



Dragana Prokin

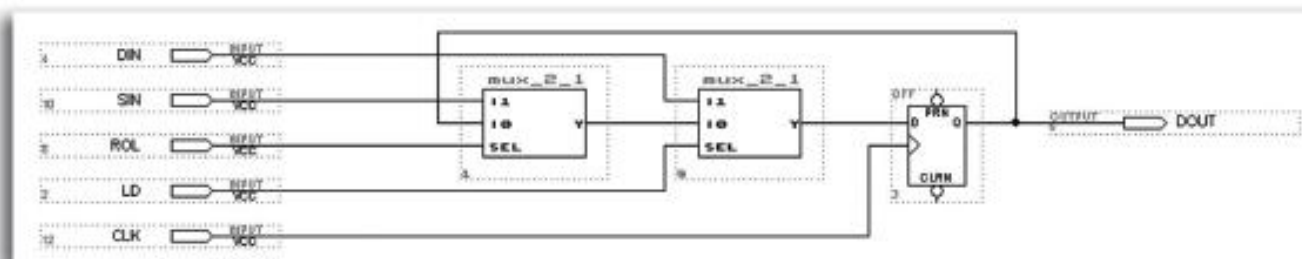
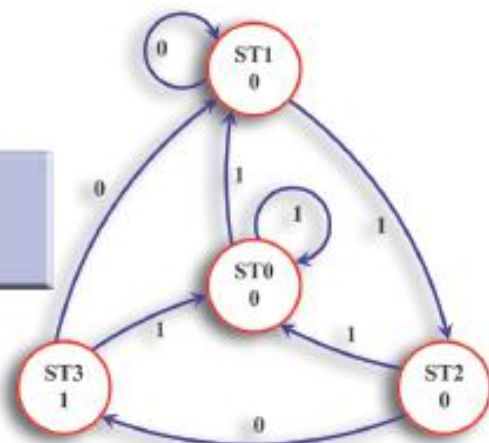
Dušan Todović

PROGRAMABILNA LOGIČKA KOLA

ZBIRKA ZADATAKA

```

ARCHITECTURE Behavior OF priority IS
BEGIN
  PROCESS ( w )
  BEGIN
    IF w(3) = '1' THEN
      y <= "11";
    ELSIF w(2) = '1' THEN
      y <= "10";
    ELSIF w(1) = '1' THEN
      y <= "01";
    ELSE
      y <= "00";
    END IF;
  END PROCESS;
  z <= '0' WHEN w = "0000" ELSE '1';
END Behavior;
  
```



BEOGRAD 2007.

VIŠA ELEKTROTEHNIČKA ŠKOLA

mr Dragana Prokin
Dušan Todović

**ZBIRKA ZADATAKA IZ
PROGRAMABILNIH LOGIČKIH
KOLA**

Beograd, 2007.

Autori: mr Dragana Prokin,
Dušan Todović, dipl. ing.

Recenzenti: mr Borislav Hadžibabić
mr Milan Mijalković

Izdavač: Viša elektrotehnička škola, Beograd
tel. 011 2471 099, Vojvode Stepe, 283

Dizajn korica: Nenad Tolić

Štamparija: “Akademska izdanja”, Beograd

Tiraž: 150 primeraka

ISBN 978-86-85081-84-2

CIP – Katalogizacija u publikaciji
Narodna biblioteka Srbije, Beograd

004 42 004 312(075.8)(076)

PROKIN. Dragana

Zbirka zadataka iz programabilnih
Logičkih kola / Dragana Prokin, Dušan
Todović - Beograd, Viša elektrotehnička
škola, 2007 (Beograd, Akademska
izdanja). - 156 str. : ilustr. ; 30cm
tr. 155.

Tiraž 150. - Bibliografija:

ISBN 978-86-85081-84-2

1. Todović, Dušan

a) Logička kola - Programiranje - Zadaci
COBISS.SR-ID 140224268

Sadržaj

<u>Predgovor</u>	1
<u>Specifikacija sistema grafičkim opisom</u>	3
Zadatak 1	3
<i>Konvertor vrednosti iz binarnog u Gray-ov kod</i>	
Zadatak 2	5
<i>Kolo za aktiviranje sedmosegmentnog displeja na bazi četvorobitne vrednosti</i>	
Zadatak 3	12
<i>Realizacija jednostavne logičke jedinice</i>	
Zadatak 4	15
<i>Demultiplekser 1/4 i 4/16</i>	
Zadatak 5	18
<i>Dekoder 3/8 i 4/16</i>	
Zadatak 6	21
<i>Komparator trobitnih celih pozitivnih brojeva</i>	
Zadatak 7	23
<i>Desetobitni registar sa funkcijom rotiranja vrednosti</i>	
Zadatak 8	25
<i>Osmobitni pomerački registar</i>	
Zadatak 9	28
<i>Osmobitni brojač nagore</i>	
Zadatak 10	31
<i>Četvorobitni brojač i indikator stanja brojača</i>	
Zadatak 11	35
<i>Pomerački registar na bazi „barrel shifter“ kola</i>	
<u>Opis sistema primenom AHDL jezika</u>	39
Zadatak 12	39
<i>Opis sekvencijalne mreže zadate preko eksitacionih jednačina</i>	
Zadatak 13	42
<i>Analiza rada sekvencijalne mreže na osnovu električne šeme</i>	

Zadatak 14	46
<i>Analiza rada sekvencijalne mreže na osnovu električne šeme</i>	
Zadatak 15	49
<i>Opis sekvencijalne mreže zadate preko eksitacionih jednačina</i>	
Zadatak 16	52
<i>Analiza rada sekvencijalne mreže na osnovu električne šeme</i>	
Zadatak 17	55
<i>Obostrani brojač zadate sekvence brojanja</i>	
Zadatak 18	59
<i>Obostrani brojač zadate sekvence brojanja</i>	
Zadatak 19	64
<i>Detektor zadate sekvence</i>	
Zadatak 20	68
<i>Detektor zadate sekvence i mehanizam indikacije detekcije</i>	
Zadatak 21	71
<i>Detektor zadate sekvence i mehanizam indikacije detekcije</i>	
Zadatak 22	79
<i>Sistem za obradu signala sa optičkog enkodera</i>	
<u>Opis sistema primenom VHDL jezika</u>	87
Zadatak 23	87
<i>Programabilan sinhroni brojač nadole sa definisanom granicom brojanja</i>	
Zadatak 24	90
<i>Aritmetička jedinica sa prihvatnim registrom</i>	
Zadatak 25	96
<i>Aritmetičko-logička jedinica sa prihvatnim registrom</i>	
Zadatak 26	101
<i>Sistem za usrednjavanje četiri osmobarne celobrojne binarne vrednosti</i>	
Zadatak 27	105
<i>Generator impulsno-širinski modulisanog signala (PWM generator)</i>	
Zadatak 28	107
<i>Četvorobitni obostrani pomerački registar na bazi „barrel shifter“ kola</i>	
Zadatak 29	111
<i>Jednostavan sistem za upravljanje radom motora</i>	
Zadatak 30	115
<i>RAM memorija</i>	
Zadatak 31	119
<i>Realizacija sistema specificiranog preko dijagrama stanja Murovog tipa</i>	
Zadatak 32	123
<i>Realizacija sistema specificiranog preko dijagrama stanja Milijevog tipa</i>	

<u>Opis sistema preko mega-funkcija i kombinovanom metodom</u>	127
Zadatak 33	127
<i>Brojač i indikator stanja brojača primenom megafunkcija</i>	
Zadatak 34	131
<i>Generator impulsno-širinski modulisanog signala (PWM generator)</i>	
Zadatak 35	135
<i>Generator periodičnog signala</i>	
Zadatak 36	141
<i>Realizacija UART-a u programabilnoj logici</i>	
Zadatak 37	148
<i>Sistem sprege sa AD-konvertorom tipa ADC0831</i>	
<u>Literatura</u>	155

Predgovor

Ova zbirka rešenih zadataka prvenstveno je namenjena studentima koji prate nastavu iz predmeta Programabilna logička kola i svojim sadržajem u potpunosti je usaglašena sa programom navedenog predmeta. Studentima se kroz odabrane urađene primere pruža mogućnost da se bolje upoznaju sa načinom projektovanja i implementiranja digitalnog hardvera primenom programabilnih logičkih kola visokog stepena integracije i samim tim uspešno pripreme za polaganje ispita iz predmeta Programabilna logička kola, na kome se proverava praktično znanje stečeno kroz predavanja, auditorne i laboratorijske vežbe istoimenog predmeta. Zadaci su formirani na osnovu ispitnih zadataka i zadataka rađenih na auditornim vežbama i grupisani su u zavisnosti od načina opisa dizajna. Svaki zadatak obuhvata opis problema, zahtev, neophodnu teorijsku osnovu za rešavanje zadatka, opis dizajna saglasno zahtevu zadatka i proveru ispravnosti rada dizajna primenom simulacionih dijagrama.

Prvo poglavlje sadrži primere realizacije dizajna na bazi grafičkog opisa. Grafičkim opisom rešeni su zadaci jednostavnih digitalnih sistema i to klasičnim postupkom određivanja prenosnih funkcija sistema i implementacijom preko standardnih logičkih kola.

U drugom poglavlju nalaze se primeri realizacije dizajna AHDL opisom. Svi zadaci u okviru ovog poglavlja realizuju sekvencijalne mreže različite namene pri čemu je akcenat stavljen na opis mašina stanja Milijevog i Murovog tipa.

Primeri sa VHDL opisom dizajna složenijih kombinacionih i sekvencijalnih mreža nalaze se u trećem poglavlju.

Četvrto poglavlje obuhvata primere realizacije dizajna u kojima se uočava prednost primene modernih tehnika opisa sistema preko parametrizovanih funkcionalnih blokova, odnosno mega-funkcija. Primenjene su različite tehnike opisa sistema pri čemu je grafički opis sistema korišćen za opis najvišeg hijerarhijskog nivoa dizajna.

Svojom tematikom i složenošću pojedini primeri prevazilaze okvire programa predmeta Programabilna logička kola i oni su namenjeni studentima koji žele da nauče nešto više o projektovanju digitalnog hardvera ili mogu da posluže kao ideja na osnovu koje mogu da se realizuju kompleksniji projekti ili diplomski radovi. Zahvaljujući teorijskim objašnjenjima, koja pomažu boljem razumevanju celokupnog toka rešavanja zadataka, ova zbirka predstavlja zaokruženu celinu, koja kao dopunski udžbenik može da posluži svima onima koji žele da steknu neophodna znanja iz primene programabilnih logičkih kola i principa projektovanja savremenih digitalnih sistema.

Pošto je za realizaciju, testiranje i implementaciju dizajna u programabilnoj logici neophodno poznavanje rada u odgovarajućem softverskom razvojnom okruženju, koje za svoje familije čipova nude proizvođači programabilnih logičkih kola, autori su se opredelili za softversko razvojno okruženje i programabilna logička kola firme Altera, sa kojima se studenti upoznaju u okviru laboratorijskih vežbi. Međutim, u rešenjima zadataka je predstavljen opšti princip realizacije koji može da se posmatra nezavisno od softverske platforme na kojoj se vrši implementacija dizajna, tako da je ova zbirka pogodna i za korisnike koji za svoj dizajn žele da koriste softverska razvojna okruženja drugih proizvođača.

Iskreno se nadamo se da će ova Zbirka biti od koristi sadašnjim studentima i svima onima koji će se u inženjerskoj praksi baviti razvojem digitalnog hardvera.

Specifikacija sistema grafičkim opisom

ZADATAK 1 – Konvertor vrednosti iz binarnog u Gray-ov kod

Realizovati konvertor četvorobitnih vrednosti iz binarnog u *Gray*-ov kodni zapis. U tabeli br. 1.1 dat je niz četvorobitnih vrednosti u binarnom ($Xbin[3..0]$) i njemu odgovarajući niz vrednosti u *Gray*-ovom ($Xgray[3..0]$) kodnom zapisu.

Table 1.1: Binarni i *Gray*-ov kodni zapis brojeva (od 0_{10} do 15_{10})

$Xbin[3..0]$	$Xgray[3..0]$	$Xbin[3..0]$	$Xgray[3..0]$
0000	0000	1000	1100
0001	0001	1001	1101
0010	0011	1010	1111
0011	0010	1011	1110
0100	0110	1100	1010
0101	0111	1101	1011
0110	0101	1110	1001
0111	0100	1111	1000

Odrediti i napisati jednačine konvertora za signale na izlazu $Xgray[3..0]$ u funkciji ulaznih signala $Xbin[3..0]$.

Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati kolo konvertora u grafičkom editoru.

Primenom alata za simulaciju izvršiti simulaciju rada konvertora pravilnim izborom vremenskih oblika signala, vremena trajanja simulacije i vremenskog inkrementa.

Formirati simbol opisanog konvertora.

REŠENJE

Na osnovu specifikacije rada konvertora (tabela br. 1.1), tj. veze između binarnog i *Gray*-ovog zapisa brojeva, formiraju se Karnoove mape za svaki od signala na izlazu ($Xgray3$, $Xgray2$, $Xgray1$ i $Xgray0$) u funkciji signala na ulazu konvertora ($Xbin3$, $Xbin2$, $Xbin1$ i $Xbin0$).

		$Xbin[1..0]$			
$Xbin[3..2]$		00	01	11	10
00					
01					
11		1	1	1	1
10		1	1	1	1

$$Xgray3 = Xbin3$$

		$Xbin[1..0]$			
$Xbin[3..2]$		00	01	11	10
00					
01		1	1	1	1
11					
10		1	1	1	1

$$Xgray2 = \overline{Xbin3} \cdot Xbin2 + Xbin3 \cdot \overline{Xbin2}$$

$$Xgray2 = Xbin3 \oplus Xbin2$$

		$Xbin[1..0]$			
$Xbin[3..2]$		00	01	11	10
	00			1	1
	01	1	1		
	11	1	1		
	10			1	1

$$Xgray1 = Xbin2 \cdot \overline{Xbin1} + \overline{Xbin2} \cdot Xbin1$$

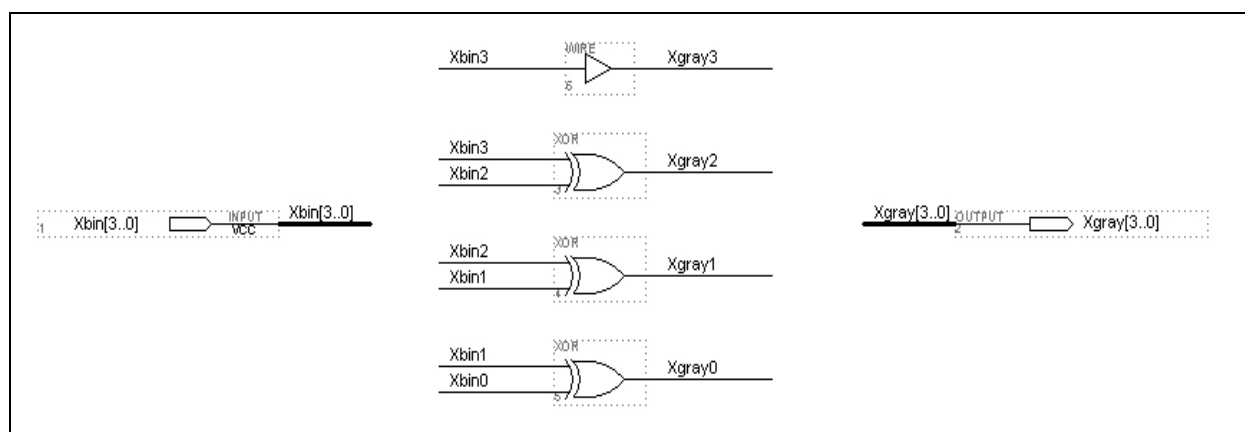
$$Xgray1 = Xbin2 \oplus Xbin1$$

		$Xbin[1..0]$			
$Xbin[3..2]$		00	01	11	10
	00		1		1
	01		1		1
	11		1		1
	10		1		1

$$Xgray0 = \overline{Xbin1} \cdot Xbin0 + Xbin1 \cdot \overline{Xbin0}$$

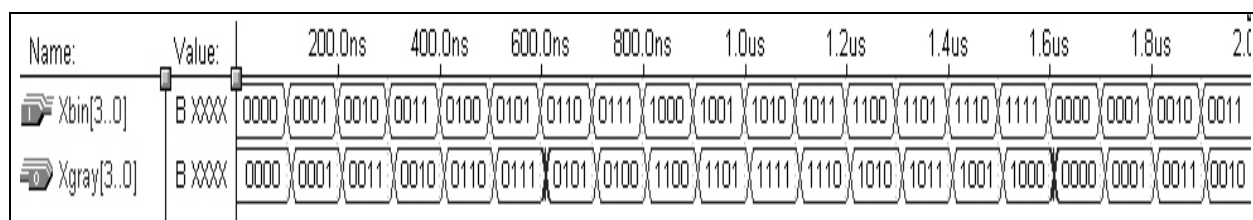
$$Xgray0 = Xbin1 \oplus Xbin0$$

Implementacija izvedenih jednačina koje specificiraju rad konvertora vrši se u grafičkom editoru softverskog razvojnog okruženja i to isključivo primenom elementarnih logičkih kola. Na slici br. 1.1 dat je grafički opis zahtevanog konvertora.



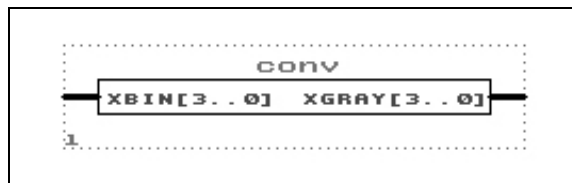
Slika 1.1: Grafički opis postavljenog zadatka

Kako bi se uverili da opisana realizacija zadovoljava zahteve zadatka potrebno je izvršiti simulaciju dizajna. Simulacija omogućava verifikaciju opisanog dizajna u okviru razvojnog alata, tj. pre provere i primene sistema u realnim uslovima rada. Formiranje simulacionih dijagrama za verifikaciju opisanog sistema sastoji se u uključivanju svih ulaznih signala i signala na izlazu koje želimo testirati. Nakon toga, vrši se specifikacija oblika ulaznih signala i zatim aktiviranje procesa simulacije. Vremenski dijagram simulacije dat je na slici br. 1.2.



Slika 1.2: Vremenski dijagram simulacije konvertora vrednosti

Na slici br. 1.3 dat je prikaz simbola konvertora vrednosti iz binarnog u *Gray*-ov kodni zapis. Imena signala *XBIN[3..0]* i *XGRAY[3..0]* u okviru simbola napisana su velikim slovima i to je proizvod realizacije simbola u okviru ALTERINO razvojnog alata. *XBIN[3..0]* i *XGRAY[3..0]* odgovaraju signalima *Xbin[3..0]* i *Xgray[3..0]* sa slike br. 1.1.

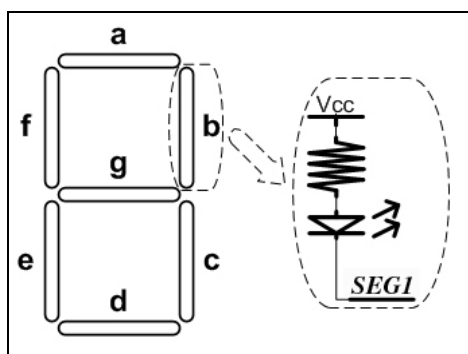


Slika 1.3: Simbol elementa konvertora vrednosti

Gray-ov (Grejov) kod se odlikuje nizom brojeva kod koga se dva susedna broja razlikuju samo po vrednosti na jednoj poziciji bita. Ovakav kod je jako povoljan za realizaciju brojača gde pri inkrementiranju ili dekrementiranju ne dolazi do promene većeg broja bita kao što je slučaj kod binarnog koda i samim tim ne dolazi do pojave lažnih vrednosti.

ZADATAK 2 – Kolo za aktiviranje sedmosegmentnog displeja na bazi četvorobitne vrednosti

Realizovati konvertor binarne vrednosti u niz signala preko kojih se pobuđuje sedmosegmentni displej. Konvertor treba realizovati kao kombinacionu mrežu za aktiviranje sedmosegmentnog displeja na bazi informacije koja se dovodi četvorobitnim ulaznim vektorom signala *B[3..0]*. Sedmosegmentni displej se još uvek često koristi za prikaz manjeg broja cifara, npr. kod časovnika, raznih tajmera, kućnih aparata, cenovnika i sl. zbog dobre uočljivosti pri različitim uslovima osvetljenja radnog okruženja. Brojnu vrednost u opsegu od $0_{(10)}$ do $15_{(10)}$ koju želimo da prikazemo na sedmosegmentnom displeju dovodimo kao odgovarajuću binarnu vrednost signalima *B[3..0]*. Broj se na sedmosegmentnom displeju prikazuje kao heksadecimalni zapis u opsegu od $0_{(16)}$ do $F_{(16)}$. Sedmosegmentni displej se sastoji od sedam svetlećih segmenata *a, b, c, d, e, f* i *g* čiji je raspored, način označavanja i pobuđivanja prikazan na slici br. 2.1.



Slika 2.1: Raspored, način označavanja segmenata i način pobuđivanja sedmosegmentnog displeja

Signal *B0* sadrži bit najmanje težine u binarnom zapisu broja koji se signalima *B[3..0]* dovodi na ulaz. Signali za aktiviranje sedmosegmentnog displeja su *SEG0, SEG1, SEG2, SEG3, SEG4, SEG5* i *SEG6* koji redom odgovaraju segmentima *a, b, c, d, e, f* i *g*. Segment displeja je aktivan, tj. svetli, kada je nivo odgovarajuće linije signala *SEGx* koja je pridružena tom segmentu jednaka '0'.

Na osnovu prethodnog opisa formirati tablicu pobuđivanja segmenata sedmosegmentnog displeja (tablicu istinitosti) za sve vrednosti podataka na ulazu $B[3..0]$.

Odrediti i napisati jednačine konvertora za signale na izlazu $SEG0$, $SEG1$, $SEG2$, $SEG3$, $SEG4$, $SEG5$ i $SEG6$ u funkciji ulaznih signala $B[3..0]$.

Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati kolo konvertora u grafičkom editoru.

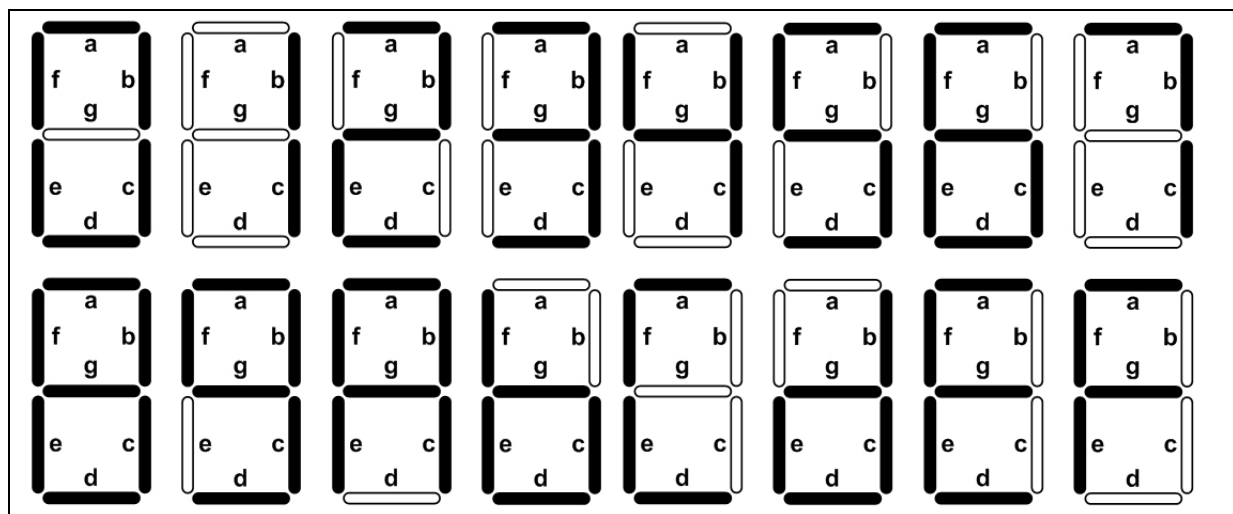
Primenom alata za simulaciju iz softverskog razvojnog okruženja, izvršiti simulaciju rada konvertora pravilnim izborom vremenskih oblika signala, vremena trajanja simulacije i vremenskog inkrementa.

Formirati simbol opisanog konvertora.

U zavisnosti od načina povezivanja sedmosegmentnog displeja, postoje realizacije u kojima se segmenti aktiviraju visokim odnosno niskim nivoom signala. Predložiti rešenje kojim se brzo i efikasno od realizovanog konvertora formira novi konvertor sa mogućnošću izbora nivoa signala za pobuđivanje segmenata displeja. Izbor se vrši preko kontrolnog signala POL . Za vrednost kontrolnog signala $POL='0'$ segmenti displeja se aktiviraju niskim a za $POL='1'$ visokim logičkim nivoom.

