d
                                          );

reg4_dx         :       component reg_N
                                generic map (18)
                                port map ( clk      => clk,
                                           rst      => rst,
                                           d_in     => dx_3,

                                           d_out    => dx_4
                                          );


--------------------------5. FAZA----------------------------------

mull1_dx(17 downto 12)      <= dx_4(17 downto 12);
mull1_dx(11 downto 6)       <= (others => '0');
mull1_dx(5 downto 0)        <= dx_9abs(5 downto 0);

mull1_a(17 downto 14)       <= koef_d;
mull1_a(13 downto 6)        <= (others => '0');
mull1_a(5 downto 0)         <= sum2(5 downto 0);

-- gornjih 7 bitova od dx_9 su relevantni
dx_9msb <= dx_9(17 downto 11);
-- ugradeni multiplekser
dx_9abs <= dx_9neg when dx_9msb(6)='1' else dx_9msb;
-- ugradeno zbrajalo
dx_9neg <= 0 - dx_9msb;
```

```
prosireno_mnozenje:     component mull
                            port map ( clk      => clk,
                                       rst      => rst,
                                       a        => mull1_a,
                                       b        => mull1_dx,

                                       p        => p1
                                     );


reg5_dx         :       component reg_N
                            generic map (18)
                            port map ( clk      => clk,
                                       rst      => rst,
                                       d_in     => dx_4,

                                       d_out    => dx_5
                                     );

reg5_koef       :       component reg_N
                            generic map (71)
                            port map ( clk      => clk,
                                       rst      => rst,
                                       d_in     => koef_abc,

                                       d_out    => koef_abc_5
                                     );

-------------------------6. FAZA---------------------------------

popravi_p1_p3_low:      component popravi
                            port map ( clk      => clk,
                                       rst      => rst,
                                       data     =>p1,      -- izlaz iz prvog mnozila
                                       delta_x  =>dx_10,-- 1.6 gornji dio (10)
                                       s2_low   =>sum2_low,-- 0.6 iz 2. zbrajala; donji dio

                                       p1_p      =>p1_p, -- gornji produkt D*x + lsb za carry in
                                       p3_low   =>p3_low-- donji produkt
                                     );

reg6_dx         :       component reg_N
                            generic map (18)
                            port map ( clk      => clk,
                                       rst      => rst,
                                       d_in     => dx_5,

                                       d_out    => dx_6
                                                                                    );

reg6_koef       :       component reg_N
                            generic map (71)
                            port map ( clk      => clk,
                                       rst      => rst,
                                       d_in     => koef_abc_5,

                                       d_out    => koef_abc_6
                                     );
```

```
----------------------------7. FAZA----------------------------------

shf_rnd_add_1     :     component shf_n_add_m
                              generic map( 10, 15)-- broj posmaka u desno, sirina ulaza i izlaza
                              port map ( clk      => clk,
                                         rst      => rst,
                                         c_in     =>p1_p(0),-- prvi desni nakon odbacivanja
                                         a        =>koef_abc_6(14 downto 0),-- koeficijent
                                         b        =>p1_p(5 downto 1),-- produkt koji se zaokruzuje

                                         sum      => sum1
                                        );

reg7_dx           :     component reg_N
                              generic map (18)
                              port map ( clk      => clk,
                                         rst      => rst,
                                         d_in     => dx_6,

                                         d_out    => dx_7
                                        );

reg7_koef         :     component reg_N
                              generic map (56)
                              port map ( clk      => clk,
                                         rst      => rst,
                                         d_in     => koef_abc_6(70 downto 15),

                                         d_out    => koef_ab_7
                                        );

----------------------------8. FAZA----------------------------------

mull2_a(17 downto 15)     <= (others => sum1(14));
mull2_a(14 downto 0)       <= sum1(14 downto 0);
mull2_b(17)               <= dx_7(17);
mull2_b(16 downto 0)      <= dx_7(17 downto 1);

mnozilo2          :     component mull
                              port map ( clk      => clk,
                                         rst      => rst,
                                         a        => mull2_a,
                                         b        => mull2_b,

                                         p        => p2
                                        );

reg8_dx           :     component reg_N
                              generic map (18)
                              port map ( clk      => clk,
                                         rst      => rst,
                                         d_in     => dx_7,

                                         d_out    => dx_8
                                        );

reg8_koef         :     component reg_N
                              generic map (56)
                              port map ( clk      => clk,
                                         rst      => rst,
                                         d_in     => koef_ab_7,

                                         d_out    => koef_ab_8
                                        );
```

```
--------------------------9. FAZA---------------------------------

shf_rnd_add_2     :     component shf_n_add_m
                              generic map( 9, 24)-- broj posmaka u desno, sirina ulaza i izlaza
                               port map ( clk     => clk,
                                          rst     => rst,
                                          c_in    =>p2(15),-- prvi desni nakon odbacivanja
                                          a       =>koef_ab_8(23 downto 0), -- koeficijent
                                          b       =>p2(30 downto 16),-- produkt koji se zaokruzuje

                                          sum     => sum2
                                         );

reg9_dx           :     component reg_N
                              generic map (18)
                              port map ( clk     => clk,
                                         rst     => rst,
                                         d_in    => dx_8,

                                         d_out   => dx_9
                                        );


reg9_koef         :     component reg_N
                              generic map (32)
                              port map ( clk     => clk,
                                         rst     => rst,
                                         d_in    => koef_ab_8(55 downto 24),

                                         d_out   => koef_a_9
                                                                                    );

--------------------------10. FAZA---------------------------------

mnozilo3          :     component mull
                              port map ( clk => clk,
                                         rst => rst,
                                         a       => sum2(23 downto 6),
                                         b       => dx_9,

                                         p       => p3
                                        );

reg10_dx          :     component reg_N
                              generic map (7)
                              port map ( clk     => clk,
                                         rst     => rst,
                                         d_in    => dx_9(17 downto 11),

                                         d_out   => dx_10
                                                                                    );

reg10_koef        :     component reg_N
                              generic map (32)
                              port map ( clk     => clk,
                                         rst     => rst,
                                         d_in    => koef_a_9,

                                         d_out   => koef_a_10
                                        );
```

```
reg10_sum2_low   :        component reg_N
                                generic map (6)
                                port map ( clk => clk,
                                           rst => rst,
                                           d_in => sum2(5 downto 0),

                                           d_out => sum2_low
                                          );


----------------------------11. FAZA-----------------------------

reg11_koef       :        component reg_N
                                generic map (32)
                                port map ( clk     => clk,
                                           rst     => rst,
                                           d_in    => koef_a_10,

                                           d_out   => koef_a_11
                                                                                );

reg11_p3_hi      :        component reg_N
                                generic map (30)
                                port map ( clk     => clk,
                                           rst     => rst,
                                           d_in    => p3(34 downto 5),

                                           d_out   => p3_11
                                                                                );


--------------------------12. FAZA-----------------------------

shf_rnd_add_m    :        component shf_n_add_m
                                generic map( 17, 30)-- broj posmaka u desno, sirina ulaza i izlaza
                                 port map ( clk     => clk,
                                            rst     => rst,
                                            c_in    =>         '0',
                                            a       =>         p3_11,    -- izlaz iz mnozila 3
                                            b       =>         p3_low,   -- donji popravljeni dio iz mnozil 1