REŠENJE

U cilju jednostavnijeg rešavanja problema, na početku izrade zadatka formira se skica svih vrednosti od $0x0$ do $0xF$ koji se mogu prikazati na sedmosegmentnom displeju (slika br. 2.2). Zatamljeni segmenti na slici br. 2.2 odgovaraju segmentima koji su aktivirani, tj. upaljeni. Na osnovu slike br. 2.2 formira se tablica istinitosti kombinacione mreže konvertora.



Slika 2.2: Izgled aktiviranih segmenata za sve vrednosti iz opsega od $0x0$ do $0xF$

U tabeli br. 2.1 data je tablica istinitosti konvertora, tj. tablica koja povezuje signale na izlazu $SEG0$, $SEG1$, $SEG2$, $SEG3$, $SEG4$, $SEG5$ i $SEG6$ sa signalima na ulazu $B[3..0]$. Polja u tabeli br. 2.1 koja odgovaraju izlazima konvertora nisu popunjena nulama zbog postizanja preglednije forme.

Na osnovu tablice istinitosti (tabela br. 2.1) formiraju se Karnoove mape za svaki od izlaznih signala $SEGx$ (gde je $x=0..6$) kako bi se utvrdile jednačine izlaza u funkciji signala na ulazu $B3$, $B2$, $B1$ i $B0$.

Tabela 2.1: Tablica istinitosti

<i>Ulaz</i>				<i>Segment / Izlaz</i>						
				<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>
<i>B3</i>	<i>B2</i>	<i>B1</i>	<i>B0</i>	<i>SEG0</i>	<i>SEG1</i>	<i>SEG2</i>	<i>SEG3</i>	<i>SEG4</i>	<i>SEG5</i>	<i>SEG6</i>
0	0	0	0							1
0	0	0	1	1			1	1	1	1
0	0	1	0			1			1	
0	0	1	1					1	1	
0	1	0	0	1			1	1		
0	1	0	1		1			1		
0	1	1	0		1					
0	1	1	1				1	1	1	1
1	0	0	0							
1	0	0	1					1		
1	0	1	0				1			
1	0	1	1	1	1					
1	1	0	0		1	1				1
1	1	0	1	1					1	
1	1	1	0		1	1				
1	1	1	1		1	1	1			

U nastavku je dato sedam Karnoovih mapa kao i funkcije signala *SEG0*, *SEG1*, ..., *SEG6*.

		B1 B0			
B3 B2	00	00	01	11	10
			1		
		1			
			1		
					1

$$\begin{aligned}
 SEG0 = & \\
 & \overline{B3} \cdot \overline{B2} \cdot \overline{B1} \cdot B0 + \\
 & \overline{B3} \cdot B2 \cdot \overline{B1} \cdot \overline{B0} + \\
 & B3 \cdot \overline{B2} \cdot B1 \cdot B0 + \\
 & B3 \cdot B2 \cdot \overline{B1} \cdot B0
 \end{aligned}$$

		B1 B0			
B3 B2	00	00	01	11	10
			1		
					1
		1		1	1
				1	

$$\begin{aligned}
 SEG1 = & \\
 & \overline{B3} \cdot B2 \cdot \overline{B1} \cdot B0 + \\
 & B3 \cdot B2 \cdot \overline{B0} + \\
 & B2 \cdot B1 \cdot \overline{B0} + \\
 & B3 \cdot B1 \cdot B0
 \end{aligned}$$

		B1 B0			
B3 B2		00	01	11	10
	00				1
	01				
	11	1		1	1
	10				

$$\begin{aligned}
 SEG2 = & \\
 & \overline{B3} \cdot \overline{B2} \cdot B1 \cdot \overline{B0} + \\
 & B3 \cdot B2 \cdot \overline{B0} + \\
 & B3 \cdot B2 \cdot B1
 \end{aligned}$$

		B1 B0			
B3 B2		00	01	11	10
	00		1		
	01	1		1	
	11			1	
	10				1

$$\begin{aligned}
 SEG3 = & \\
 & \overline{B3} \cdot B2 \cdot \overline{B1} \cdot \overline{B0} + \\
 & \overline{B3} \cdot \overline{B2} \cdot \overline{B1} \cdot B0 + \\
 & B3 \cdot \overline{B2} \cdot B1 \cdot \overline{B0} + \\
 & B2 \cdot B1 \cdot B0
 \end{aligned}$$

		B1 B0			
B3 B2		00	01	11	10
	00		1	1	
	01	1	1	1	
	11				
	10		1		

$$\begin{aligned}
 SEG4 = & \\
 & \overline{B3} \cdot B0 + \\
 & \overline{B3} \cdot B2 \cdot \overline{B1} + \\
 & \overline{B2} \cdot \overline{B1} \cdot B0
 \end{aligned}$$

		B1 B0			
B3 B2		00	01	11	10
	00		1	1	1
	01			1	
	11		1		
	10				

$$\begin{aligned}
 SEG5 = & \\
 & B3 \cdot B2 \cdot \overline{B1} \cdot B0 + \\
 & \overline{B3} \cdot B1 \cdot B0 + \\
 & \overline{B3} \cdot \overline{B2} \cdot B0 + \\
 & \overline{B3} \cdot \overline{B2} \cdot B1
 \end{aligned}$$

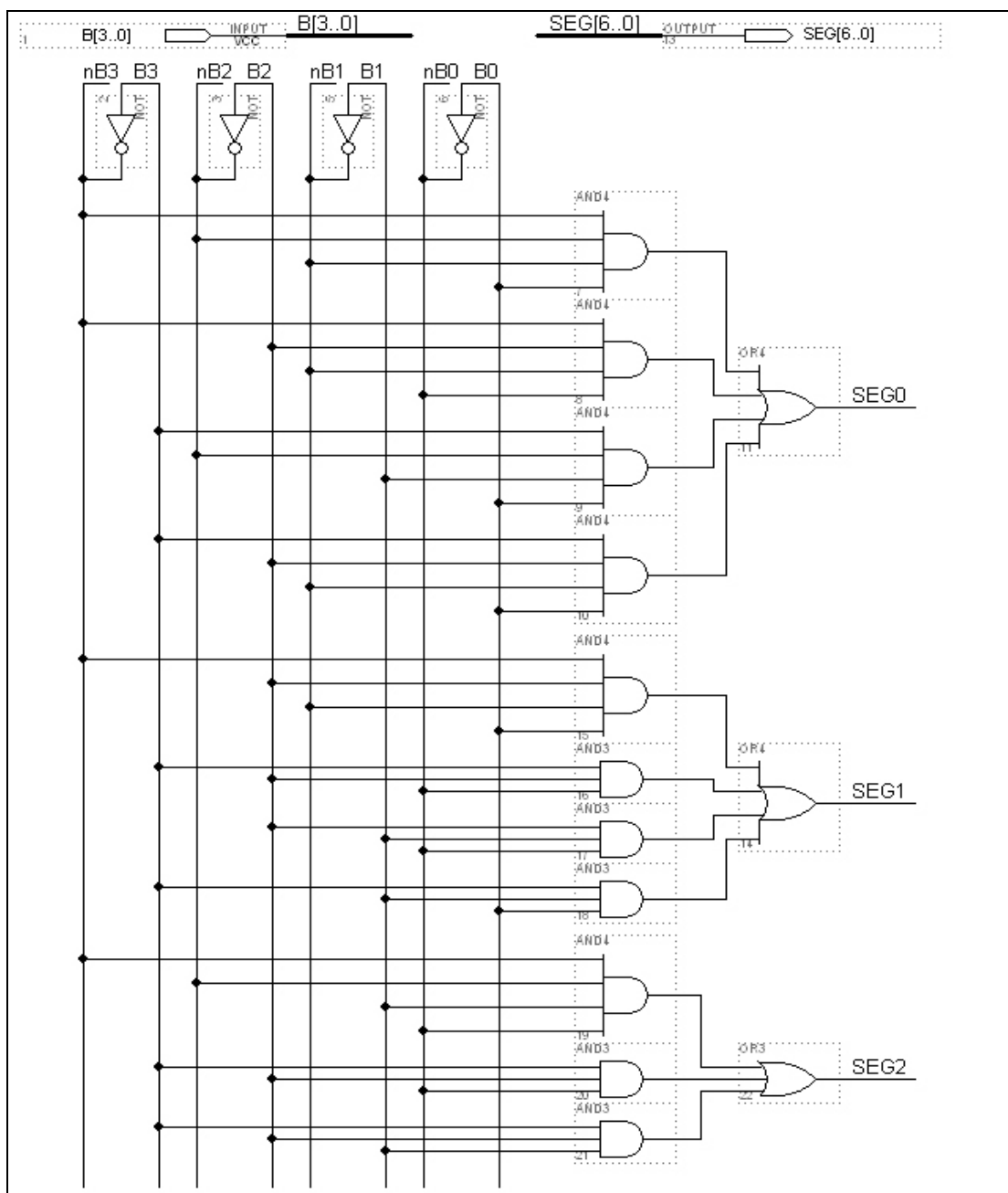
		B1 B0			
B3 B2		00	01	11	10
	00	1	1		
	01			1	
	11	1			
	10				

$$\begin{aligned}
 SEG6 = & \\
 & \overline{B3} \cdot \overline{B2} \cdot \overline{B1} + \\
 & \overline{B3} \cdot B2 \cdot B1 \cdot B0 + \\
 & B3 \cdot B2 \cdot \overline{B1} \cdot \overline{B0}
 \end{aligned}$$

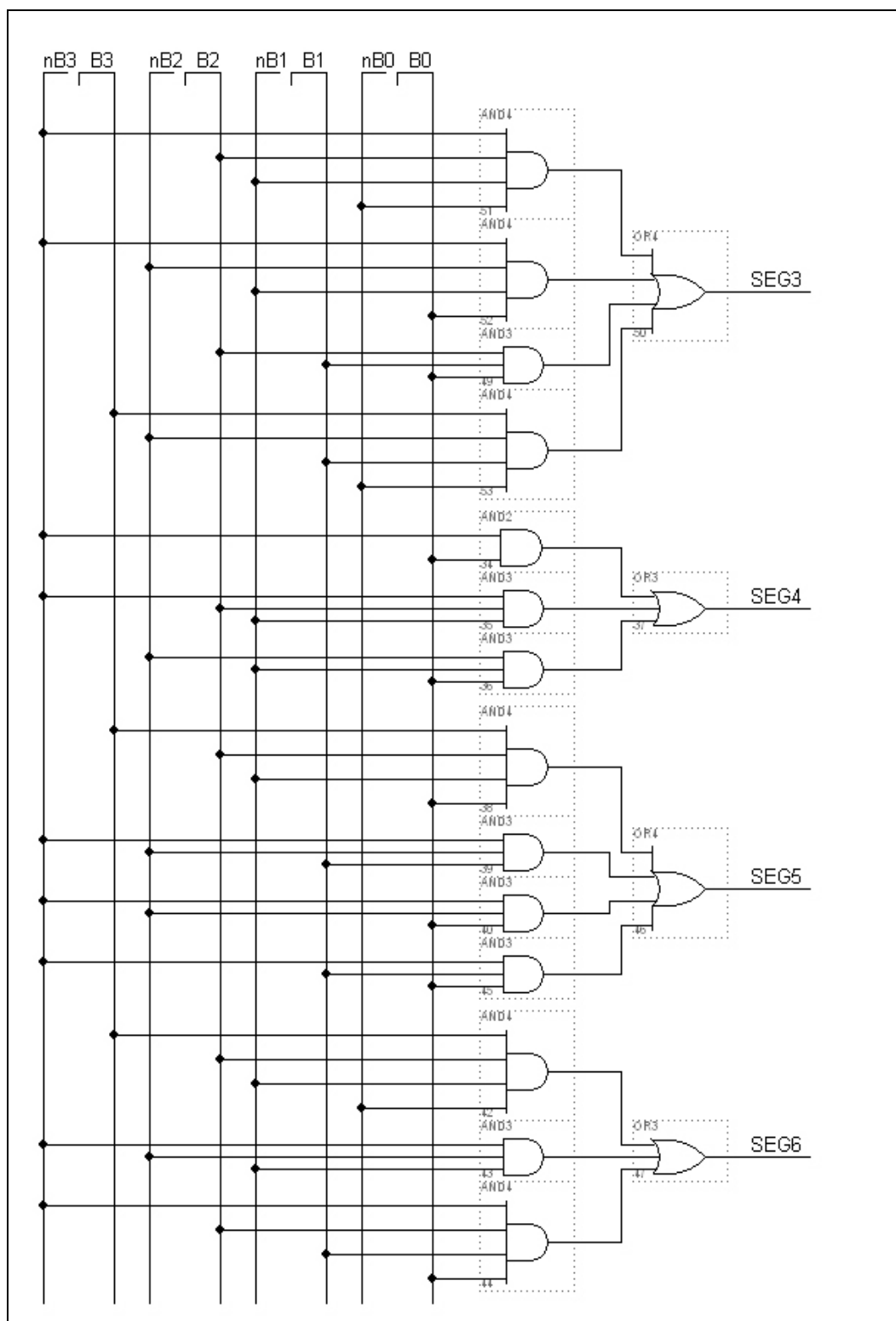
Direktnom implementacijom jednačina signala na izlazu:

$$\begin{aligned}
 SEG1 &= \overline{B3} \cdot \overline{B2} \cdot \overline{B1} \cdot B0 + \overline{B3} \cdot \overline{B2} \cdot B1 \cdot \overline{B0} + \overline{B3} \cdot B2 \cdot \overline{B1} \cdot \overline{B0} + \overline{B3} \cdot B2 \cdot B1 \cdot B0, \\
 SEG2 &= \overline{B3} \cdot \overline{B2} \cdot B1 \cdot \overline{B0} + \overline{B3} \cdot \overline{B2} \cdot \overline{B1} \cdot B0 + \overline{B3} \cdot B2 \cdot \overline{B1} \cdot B0 + \overline{B3} \cdot B2 \cdot B1 \cdot \overline{B0}, \\
 SEG3 &= \overline{B3} \cdot \overline{B2} \cdot \overline{B1} \cdot \overline{B0} + \overline{B3} \cdot \overline{B2} \cdot \overline{B1} \cdot B0 + \overline{B3} \cdot \overline{B2} \cdot B1 \cdot \overline{B0} + \overline{B3} \cdot \overline{B2} \cdot B1 \cdot B0, \\
 SEG4 &= \overline{B3} \cdot B0 + \overline{B3} \cdot B2 \cdot \overline{B1} + \overline{B2} \cdot \overline{B1} \cdot B0, \\
 SEG5 &= B3 \cdot \overline{B2} \cdot \overline{B1} \cdot B0 + \overline{B3} \cdot B1 \cdot B0 + \overline{B3} \cdot \overline{B2} \cdot B0 + \overline{B3} \cdot \overline{B2} \cdot B1 \cdot B0, \\
 SEG6 &= \overline{B3} \cdot \overline{B2} \cdot \overline{B1} + \overline{B3} \cdot \overline{B2} \cdot B1 \cdot B0 + \overline{B3} \cdot B2 \cdot \overline{B1} \cdot \overline{B0}
 \end{aligned}$$

u grafičkom editoru razvojnog alata za rad sa programabilnim logičkim kolima firme ALTERA dobija se šema kola konvertora koje su prikazane na slikama br. 2.3 i 2.4. Zbog obimnosti realizacije, šema kola je podeljena na dve slike.



Slika 2.3: Šema konvertora (I deo)

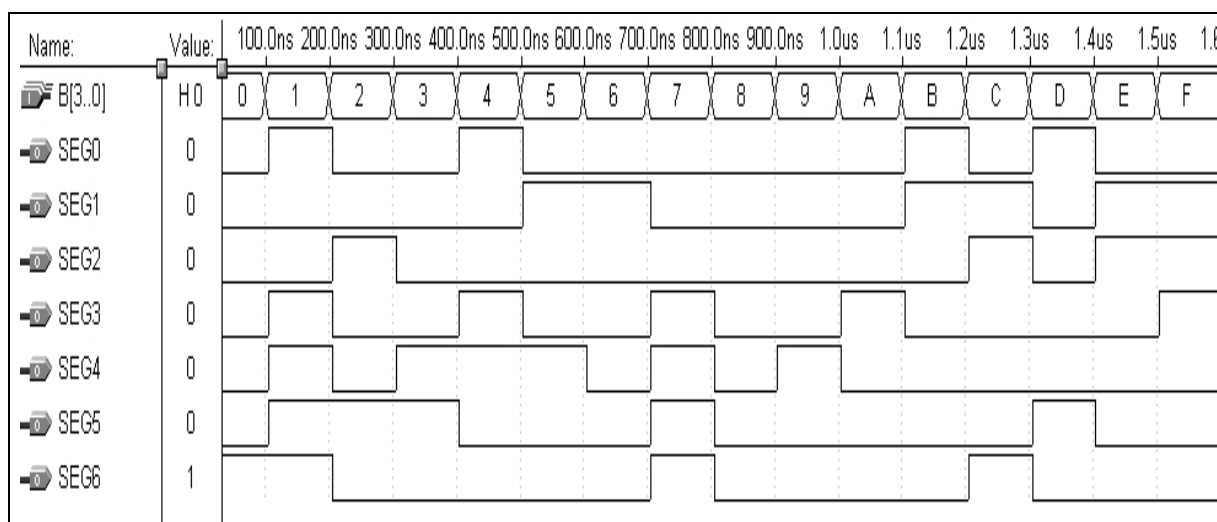


Slika 2.4: Šema konvertora (II deo)

Na slici br. 2.5 dat je vremenski dijagram simulacije rada konvertora. Na dijagramu su pokrivene sve vrednosti signala na ulazu. Na slici br. 2.6 dat je simbol kola konvertora.

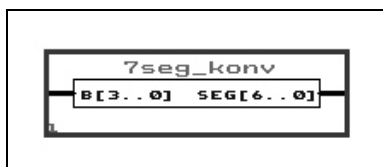
U praksi se podjednako koriste sedmosegmentni displeji koji su u konfiguraciji sa zajedničkom anodom (slučaj u okviru ovog zadatka koji je prikazan na slici br. 2.1) i zajedničkom katodom. U prvom slučaju konfiguracije, segment displeja se aktivira

postavljanjem vrednosti '0' na odgovarajuću liniju signala koja je pridružena segmentu dok se u drugom slučaju segment aktivira dovođenjem nivoa '1'.



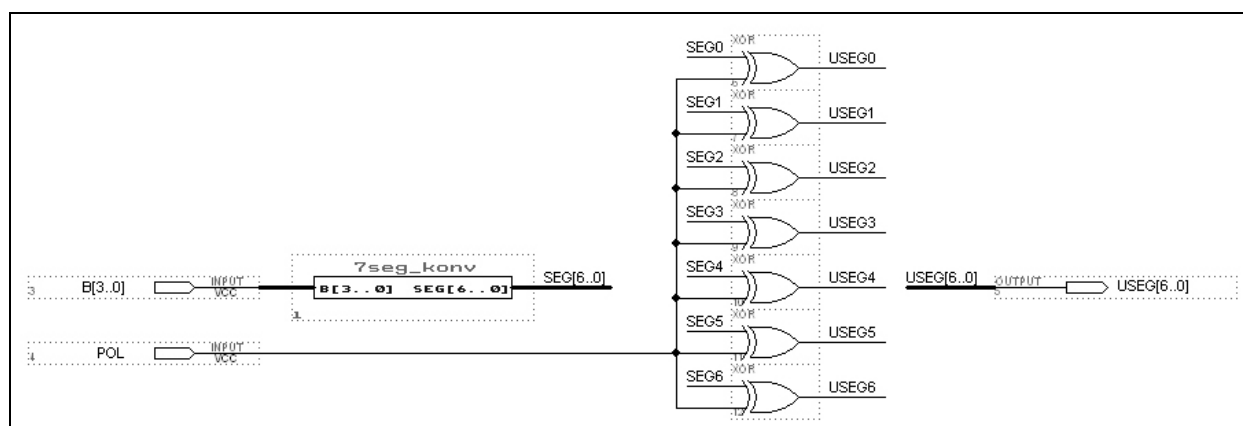
Slika 2.5: Vremenski dijagram simulacije konvertora

Na bazi realizacije konvertora za primenu kod konfiguracije sedmosegmentnog displeja sa zajedničkom anodom, formira se konvertor koji ima mogućnost podešavanja aktivnog nivoa signala na izlazu, tj. signala kojim se pobuđuje svetlosni segment displeja – za primenu u konfiguracijama displeja sa zajedničkom anodom i zajedničkom katodom. Na slici br. 2.7 data je realizacija poboljšanog kola konvertora sa kontrolnim signalom *POL*.



Slika 2.6: Simbol konvertora

Promena polariteta signala na izlazu konvertora u poboljšanoj verziji (slika br. 2.7) postiže se preko sedam XOR logičkih kola.



Slika 2.7: Šema poboljšanog kola konvertora sa signalom selekcije nivoa signala za pobuđivanje svetlosnog segmenta sedmosegmentnog displeja

ZADATAK 3 – Realizacija jednostavne logičke jedinice

Realizovati jednostavnu logičku jedinicu koja ima mogućnost određivanja osam različitih operacija nad dva jednobitna podatka u zavisnosti od vrednosti kontrolnih signala. Jednostavna logička jedinica sadrži sledeće signale:

- dva jednobitna ulazna podatka A i B ;
- tri kontrolna signala $C0$, $C1$ i $C2$ preko kojih se obavlja selekcija moda rada jedinice tj. izbor tipa logičke operacije i
- jedan jednobitni izlazni podatak F – rezultat obrade logičke jedinice.

Izbor operacije logičke jedinice obavlja se preko tri jednobitna kontrolna signala $C0$, $C1$ i $C2$. U tabeli br. 3.1 data je specifikacija rada logičke jedinice na bazi vrednosti kontrolnih signala.

Tabela 3.1: Specifikacija rada jednostavne logičke jedinice

kontrolni signali			izlaz	Opis funkcije
$C0$	$C1$	$C2$	F	
0	0	0	1	»uvek 1«
0	0	1	$A + B$	logička operacija OR
0	1	0	$\overline{A \cdot B}$	logička operacija NAND
0	1	1	$A \otimes B$	logička operacija XOR
1	0	0	$\overline{A \otimes B}$	logička operacija XNOR
1	0	1	$A \cdot B$	logička operacija AND
1	1	0	$\overline{A + B}$	logička operacija NOR
1	1	1	0	»uvek 0«

Odrediti i napisati jednačinu izlaznog podatka F u funkciji ulaznih signala A , B , $C0$, $C1$ i $C2$. Primenom softverskog paketa za rad sa proizvodima firme ALTERA realizovati jednostavnu logičku jedinicu u grafičkom editoru.

Primenom alata za simulaciju i pravilnim izborom vremenskih oblika signala, vremena trajanja simulacije i vremenskog inkrementa izvršiti simulaciju rada opisanog sistema.

REŠENJE

Pri projektovanju logičke jedinice, treba uzeti u obzir da je to kombinaciona mreža sa pet ulaza i jednim izlazom, pri čemu vrednost na izlazu zavisi od kombinacije binarnih vrednosti kontrolnih signala ($C0..C2$), kojima se obavlja izbor logičke operacije nad binarnim vrednostima koje se dovode na ulaze A i B . Da bi se izvršila realizacija ove kombinacione mreže treba formirati tabelu istinitosti, na osnovu koje će se dobiti analitički izraz za izlazni signal F . Na osnovu specifikacije rada logičke jedinice (tabela br. 3.1) formira se tablica istinitosti koja je prikazana u tabeli br. 3.2.

Tabela 3.2: Tablica istinitosti

<i>C0</i>	<i>C1</i>	<i>C2</i>	<i>A</i>	<i>B</i>	<i>F</i>
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	0

<i>C0</i>	<i>C1</i>	<i>C2</i>	<i>A</i>	<i>B</i>	<i>F</i>
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

Pošto je predviđeno da se unos dizajna obavi u grafičkom editoru, neophodno je da se na osnovu tabele istinitosti izvrši minimizacija analitičkog izraza za izlazni signal *F* primenom Karnoove mape. Na osnovu tablice istinitosti (tabela br. 3.2) i preko formiranja Karnoovih mapa određuje se izraz za signal *F* u funkciji pet ulaznih signala *C0*, *C1*, *C2*, *A* i *B*. Problem minimizacije funkcije sa pet ulaznih promenljivih može da se svede na minimizaciju funkcije sa četiri ulazne promenljive, ukoliko se promenljiva koja predstavlja najviši bit u kombinaciji ulaznih vrednosti (*C0*) proglasi za konstantu.

Određivanje izraza za signal *F* na bazi pet ulaznih signala obavlja se preko dve Karnoove mape 4x4 i to za *C0*=0 i *C0*=1.

C0=0

		A B				
		C1 C2	00	01	11	10
C1 C2	00		1	1	1	1
	01			1	1	1
	11			1		1
	10		1	1		1

C0=1

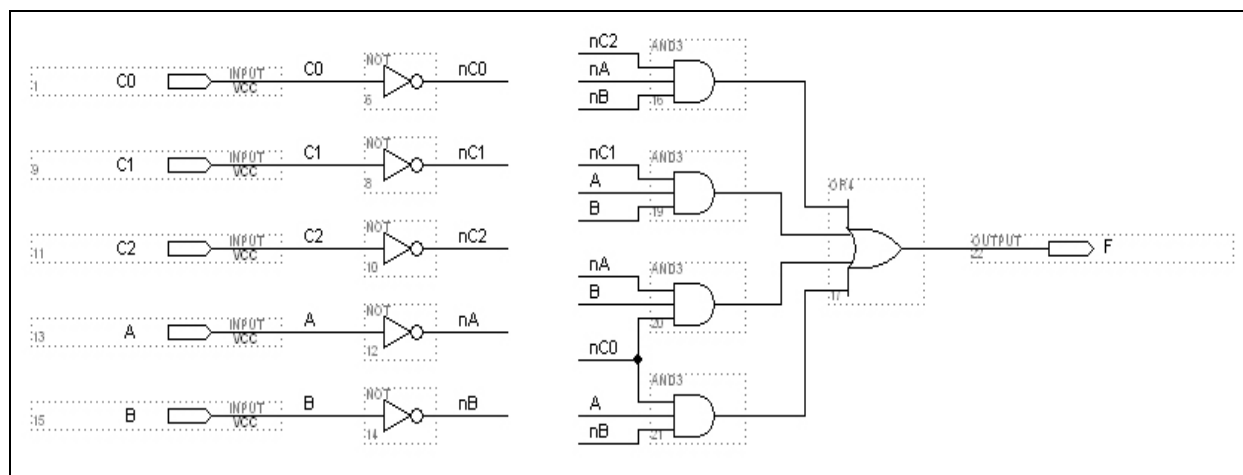
		A B				
		C1 C2	00	01	11	10
C1 C2	00		1		1	
	01				1	
	11					
	10		1			

Izraz izlaznog signala *F* je:

$$F = \overline{C0} \cdot (\overline{C2} \cdot \overline{A} \cdot \overline{B} + \overline{C1} \cdot A \cdot B + \overline{A} \cdot B + A \cdot \overline{B}) + C0 \cdot (\overline{C2} \cdot \overline{A} \cdot \overline{B} + \overline{C1} \cdot A \cdot B)$$

$$F = \overline{C2} \cdot \overline{A} \cdot \overline{B} \cdot (\overline{C0} + C0) + \overline{C1} \cdot A \cdot B \cdot (\overline{C0} + C0) + \overline{C0} \cdot \overline{A} \cdot B + \overline{C0} \cdot A \cdot \overline{B}$$

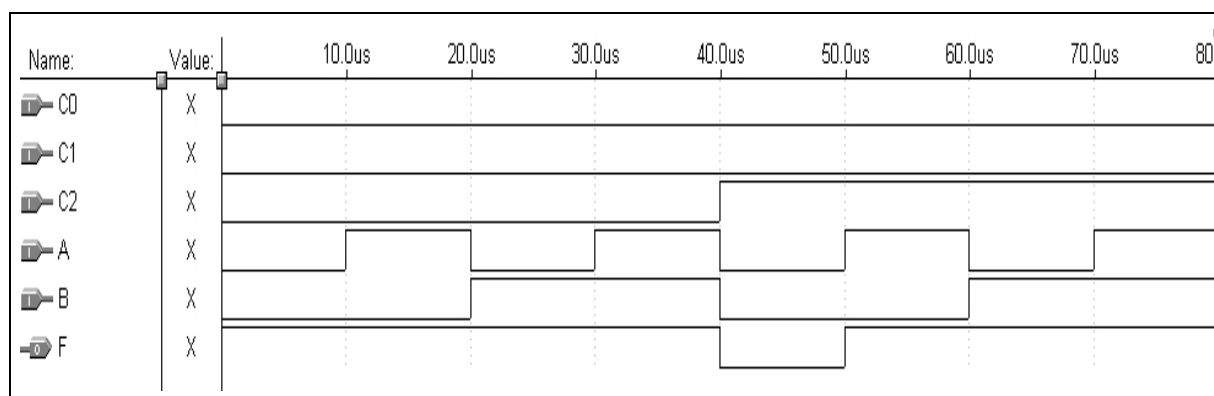
$$F = \overline{C2} \cdot \overline{A} \cdot \overline{B} + \overline{C1} \cdot A \cdot B + \overline{C0} \cdot \overline{A} \cdot B + \overline{C0} \cdot A \cdot \overline{B}$$



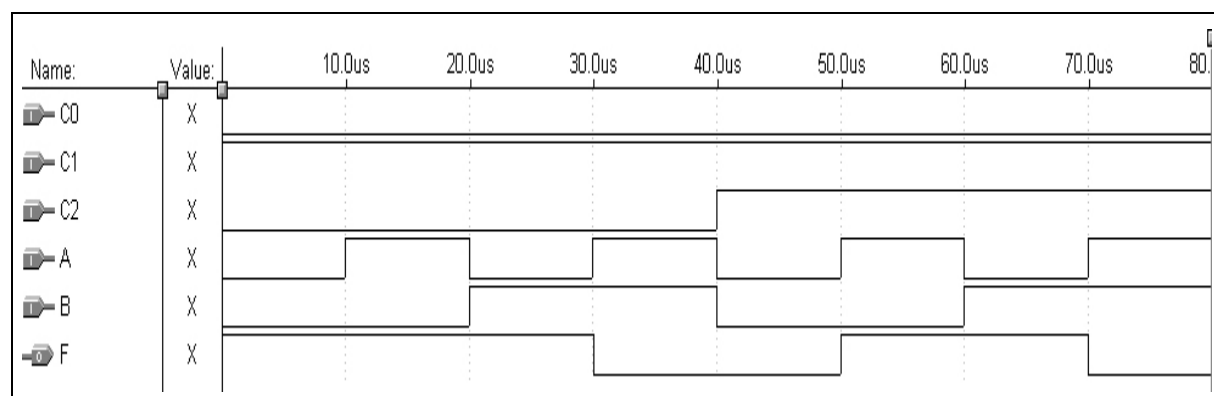
Slika 3.1: Grafički opis postavljenog zadatka

U grafičkom editoru razvojnog alata i to isključivo primenom elementarnih logičkih kola vrši se implementacija dobijenog izraza za signal F . Na slici br. 3.1 dat je grafički opis zahtevane logičke jedinice.

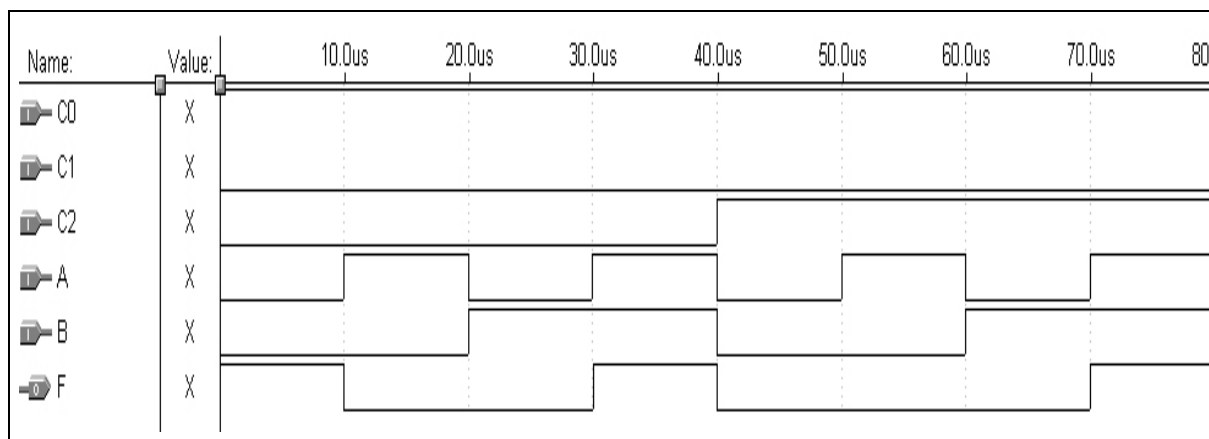
Kako bi se uverili da opisana realizacija obavlja zahteve zadatka potrebno je izvršiti simulaciju dizajna. Formiranje simulacionih dijagrama za verifikaciju opisanog sistema sastoji se u uključivanju svih ulaznih signala (A , B , $C0$, $C1$ i $C2$) i signala na izlazu F . Nakon toga, vrši se specifikacija oblika ulaznih signala i zatim aktiviranje procesa simulacije. Vremenski dijagrami simulacije za sve modove rada logičke jedinice dati su na slikama br. 3.2, 3.3, 3.4 i 3.5.



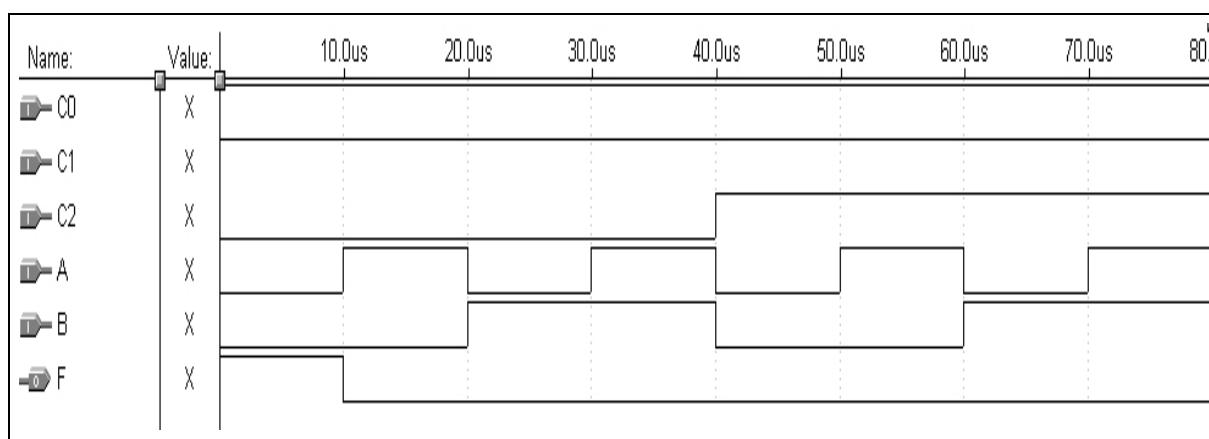
Slika 3.2: Vremenski dijagram simulacije operacije uvek "1" i OR



Slika 3.3: Vremenski dijagram simulacije operacije NAND i XOR



Slika 3.4: Vremenski dijagram simulacije operacije XNOR i AND



Slika 3.5: Vremenski dijagram simulacije operacije NOR i "uvek 0"

ZADATAK 4 – Demultiplekser 1/4 i 4/16

Realizovati demultiplekser 1/4 (*demux14*) koji sadrži tri jednobitna signala na ulazu (*I*, *SEL0* i *SEL1*) i četiri jednobitna signala na izlazu (*Q3*, *Q2*, *Q1* i *Q0*). *I* je signal podatka dok su *SEL0* i *SEL1* signali selekcije izlaza, odnosno signali za kontrolu rada demultipleksera.

Napisati jednačine za signale na izlazu *Q3*, *Q2*, *Q1* i *Q0* u funkciji ulaznih signala *I*, *SEL0* i *SEL1*.

Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati kolo demultipleksera u grafičkom editoru. Formirati simbol realizovanog demultipleksera *demux14*.

Na bazi realizovanog demultipleksera 1/4, tj. kola *demux14*, formirati demultiplekser kojim se demultipleksira četvorobitni podatak koji je doveden preko ulaznih priključaka *BUS[3..0]* u četiri četvorobitna podatka na izlazu: *A[3..0]*, *B[3..0]*, *C[3..0]* i *D[3..0]*. Rad demultipleksera kontrolisan je signalima *SEL1* i *SEL0*. Obezbediti hijerarhijsku organizaciju sistema.

Primenom alata za simulaciju izvršiti simulaciju rada demultipleksera pravilnim izborom vremenskih oblika signala, vremena trajanja simulacije i vremenskog inkrementa.

REŠENJE

Opis rada demultipleksera 1/4 prikazan je u tabeli br. 4.1. U zavisnosti od vrednosti signala *SEL1*, *SEL0* vrši se preusmeravanje ulaznog signala *I* na jedan od četiri izlaza *Q3*, *Q2*, *Q1* ili *Q0*.

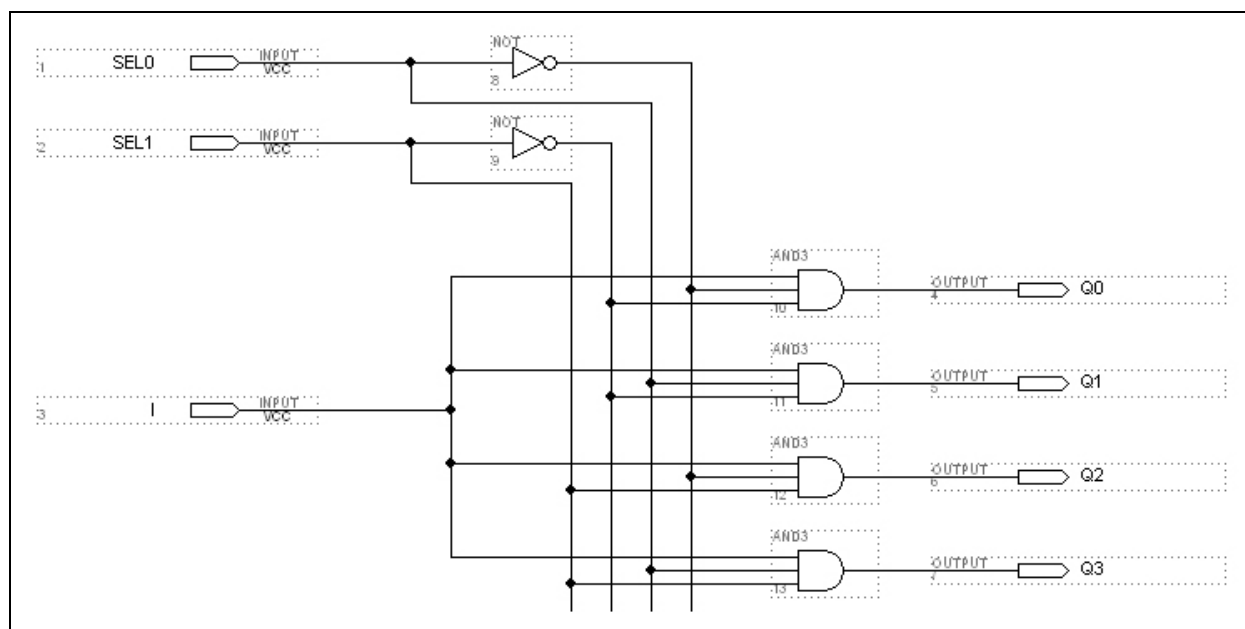
Tabela 4.1: Specifikacija rada demultipleksera 1/4

Ulazi		Izlazi			
SEL1	SEL0	Q3	Q2	Q1	Q0
0	0	0	0		I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

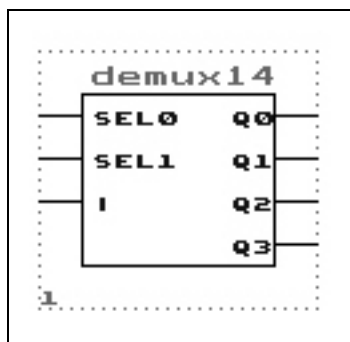
Zbog velike simetrije problema, određivanje jednačina izlaza za zahtevani demultiplekser 1/4 može se obaviti bez formiranja Karnoovih mapa. Jednačine izlaza u funkciji ulaznih signala su:

$$Q0 = I \cdot \overline{SEL1} \cdot \overline{SEL0}; \quad Q1 = I \cdot \overline{SEL1} \cdot SEL0;$$

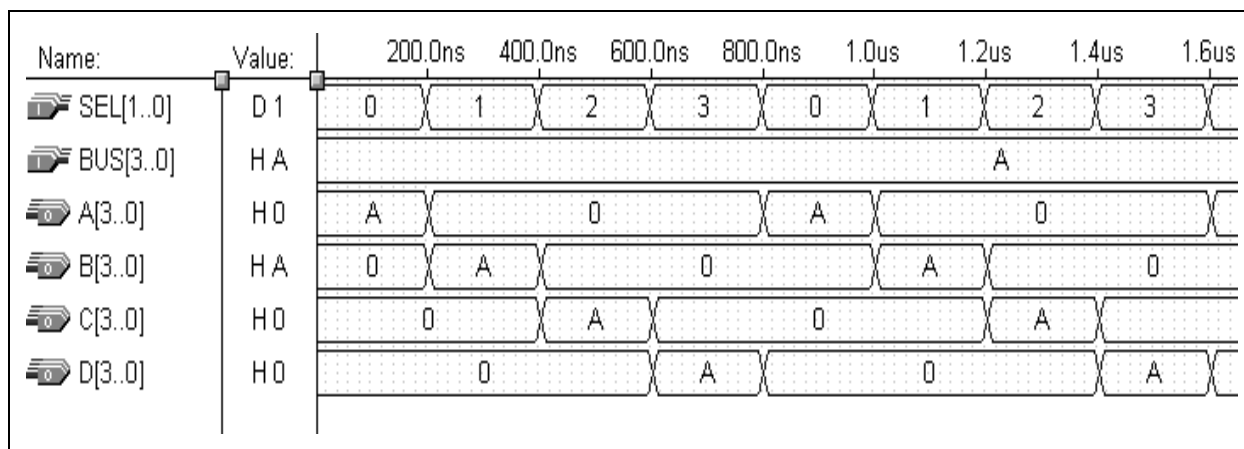
$$Q2 = I \cdot SEL1 \cdot \overline{SEL0} \text{ i } Q3 = I \cdot SEL1 \cdot SEL0.$$

**Slika 4.1:** Šema demultipleksera 1/4

Na bazi jednačina izlaza u funkciji ulaznih signala, u grafičkom editoru razvojnog alata formira se šema demultipleksera 1/4. Na slici br. 4.1 data je šema demultipleksera 1/4 a na slici br. 4.2 simbol opisanog demultipleksera 1/4.

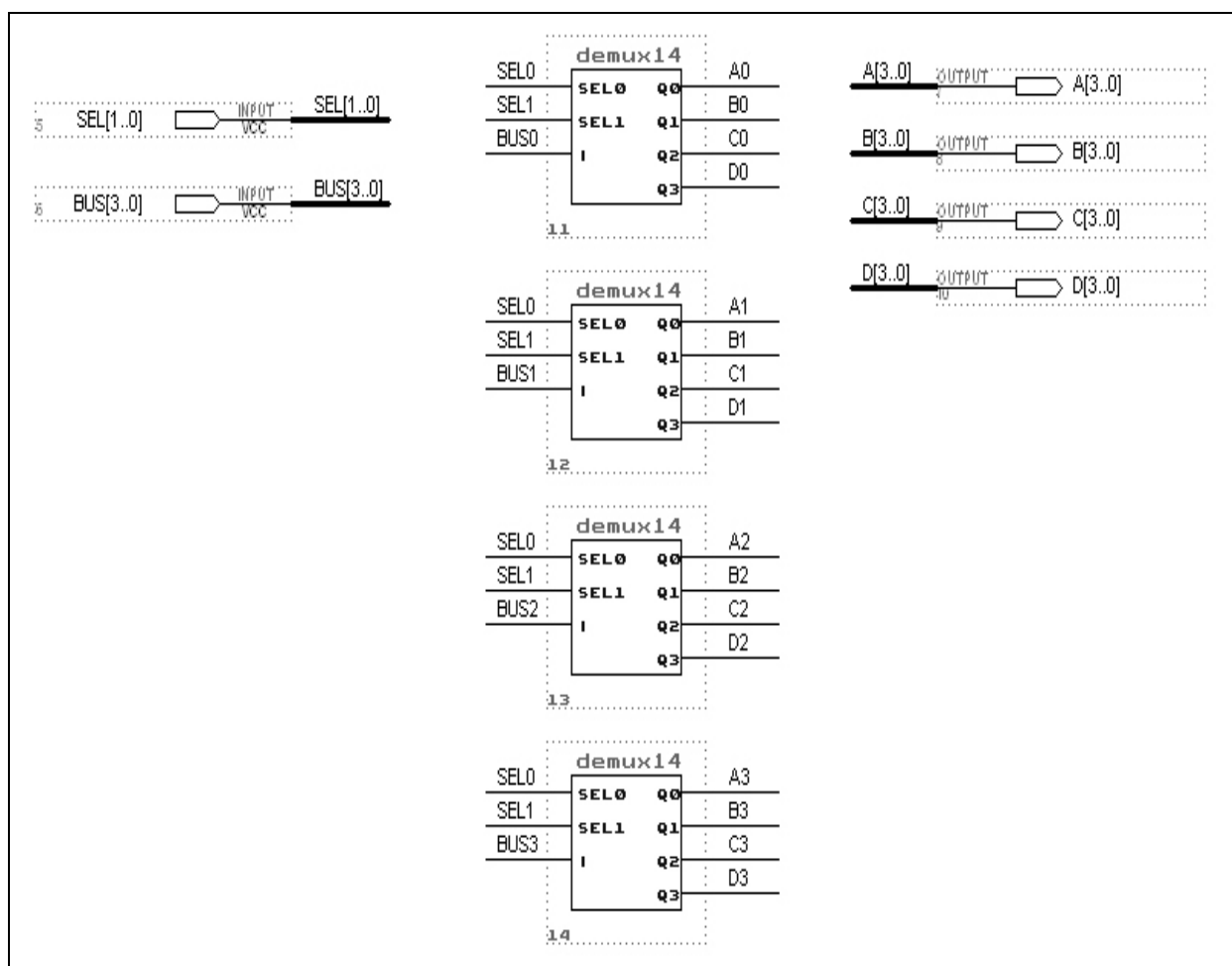
**Slika 4.2:** Simbol demultipleksera 1/4

Realizacija demultipleksera četvorobitnog podatka na ulazu na bazi opisanog demultipleksera 1/4 (*demux14*) ostvaruje se na način koji je prikazan na slici br. 4.4.



Slika 4.3: Vremenski dijagram simulacije za demultiplekser četvorobitnog podatka

Simulacioni dijagram kojim se proverava funkcija sistema demultipleksera četvorobitnog podatka prikazan je na slici br. 4.3.



Slika 4.4: Šema demultipleksera četvorobitnog podatka

ZADATAK 5 – Dekoder 3/8 i 4/16

Realizovati standardni dekodera 3/8 koji sadrži sledeće signale:

- $X[2..0]$ – tri signala kodirane vrednosti na ulazu;
- $Y[7..0]$ – osam signala dekodirane vrednosti na izlazu i
- E – kontrolni signal, tj. signal dozvole rada dekodera.

Kontrolni signal E omogućava rad dekodera tj. za $E='1'$ kolo obavlja funkciju dekodera dok je za $E='0'$ na svim izlaznim priključcima prisutna logička nula. Na primer, za kombinaciju signala na ulazu $X[2..0]='010'$ i $E='1'$ aktivira se izlazna linija signala $Y2$ ($Y2='1'$) dok su ostale linije izlaznih signala na niskom logičkom nivou.

Popuniti tabelu istinitosti koja opisuje rad standardnog dekodera 3/8. Odrediti jednačine izlaznih signala u funkciji signala na ulazu.

Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati kolo dekodera 3/8 u grafičkom editoru.

Izvršiti simulaciju rada sistema valjanom izborom vremenskog inkrementa, vremena trajanja simulacije i test signala ulaznih priključaka dekodera.

Na bazi realizovanog dekodera 3/8 formirati dekodera 4/16. Obezbediti hijerarhijsku organizaciju sistema i realizaciju grafičkim putem.

Izvršiti simulaciju rada dekodera 4/16.

REŠENJE

Na osnovu poznavanja osobine rada standardnog dekodera 3/8 popunjava se tabela istinitosti kombinacione mreže. Tabela istinitosti dekodera 3/8 data je u tabeli br. 5.1.