                                            sum     => p3_all
                                           );

reg12_koef       :        component reg_N
                                generic map (32)
                                port map ( clk     => clk,
                                           rst     => rst,
                                           d_in    => koef_a_11,

                                           d_out   => koef_a_12
                                          );

--------------------------13. FAZA-----------------------------

shf_rnd_add_3    :        component shf_n_add_m
                                generic map( 8, 32)-- broj posmaka u desno, sirina ulaza i izlaza
                                port map ( clk     => clk,
                                           rst     => rst,
                                           c_in    =>p3_all(5),
                                           a       =>koef_a_12,-- koef a (12)
                                           b       =>p3_all(29 downto 6),      -- donji popravljeni dio
                                                                               iz mnozil 1
```

```
                                        sum         => sum3
                                        );

--------------------------14. FAZA-----------------------------

reg14_out           :       component reg_N
                                generic map (32)
                                port map ( clk      => clk,
                                           rst      => rst,
                                           d_in     => sum3,

                                           d_out    => rez
                                          );

end rtl;



entity akumulator is
    Port ( clk              : in  STD_LOGIC;
           rst              : in  STD_LOGIC;
           faza             : in  STD_LOGIC_VECTOR (27 downto 0);
           korak            : in  STD_LOGIC_VECTOR (27 downto 0);
           faza_out         : out  STD_LOGIC_VECTOR (27 downto 0)
            );
end akumulator;

architecture rtl of akumulator is

signal akum : STD_LOGIC_VECTOR (27 downto 0) := (others => '0');

begin

        process (clk)
        begin
                if rising_edge(clk) then
                        if rst = '1' then
                                faza_out <= (others => '0');
                                akum <= (others => '0');
                        else
                                faza_out <= faza + akum;
                                akum             <= akum + korak;
                        end if;
                end if;
        end process;


end rtl;



entity akumulator0 is
    Port ( clk              : in  STD_LOGIC;
           rst              : in  STD_LOGIC;
           modulacija       : in        std_logic_vector (1 downto 0);
           faza             : in  STD_LOGIC_VECTOR (27 downto 0);
           delta_faze       : in  STD_LOGIC_VECTOR (28 downto 0);
           korak            : in  STD_LOGIC_VECTOR (27 downto 0);
           faza_out         : out  STD_LOGIC_VECTOR (27 downto 0)
            );
end akumulator0;
```

```
architecture rtl of akumulator0 is

signal akum : STD_LOGIC_VECTOR (27 downto 0) := (others => '0');

begin

	process (clk)
	begin
		if rising_edge(clk) then
			if rst = '1' then
				faza_out <= (others => '0');
				akum <= (others => '0');
			elsif modulacija = "01" then -- ako je PM
				if delta_faze(28) = '0' then
					faza_out <= faza + akum + delta_faze(27 downto 0);
					akum		<= akum + korak;
				else
					faza_out <= faza + akum - not(delta_faze(27 downto 0));
					akum		<= akum + korak;
				end if;
			elsif modulacija = "10" then -- ako je FM
				if delta_faze(28) = '0' then
					faza_out <= faza + akum;
					akum	<= akum + korak + delta_faze(27 downto 0);
				else
				    faza_out <= faza + akum;
				    akum  <= akum + korak - not(delta_faze(27 downto 0)) - x"0000001";
				end if;
			else
				faza_out <= faza + akum;
				akum		<= akum + korak;

			end if;
		end if;
	end process;


end rtl;


entity priprema is
   Port ( clk		: in  STD_LOGIC;
          rst		: in  STD_LOGIC;
          faza		: in  STD_LOGIC_VECTOR (27 downto 0);
          addr		: out  STD_LOGIC_VECTOR (7 downto 0);
          kvadrant	: out  STD_LOGIC_VECTOR (1 downto 0);
          deltaX	: out  STD_LOGIC_VECTOR (17 downto 0)
			 );
end priprema;

architecture rtl of priprema is

begin

	process (clk, rst)
	begin
		if rising_edge(clk) then
			if rst = '1' then
				addr		<= (others => '0');
				kvadrant <= (others => '0');
				deltaX	<= (others => '0');
			else
				if ((faza(27 downto 26) = "01") or (faza(27 downto 26) = "11")) then
					addr <= not (faza(25 downto 18));
```

```
                                    else
                                            addr <= faza(25 downto 18);
                                    end if;

                                    kvadrant                        <= faza(27 downto 26);
                                    deltaX(17)                              <= not (faza(17));
                                    deltaX(16 downto 0)         <= faza(16 downto 0);
                            end if;
                    end if;
            end process;

end rtl;


entity ROM is
   Port ( clk           : in  STD_LOGIC;
                        rst         : in  STD_LOGIC;
          addr          : in  STD_LOGIC_VECTOR (7 downto 0);
          data          : out STD_LOGIC_VECTOR (71 downto 0)
                        );
end ROM;

architecture rtl of ROM is
begin

  -- RAMB16_S36_S36: Virtex-II/II-Pro, Spartan-3/3E 512 x 32 + 4 Parity bits Dual-Port RAM
  -- Xilinx HDL Language Template, version 10.1

  RAMB16_S36_S36_inst : RAMB16_S36_S36
  generic map (
    INIT_A => X"000000000",  --  Value of output RAM registers on Port A at startup
    INIT_B => X"000000000",  --  Value of output RAM registers on Port B at startup
    SRVAL_A => X"000000000", --  Port A ouput value upon SSR assertion
    SRVAL_B => X"000000000", --  Port B ouput value upon SSR assertion
    WRITE_MODE_A => "WRITE_FIRST", --  WRITE_FIRST, READ_FIRST or NO_CHANGE
    WRITE_MODE_B => "WRITE_FIRST", --  WRITE_FIRST, READ_FIRST or NO_CHANGE
    SIM_COLLISION_CHECK => "ALL", -- "NONE", "WARNING", "GENERATE_X_ONLY", "ALL"
                        -- The following INIT_xx declarations specify the intiial contents of the RAM
                        -- Address 0 to 127
                        INIT_00 =>
X"05E36EA9051A8E5C0451A1770388A9EA02BFA9A401F6A297012D96B1006487E3",
                        INIT_01 =>
X"8C27C3890B5F8D9F8A973BA509CECF8909064B3A083DB0A7877501BE86AC406F",
                        INIT_02 =>
X"9264994E919D894110D64DBD100EE8AD8F475BFF8E7FA99E8DB7D3760CEFDB76",
                        INIT_03 =>
X"1896172897D0A7BC970AFD8D16451A83157F008614B8B17F13F22F58932B7BF9",
                        INIT_04 =>
X"9EB86B461DF5163F9D31774D1C6D90531BA963359AE4F1D61A203E1B995B49EA",
                        INIT_05 =>
X"A4C7CD3324070B082345EFF8A2847DE0A1C2B69C21009C0CA03E300D1F7B7481",
                        INIT_06 =>
X"AAC080262A02C7B8A944A7A2A88621B927C737D3A707EBC726483F6C2588349D",
                        INIT_07 =>
X"309ED556AFE49BA72F29EBCC2E6EC792ADB330C72CF72939AC3AB2B92B7DCF17",
                        INIT_08 =>
X"B65F2E3B35A8E625B4F219A8343ACA873382FA88B2CAAB6F3211DF043158970E",
                        INIT_09 =>
X"BBFDFECD3B4C18BA3A99A057B9E6975EB932FF87B87EDA8E37CA2A303714F02A",
                        INIT_0A =>
X"4177CFB140CAB958401D0321BF6EAEB8BEBFBDCD3E10320DBD600D2C3CAF50DA",
                        INIT_0B =>
X"46C9405CC621647D4578DB93C4CFA740C425C923437B42E1C2D0161EC2244481",
```

```
                INIT_0C =>
X"4BEF092D4B4CCF4D4AA9DBA2CA062FBDC961CD33C8BCB599C816EA86C7706D93",
                INIT_0D =>
X"D0E5FD6DD049C999CFACCFABCF0F1126CE708F8F4DD14C6E4D31494BCC9087B1",
                INIT_0E =>
X"55AB0D4655153FD4547EA073D3E73097534EF1B5D2B5E546521C0CC2D18169A5",
                INIT_0F =>
X"DA3B47ABD9AC3CFDD91C550ED88B9140D7F9F2F857677B9D56D42C99D6400758",
                -- Address 128 to 255
                INIT_10 =>
X"DE93DC1F5E0BEC6E5D8314B1DCF95638DC6EB2585BE32A675B56BFBD5AC973B5",
                INIT_11 =>
X"62B21C7B62319B9D61B02876E12DC447E0AA705060262DD65FA0FE1F5F1AE274",
                INIT_12 =>
X"66937E91661ABBC5E5A0FD0B6526438F64AA907F642DE50D63B0426DE331A9D4",
                INIT_13 =>
X"EA359DB9E9C4E37A6953244268E06129E86C9B4BE7F7D3C5E7820BB7670B4444",
                INIT_14 =>
X"6D963C54ED2DD027ECC45698EC59D0A9EBEE3F626B81A3CD6B13FEF56AA551E9",
                INIT_15 =>
X"F0B34525F0536771EFF27497EF906D84EF2D532C6EC92683EE63E87F6DFD9A1C",
                INIT_16 =>
X"738ACC9E7333B883F2DB8828F2823C67F227D61C71CC5626716FBD6871120CC5",
                INIT_17 =>
X"F61B121175CCFD42757DC5CAF52D6C6CF4DBF1EF7489571CF4359CBDF3E0C3A3",
                INIT_18 =>
X"F86280BF781D9B6577D78DAA7790583EF747FBCE76FE790E76B3D0B476680376",
                INIT_19 =>
X"7A5FB0D8FA24256FF9E76CA7F9A98715F96A7554792A37FEF8E8CFB278A63D11",
                INIT_1A =>
X"7C116853FBDF5B947BAC1D31FB77ADA87B420D7AFB0B3D2CFAD33D45FA9A0E50",
                INIT_1B =>
X"FD769BB5FD4E2C7F7D24881BFCF9AEF0FCCDA1697CA05FF1FC71EAF9FC4242F2",
                INIT_1C =>
X"7E8E6EB2FE6FB5F47E4FC53E7E2E9CDFFE0C3D297DE8A670FDC3D90D7D9DD55A",
                INIT_1D =>
X"7F5834B7FF4345637F2D1C0EFF15B8EEFEFD1C3C7EE346367EC8371A7EABEF2C",
                INIT_1E =>
X"7FD37153FFC858547FBC040A7FAE74957F9FAA157F8FA4B0FF7E648CFF6BE9D4",
                INIT_1F =>
X"FFFFD8867FFE9CB2FFFC250F7FF871A2FFF382747FED5791FFE5F108FFDD4EEC",
    -- drugi dio
    INIT_20 => X"07472365064F239B055723CA045F23F00367240F026F242701772436007F243E",
                INIT_21 =>
X"0F03209C0E0B21110D13217D0C1B21E20B27223F0A2F2294093722E1083F2327",
                INIT_22 =>
X"16B31BE615BF1C9814C71D4213D31DE412DB1E7F11E71F1210EF1F9D0FF72021",
                INIT_23 =>
X"1E5715451D6316341C6F171B1B7B17FB1A8718D3199319A3189F1A6C17AB1B2D",
                INIT_24 =>
X"25EB0CBD24F70DE924070F0D231710292223113E212F124B203F13511F4B144F",
                INIT_25 =>
X"2D6302542C7703BB2B87051B2A97067329AB07C428BB090E27CB0A5026DB0B8A",
                INIT_26 =>
X"34C2F60F33D6F7B232EAF94D3202FAE03116FC6D302AFDF22F3EFF6F2E4F00E5",
                INIT_27 =>
X"3BFEE7F83B16E9D43A32EBA9394AED773862EF3D377AF0FD3692F2B535AAF466",
                INIT_28 =>
X"4316D8154236DA2A4152DC384072DE3F3F8EE03F3EAAE2383DC6E4293CE2E614",
                INIT_29 =>
X"4A06C671492AC8BE484ECB034772CD424692CF7A45B6D1AB44D6D3D543F6D5F9",
                INIT_2A =>
X"50C6B3174FF2B59A4F1AB8164E42BA8B4D6ABCFA4C92BF624BBAC1C34ADEC41D",
```