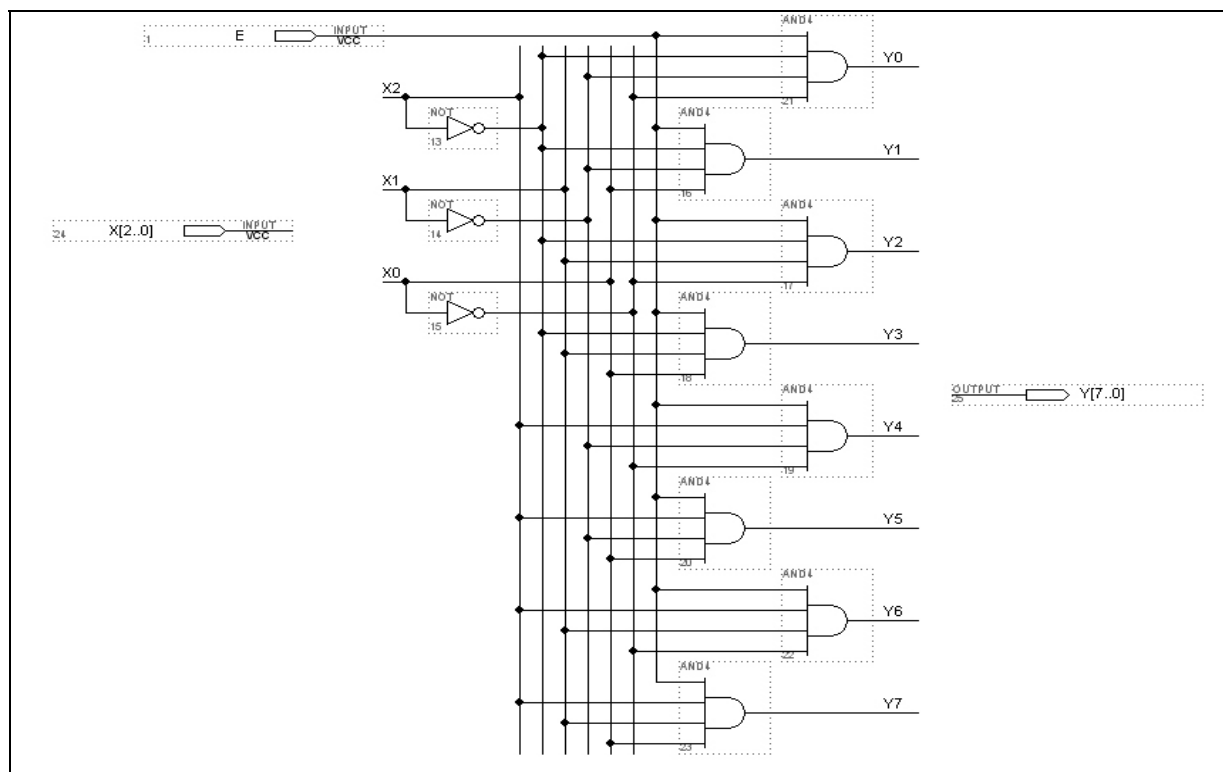
Tabela 5.1: Tabela istinitosti dekodera 3/8

<i>Ulazni signali</i>				<i>Izlazni signali</i>							
E	$X2$	$X1$	$X0$	$Y7$	$Y6$	$Y5$	$Y4$	$Y3$	$Y2$	$Y1$	$Y0$
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Kako je reč o simetričnom sistemu gde je svaki signal izlaza aktivan (ima vrednost 1) samo pri jednoj kombinaciji signala na ulazu, jednačine izlaza mogu se jednostavno dobiti bez primene Karnoovih mapa. U nastavku su date jednačine signala na izlazu u funkciji ulaznih signala.

$$\begin{aligned}
 Y0 &= E \cdot \overline{X2} \cdot \overline{X1} \cdot \overline{X0} & Y4 &= E \cdot X2 \cdot \overline{X1} \cdot \overline{X0} \\
 Y1 &= E \cdot \overline{X2} \cdot \overline{X1} \cdot X0 & Y5 &= E \cdot X2 \cdot \overline{X1} \cdot X0 \\
 Y2 &= E \cdot \overline{X2} \cdot X1 \cdot \overline{X0} & Y6 &= E \cdot X2 \cdot X1 \cdot \overline{X0} \\
 Y3 &= E \cdot \overline{X2} \cdot X1 \cdot X0 & Y7 &= E \cdot X2 \cdot X1 \cdot X0
 \end{aligned}$$

Na bazi tabele istinitosti i dobijenih jednačina izlaznih u funkciji ulaznih signala formira se kolo standardnog dekodera 3/8. Šema dekodera 3/8 data je na slici br. 5.1.

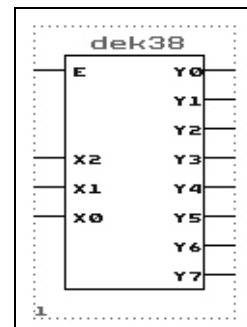


Slika 5.1: Realizacija dekodera 3/8 preko logičkih kola

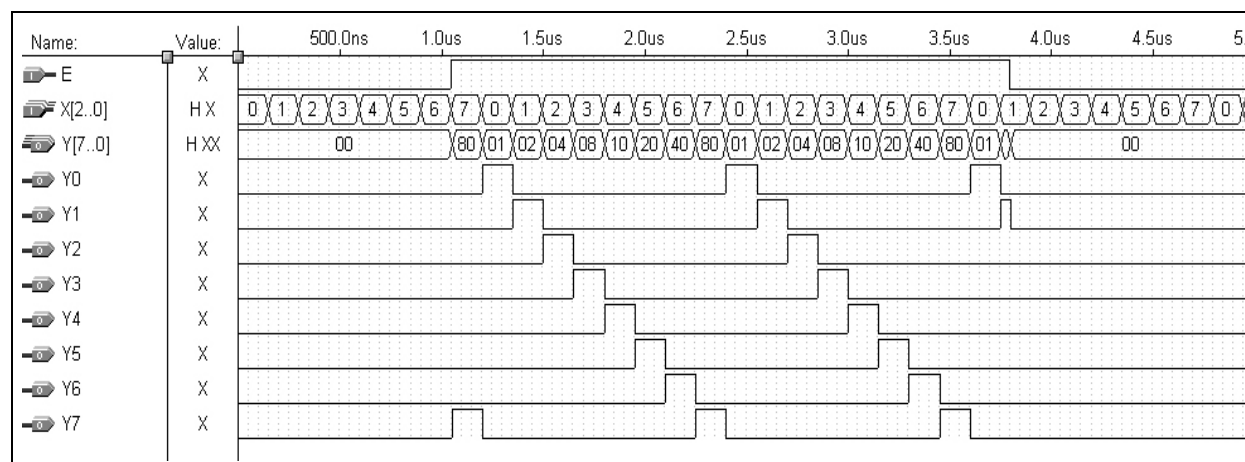
Realizacija dekodera 3/8 u grafičkom editoru obavlja se primenom logičkih kola *NOT* i *AND4* kao i ulaznih i izlaznih priključaka *INPUT* i *OUTPUT*.

Nakon uspešnog prevođenja dizajna standardnog dekodera 3/8 (*dek38*) dobija se simbol kola (slika br. 5.2) koji se može koristiti u drugim realizacijama sistema tj. u višim hijerarhijskim nivoima.

Simulacija rada standardnog dekodera 3/8 data je na slici br. 5.3. Simulacijom je pokriven rad sistema u uslovima $E=0$ i $E=1$ pri svim kombinacijama signala kodirane vrednosti $X[2..0]$. Treba uočiti da je u ovom i u slučajevima sličnim ovom pogodnije prikazati grupni signal $Y[7..0]$ kao niz pojedinačnih signala $Y0, Y1, \dots, Y7$. Proveru regularnosti rada opisanog sistema lakše je obaviti na ovaj način nego analizom osmobarne vrednosti $Y[7..0]$.

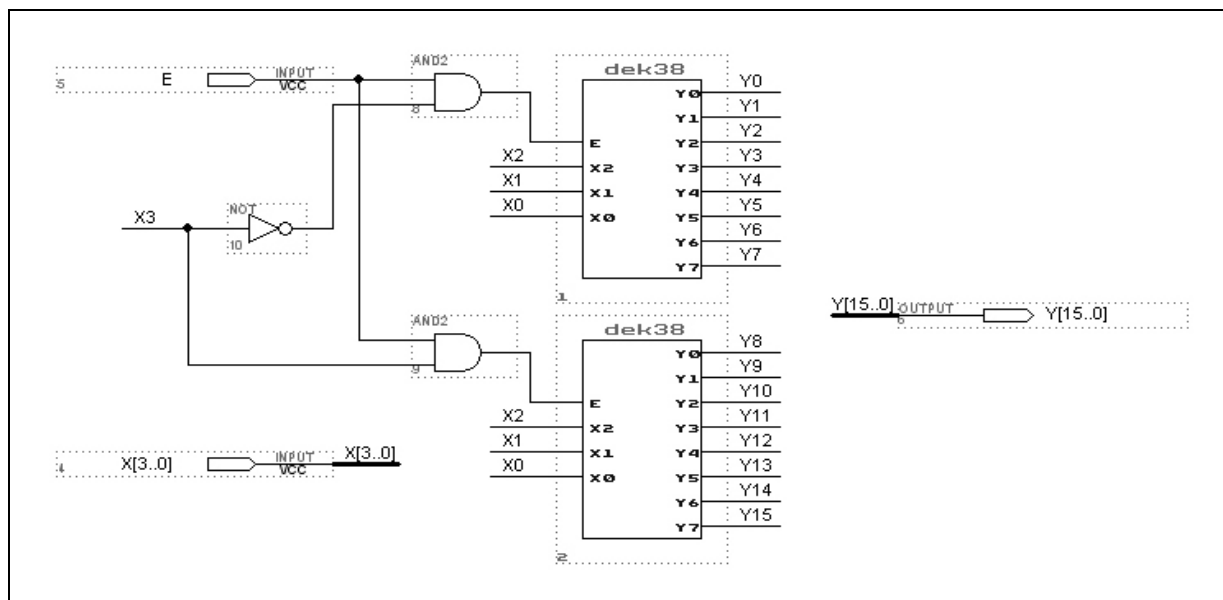


Slika 5.2: Simbol dekodera 3/8



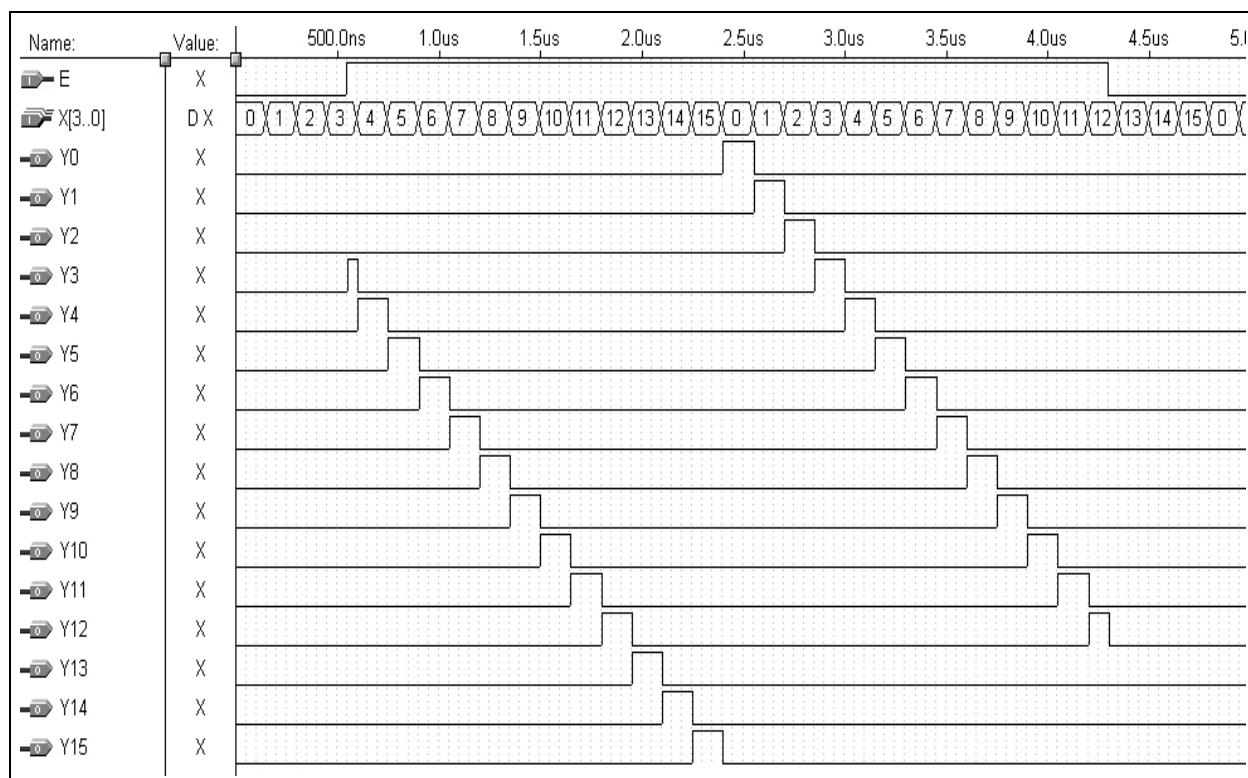
Slika 5.3: Vremenski dijagram simulacije dekodera 3/8

Implementacija dekodera 4/16 primenom realizovanog dekodera 3/8 ostvaruje se preko kola prikazanog na slici br. 5.4. Bit najveće težine (X_3) četvorobitne kodirane vrednosti $X[3..0]$ u kombinaciji sa signalom dozvole rada dekodera formira signale dozvole rada za dva dekodera *dek38*. Na ovaj način obezbeđuje se da za vrednosti na ulazu $X[3..0]$ od 0_{10} do 7_{10} radi jedan, dok za vrednosti od 8_{10} do 15_{10} radi drugi dekodera *dek38*.



Slika 5.4: Realizacija dekodera 4/16 primenom standardnog dekodera 3/8 (*dek38*)

Vremenski dijagram simulacije rada dekodera 4/16 prikazan je na slici br. 5.5. Kratki impulsi signala Y_3 i Y_{12} za $t=600\text{nsec}$ i $t=4,25\mu\text{sec}$ rezultat su deaktiviranja signala dozvole rada dekodera E .



Slika 5.5: Vremenski dijagram simulacije rada dekodera 4/16

ZADATAK 6 – Komparator trobitnih celih pozitivnih brojeva

Realizovati komparator trobitnih celih pozitivnih brojeva. Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati komparator u grafičkom editoru. Obezbediti hijerarhijsku organizaciju sistema.

Primenom alata za simulaciju izvršiti verifikaciju rada komparatora pravilnim izborom vremenskih oblika signala, vremena trajanja simulacije i vremenskog inkrementa.

REŠENJE

Za realizaciju komparatora višebitnih brojeva pogodno je koristiti hijerarhijsku organizaciju u procesu projektovanja sistema. Ovakav pristup omogućava znatno efikasniju realizaciju u pogledu sinteze i verifikacije sistema. Primera radi, za realizaciju šesnaestobitnog komparatora polazi se od komparatora jednobitnih vrednosti na bazi koga se generiše četvorobitni komparator. Pravilnom organizacijom četiri četvorobitna komparatora dobija se komparator šesnaestobitne vrednosti. U procesu realizacije komparatora trobitnih brojeva, takođe se polazi od jednobitnog komparatora.

Jednobitni komparator je kombinaciona mreža koja na ulazu ima dva jednobitna signala A i B, a na izlazu tri signala LT, EQ i GT koji predstavljaju rezultat upoređivanja vrednosti na ulazu. Vrednosti signala na izlazu su:

- $(LT, EQ, GT) = (1, 0, 0)$ za $A > B$;
- $(LT, EQ, GT) = (0, 1, 0)$ za $A = B$;
- $(LT, EQ, GT) = (0, 0, 1)$ za $A < B$.

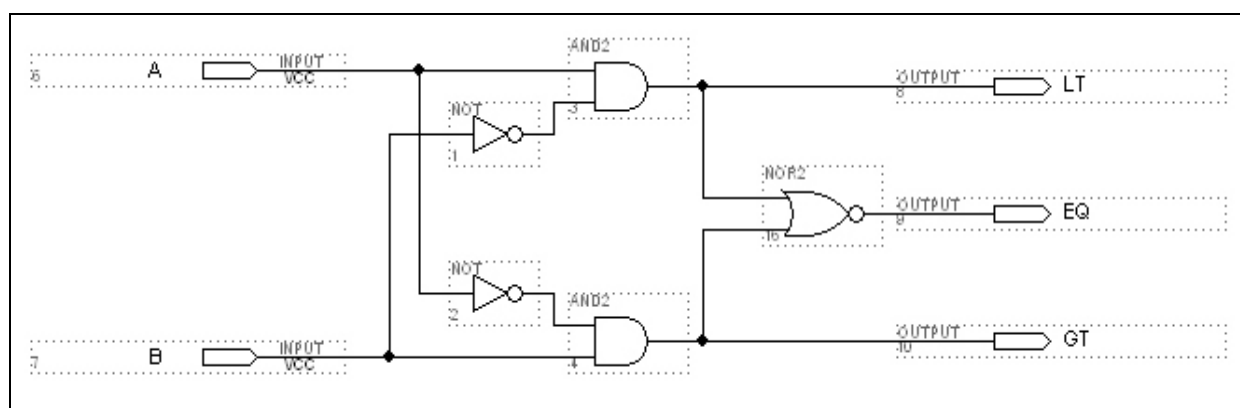
Na bazi prethodne specifikacije rada jednobitnog komparatora izvode se jednačine signala na izlazu u funkciji signala na ulazu:

$$A < B: LT = A \cdot \bar{B},$$

$$A > B: GT = \bar{A} \cdot B \text{ i}$$

$$A = B: EQ = \bar{A} \cdot \bar{B} + A \cdot B = \overline{\bar{A} \cdot B + A \cdot \bar{B}} = \overline{LT + GT}.$$

Realizacija jednobitnog komparatora u grafičkom editoru prikazana je na slici br. 6.1.



Slika 6.1: Šema jednobitnog komparatora

Iz šeme jednobitnog komparatora formira se simbol (*komp_1b*) koji se koristi za realizaciju komparatora višebitnih vrednosti.

Trobitni komparator dobija se formiranjem kombinacione mreže od tri jednobitna komparatora i dodatne logike. Dodatna logika u šemi trobitnog komparatora obezbeđuje formiranje signala komparacije na izlazu u zavisnosti od rezultata komparacije trobitnih brojeva na nivou bita.

Jednačine koje opisuju rad dodatne logike u okviru trobitnog komparatora date su u nastavku:

$$EQ = (A_2 = B_2) \cdot (A_1 = B_1) \cdot (A_0 = B_0)$$

$$EQ = EQ_2 \cdot EQ_1 \cdot EQ_0$$

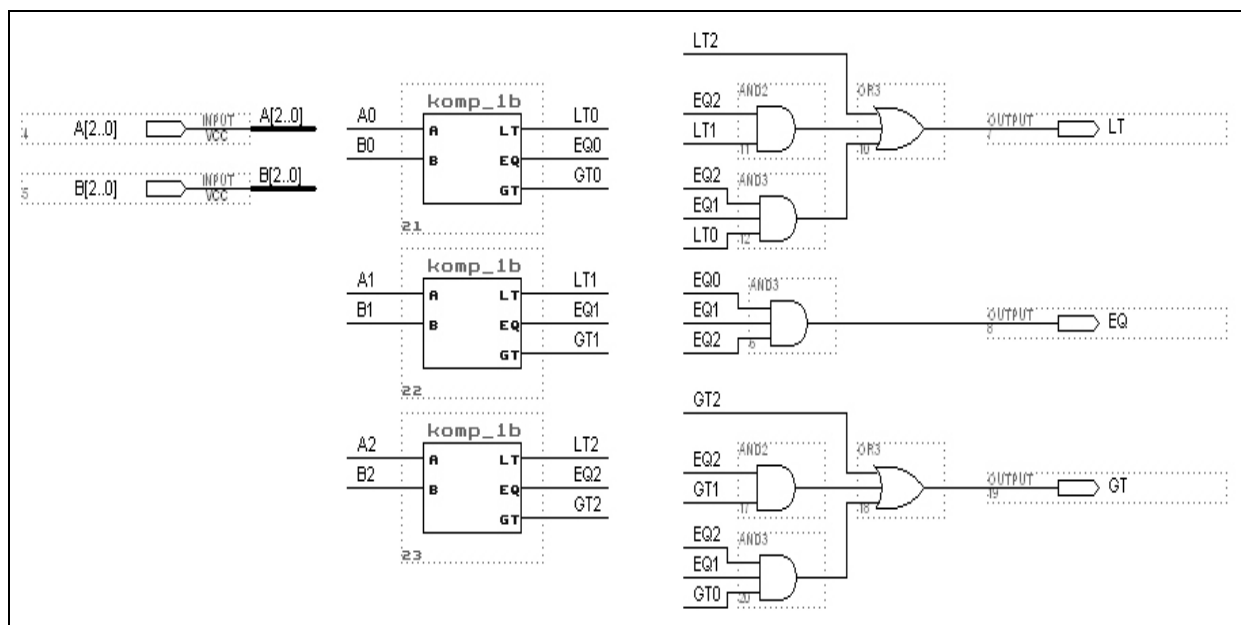
$$LT = (A_2 > B_2) + (A_2 = B_2) \cdot (A_1 > B_1) + (A_2 = B_2) \cdot (A_1 = B_1) \cdot (A_0 > B_0)$$

$$LT = LT_2 + EQ_2 \cdot LT_1 + EQ_2 \cdot EQ_1 \cdot LT_0$$

$$GT = (A_2 < B_2) + (A_2 = B_2) \cdot (A_1 < B_1) + (A_2 = B_2) \cdot (A_1 = B_1) \cdot (A_0 < B_0)$$

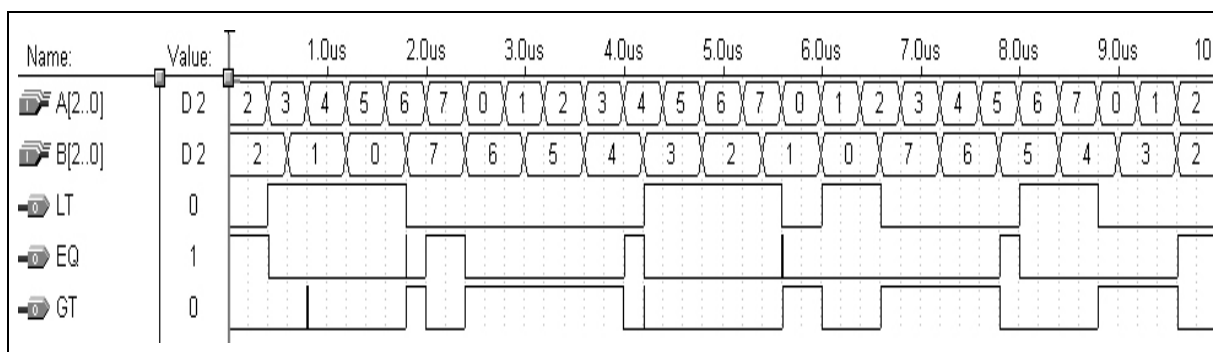
$$GT = GT_2 + EQ_2 \cdot GT_1 + EQ_2 \cdot EQ_1 \cdot GT_0$$

Signali sa indeksom 0, 1 i 2 (A_0 , B_2 , GT_2 , ...) predstavljaju ulazne i izlazne signale komparatora na nivou bita u okviru šeme trobitnog komparatora. Signali $A[2..0]$ i $B[2..0]$ predstavljaju trobitne vrednosti koje se dovode na ulaz komparatora. LT , EQ i GT su signali rezultata upoređivanja vrednosti na izlazu komparatora trobitnih brojeva.



Slika 6.2: Šema trobitnog komparatora

Na slici br. 6.2 data je šema trobitnog komparatora na bazi tri jednobitna komparatora i dodatne logike. Od simbola trobitnog komapratora i na bazi prethodne analize vrlo lako se može formirati komparator devetobitnih vrednosti ili bilo koji drugi komparator.



Slika 6.3: Simulacioni dijagram trobitnog komparatora

Na slici br. 6.3 dat je jedan od vremenskih dijagrama simulacije trobitnog komparatora. Na dijagramu se lako verifikuje ispravan rad sistema. U pojedinim trenucima vremena uočavaju se kratki glicevi signala *EQ* i *GT*. Ove lažne vrednosti kratkog trajanja potiču od kaskadne organizacije sistema i velike razlike u vremenima formiranja signala.