```
                INIT_2B =>
X"57569E135686A0CA55B6A37B54E6A6255416A8C95342AB67526EADFD519AB08D",
                INIT_2C =>
X"5DAE87715CE68A5C5C1E8D405B56901D5A8A92F559BE95C658F2989158269B55",
                INIT_2D =>
X"63CE6F41630E725C624E7571618A788160C67B8A60027E8D5F3E818A5E768481",
                INIT_2E =>
X"69B2559068FA58DA683E5C1F67865F5E66CA6297660E65CA654E68F8648E6C1F",
                INIT_2F =>
X"6F523A6E6EA23DE66DF241586D3E44C56C8A482C6BD64B8D6B224EE96A6A523F",
                -- Address
                INIT_30 =>
X"74B21DEC740A218F735E252D72B628C5720A2C58715E2FE670B2336E700236F1",
                INIT_31 =>
X"79C6001D792603E8788607AF77E60B7177420F2D76A212E575FE169775561A44",
                INIT_32 =>
X"7E8DE1117DF9E5037D61E8F17CC9ECD97C31F0BD7B99F49C7AFDF8777A61FC4C",
                INIT_33 =>
X"8309C0DD827DC4F381F1C9058165CD1280D5D11B8045D51F7FB5D91F7F21DD1B",
                INIT_34 =>
X"87359F9486B1A3CC8631A7FF85ADAC2F8529B05A84A1B481841DB8A48391BCC3",
                INIT_35 =>
X"8B097D4B8A9581A18A1D85F489A58A4389298E8E88AD92D68831971987B19B59",
                INIT_36 =>
X"8E8D5A168E215E898DB562F88D4567648CD56BCD8C6570328BF574938B8178F1",
                INIT_37 =>
X"91B5360C91553A9990F53F22909143A8902D482B8FC54CAA8F6151278EF555A0",
                INIT_38 =>
X"94851143943115E693D91A8693811F24932923BE92CD285692712CEB9215317D",
                INIT_39 =>
X"96F8EBD296B0F0889664F53D9618F9EF95CCFE9E957D034B952D07F694D90C9E",
                INIT_3A =>
X"9910C5CF98D4CA969894CF5C9854D41F9810D8E197CCDDA09788E25E9740E719",
                INIT_3B =>
X"9AC89F519A98A4279A64A8FB9A30ADCD99F8B29E99C0B76D9988BC3A994CC105",
                INIT_3C =>
X"9C2078729BFC7D539BD482329BAC87109B808BED9B5490C89B2895A29AF89A7A",
                INIT_3D =>
X"9D1C51499D0056319CE45B199CC85FFF9CAC64E59C8C69CA9C686EAE9C487391",
                INIT_3E =>
X"9DB429ED9DA42EDA9D9433C79D8438B49D743D9F9D60428B9D4847759D344C5F",
                INIT_3F =>
X"9DE802779DE807669DE40C569DE011459DDC16349DD41B229DC820119DC024FF",
    -- The next set of INITP_xx are for the parity bits
                -- Address 0 to 127
                INITP_00 =>
X"0464E576F43D209B6B9FE65DC3174722A7B55A55B67C6592F17214B77B3FF3B7",
                INITP_01 =>
X"52613B890A77AE4A18E37984D23E56164A9FD2DF746C768E61ECCDF1356763F9",
                -- Address 128 to 255
                INITP_02 =>
X"E1332F7CD8CA10624B77A07092A169839A6B8EC1CD3EDF4B4E82B4AEEC598251",
                INITP_03 =>
X"AF242E574A90E3DC071C9740A166176FF76A4349193C5C132C109B45CAC3DBBC",
     -- drugi dio
     INITP_04 => X"5555555555555555555555555555555555555555555555555555555555555555555555555555",
                INITP_05 =>
X"4444444444444444444444444444444444444444444444555555555555555555555",
                --
                INITP_06 =>
X"22222222222222222233333333333333333333333333333333333333333334444444",
                INITP_07 =>
X"00000000000000001111111111111111111111111111111122222222222222222")
```

```
port map (
    DOA => data (31 downto 0),     -- Port A 32-bit Data Output      ;A koef i B (LSB, 1 bit)
    DOB => data (67 downto 36),    -- Port B 32-bit Data Output      ;C i B koef (ostatak B-a)
    DOPA           => data (35 downto 32) ,   -- Port A 4-bit Parity Output     ;B koef (4 bita)
    DOPB           => data (71 downto 68),    -- Port B 4-bit Parity Output     ;D koef
    ADDRA(8) => '0',
    ADDRA (7 downto 0) => addr,  -- Port A 9-bit Address Input                ;A koef, B1 koef (dio)
                ADDRB (8) => '1',
    ADDRB (7 downto 0) => addr,  -- Port B 9-bit Address Input                ;B2 koef (drugi dio), C i D koef
    CLKA => clk,    -- Port A Clock
    CLKB => clk,    -- Port B Clock
    DIA => X"00000000",      -- Port A 32-bit Data Input
    DIB => X"00000000",      -- Port B 32-bit Data Input
    DIPA => X"0",    -- Port A 4-bit parity Input
    DIPB => X"0",    -- Port-B 4-bit parity Input
    ENA => '1',      -- Port A RAM Enable Input
    ENB => '1',      -- PortB RAM Enable Input
    SSRA => rst,     -- Port A Synchronous Set/Reset Input
    SSRB => rst,     -- Port B Synchronous Set/Reset Input
    WEA => '0',      -- Port A Write Enable Input
    WEB => '0'       -- Port B Write Enable Input
  );

  -- End of RAMB16_S36_S36_inst instantiation

end rtl;



entity reg_N is
          generic ( N : positive);
   Port ( clk        : in  STD_LOGIC;
          rst        : in  STD_LOGIC;
          d_in       : in  STD_LOGIC_VECTOR (N-1 downto 0);
          d_out      : out STD_LOGIC_VECTOR (N-1 downto 0)
                       );
end reg_N;

architecture rtl of reg_N is

begin

          process (clk)
          begin
                    if rising_edge(clk) then
                              if rst = '1' then
                                        d_out <= (others => '0');
                              else
                                        d_out <= d_in;
                              end if;
                    end if;
          end process;

end rtl;
```

```
entity predznak is
   Port ( clk                    : in  STD_LOGIC;
          rst                    : in  STD_LOGIC;
          u_koef                 : in  STD_LOGIC_VECTOR (70 downto 0);
          kvadrant               : in  STD_LOGIC_VECTOR (1 downto 0);
          a_koef                 : out  STD_LOGIC_VECTOR (31 downto 0);
          b_koef    : out  STD_LOGIC_VECTOR (23 downto 0);
          c_koef    : out  STD_LOGIC_VECTOR (14 downto 0);
          d_koef    : out  STD_LOGIC_VECTOR (3 downto 0)
                     );
end predznak;

architecture rtl of predznak is

begin

        process (clk)
        begin
              if rising_edge(clk) then
                  if rst = '1' then
                        a_koef (31 downto 0) <= (others => '0');
                        b_koef (23 downto 0) <= (others => '0');
                        c_koef (14 downto 0) <= (others => '0');
                        d_koef (3 downto 0)  <= (others => '0');
                    else
                        case kvadrant is
                              when "00" =>
                                    a_koef (31 downto 0) <= '0'&u_koef(30 downto 0);
                                    b_koef (23 downto 0) <= '0'&u_koef(53 downto 31);
                                    c_koef (14 downto 0) <= -('0'&u_koef(67 downto 54));
                                    d_koef (3 downto 0)  <= -('0'&u_koef(70 downto 68));
                              when "01" =>
                                    a_koef (31 downto 0) <= '0'&u_koef(30 downto 0);
                                    b_koef (23 downto 0) <= -('0'&u_koef(53 downto 31));
                                    c_koef (14 downto 0) <= -('0'&u_koef(67 downto 54));
                                    d_koef (3 downto 0)  <= '0'&u_koef(70 downto 68);
                              when "10" =>
                                    a_koef (31 downto 0) <= -('0'&u_koef(30 downto 0));
                                    b_koef (23 downto 0) <= -('0'&u_koef(53 downto 31));
                                    c_koef (14 downto 0) <= '0'&u_koef(67 downto 54);
                                    d_koef (3 downto 0)  <= '0'&u_koef(70 downto 68);
                              when "11" =>
                                    a_koef (31 downto 0) <= -('0'&u_koef(30 downto 0));
                                    b_koef (23 downto 0) <= '0'&u_koef(53 downto 31);
                                    c_koef (14 downto 0) <= '0'&u_koef(67 downto 54);
                                    d_koef (3 downto 0)  <= -('0'&u_koef(70 downto 68));
                              -- nije potrebno, ali zbog lakseg debugiranja u slucaju neke greske
                              when others =>
                                    a_koef (31 downto 0) <= (others => '0');
                                    b_koef (23 downto 0) <= (others => '0');
                                    c_koef (14 downto 0) <= (others => '0');
                                    d_koef (3 downto 0)  <= (others => '0');
                        end case;
                    end if;
              end if;

        end process;


end rtl;
```

```
entity mull is
   Port ( clk        : in  STD_LOGIC;
          rst        : in  STD_LOGIC;
          a          : in  STD_LOGIC_VECTOR (17 downto 0);
          b          : in  STD_LOGIC_VECTOR (17 downto 0);
          p          : out  STD_LOGIC_VECTOR (35 downto 0)
                     );
end mull;

architecture rtl of mull is
begin

  MULT18X18S_inst : MULT18X18S
  port map (
    P => p,    -- 36-bit multiplier output
    A => a,    -- 18-bit multiplier input
    B => b,    -- 18-bit multiplier input
    C => clk,  -- Clock input
    CE => '1', -- Clock enable input
    R => rst   -- Synchronous reset input
  );

end rtl;


entity popravi is
   Port ( clk        : in  STD_LOGIC;
          rst        : in  STD_LOGIC;
          data       : in  STD_LOGIC_VECTOR (35 downto 0);     -- izlaz iz prvog mnozila
          delta_x    : in  STD_LOGIC_VECTOR (6 downto 0);      -- 1.6 gornji dio
          s2_low     : in  STD_LOGIC_VECTOR (5 downto 0);      -- 0.6 iz 2. zbrajala; donji dio
          p1_p       : out STD_LOGIC_VECTOR (5 downto 0);      -- gornji produkt D*x 1.4 + lsb za carry in
          p3_low     : out STD_LOGIC_VECTOR (12 downto 0)      -- donji produkt
                     );
end popravi;

architecture rtl of popravi is

signal p1 :          std_logic_vector (9 downto 0) := (others => '0');
signal p3 :          std_logic_vector (12 downto 0) := (others => '0');

begin
  -- zbog prosirenog mnozenja da popravimo rezultat na vrhu
  -- generator p1
          process (clk)
          begin
                  if rising_edge(clk) then
                          if rst = '1' then
                            p1 <= (others => '0');
      else
              p1 <= data(34 downto 25) + 1;
                    end if;
                  end if;
          end process;

  -- generator p3
          process (clk)
          begin
                  if rising_edge(clk) then
                          if rst = '1' then
                            p3 <= (others => '0');
                          elsif delta_x = "1000000" then
                                  p3 <= 0 - ('0' & s2_low & "000000");  -- low donj dio od 1.23 nakon
zbarajanja produkta2 i b koef
```

```
                        elsif delta_x(6) = '1' then      -- ako je deltaX negativan produkt nižih bitova mora
biti negativan
                                p3 <= 0 - ('0' & data(11 downto 0));    -- donjih 12 bita iz prvog mnozila
                        else
                                p3 <= ('0' & data(11 downto 0));        -- pozitivan je
                        end if;
                end if;

        end process;

  -- izlazno mapiranje
        p1_p   <= p1(9 downto 4);               -- lsb je carry in
        p3_low <= p3;                            -- sredjeni donji dio produkta

end rtl;


entity shf_n_add_m is
         generic ( N        :        positive;  -- broj posmaka u desno
                   M        :        positive);-- širina prvog ulaznog podatak i izlaza
   Port ( clk       : in  STD_LOGIC;
          rst       : in  STD_LOGIC;
          c_in      : in  STD_LOGIC;
            -- prvi desni nakon odbacivanja kod produkta
          a         : in  STD_LOGIC_VECTOR (M-1 downto 0);  -- koeficijent
          b         : in  STD_LOGIC_VECTOR (M-N-1 downto 0);-- produkt iz množila koji se zaokružuje
          sum       : out STD_LOGIC_VECTOR (M-1 downto 0)
                    );
end shf_n_add_m;


architecture rtl of shf_n_add_m is
        signal b_signext :  std_logic_vector (M-1 downto 0);
begin

        b_signext (M-1 downto M-N) <= (others => ( b(M-N-1) ));
        b_signext (M-N-1 downto 0) <= b;

  process (clk)
  begin
          if rising_edge(clk) then
                if rst = '1' then
                        sum <= (others => '0');
                else
                        sum <= a + b_signext + c_in;
                end if;
               end if;
  end process;

end rtl;

entity PM_i_FM is
   Port ( clk                   : in  STD_LOGIC;
          rst                   : in  STD_LOGIC;
          PM_FM_off             : in  STD_LOGIC;  -- odabir vrste modulacije/gasenje("00")
          mod_sig               : in  STD_LOGIC_VECTOR (31 downto 0); -- modulacijski signal
          korak                 : in  STD_LOGIC_VECTOR (27 downto 0); -- korak faze mod. sig.
           PM_led               : out STD_LOGIC; -- ld5
          FM_led                : out STD_LOGIC; -- ld4
          modulacija_o          : out std_logic_vector (1 downto 0);
          delta_faze            : out STD_LOGIC_VECTOR (28 downto 0)
                                );
end PM_i_FM;
```

```
architecture rtl of PM_i_FM is

signal modulacija : std_logic_vector (1 downto 0) := (others => '0');

begin

        modulacija_o <= modulacija;

        process (rst, clk, mod_sig)

        variable temp : std_logic := '0';

        begin


                if rising_edge(clk) then
                        if rst = '1' then
                                delta_faze <= (others => '0');
                                modulacija <= "10";
                        else

                                if PM_FM_off = '1' AND temp = '0' then
                                        modulacija <= modulacija + 1;
                                end if;
                                if PM_FM_off = '1' then
                                        temp := '1';
                                else
                                        temp := '0';
                                end if;

                                PM_led <= modulacija (0); -- upali LD5 ako je PM ukljucena
                                FM_led <= modulacija (1); -- upali LD4 ako je FM ukljucena


                                case modulacija is
                                when "01" =>                    -- fazna modulacija
                                        delta_faze <= mod_sig(31 downto 3);

                                when "10" =>                    -- frekvencijska modulacija
                                        delta_faze(28 downto 17) <= (others => mod_sig(31));
                        if mod_sig(13 downto 12) ="11" and mod_sig(31)='0' and mod_sig(30 downto 14)
                                        /= "11111111111111111" then -- popravak zbog zokruživanja
                                delta_faze(16 downto 0) <= mod_sig(30 downto 14) + "00000000000000001";
                        elsif mod_sig(13 downto 12) ="00" and mod_sig(31)='1' and mod_sig(30 downto 14)
                                        /= "00000000000000000" then -- popravak zbog zokruživanja
                                delta_faze(16 downto 0) <= mod_sig(30 downto 14) - "00000000000000001";
                                        else
                                                delta_faze(16 downto 0) <= mod_sig(30 downto 14);
                                        end if;

                                when "11" =>
                                        modulacija <= "00";

                                when "00" =>
                                        delta_faze <= (others => '0');