ZADATAK 7 – Desetobitni registar sa funkcijom rotiranja vrednosti

Realizovati desetobitni registar sa funkcijom paralelnog upisa i rotiranja vrednosti registra ulevo. Registar sadrži kontrolne signale dozvole upisa vrednosti u registar (*EN*) i dozvole rotiranja (*ROL*). Funkcija paralelnog upisa ima veći prioritet nad dozvolom rotiranja. Ulazni signali registra su:

- *CLK* – sinhronizacioni signal;
- *LD* – kontrolni signal dozvole upisa vrednosti u registar brojača;
- *ROL* – kontrolni signal dozvole rotiranja vrednosti ulevo i
- *DIN[9..0]* – desetobitna vrednost podatka na ulazu.

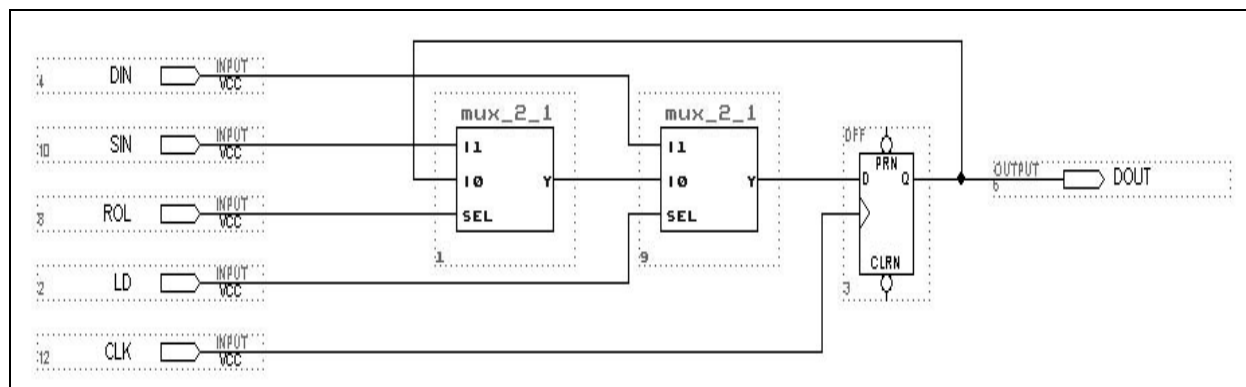
Na izlazu iz registra nalazi se desetobitna vrednost trenutnog stanja registra *DOUT[9..0]*. Oba kontrolna signala su u pozitivnoj logici a sadržaj registra ostaje nepromenjen ukoliko je *EN=ROL='0'*.

Na bazi opšteg modela registra sa funkcijom paralelnog upisa realizovati desetobitni sinhroni rotirajući registar ulevo. Realizovati registar hijerarhijskom organizacijom sistema. Osnovni razred registra realizovati koristeći *D-FF* i multiplekser 2/1. Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati registar u grafičkom editoru.

Primenom alata za simulaciju izvršiti simulaciju rada sistema pravilnim izborom vremenskih oblika signala, vremena trajanja simulacije i vremenskog inkrementa. Formirati simbol opisanog sistema.

REŠENJE

Opšti model registra omogućava realizaciju višebitnog registra sa mogućnošću dodavanja proizvoljnog broja funkcija. Centralno mesto u sistemu čine memorijski elementi (npr. *D-FF*). Realizacija većeg broja funkcija postiže se preko sistema multipleksera. Princip multipleksiranja obezbeđuje jednostavnu implementaciju prioriteta funkcija. Nezavisno od sistema multipleksera postoji logika koja implementira svaku od dodatnih funkcija registra. Odgovarajućim kontrolnim linijama preko sistema multipleksera, vrši se izbor signala tj. rezultata određene funkcije sistema za upis u registar. Pri aktivnoj ivici sinhronizacionog takta selektovana vrednost se upisuje u memorijski blok registra.

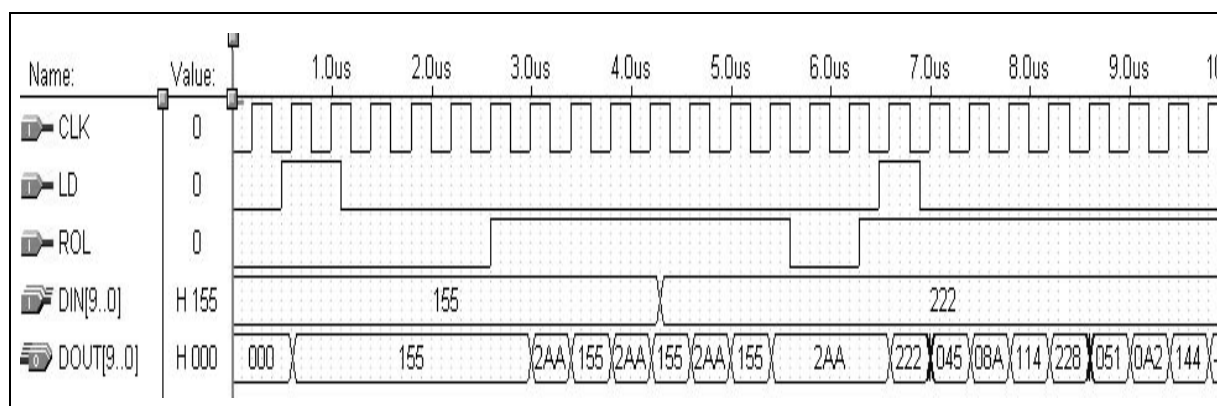


Slika 7.1: Jedna ćelija registra sa funkcijom paralelnog upisa i pomeranja

Na slici br. 7.3 data je šema desetobitnog registra sa funkcijom paralelnog upisa i rotiranja u levo.

Jedan od vremenskih dijagrama simulacije rada registra dat je na slici br. 7.4.

Zbog jednostavnosti funkcije rotiranja, dodatna logika koja implementira ovu funkciju predstavlja mrežu za povezivanje signala na višem hijerarhijskom nivou (slika br. 7.3). Izlaz iz nižeg razreda registra *DOUT* povezuje se sa *SIN* ulazom višeg razreda registra. Izlaz iz desetog razreda registra *DOUT9* vezuje se za *SIN* ulaz razreda najmanje težine.



Slika 7.4: Simulacioni dijagram osmобitnog pomeračkog registra

NAPOMENA

Za regularan rad sistema koji je prikazan na slici br. 7.1 ili na slici br. 7.3 potrebno je obezbediti da signal na ulazu D-FF bude stabilan u vreme usponske ivice *CLK* signala, tj. pri aktivnoj promeni *CLK* signala. Signal na ulazu D-FF dobija se iz signala *DIN*, *SIN*, *ROL* i *LD* što povlači da ovi signali treba da imaju stabilan nivo u vreme usponske ivice *CLK* signala. Za detaljnu analizu i ostvarivanje stabilnosti rada sistema treba uzeti u obzir propagaciju signala kroz multipleksere sistema. Ovaj problem se može rešiti obezbeđivanjem da signali *DIN*, *SIN*, *ROL* i *LD* imaju sinhronu promenu sa opadajućom ivicom *CLK* signala. Ovakvom organizacijom, signali *DIN*, *SIN*, *ROL* i *LD* pa samim tim i signal na ulazu D-FF ostaju stabilni pri usponskoj ivici *CLK* signala što dovodi do sigurne promene stanja D-FF.

ZADATAK 8 – Osmобitni pomerački registar

Realizovati osmобitni registar sa funkcijom paralelnog upisa i pomeranja vrednosti registra ulevo i udesno. Registar sadrži signal dozvole upisa vrednosti u registar (*EN*) i kontrolne signale koji se odnose na pomeranje vrednosti registra ulevo ili udesno (*SHIFT*, *LEFT_RIGHT* i *SIN*). Funkcija paralelnog upisa ima veći prioritet nad funkcijom pomeranja vrednosti. Ulazni signali registra su:

- *CLK* – sinhronizacioni signal;
- *LD* – kontrolni signal dozvole upisa vrednosti u registar brojača;
- *SHIFT* – kontrolni signal dozvole pomeranja vrednosti;
- *LEFT_RIGHT* – kontrolni signal koji određuje smer pomeranja vrednosti registra. Za *LEFT_RIGHT*= '0' sadržaj registra pomera se udesno dok se za *LEFT_RIGHT*= '1' pomera ulevo;
- *DIN*[7..0] – osmобitna vrednost podatka na ulazu i
- *SIN* – jednobitna vrednost koja se serijski priključuje sadržaju registra pri funkciji pomeranja.

Na izlazu iz registra prisutni su:

- $DOUT[7..0]$ – osmobitna vrednost trenutnog stanja registra i
- $SOUT$ – jednobitna vrednost podatka koji se potiskuje iz registra pri pomeranju.

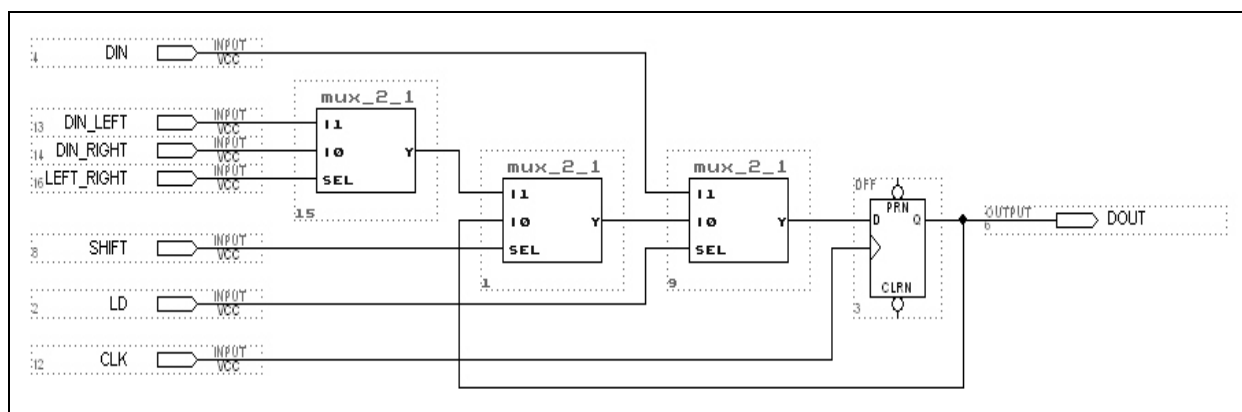
Svi kontrolni signali aktivni su u pozitivnoj logici a sadržaj registra ostaje nepromenjen ukoliko je $EN=SHIFT='0'$.

Na bazi opšteg modela registra sa funkcijom paralelnog upisa realizovati osmobitni sinhroni pomerački registar. Realizovati registar hijerarhijskom organizacijom sistema. Osnovnu ćeliju registra realizovati koristeći D -FF i multiplekser 2/1. Primenom softverskog paketa za rad sa proizvodima firme ALTERA realizovati registar u grafičkom editoru.

Primenom alata za simulaciju izvršiti simulaciju rada sistema pravilnim izborom vremenskih oblika signala, vremena trajanja simulacije i vremenskog inkrementa. Formirati simbol opisanog sistema.

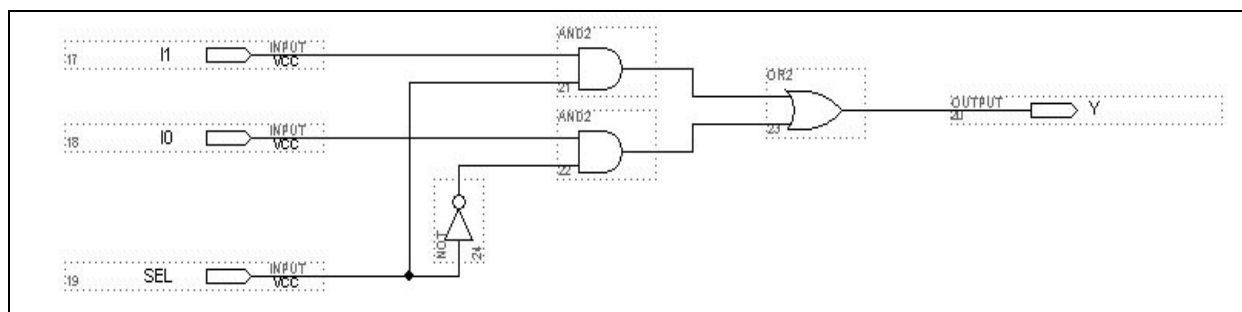
REŠENJE

Realizacija sistema osmobitnog pomeračkog registra sprovodi se na sličan način kao i u slučaju registra sa funkcijom rotiranja vrednosti (prethodni zadatak). Realizacija registra počinje implementacijom jedne ćelije registra na bazi specifikacije zahteva u pogledu funkcija i prioriteta izvršavanja više aktiviranih zahteva. Na slici br. 8.1 data je šema jedne ćelije pomeračkog registra koji ima mogućnost pomeranja vrednosti ulevo ili udesno.



Slika 8.1: Šema jedne ćelije pomeračkog registra (*shift_cell*)

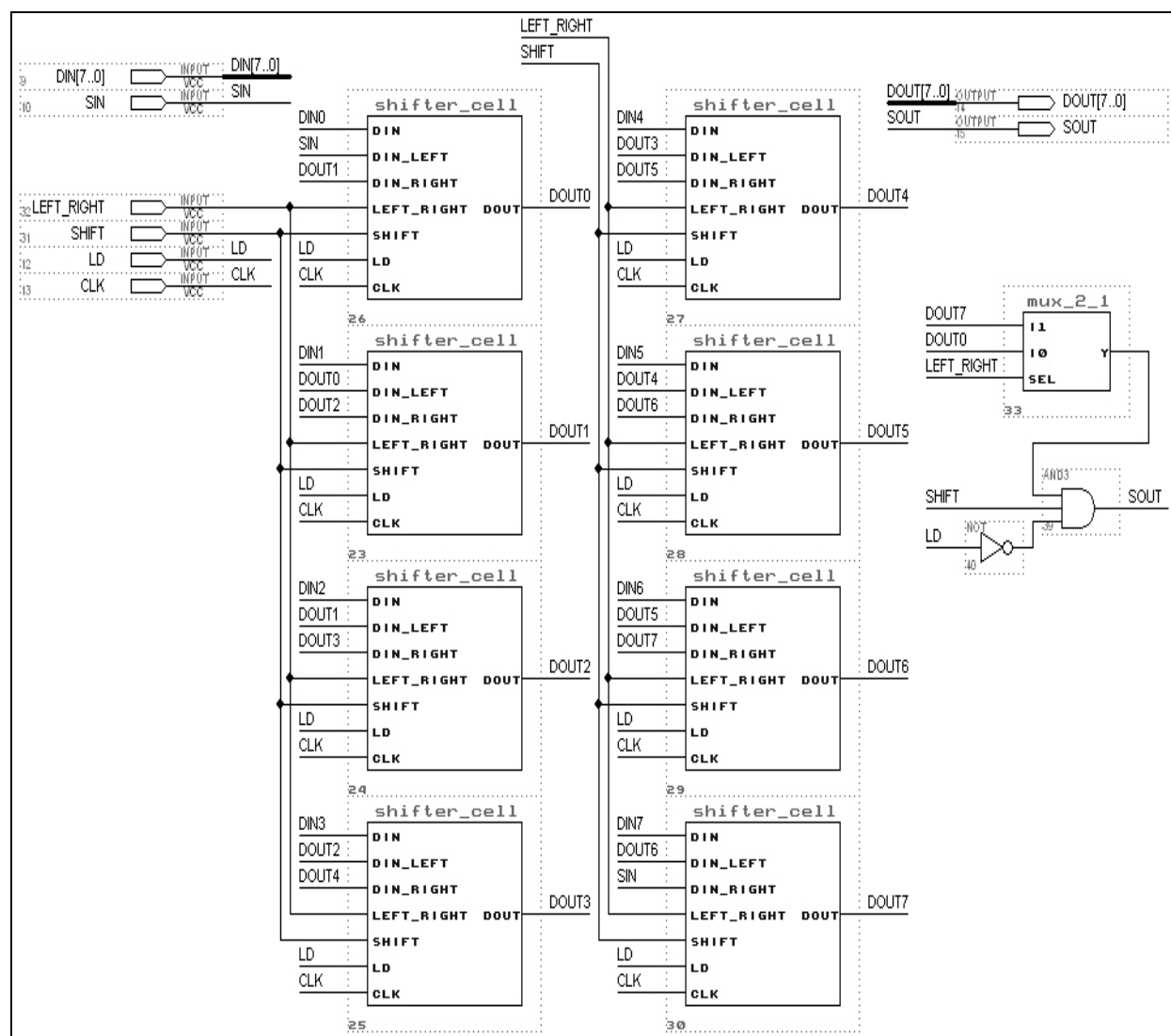
U okviru šeme jedne ćelije pomeračkog registra uočavaju se tri multipleksa 2/1 (*mux_2_1*). Šema multipleksa 2/1 data je na slici br. 8.2. Uloga multipleksa je da obezbede selekciju funkcije rada registra kao i prioritet izvršavanja više aktiviranih funkcija u isto vreme. Prvi multiplekser sa desne strane obezbeđuje najveći prioritet funkcije paralelnog upisa. Srednji multiplekser, pri $EN='1'$, realizuje funkciju pomeranja vrednosti za $SHIFT='0'$ i zadržavanja vrednosti registra za $SHIFT='1'$.



Slika 8.2: Šema multipleksa 2/1

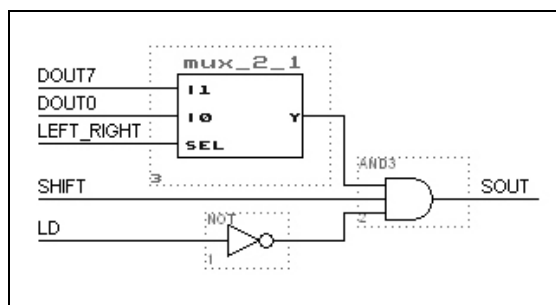
Multiplexer sa leve strane na slici br. 8.1 implementira funkciju pomeranja na levo ili desno u zavisnosti od kontrolnog signala *LEFT_RIGHT*. Kada je ispunjen uslov pomeranja *EN='0'* i *SHIFT='1'*, vrši se upis vrednosti *DIN_LEFT* odnosno *DIN_RIGHT* u registar u zavisnosti od kontrolnog signala *LEFT_RIGHT*.

Šema osmobitnog pomeračkog registra sa funkcijom paralelnog upisa na bazi ćelije pomeračkog registra (*shift_cell*) data je na slici br. 8.3.



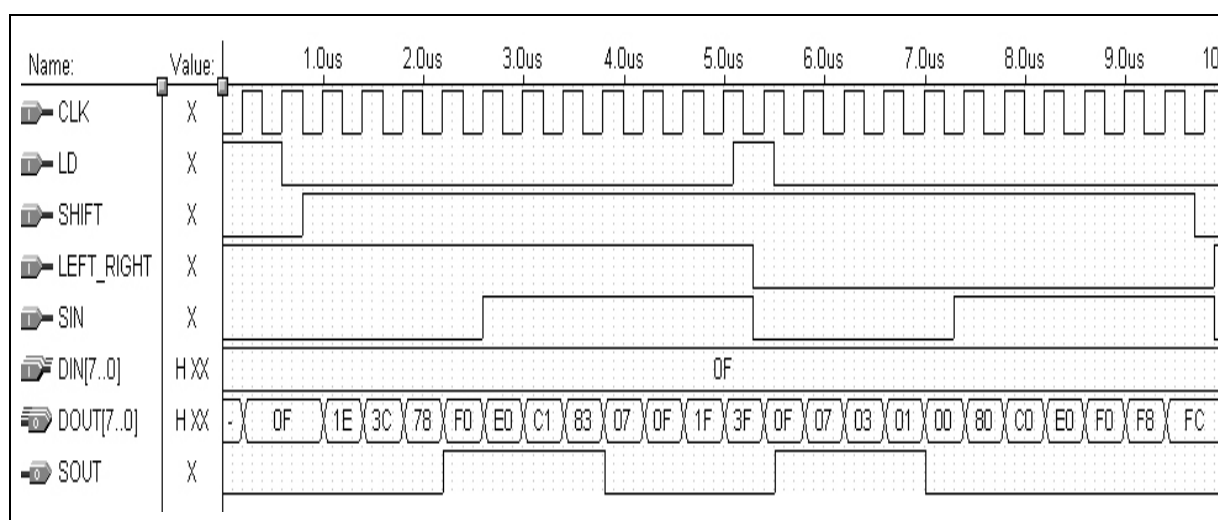
Slika 8.3: Šema osmobitnog pomeračkog registra sa funkcijom paralelnog upisa

Osmobitni pomerački registar sa funkcijom paralelnog upisa dobija se pravilnim povezivanjem osam ćelija pomeračkog registra (*shift_cell*). Na mesto *DIN_LEFT* svake ćelije registra dovodi se vrednost izlaza nižeg razreda registra. Linija signala *DIN_RIGHT* svake ćelije registra povezuje se za izlazom višeg razreda registra. Krajnje ćelije registra prihvataju signal *SIN* kao jedan od ulaznih signala pri pomeranju vrednosti registra. Pored osam ćelija pomeračkog registra u sistemu osmobitnog registra prisutna je i dodatna logika koja vrši formiranje signala *SOUT*. Logika koja generiše izlazni signal *SOUT* data je na slici br. 8.4. Specifičnost ovog dela sistema je da obezbedi vrednost signala koji se potiskuje iz registra pri pomeranju vrednosti.



Slika 8.4: Šema logike za formiranje signala *SOUT*

Na slici br. 8.5 dat je vremenski dijagram simulacije rada pomeračkog registra. Na dijagramu se uočava pravilan rad registra u režimu upisa vrednosti, pomeranja ulevo i udesno.



Slika 8.5: Vremenski dijagram simulacije rada registra

I za rešenje ovog zadatka važi ista napomena koja je izneta na kraju zadatka br. 7, a odnosi se na obezbeđivanje stabilnosti signala na ulazu u toku aktivne ivice *CLK* signala.

ZADATAK 9 – Osmobitni brojač nagore

Realizovati osmobitni sinhroni brojač nagore sa funkcijom paralelnog upisa vrednosti u registar brojača. Realizaciju sistema bazirati na opštem modelu brojača sa kolom za inkrementiranje. Na ulazu brojača prisutni su signali:

- *CLK* – sinhronizacioni signal;
- *LD* – kontrolni signal dozvole upisa vrednosti u registar brojača;
- *EN* – kontrolni signal dozvole brojanja i
- *DIN[7..0]* – osmobitna vrednost podatka na ulazu.

Na izlazu iz sistema prisutna je osmobitna vrednost stanja brojača tj. signali *DOUT[7..0]*. Funkcija paralelnog upisa ima veći prioritet izvršavanja nad funkcijom dozvole brojanja. Paralelni upis podatka u registar brojača postiže se preko kontrolnog signala *LD*. Za vrednost *LD*= '1' aktivira se upis vrednosti *DIN[7..0]* u registar brojača. Pri vrednosti *EN*= '0' postoji mogućnost aktiviranja funkcije brojanja, tj. ukoliko je tada signal dozvole brojanja *EN*= '1' brojač broji nagore odnosno za *EN*= '0' brojač ostaje u zatečenom stanju.

Sistem treba biti hijerarhijski organizovan. Osnovni razred brojača realizovati koristeći *D-FF*, multiplexer 2/1 i kolo za inkrementiranje. Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati brojač u grafičkom editoru.

Primenom alata za simulaciju izvršiti simulaciju rada konvertora pravilnim izborom vremenskih oblika signala, vremena trajanja simulacije i vremenskog inkrementa. Formirati simbol opisanog sistema.

REŠENJE

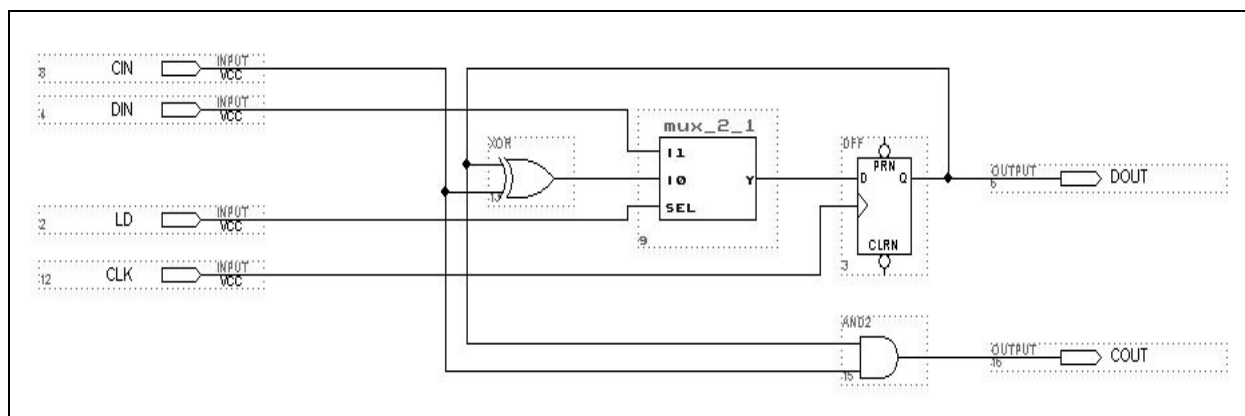
Brojač koji je specificiran u zadatku može se posmatrati kao osmobarbitni registar sa funkcijom paralelnog upisa i brojanja nagore. Brojač se može analizirati kao sistem od osam razreda pri čemu svaki razred sadrži:

- memorijski element;
- kolo za selekciju funkcije rada i
- kolo za obezbeđivanje brojanja.