                                when others => null;
                                end case;
                        end if;
                end if;
        end process;

end rtl;
```

```
entity AM is
                port(   clk             : in  STD_LOGIC;
                        rst             : in  STD_LOGIC;
                        am_mod_in       : in  STD_LOGIC; -- AM uključena / isključena - tipka
                        mod_sig         : in  STD_LOGIC_VECTOR (31 downto 0); -- mod signal
                        d_in            : in  STD_LOGIC_VECTOR (31 downto 0);
                        AM_led          : out STD_LOGIC; -- ld7
                        AM_SC_led       : out STD_LOGIC; -- ld6
                        d_out           : out STD_LOGIC_VECTOR (31 downto 0)
                                );
end AM;

architecture rtl of AM is

component mull32 is
   Port ( clk       : in  STD_LOGIC;
          rst       : in  STD_LOGIC;
          a         : in  STD_LOGIC_VECTOR (31 downto 0);
          b         : in  STD_LOGIC_VECTOR (31 downto 0);
          rez       : out STD_LOGIC_VECTOR (31 downto 0)
                        );
end component mull32;

signal d_out_int, d_out_int2, d_in2 : std_logic_vector (31 downto 0) := (others => '0');
signal polozaj: std_logic_vector (1 downto 0) := "00";

begin

        mnozenje32 : component mull32
                port map (      clk     => clk,
                                rst     => rst,
                                a       => mod_sig (31 downto 0),
                                b       => d_in (31 downto 0),

                                rez     => d_out_int
                        );


        process (rst, clk, am_mod_in)

        variable temp : std_logic := '0';

        begin
                if rising_edge(clk) then
                        if rst = '1' then
                                d_out <= (others => '0');
                                d_in2 <= (others => '0');
                                d_out_int2 <= (others => '0');
                                polozaj <= "00";
                        else
                                if temp = '0' and am_mod_in = '1' then
                                        polozaj <= polozaj + 1;
                                end if;
                                if am_mod_in = '1' then
                                        temp := '1';
                                else
                                        temp := '0';
                                end if;

                                -- djeljenj s 2 da bi dinamika ostala u +-1
                                d_in2(31) <= d_in(31);
                                d_in2(30 downto 0) <= d_in(31 downto 1);
                                d_out_int2(31) <= d_out_int(31);
                                d_out_int2(30 downto 0) <= d_out_int(31 downto 1);
```

```
                                        AM_led <= polozaj (0); -- upali LD7 ako je AM s nosiocem ukljucena
                                        AM_SC_led <= polozaj (1); -- upali LD6 ako je AM s potisnutim
                                                                     nosiocem ukljucena

                                        if polozaj = "01" then
                                                d_out <= d_out_int2 + d_in2; -- 1/2*cos(w0t)*(1+um); U0=1
                                        elsif polozaj = "10" then
                                                d_out <= d_out_int; -- cos(w0t)*um; U0=1
                                        elsif polozaj = "11" then
                                                polozaj <= "00";
                                        else
                                                d_out <= d_in;
                                        end if;
                                end if;
                        end if;
                end process;

end rtl;


entity mull32 is
   Port ( clk           : in  STD_LOGIC;
          rst           : in  STD_LOGIC;
          a             : in  STD_LOGIC_VECTOR (31 downto 0);
          b             : in  STD_LOGIC_VECTOR (31 downto 0);
          rez           : out STD_LOGIC_VECTOR (31 downto 0)
                        );
end mull32;

architecture rtl of mull32 is

component mull is
   Port ( clk           : in  STD_LOGIC;
          rst           : in  STD_LOGIC;
          a             : in  STD_LOGIC_VECTOR (17 downto 0);
          b             : in  STD_LOGIC_VECTOR (17 downto 0);
          p             : out STD_LOGIC_VECTOR (35 downto 0)
                        );
end component mull;

component reg_N is
          generic ( N : positive);
          Port (    clk         : in  STD_LOGIC;
                    rst         : in  STD_LOGIC;
                     d_in       : in  STD_LOGIC_VECTOR (N-1 downto 0);
                    d_out       : out  STD_LOGIC_VECTOR (N-1 downto 0)
                                );
end component reg_N;

signal hi_hi, hi_lo, lo_hi, hi_hi1, hi_hi2, hi_lo1, lo_hi1, low2, rez_int : std_logic_vector (35 downto 0) :=
(others => '0');
signal a_lo, b_lo, a_hi, b_hi : std_logic_vector (17 downto 0) := (others => '0');
signal low : std_logic_vector (18 downto 0) := (others => '0');

begin
          mnozenje_hi_hi : component mull
                                        port map (clk       => clk,
                                                  rst       => rst,
                                                  a         => a_hi,
                                                  b         => b_hi,

                                                  p         => hi_hi
                                                  );
```

```
mnozenje_hi_lo : component mull
                        port map (clk       => clk,
                                  rst       => rst,
                                  a         => a_hi,
                                  b         => b_lo,

                                  p         => hi_lo
                                  );

mnozenje_lo_hi : component mull
                        port map (          clk       => clk,
                                            rst       => rst,
                                            a         => a_lo,
                                            b         => b_hi,

                                            p         => lo_hi
                                                                 );

reg_hi_hi1 : component reg_N
                        generic map (36)
                        port map ( clk      => clk,
                                   rst      => rst,
                                   d_in     => hi_hi,

                                   d_out    => hi_hi1
                                  );

reg_hi_hi2 : component reg_N
                        generic map (36)
                        port map ( clk      => clk,
                                   rst      => rst,
                                   d_in     => hi_hi1,

                                   d_out    => hi_hi2
                                  );

reg_hi_lo : component reg_N
                        generic map (36)
                        port map ( clk      => clk,
                                   rst      => rst,
                                   d_in     => hi_lo,

                                   d_out    => hi_lo1
                                  );

reg_lo_hi : component reg_N
                        generic map (36)
                        port map ( clk      => clk,
                                   rst      => rst,
                                   d_in     => lo_hi,

                                   d_out    => lo_hi1
                                  );

reg_low : component reg_N
                        generic map (19)
                        port map ( clk      => clk,
                                   rst      => rst,
                                   d_in     => low,

                                   d_out    => low2 (18 downto 0)
                                  );
```

```
        a_lo (16 downto 3)      <= a (13 downto 0);
        a_hi (17 downto 0)      <= a (31 downto 14);
        b_lo (16 downto 3)      <= b (13 downto 0);
        b_hi (17 downto 0)      <= b (31 downto 14);


        low <= lo_hi1 (35 downto 17) + hi_lo1 (35 downto 17);

        low2 (35 downto 19) <= (others => low2(18));

        rez_int <= hi_hi2 + low2 + x"0_0000_0001"; -- popravak zbog zokruživanja


        process (rst, clk)
        begin
                if rising_edge(clk) then
                        if rst = '1' then
                                rez <= (others => '0');
                        else
                                rez <= rez_int (34 downto 3);
                        end if;
                end if;
        end process;


end rtl;


entity ATTENUATE is
                port    (clk                    : in  STD_LOGIC;
                        rst                     : in  STD_LOGIC;
                        r_event                 : in    std_logic;
                        r_direction             : in    std_logic;
                        d_in                    : in    std_logic_vector (31 downto 0);
                        d_out                   : out   std_logic_vector (31 downto 0)
                        );
end ATTENUATE;

architecture rtl of ATTENUATE is

signal n : std_logic_vector (4 downto 0) := (others => '0');

begin
        process (rst, clk)
        begin
              if rising_edge (clk) then
                   if rst = '1' then
                        d_out <= (others => '0');
                        n <= "00000";
                   else
                        if r_event = '1' then
                                if r_direction = '1' then
                                        if n /= x"1f" then
                                                n <= n + 1;
                                        end if;
                                else
                                        if n /= x"00" then
                                                n <= n - 1;
                                        end if;
                                end if;
                        end if;
                        d_out (31-to_integer(unsigned(n)) downto 0) <= d_in (31 downto
                                                                        to_integer(unsigned(n)));
                        d_out (31 downto 31-to_integer(unsigned(n))) <= (others => d_in(31));
```