Centralno mesto kod registra, pa samim tim i kod brojača, čini memorijski element. U najvećem broju slučajeva koriste se *D-FF*. Selekcija funkcije rada registra postiže se preko sistema multipleksera. U konkretnom slučaju potreban je jedan multiplexer 2/1 po razredu brojača. Kolo za inkrementiranje može se predstaviti jednačinama:

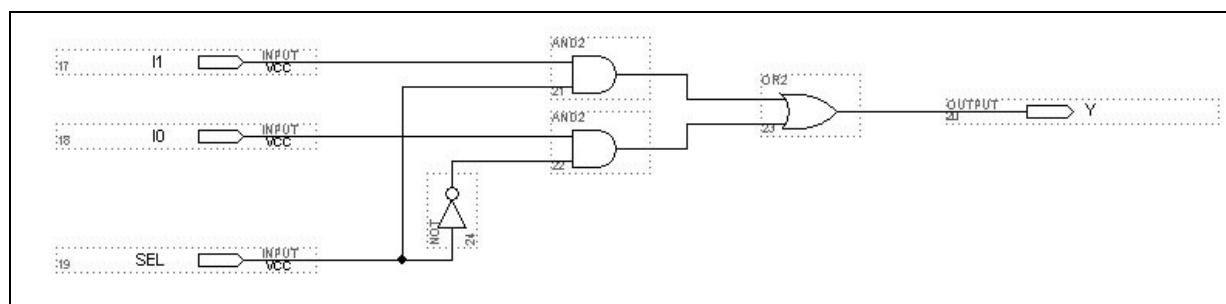
$$D = Q \otimes CIN \text{ i } COUT = Q \cdot CIN,$$

gde su *CIN* i *COUT* redom signal prenosa iz prethodnog razreda i signal prenosa u naredni razred. Signali *D* i *Q* predstavljaju signale na ulazu i izlazu iz *D-FF*.



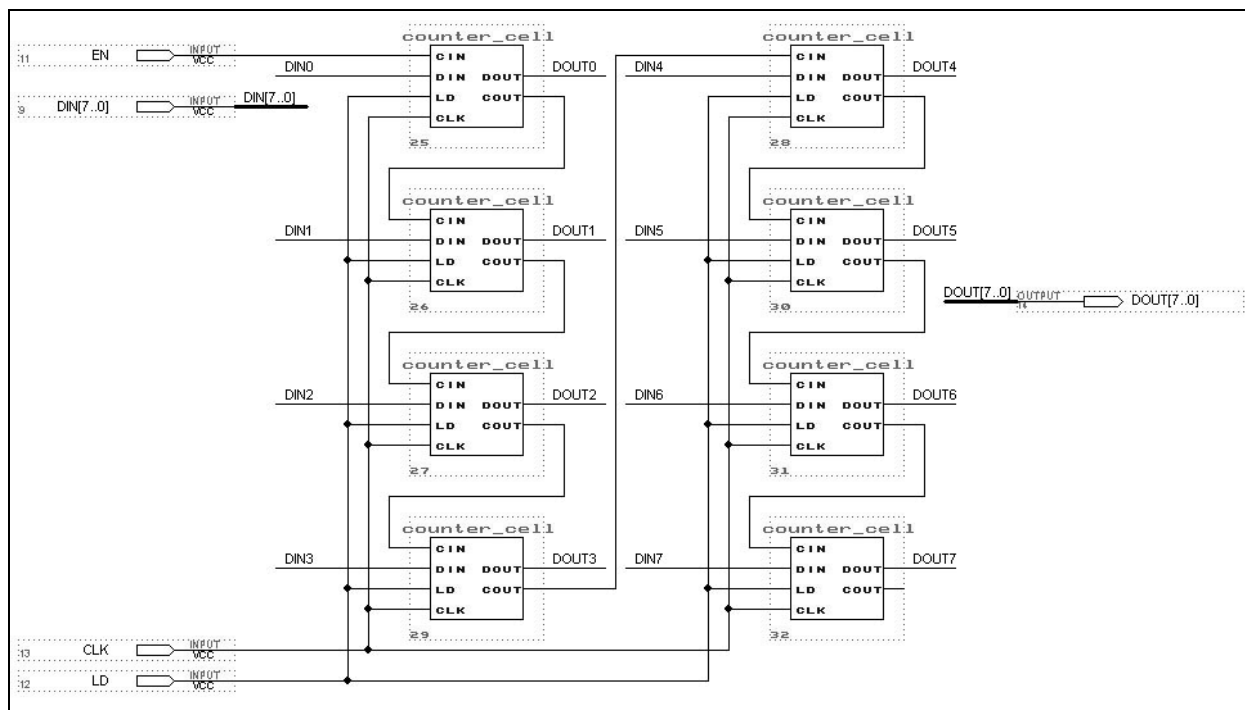
Slika 9.1: Šema jedne ćelije registra sa funkcijom brojača i paralelnog upisa (*counter_cell*)

Na slici br. 9.1 data je šema jedne ćelije registra sa funkcijom paralelnog upisa i brojanja na gore (*counter_cell*). Za realizaciju ćelije brojača koristi se multiplexer 2/1 (*mux_2_1*) čija je šema data na slici br. 9.2.

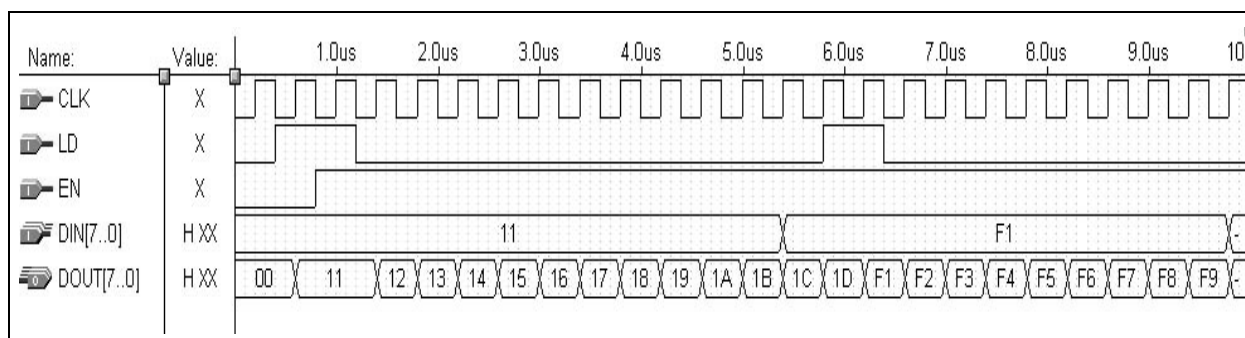


Slika 9.2: Šema multipleksera 2/1 (*mux_2_1*)

Pravilnom organizacijom osnovnih ćelija realizuje se brojač nagore sa funkcijom paralelnog upisa (slika br. 9.3). Signal dozvole brojanja *EN* vezuje se sa linijom signala prenosa iz prethodnog razreda kola za inkrementiranje prve ćelije brojača. Jedan od vremenskih dijagrama simulacije rada sistema brojača dat je na slici br. 9.4.



Slika 9.3: Osmobitni brojački registar sa funkcijom paralelnog upisa

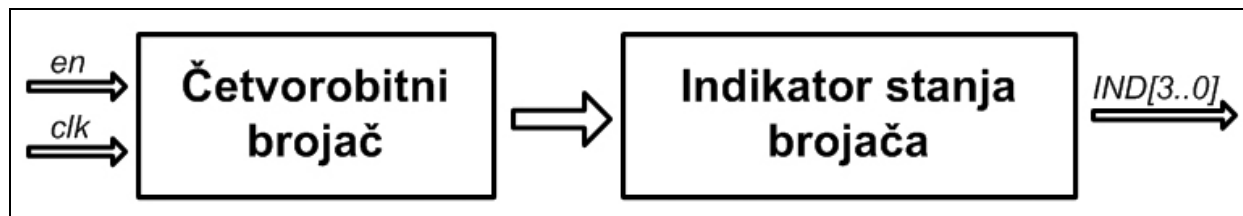


Slika 9.4: Vremenski dijagram simulacije brojača

I za rešenje ovog zadatka važi ista napomena koja je izneta na kraju zadatka br. 7 a odnosi se na obezbeđivanje stabilnosti signala na ulazu u toku aktivne ivice *CLK* signala.

ZADATAK 10 – Četvorobitni brojač i indikator stanja brojača

Realizovati sistem koji sadrži četvorobitni brojač i indikator stanja brojača. Organizacija sistema data je na slici br. 10.1.



Slika 10.1: Organizacija sistema

Brojač u okviru sistema je četvorobitni sa sekvencom brojanja nagore: $0 \rightarrow 1 \rightarrow 2 \rightarrow \dots \rightarrow 15 \rightarrow 0 \rightarrow 1 \rightarrow \dots$. Brojač sadrži kontrolni signal dozvole brojanja *en* koji dozvoljava rad brojača pri niskom logičkom nivou, tj. za $en = '0'$. Sinhronne promene u sistemu dešavaju se pri usponskoj ivici signala *clk*.

Vrednost brojača dovodi se na dekodek indikacije, tj. kombinacionu mrežu koja na bazi ulazne vrednosti vrši postavljanje signala indikacije *IND[3..0]*. U tabeli br. 10.1 dat je opis rada kola za indikaciju stanja brojača u zavisnosti od vrednosti na ulazu.

Tabela 10.1: Tabela opisa rada kola za indikaciju stanja brojača

Izlaz	Vrednost na ulazu dekodera indikacije															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
IND0		L	L	L												
IND1					L	L	L	L	L	L	L					
IND2												L	L	L	L	
IND3																L

Ukoliko je polje u tabeli br. 10.1 popunjeno slovom *L* za određenu kombinaciju ulaza i izlaza *IND_x*, na tom izlazu *IND_x* treba postaviti nizak logički nivo. U suprotnom, treba postaviti visok logički nivo signala na izlazu. Primera radi, za vrednost 11 na ulazu dekodera indikacije signal *IND2* ima nizak ($IND2 = '0'$), dok signali *IND0*, *IND1* i *IND3* imaju visoki logički nivo ($IND0 = IND1 = IND3 = '1'$).

Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA rešiti zadatak opisom četvorobitnog brojača u grafičkom editoru preko kola za inkrementiranje. Indikator stanja brojača realizovati takođe grafičkim opisom.

Primenom alata za simulaciju iz softverskog razvojnog okruženja izvršiti simulaciju rada sistema pravilnim izborom vremenskih oblika signala, vremena trajanja simulacije i vremenskog inkrementa.

REŠENJE

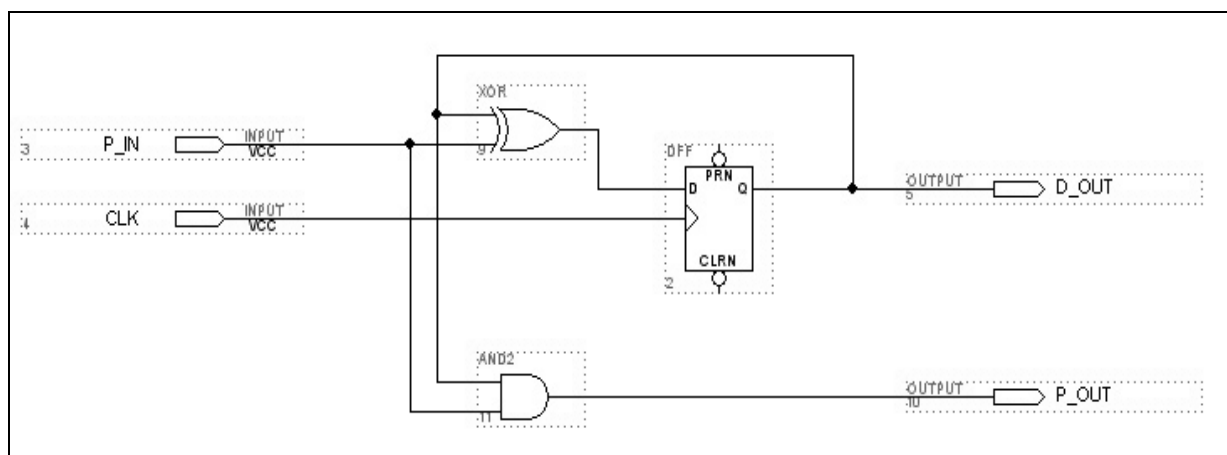
Četvorobitni brojač iz sistema realizuje se na sličan način kao i većina multifunkcionalnih registara iz prethodnih zadataka. Realizacija četvorobitnog brojača obavlja se hijerarhijskom organizacijom koristeći bazičnu ćeliju registra sa funkcijom brojanja na gore. Funkcija brojanja na gore implementira se preko kola za inkrementiranje koje je specificirano jednačinama:

$$D = Q \otimes P_IN \text{ i } P_OUT = Q \cdot P_IN,$$

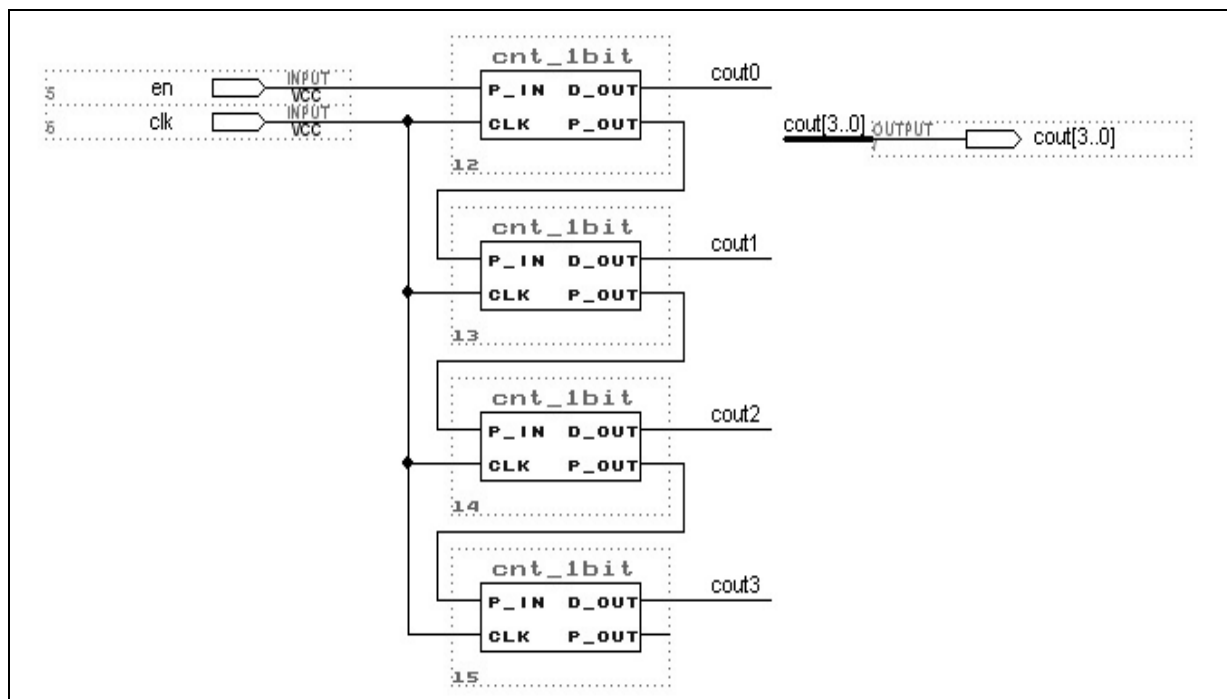
gde su P_IN i P_OUT signali ulaza i izlaza tj. prenosa iz prethodnog u naredni razred brojača. Signali D i Q su signali ulaza i izlaza D -FF elementa ćelije registra.

Šema ćelije brojača prikazana je na slici br. 10.2. Pošto u zadatku nije predviđeno da brojač sadrži neku dodatnu funkciju osim funkcije brojanja, ćelija brojača ne uključuje blok za selekciju funkcije registra, tj. sistem multipleksera. Aktiviranje funkcije brojanja postiže se preko signala iz prethodnog razreda za ćeliju najnižeg razreda brojača.

Pravilnim povezivanjem četiri ćelije brojača sa slike br. 10.2 (*cnt_1bit*) dobija se četvorobitni brojač zasnovan na kolu za inkrementiranje. Šema četvorobitnog brojača data je na slici br. 10.3.

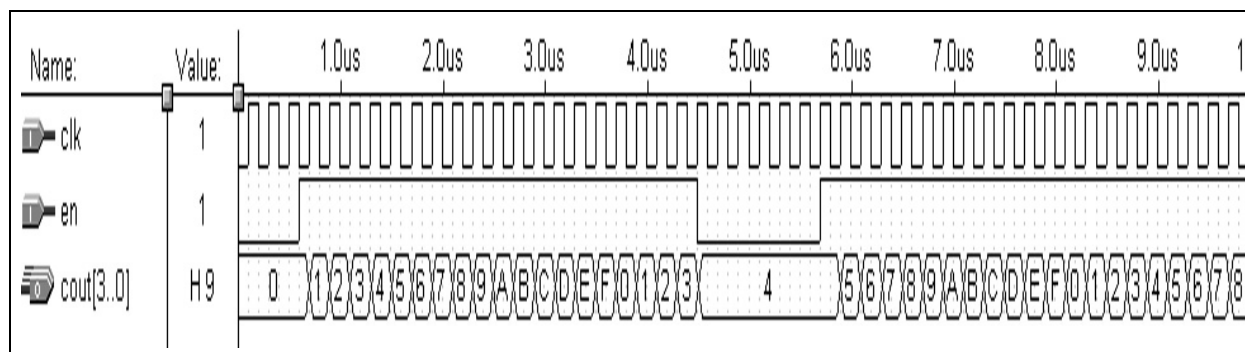


Slika 10.2: Šema ćelije brojača na bazi kola za inkrementiranje (*cnt_1bit*)



Slika 10.3: Četvorobitni brojač na bazi kola za inkrementiranje (*cnt_4bits*)

Na slici br. 10.4 prikazan je vremenski dijagram simulacije četvorobitnog brojača na bazi kola za inkrementiranje.



Slika 10.4: Dijagram simulacije četvorobitnog brojača na bazi kola za inkrementiranje

Dekoder indikacije na ulazu ima četvorobitnu vrednost $CNT[3..0]$ dok su na izlazu prisutna četiri jednobitna signala indikacije $IND0$, $IND1$, $IND2$ i $IND3$. Jednačine signala $IND0$, $IND1$, $IND2$ i $IND3$ u funkciji ulaza CNT određuju se formiranjem Karnoovih mapa za svaki signal posebno. Najpogodnije je formirati Karnoove mape za invertovane signale na izlazu.

		CNT1 CNT0			
CNT3	CNT2	00	01	11	10
00		1	1	1	
01					
11					
10					

Karnoova mapa za signal $\overline{IND0}$

$$\overline{IND0} = \overline{CNT3} \cdot \overline{CNT2} \cdot (CNT0 + CNT1)$$

$$IND0 = \overline{\overline{CNT3} \cdot \overline{CNT2} \cdot (CNT0 + CNT1)}$$

		CNT1 CNT0			
CNT3 CNT2		00	01	11	10
	00				
	01	1	1	1	1
	11				
	10	1	1		1

Karnoova mapa za signal $\overline{IND1}$

$$\overline{IND1} = \overline{CNT3} \cdot \overline{CNT2} + \overline{CNT3} \cdot \overline{CNT2} \cdot (\overline{CNT0} + \overline{CNT1})$$

$$IND1 = \overline{\overline{CNT3} \cdot \overline{CNT2} + \overline{CNT3} \cdot \overline{CNT2} \cdot (\overline{CNT0} + \overline{CNT1})}$$

		CNT1 CNT0			
CNT3 CNT2		00	01	11	10
	00				
	01				
	11	1	1		1
	10			1	

Karnoova mapa za signal $\overline{IND2}$

$$\overline{IND2} = \overline{CNT3} \cdot \overline{CNT2} \cdot (\overline{CNT0} + \overline{CNT1}) +$$

$$CNT3 \cdot \overline{CNT2} \cdot CNT1 \cdot CNT0$$

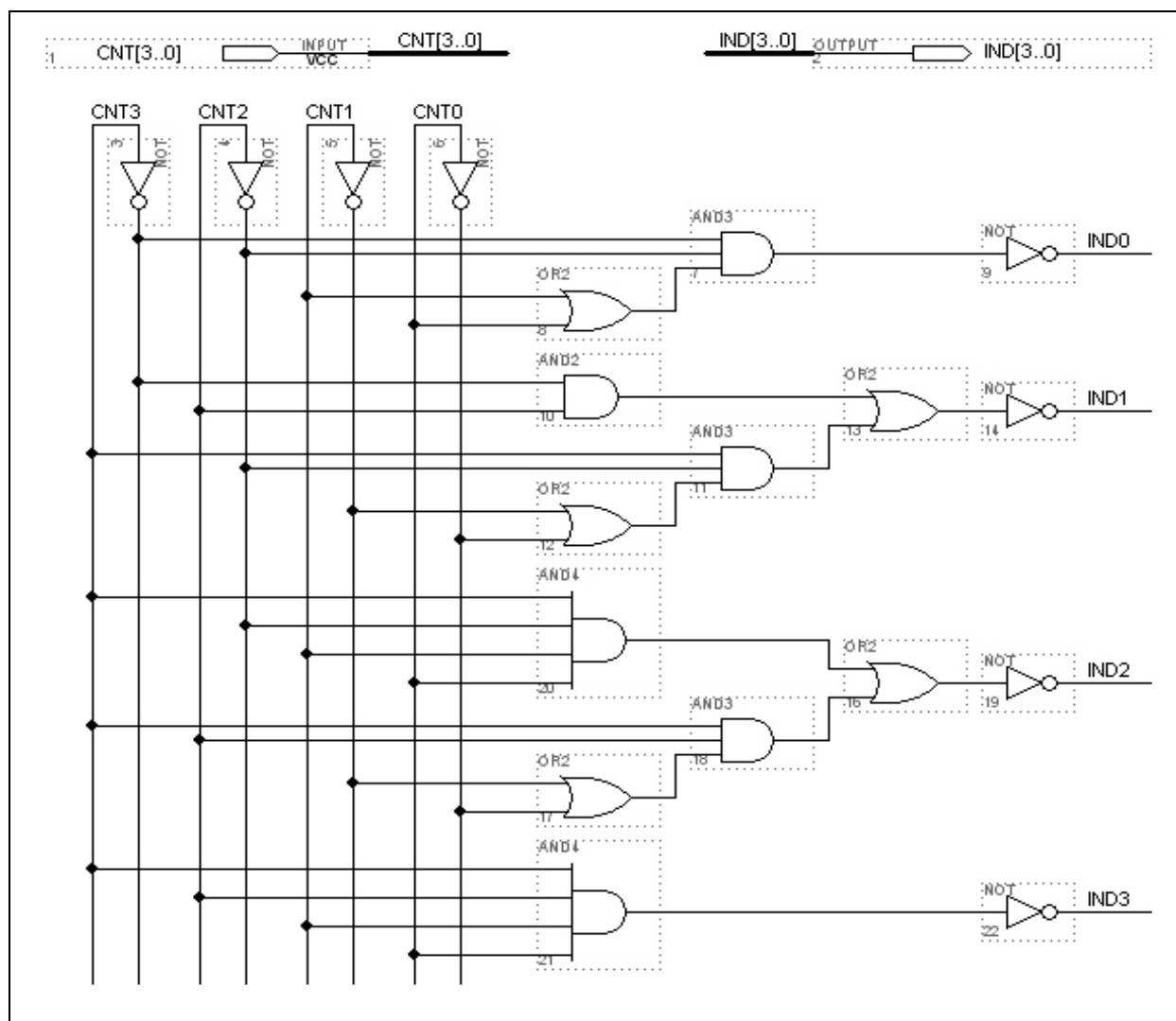
		CNT1 CNT0			
CNT3 CNT2		00	01	11	10
	00				
	01				
	11			1	
	10				

Karnoova mapa za signal $\overline{IND3}$

$$\overline{IND3} = \overline{CNT3} \cdot \overline{CNT2} \cdot \overline{CNT1} \cdot \overline{CNT0}$$

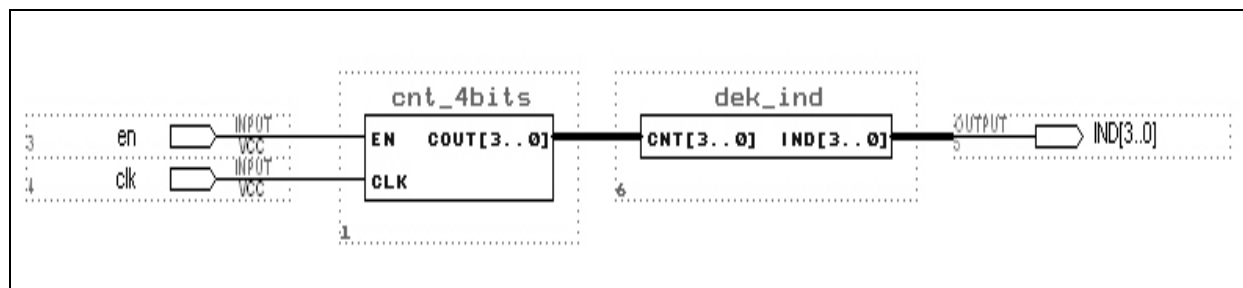
$$IND3 = \overline{\overline{CNT3} \cdot \overline{CNT2} \cdot \overline{CNT1} \cdot \overline{CNT0}}$$

Na osnovu dobijenih jednačina realizuje se kombinaciona mreža kola za indicaciju stanja brojača. Realizacija kola za indicaciju stanja brojača u grafičkom editoru prikazana je na slici br. 10.5.