```
                    end if;
                end if;
        end process;

end rtl;


entity att_select is
                port    (       clk                     : in  STD_LOGIC;
                                rst                     : in  STD_LOGIC;
                                tipka                   : in    std_logic; --sw0
                                on_off                  : in    std_logic; --sw3
                                r_event                 : in    std_logic;
                                r_direction             : in    std_logic;
                                AM_event                : out   std_logic;
                                AM_direction            : out   std_logic;
                                FPM_event               : out   std_logic;
                                FPM_direction           : out   std_logic
                               );
end att_select;

architecture rtl of att_select is

begin
        process (rst, clk, tipka)

        begin
                if rising_edge(clk) then
                        if rst = '1' then
                                AM_event        <= '0';
                                AM_direction    <= '0';
                                FPM_event       <= '0';
                                FPM_direction   <= '0';
                        elsif on_off = '0' then

                                if tipka = '0' then
                                        AM_event        <=      r_event;
                                        AM_direction    <= r_direction;
                                        FPM_event       <= '0';
                                        FPM_direction   <= '0';
                                elsif tipka = '1' then
                                        AM_event        <= '0';
                                        AM_direction    <= '0';
                                        FPM_event       <= r_event;
                                        FPM_direction   <= r_direction;
                                end if;
                        end if;
                end if;
        end process;

end rtl;


entity korak is
                port (  clk             : in  STD_LOGIC;
                        rst             : in  STD_LOGIC;
                        on_off          : in  STD_LOGIC; -- preklopka za uključivanje
                        dds_btn         : in  STD_LOGIC; -- tipka za odabiranje dds-a
                        position        : in  STD_LOGIC; -- tipka za odabir pozicije bita u koraku
                        r_event         : in    std_logic;
                        r_direction     : in    std_logic;
--                      korak0          : in  STD_LOGIC_vector (27 downto 0);
--                      korak1          : in  STD_LOGIC_vector (27 downto 0);
--                      korak2          : in  STD_LOGIC_vector (27 downto 0);
```

```
                led0            : out STD_LOGIC;
                led1            : out STD_LOGIC;
                led2            : out STD_LOGIC;
                korak0_out      : out STD_LOGIC_vector (27 downto 0);
                korak1_out      : out STD_LOGIC_vector (27 downto 0);
                korak2_out      : out STD_LOGIC_vector (27 downto 0)
                        );
end korak;

architecture rtl of korak is

signal dds : STD_LOGIC_vector (1 downto 0) := (others => '0');
signal pos : STD_LOGIC_vector (4 downto 0) := (others => '0');
signal pribroj : STD_LOGIC_vector (27 downto 0) := (others => '0');
signal korak_0 : STD_LOGIC_vector (27 downto 0) := x"007ff00"; -- nosilac 007ff00 nema spike, 0080000 ima
signal korak_1 : STD_LOGIC_vector (27 downto 0) := x"0007f00"; -- PM,FM 07f00 nema spike, 0010000 ima
signal korak_2 : STD_LOGIC_vector (27 downto 0) := x"0007f00"; -- AM 0007f00 nema spike, 0010000 ima

begin

        led0 <= not (dds(0) or dds(1));
        led1 <= dds(0);
        led2 <= dds(1);

        process (clk, rst)

        variable temp_dds, temp_pos : std_logic := '0';

        begin


                if rising_edge(clk) then
                        if rst = '1' then
                                korak0_out <= (others => '0');
                                korak1_out <= (others => '0');
                                korak2_out <= (others => '0');
                        else
                                korak0_out <= korak_0;
                                korak1_out <= korak_1;
                                korak2_out <= korak_2;
                        end if;
                end if;


                if rising_edge(clk) then
                        if rst = '1' then
                                pos <= (others => '0');
                                dds <= "00";
                        elsif on_off = '0' then
                                pos <= (others => '0');
                        else

                                if dds_btn = '1' AND temp_dds = '0' then
                                        dds <= dds + 1;
                                        pos <= (others => '0'); -- pri promjeni dds-a vrati na LSB
                                end if;
                                if dds_btn = '1' then
                                        temp_dds := '1';
                                else
                                        temp_dds := '0';
                                end if;

                                if position = '1' AND temp_pos = '0' then
                                        pos <= pos + 1;
```

```
                                        if pos = "11100" then
                                                pos <= (others => '0');
                                                end if;
                                        pribroj <= (others => '0');
                                        pribroj (to_integer(unsigned(pos))) <= '1';
                                end if;
                                if position = '1' then
                                        temp_pos := '1';
                                else
                                        temp_pos := '0';
                                end if;
                                if dds = "11" then
                                        dds <= "00";
                                end if;

novi_korak:     if r_event = '1' then
                                        case dds is
                                                when "00" =>
                                                        if r_direction = '1' then
                                                                korak_0 <= korak_0 + pribroj;
                                                        else
                                                                korak_0 <= korak_0 - pribroj;
                                                        end if;

                                                when "01" =>
                                                        if r_direction = '1' then
                                                                korak_1 <= korak_1 + pribroj;
                                                        else
                                                                korak_1 <= korak_1 - pribroj;
                                                        end if;

                                                when "10" =>
                                                        if r_direction = '1' then
                                                                korak_2 <= korak_2 + pribroj;
                                                        else
                                                                korak_2 <= korak_2 - pribroj;
                                                        end if;

                                                when "11" =>
                                                        dds <= "00";

                                                when others => null;
                                        end case;
                                end if;
                        end if;
                end if;
        end process;

end rtl;


entity sklopka32 is
                port    (       clk             : in  STD_LOGIC;
                                rst             : in  STD_LOGIC;
                                int_ext         : in  STD_LOGIC; -- tipka-sw
                                prvi_ulaz       : in    std_logic_vector (31 downto 0); -- unutarnji
                                drugi_ulaz      : in    std_logic_vector (31 downto 0); -- vanjski
                                odabrani        : out   std_logic_vector (31 downto 0)
                                );
end sklopka32;

architecture rtl of sklopka32 is
```

```
begin

        process (rst, clk, int_ext)

        begin

                if rising_edge(clk) then
                        if rst = '1' then
                                odabrani <= (others => '0');
                        else
                                if int_ext = '0' then
                                        odabrani <= prvi_ulaz;
                                elsif int_ext = '1' then
                                        odabrani <= drugi_ulaz;
                                end if;
                        end if;
                end if;
        end process;

end rtl;
```

(Vučić, 2010.)

```
entity encoder01 is
  port  (
       clk             : in std_logic;
       reset           : in std_logic;

       rot_a           : in std_logic;
       rot_b           : in std_logic;

       rotary_event      : out std_logic;
       rotary_direction  : out std_logic
       );
end encoder01;

architecture RTL of encoder01 is

  -- local signals
  signal a_reg, b_reg                       : std_logic := '0';
  signal rotary_q1, rotary_q2               : std_logic := '0';
  signal rotary_q1_delay, rot_event         : std_logic := '0';
  signal rotary_event_local, rotary_direction_local  : std_logic := '0';

begin
  -- registering input signals
  process(clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        a_reg <= '0';
        b_reg <= '0';
      else
        a_reg <= not rot_a;
        b_reg <= not rot_b;
      end if;
    end if;
  end process;

  -- generating rotary_q1 and rotary_q2
  process(clk)
    variable rotary_tmp  : std_logic_vector(1 downto 0);
  begin
    if rising_edge(clk) then
      if reset = '1' then
```

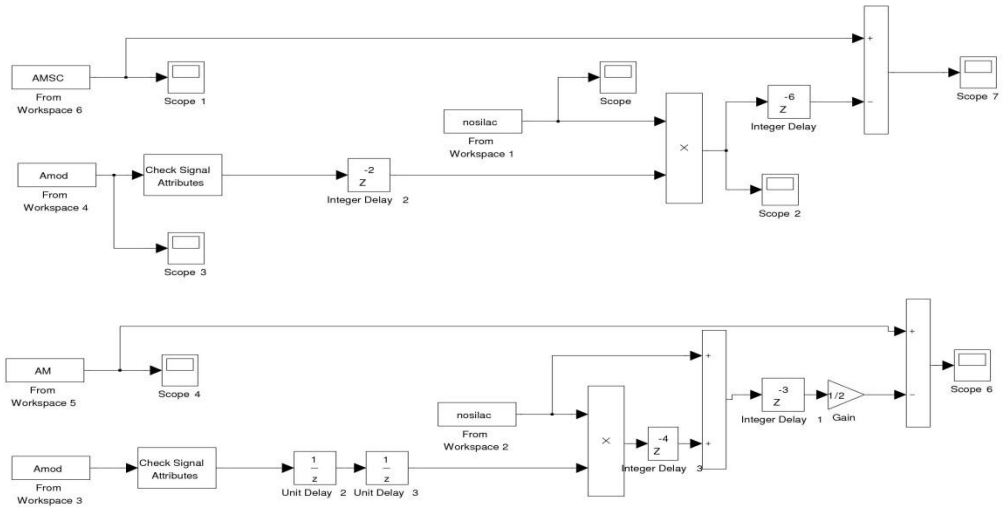
```
          rotary_q1 <= '0';
          rotary_q2 <= '0';
        else
          rotary_tmp := a_reg & b_reg;
          case rotary_tmp is
            when "00" =>   rotary_q1 <= '0';
                           rotary_q2 <= rotary_q2;
            when "01" =>   rotary_q1 <= rotary_q1;
                           rotary_q2 <= '1';
            when "10" =>   rotary_q1 <= rotary_q1;
                           rotary_q2 <= '0';
            when "11" =>   rotary_q1 <= '1';
                           rotary_q2 <= rotary_q2;
            when others => rotary_q1 <= rotary_q1;
                           rotary_q2 <= rotary_q2;
          end case;
        end if;
      end if;
    end process;

    -- registering rotary_q1
    process(clk)
    begin
      if rising_edge(clk) then
        if reset = '1' then
          rotary_q1_delay <= '0';
        else
          rotary_q1_delay <= rotary_q1;
        end if;
      end if;
    end process;

    -- generating rotary_event pulse
    rot_event <= rotary_q1 and not rotary_q1_delay;
    process(clk)
    begin
      if rising_edge(clk) then
        if reset = '1' then
          rotary_event_local <= '0';
        else
          rotary_event_local <= rot_event;
        end if;
      end if;
    end process;

    -- rotary_direction
    process(clk)
    begin
      if rising_edge(clk) then
        if reset = '1' then
          rotary_direction_local <= '0';
        elsif rot_event = '1' then
          rotary_direction_local <= rotary_q2;
        else
          rotary_direction_local <= rotary_direction_local;
        end if;
      end if;
    end process;

    -- output mapping
    rotary_event <= rotary_event_local;
    rotary_direction  <= rotary_direction_local;
  end RTL;
```

```
entity DAC is
   Port (          clk       : in  STD_LOGIC;
                   rst       : in  STD_LOGIC;
                   ulaz      : in      std_logic_vector(11 downto 0);

                   CS        : out STD_LOGIC;
                   CLR       : out STD_LOGIC; -- dac reset active low
                   SCK       : out STD_LOGIC;

                   MOSI : out  STD_LOGIC;

                   SPI_SS_B          : out STD_LOGIC;
                   AMP_CS                    : out STD_LOGIC;
                   AD_CONV                   : out STD_LOGIC;
                   SF_CE0            : out STD_LOGIC;
                   FPGA_INIT         : out STD_LOGIC
                   );
end DAC;

architecture rtl of DAC is

--signal data : std_logic_vector (12 downto 0) := (others => '0');
type dacStateType is (
            start,
            sendBit,
            clockHigh,
            csHigh
);
signal data                         : std_logic_vector (12 downto 0) := (others => '0');
signal dacState   : dacStateType := start;
signal dacData    : unsigned(23 downto 0);
signal data_i         : unsigned(11 downto 0);


begin

            CLR <= not rst;
            -- gasenje ostalih uredaja na spi-ju
            SPI_SS_B          <= '1';
            AMP_CS            <= '1';
            AD_CONV           <= '0';
            SF_CE0            <= '1';
            FPGA_INIT         <= '1';

            process (clk, rst) is

            variable ulaz_i : std_logic_vector (12 downto 0) := (others => '0');
            variable pom : std_logic_vector (12 downto 0) := "0100000000000";
            VARIABLE dacCounter : integer range 0 to 23;

            begin
                        if rising_edge(clk) then
                                    if rst ='1' then
                                                SPI_SS_B          <= '1';
                                                AMP_CS            <= '1';
                                                AD_CONV           <= '0';
                                                SF_CE0            <= '1';
                                                FPGA_INIT         <= '1';

                                                CS                <= '0';
                                                data              <= (others => '0');
                                                data_i            <= (others => '0');
                                                ulaz_i            := (others => '0');
                                                dacState          <= start;
```

```
                else
                        ulaz_i (12) := ulaz (11);
                        ulaz_i (11 downto 0) := ulaz; -- djeljenje s 2

                        data <= ulaz_i + pom;
                        data_i <= unsigned (data(11 downto 0)); -- pomicanje u raspon 0 do 1
                case dacState is
                        when start =>
                                dacData <= "0010" & "1111" & data_i & "0000";
                                CS <= '0';
                                SCK <= '0';
                                dacCounter := 23;
                                dacState <= sendBit;
                        when sendBit =>
                                SCK <= '0';
                                MOSI <= dacData(23);
                                dacData <= dacData(22 downto 0) & "0";
                                dacState <= clockHigh;
                        when clockHigh =>
                                SCK <= '1';
                                if dacCounter = 0 then
                                        dacState <= csHigh;
                                else
                                        dacCounter := dacCounter - 1;
                                        dacState <= sendBit;
                                end if;
                        when csHigh =>
                                CS <= '1';
                                dacState <= start;
                 end case;


                end if;
        end if;

    end process;

end rtl;
```

Simulacija u Simulinku

Amplitudne modulacije



Kutne modulacije