Slika 10.5: Opis kola za indicaciju stanja brojača (*dek_ind*) u grafičkom editoru

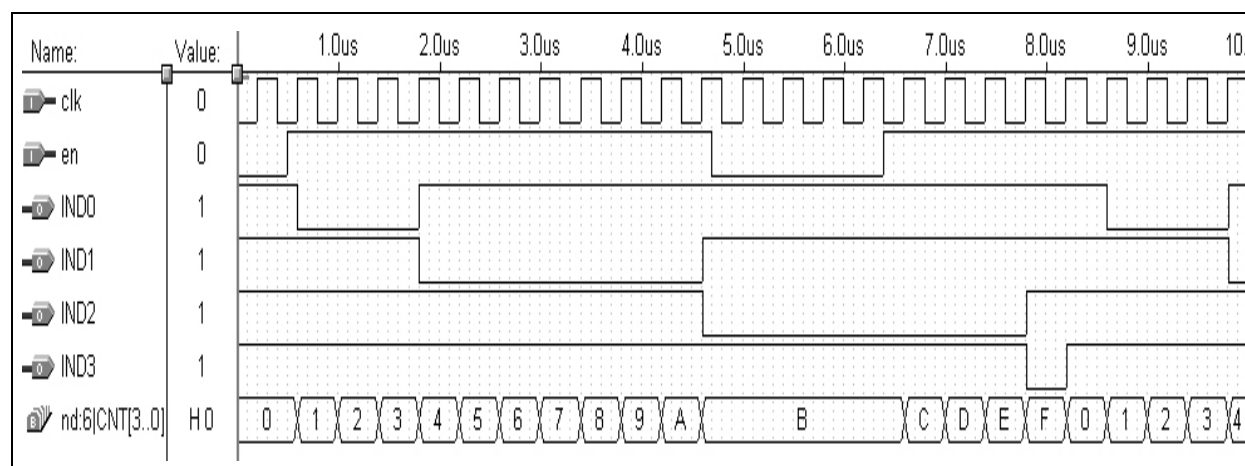
Sistem u celini dobija se povezivanjem četvorobitnog brojača (*cnt_4bits*) i kola za indicaciju stanja brojača (*dek_ind*). Šema sistema u celini prikazana je na slici br. 10.6.



Slika 10.6: Organizacija zahtevanog sistema u celini

Jedan od vremenskih dijagrama simulacije sistema u celini prikazan je na slici br. 10.7.

I za rešenje ovog zadatka važi ista napomena koja je izneta na kraju zadatka br. 7 a odnosi se na obezbeđivanje stabilnosti signala na ulazu u toku aktivne ivice *clk* signala.



Slika 10.7: Vremenski dijagram simulacije sistema u celini

ZADATAK 11 – Pomerački registar na bazi „*barrel shifter*“ kola

Realizovati sistem koji predstavlja četvorobitni prihvatni registar sa *barrel shifter* funkcijom. Pored funkcije *barrel shifter* kola, sistem treba da sadrži signale kontrole dozvole upisa podatka u registar i dozvole pomeranja vrednosti registra na levo u skladu sa funkcijom *barrel* pomerača. Takođe, pomeranje vrednosti registra na levo može da se obavlja od jedne do tri pozicije u skladu sa odgovarajućim signalom kontrole *CNT[1..0]*.

Sistem sadrži sledeće signale na ulazu i izlazu:

- sinhronizacioni signal *CLK* (promena na usponskoj ivici);
- 4-bitni ulazni podatak *DATA_IN[3..0]*;
- 4-bitni izlazni podatak *DATA_OUT[3..0]*;
- jednobitni kontrolni signal *LD* kojim se aktivira funkcija upisa podataka *DATA_IN* u registar sistema (za nivo signala *LD*= '1' vrši se upis);
- jednobitni kontrolni signal *EN* kojim se dozvoljava pomeranje vrednosti registra u skladu sa osobinom *barrel shifter* kola (pri nivou signala *EN*= '1' dozvoljava se pomeranje vrednosti registra) i
- dvobitni ulazni podatak *CNT[1..0]* koji definiše broj pozicija pomeraja vrednosti registra ulevo.

Sadržaj registra prisutan je na izlazu preko signala *DATA_OUT[3..0]*. Funkcija paralelnog upisa vrednosti u registar je većeg prioriteta od pomeranja vrednosti registra.

Realizovati sistema primenom softverskog paketa za rad sa programabilnim kolima firme ALTERA. Sistem opisati grafičkim putem preko standardnih logičkih kola. Koristiti princip hijerarhijske organizacije sistema.

Izvršiti sumulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

Barrel shifter – teorijski uvod

Standardno kolo za pomeranje, pomerač ili *shifter* (engleska reč koja se odomaćila), pomera sve bite posmatrane reči za jedno mesto ulevo u slučaju pomerača ulevo (SLL pomerača) ili udesno kod pomerača udesno (SRL pomerača), popunjavajući upražnjeno mesto na desnoj odnosno levoj strani nulom ili specificiranom vrednošću preko posebne linije signala. U osnovi, pomerač je sekvencijalna mreža koja obavlja ovu operaciju kroz jednu periodu

sinhronizacionog signala. Da bi se ostvarilo pomeranje reči za N pozicija korišćenjem standardnog pomerača potrebno je ponoviti isti postupak N puta i samim tim obaviti ovu operaciju kroz N perioda sinhronizacionog signala.

Sekvencijalna mreža koja ima mogućnost da kroz jednu periodu sinhronizacionog signala izvrši pomeranje posmatrane reči za N pozicija ulevo ili udesno naziva se *barrel shifter*. *Barrel shifter* ima osobinu kružnog pomeranja bita što će biti ilustrovano sledećim primerom u slučaju pomeranja osmobitne reči formata: D7 D6 D5 D4 D3 D2 D1 D0:

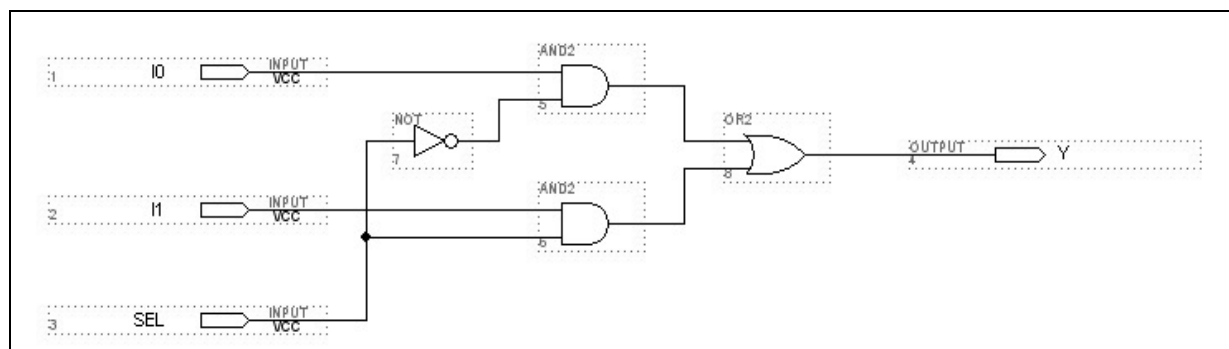
- reč pre pomeranja: D7 D6 D5 D4 D3 D2 D1 D0
- reč nakon pomeranja za 3 pozicije ulevo: D4 D3 D2 D1 D0 D7 D6 D5
- reč nakon pomeranja za 4 pozicije udesno: D3 D2 D1 D0 D7 D6 D5 D4.

REŠENJE

Zahtevani zadatak rešava se na sličan način kao i većina problema realizacije grafičkim putem sekvencijalnih mreža nižeg stepena složenosti tj. registara, brojača različitih namena i sl.

Centralno mesto u sistemu zauzima memorijski element koji se skoro redovno realizuje preko D-FF. Sistem sadrži onoliko jednobitnih memorijskih elemenata koliko bita ima registar sistema. Na ulaz D-FF dovodi se izlaz kombinacione mreže, tj. podsistema koji određuje rad sistema. Ovaj podsistem implementira se preko multipleksera gde se vrlo jednostavno ostvaruju zahtevi prioriteta više funkcija sistema (paralelni upis, zadržavanje vrednosti, pomeranje ulevo, itd.) Treći deo sistema je podsistem koji realizuje specifičnu funkciju sistema. U ovom slučaju podsistem specifične funkcije *barrel shifter* kola realizuje se preko multipleksera 4/1. Multiplekser 4/1 obavlja selekciju određene vrednosti koju treba upisati u registar na bazi signala *CNT[1..0]*.

Za implementaciju prioritetnog selektora funkcije zahtevanog sistema koristi se kolo multiplekser 2/1. Šema kola multipleksera 2/1 data je na slici br. 11.1.

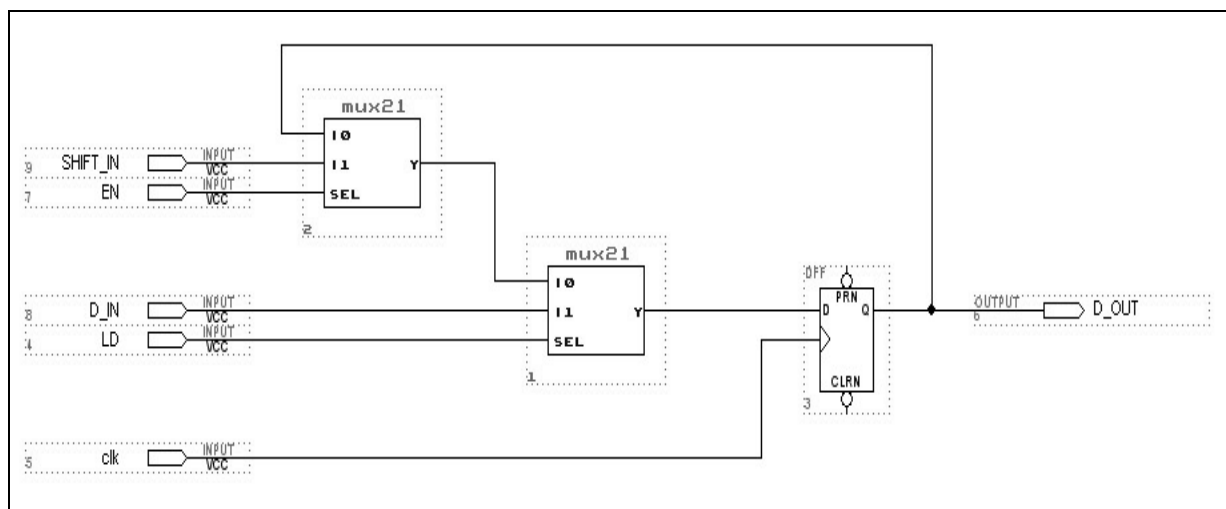


Slika 11.1: Šema multipleksera 2/1 – mux21

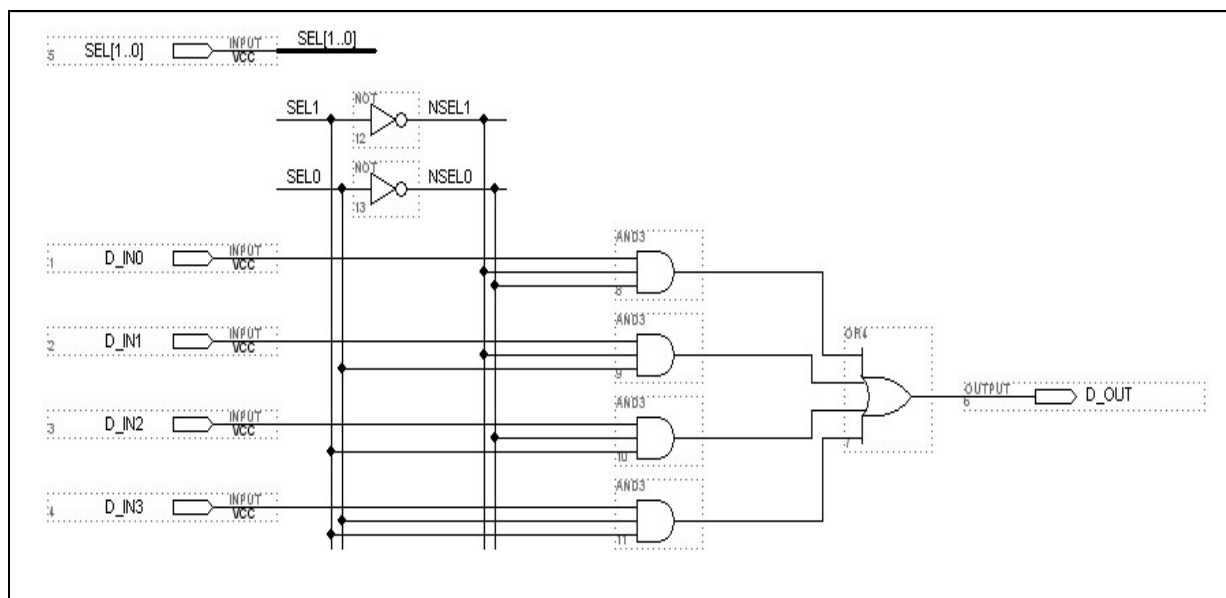
Osnovni deo jedne ćelije pomeračkog registra predstavlja D-FF kao memorijski element i kolo kojim se obezbeđuju prioritetne funkcije rada sistema. Selekcija funkcije sistema ostvaruju se preko signala *LD* i *EN* čime se aktiviraju paralelni upis i dozvola pomeranja. Šema jedne ćelije opisanog podsistema data je na slici br. 11.2.

Na slici br. 11.2 simbolom *mux21* predstavljeno je kolo sa slike br. 11.1. Podsistem na slici br. 11.2 je deo jedne ćelije većine multifunkcionalnih registara kao što je pomerač, brojač i sl.

Za implementaciju zahtevanog sistema sa funkcijom *barrel shifter*-a potreban je multiplekser 4/1. Šema multipleksera 4/1 data je na slici br. 11.3.



Slika 11.2: Šema jedne ćelije (jednog razreda) registra sa funkcijom paralelnog upisa i dozvolom izvršenja spoljne operacije tj. pomeranja vrednosti – *reg_cell*

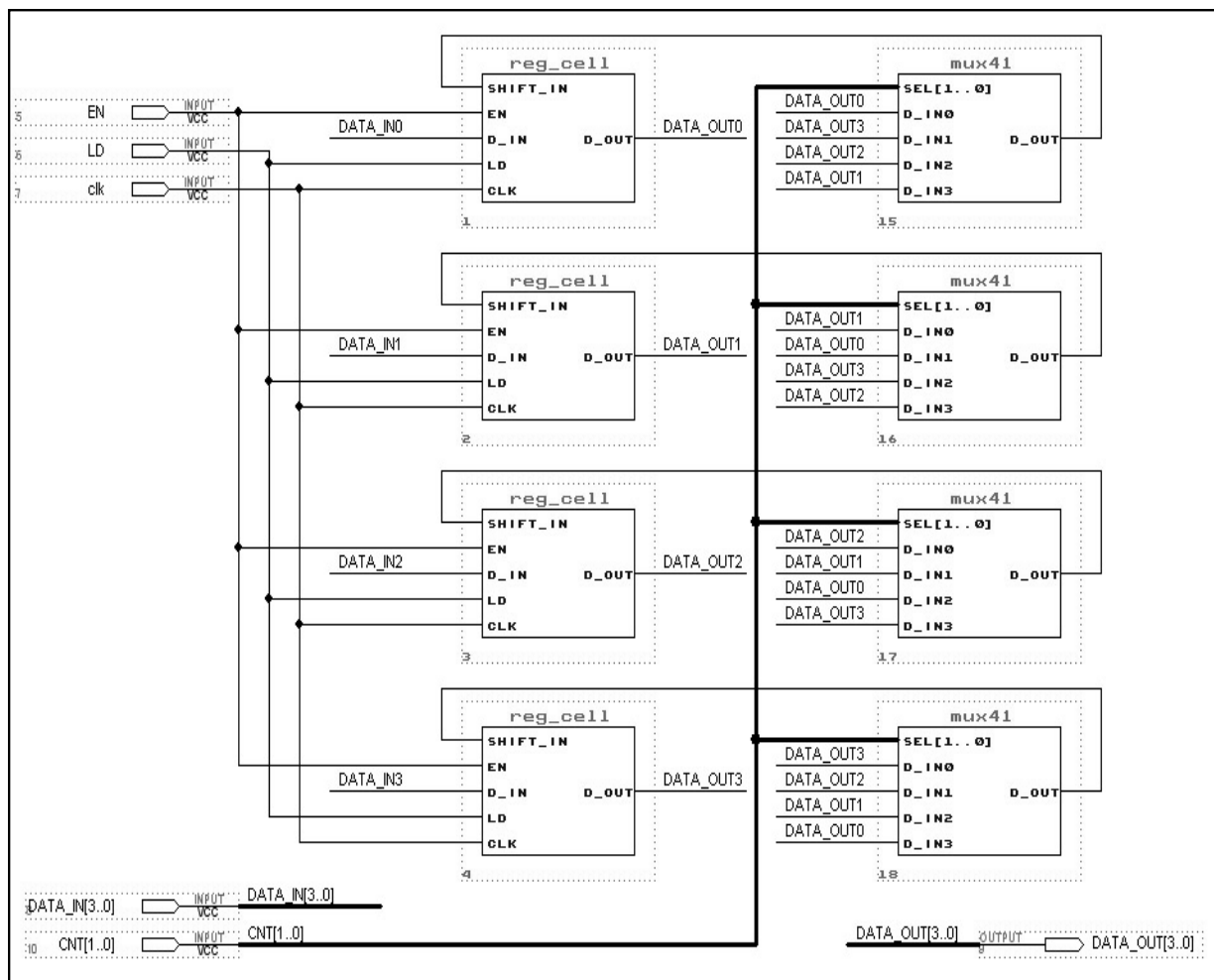
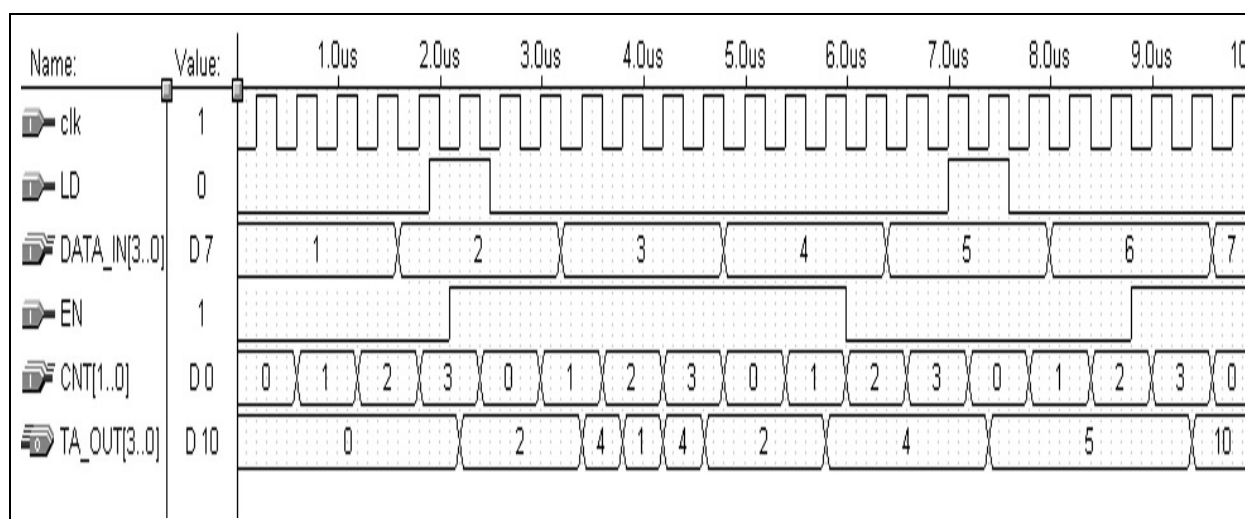


Slika 11.3: Šema multipleksera 4/1 – *mux41*

Na slici br. 11.4 data je organizacija sistema sa funkcijom *barrel shifter*-a na bazi opisane ćelije (slika br. 11.2) i multipleksera 4/1 (slika br. 11.3).

Na slici br. 11.4 simbolom *reg_cell* predstavljen je podsistem sa slike br. 11.2 a simbolom *mux41* podistem sa slike br. 11.3. Pravilnim povezivanjem ovih podistema za svaki razred sistema ostvaruje se funkcija *barrel shifter* kola. Signalima *CNT[1..0]* vrši se selekcija signala za svaki od četiri razreda preko četiri *mux41* kola.

Jedan od simulacionih dijagrama za proveru rada zahtevanog sistema sa funkcijom *barrel shifter*-a dat je na slici br. 11.5.

Slika 11.4: Šema zahtevanog sistema *barrel shifter-a*Slika 11.5: Jedan deo simulacionog dijagrama *barrel shifter-a*

Za detaljnu analizu rada sistema potreban je veći broj simulacionih dijagrama sa različitim kombinacijama signala na ulazu.

Opis sistema primenom AHDL jezika

ZADATAK 12 – Opis sekvencijalne mreže zadate preko eksitacionih jednačina

Rad sekvencijalne mreže na bazi dva memorijska elementa (dva D flip-flopa) opisan je sledećim eksitacionim jednačinama:

$$D_1 = Q_1 + \bar{X} \cdot Q_2$$

$$D_2 = X \cdot \bar{Q}_1 + \bar{X} \cdot Q_2$$

$$Z = \bar{X} \cdot Q_1 \cdot Q_2 + X \cdot \bar{Q}_1 \cdot \bar{Q}_2$$

pri čemu su D_1 i D_2 ulazi a Q_1 i Q_2 izlazi D-FF. Signal X je ulaz a Z izlaz sekvencijalne mreže. Na osnovu prethodnih jednačina:

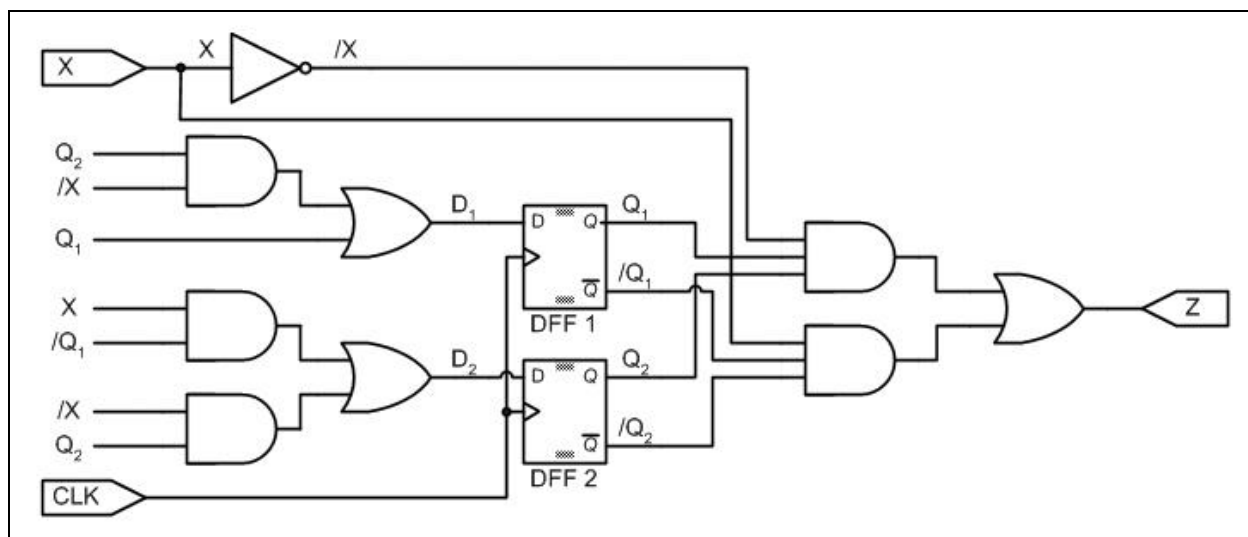
- formirati sekvencijalnu mrežu primenom dva D-FF i elementarnih logičkih kola;
- definisati stanja sistema i formirati tabelu stanja i
- formirati tabelu prelaza i nacrtati dijagram stanja.

Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati mrežu AHDL opisom.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa, vremena trajanja simulacije i test signala ulaznih priključaka sistema.

REŠENJE

Sekvencijalna mreža koja je prikazana na slici br. 12.1 dobija se direktnom realizacijom postavljenih jednačina primenom elementarnih logičkih kola i flip-flopa (FF). Ulazni i izlazni signali D-FF kola obeleženi su oznakama D_1 , Q_1 i $\bar{Q}_1$ za prvi i D_2 , Q_2 i $\bar{Q}_2$ za drugi FF. U cilju bolje preglednosti realizacije sekvencijalne mreže grafičkim putem, povezanost pojedinih linija signala na slici br. 12.1 ostvarena je istim imenovanjem linija (npr. linije signala Q_1 , Q_2 , $\bar{Q}_1$, X ...)



Slika 12.1: Električna šema sekvencijalne mreže

Pre formiranja tabele stanja treba izvršiti specifikaciju stanja sekvencijalne mreže. Kako je reč o sistemu koji ima dva FF (dva binarna memorijska elementa) jasno je da je maksimalan broj stanja u sistemu $2^2=4$.

Tabela 12.1: Specifikacija stanja sekvencijalne mreže

Stanje	Q1 (Q₁)	Q2 (Q₁)
S0	0	0
S1	0	1
S2	1	0
S3	1	1

Signali Q_1 , Q_2 sa šeme na slici br. 12.1 označeni su signalima $Q1$ i $Q2$ u daljem tekstu. Invertovani signali linija signala X , Q_1 i Q_2 označeni su $/X$, $/Q_1$ i $/Q_2$. U tabeli br. 12.1 data je specifikacija stanja sistema u funkciji sadržaja FF, tj. $Q1$ i $Q2$.

Na osnovu jednačina koje su date u tekstu zadatka i funkcije prenosa memorijskog elementa D-FF:

$$Q_i(t_{n+1}) = D_i(t_n),$$

vrši se formiranje tabele stanja (tabela br. 12.2). Postupak formiranja tabele stanja sastoji se u posmatranju i analizi svih mogućih kombinacija signala na ulazu i stanja D-FF i određivanju stanja u koje će sistem preći pri sledećoj aktivnoj ivici sinhronizacionog signala.

Tabela 12.2: Tabela stanja opisanog sistema

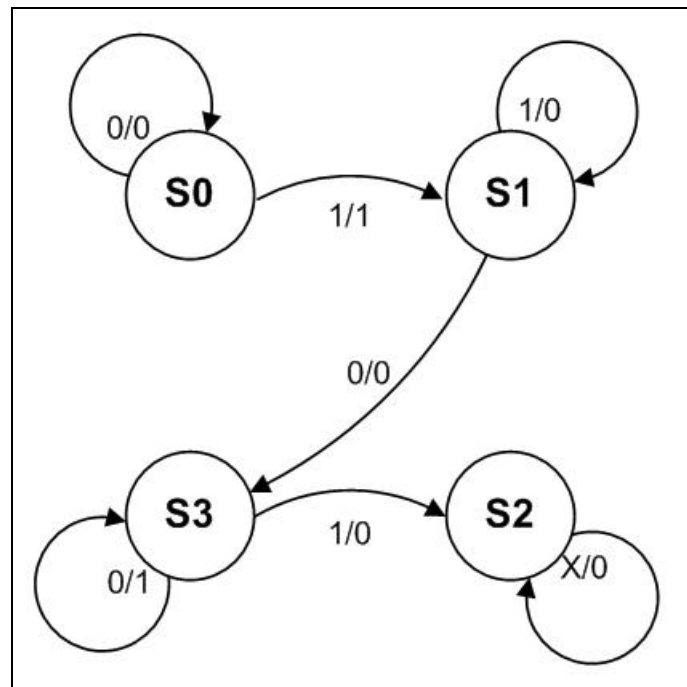
Ulaz	stanje		stanje		Izlaz
X	t=t_n		t=t_{n+1}		Z
	Q1	Q2	Q1	Q2	
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	0	0

Na bazi tabele stanja izvodi se tabela prelaza izdvajanjem signala ulaza, izlaza i stanja i odgovarajućim grupisanjem signala kao što je prikazano u tabeli br. 12.3.

Tabela 12.3: Tabela prelaza

t=t_n			t=t_{n+1}
stanje	X	Z	stanje
S0	0	0	S0
S0	1	1	S1
S1	0	0	S3
S1	1	0	S1
S2	0	0	S2
S2	1	0	S2
S3	0	1	S3
S3	1	0	S2

Na osnovu tabele prelaza formira se dijagram stanja koji je prikazan na slici br. 12.2.



Slika 12.2: Dijagram stanja

Kod sekvencijalnih mreža Milijevo tipa signali na izlazu zavise od trenutnog stanja sistema i vrednosti signala na ulazu. Na osnovu jednačine za signal na izlazu Z može se zaključiti da je ova sekvencijalna mreža Milijevo tipa. Uslov promene stanja sistema, tj. vrednost signala na ulazu X , kao i vrednost izlaznog signala Z naznačavaju se na liniji prelaska iz stanja u stanje u poretku X/Z .

Sekvencijalne mreže najjednostavnije se u AHDL jeziku opisuju preko obrasca mašine stanja (FSM). Ključne reči koje opisuju mehanizam mašine stanja su *MACHINE OF BITS...WITH STATES...TABLE...END TABLE*. Tabela prelaza i dijagram stanja predstavljaju polazne elemente u opisu i formiranju AHDL koda. AHDL ne dozvoljava korišćenje signala proizvoljnog imena. Među imenima signala koje AHDL ne podržava je ime signala X . Umesto imena ulaznog signala X koristi se ime X_ul . Predefinisani signali, tj. signali sa podrazumevanim prisustvom, u obrascu mašine stanja u AHDL-u su clk (sinhronizacioni signal) i $reset$ (asinhroni reset). U nastavku je dat AHDL opis sekvencijalne mreže postavljenog zadatka.

```

CONSTANT ON      = B"1";
CONSTANT OFF     = B"0";

SUBDESIGN fsm
(
  X_ul, clk, reset  : INPUT;
  Z                 : OUTPUT;
)

VARIABLE
  stanje: MACHINE
          OF BITS (Q1,Q2)
          WITH STATES ( S0, S1, S2, S3 );

```

```

BEGIN
    stanje.clk    = clk;
    stanje.reset  = reset;

    TABLE
    stanje, X_ul =>    stanje,    Z;
    %-----%
    S0,      OFF =>    S0,        0;
    S0,      ON  =>    S1,        1;

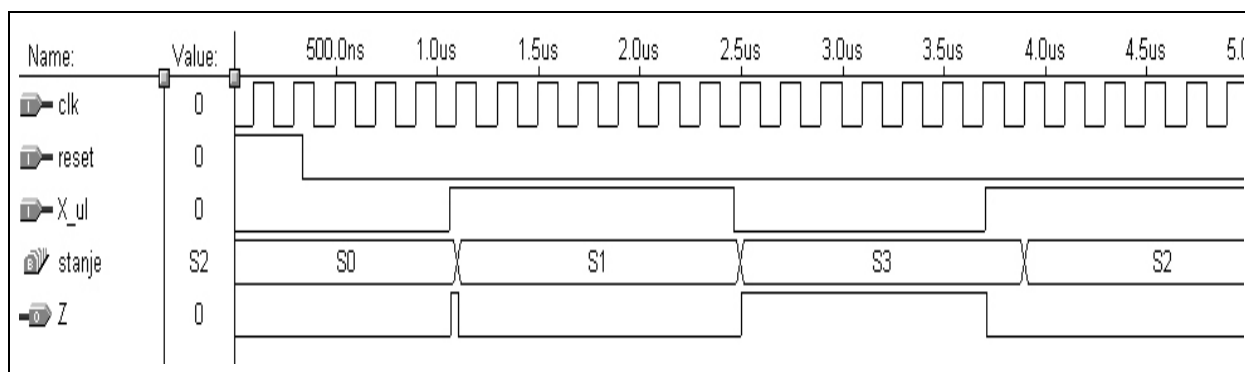
    S1,      OFF =>    S3,        0;
    S1,      ON  =>    S1,        0;

    S2,      OFF =>    S2,        0;
    S2,      ON  =>    S2,        0;

    S3,      OFF =>    S3,        1;
    S3,      ON  =>    S2,        0;
    END TABLE;
END;

```

Verifikacija rada opisanog sistema obavlja se kroz postupak simulacije. Jedan od simulacionih dijagrama kojim se verifikuje rad sistema dat je na slici br. 12.3. Iz tabele prelaza, dijagrama stanja i simulacionog dijagrama uočava se stanje S2 koje ima tu osobinu da iz njega sistem može izaći samo aktiviranjem signala *reset*.



Slika 12.3: Simulacioni dijagram opisanog sistema

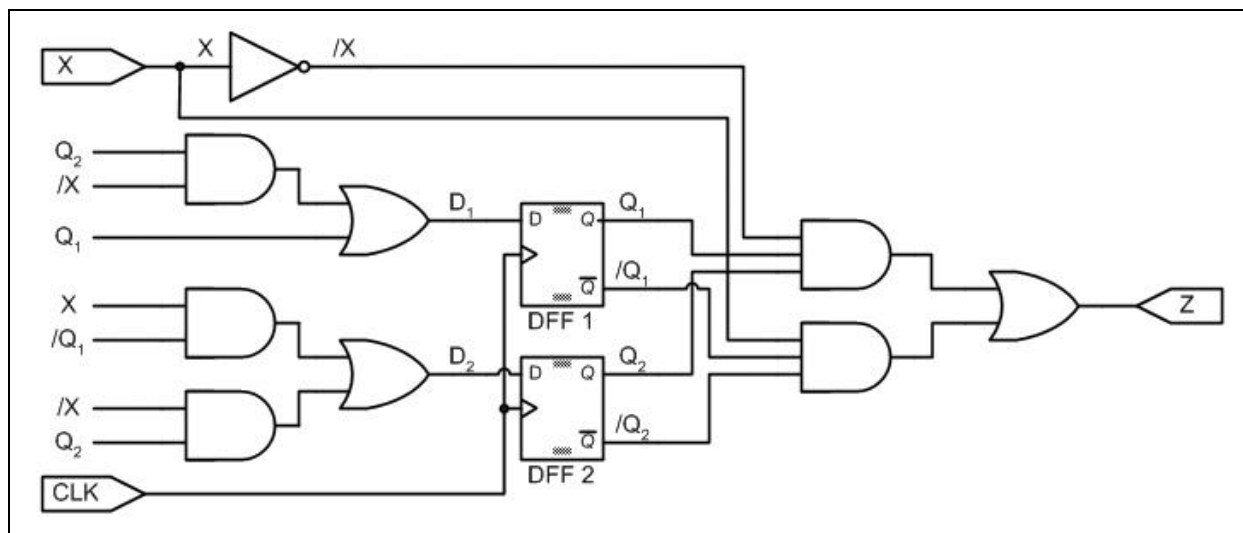
ZADATAK 13 – Analiza rada sekvencijalne mreže na osnovu električne šeme

Za sekvencijalnu mrežu koja je prikazana na slici br. 13.1:

- napisati eksitacione jednačine, odnosno izraze za logičke funkcije signala na ulazu flip-floпова (*D0* i *D1*) i signala na izlazu sistema *IZL*;
- odrediti stanja sistema i popuniti tabelu stanja;
- nacrtati dijagram stanja i formirati tabelu prelaza.

Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati mrežu AHDL opisom.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.



Slika 13.1: Električna šema sekvencijalne mreže

REŠENJE

$D1$ i $D0$ su signali na ulazu D flip-flopa (FF) FF0 i FF1 (slika br. 13.1). Na osnovu zadate šeme sekvencijalne mreže lako se određuju eksitacione jednačine za signale $D1$ i $D0$ i jednačina signala na izlazu IZL . Eksitacione jednačine i jednačina signala izlaza date su u nastavku:

$$\begin{aligned} D0 &= Q0 \oplus UL, \\ D1 &= (\overline{UL} \cdot Q1) + (UL \cdot \overline{Q0} \cdot Q1) + (UL \cdot Q0 \cdot \overline{Q1}), \\ IZL &= Q0 \cdot Q1. \end{aligned}$$

Pre formiranja tabele stanja treba izvršiti specifikaciju stanja sekvencijalne mreže tj. određivanje imena stanja i pridruživanje svakom stanju kombinaciju vrednosti FF elemenata iz mreže. Sistem sa dva FF, tj. dva binarna memorijska elementa, može da ima maksimalno $2^2=4$ stanja. U tabeli br. 13.1 data je specifikacija stanja sistema u funkciji sadržaja D-FF elemenata, tj. $Q1$ i $Q0$.

Tabela 13.1: Specifikacija stanja sistema sekvencijalne mreže sa slike br. 13.1

Stanje	$Q1$	$Q0$
S0	0	0
S1	0	1
S2	1	0
S3	1	1

Koristeći izvedene eksitacione jednačine i funkciju prelaza za D-FF element:

$$Q_i(t_{n+1}) = D_i(t_n)$$

formira se tabela stanja (tabela br. 13.2).

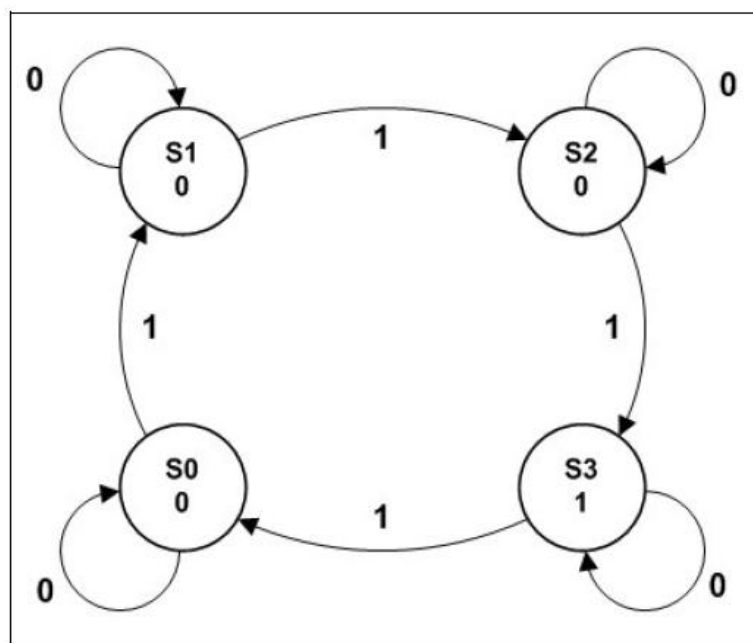
Tabela 13.2: Tabela stanja

$t=t_n$							$t=t_{n+1}$		
<i>UL</i>	<i>Q1</i>	<i>Q0</i>	<i>Stanje</i>	<i>IZL</i>	<i>D1</i>	<i>D0</i>	<i>Q1</i>	<i>Q0</i>	<i>Stanje</i>
0	0	0	S0	0	0	0	0	0	S0
0	0	1	S1	0	0	1	0	1	S1
0	1	0	S2	0	1	0	1	0	S2
0	1	1	S3	1	1	1	1	1	S3
1	0	0	S0	0	0	1	0	1	S1
1	0	1	S1	0	1	0	1	0	S2
1	1	0	S2	0	1	1	1	1	S3
1	1	1	S3	1	0	0	0	0	S0

Izdvajanjem kolona stanja sistema (*Stanje* za t_n i t_{n+1}), signala na ulazu (*UL*) i izlazu (*IZL*) iz tabele stanja (tabele br. 13.2) i odgovarajućim grupisanjem vrednosti u cilju dobijanja preciznog opisa prelaska sistema iz stanja u stanje formira se tabela prelaza (tabela br. 13.3).

Tabela 13.3: Tabela prelaza

$t=t_n$			$t=t_{n+1}$
<i>Stanje</i>	<i>Ulaz</i>	<i>Izlaz</i>	<i>Stanje</i>
	<i>UL</i>	<i>IZL</i>	
S0	0	0	S0
S0	1	0	S1
S1	0	0	S1
S1	1	0	S2
S2	0	0	S2
S2	1	0	S3
S3	0	1	S3
S3	1	1	S0

**Slika 13.2:** Dijagram stanja

Na osnovu tabele prelaza formira se dijagram stanja koji je prikazan na slici br. 13.2. Kod sekvencijalnih mreža Murovog tipa signali izlaza zavise samo od trenutnog stanja sistema.

Na osnovu jednačine za signal izlaza *IZL* može se zaključiti da je ova sekvencijalna mreža Murovog tipa. Kod mreža Murovog tipa uobičajeno je da se vrednost izlaza vezuje za trenutno stanje u kojem se sistem nalazi i da se ta vrednost u dijagramu stanja upisuje u okviru simbola za stanje. Uslov promene stanja sistema, tj. vrednost signala na ulazu *UL*, naznačava se na strelici prelaska iz stanja u stanje.

Na bazi dijagrama stanja i tabele prelaza formiran je AHDL opis sekvencijalne mreže koji je u celini dat u nastavku. Simulacioni dijagram kojim se verifikuje rad opisanog sistema dat je na slici br. 13.3.

SUBDESIGN zadatak

```
(
  UL, CLK, RESET : INPUT;
  IZL              : OUTPUT;
)
```

VARIABLE

```
  Stanje: MACHINE
          OF BITS ( q1, q0 )
          WITH STATES ( ST0, ST1, ST2, ST3 );
```

BEGIN

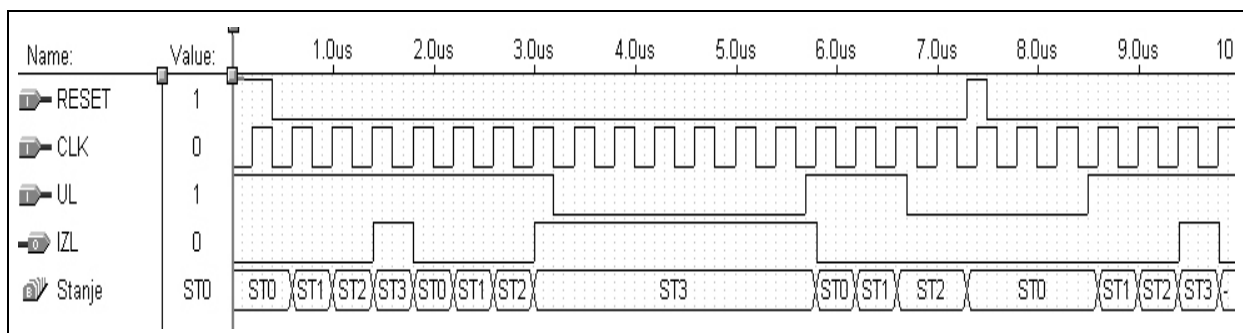
```
  Stanje.clk = CLK;
  Stanje.reset = RESET;
```

TABLE

Stanje,	UL	=>	Stanje,	IZL;
ST0,	0	=>	ST0,	0;
ST0,	1	=>	ST1,	0;
ST1,	0	=>	ST1,	0;
ST1,	1	=>	ST2,	0;
ST2,	0	=>	ST2,	0;
ST2,	1	=>	ST3,	0;
ST3,	0	=>	ST3,	1;
ST3,	1	=>	ST0,	1;

END TABLE;

END;

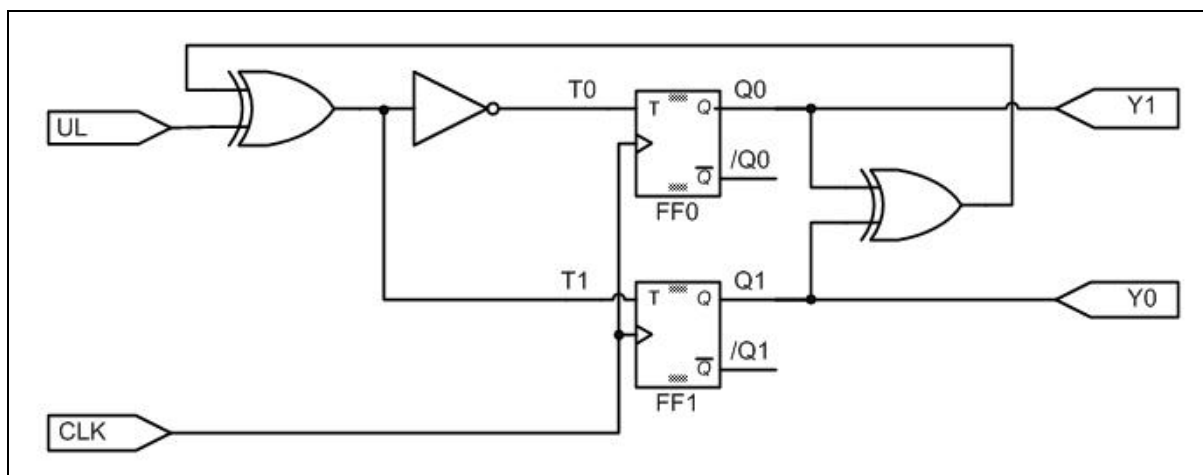


Slika 13.3: Simulacioni dijagram verifikacije rada sistema sekvencijalne mreže

ZADATAK 14 – Analiza rada sekvencijalne mreže na osnovu električne šeme

Za sekvencijalnu mrežu čija je električna šema prikazana na slici br. 14.1:

- odrediti eksitacione jednačine, odnosno izraze za logičke funkcije ulaznih signala memorijskih elementata, flip-flopova FF0 i FF1;
- odrediti logičke funkcije signala na izlazu sekvencijalne mreže ($Y0$ i $Y1$);
- odrediti stanja sistema i tabelu stanja i
- formirati dijagram stanja i tabelu prelaza stanja.



Slika 14.1: Električna šema sekvencijalne mreže

Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati mrežu AHDL opisom.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

Na osnovu zadate električne šeme sekvencijalne mreže (slika br. 14.1) lako se određuju eksitacione jednačine za memorijske elemente, tj. signale $T1$ i $T0$, i jednačine signala na izlazu mreže $Y0$ i $Y1$. Eksitacione jednačine i jednačine signala na izlazu dati su u nastavku:

$$T0 = \overline{(UL \oplus (Q1 \oplus Q0))}$$

$$T1 = (UL \oplus (Q1 \oplus Q0))$$

$$Y1 = Q0$$

$$Y0 = Q1$$

Sistem sa dva FF, tj. dva binarna memorijska elementa, može da radi najviše sa $2^2=4$ stanja. U tabeli br. 14.1 data je specifikacija stanja sistema u funkciji sadržaja T-FF elemenata, tj. signala $Q1$ i $Q0$.

Tabela 14.1: Specifikacija stanja sekvencijalne mreže sa slike br. 14.1

Stanje	$Q1$	$Q0$
S0	0	0
S1	0	1
S2	1	0
S3	1	1

Opis rada T-FF memorijskog elementa opisan je funkcijom u nastavku i tablicom prenosa (tabela br. 14.2).

$$Q(t_{n+1}) = T \cdot \overline{Q(t_n)} + \overline{T} \cdot Q(t_n) = T \oplus Q(t_n)$$

Tabela 14.2: Funkcionalna tabela T-FF elementa

$t=t_n$		$t=t_{n+1}$
T	Q	Q
0	0	0
0	1	1
1	0	1
1	1	0

Na osnovu kombinacija vrednosti signala trenutnog stanja sistema ($Q1$ i $Q0$), ulaza (UL) kao i jednačina prenosa popunjavaju se kolona $T1$, $T0$, $Y1$ i $Y0$ u tabeli stanja (tabela br. 14.3). Vrednosti signala izlaza iz elemenata T-FF u sledećem trenutku ($t=t_{n+1}$) određuju se na osnovu signala $Q1$, $T1$ odnosno $Q0$, $T0$ u sadašnjem trenutku ($t=t_n$) i funkcionalne tabele T-FF.

Tabela 14.3: Tabela stanja

$t=t_n$								$t=t_{n+1}$		
UL	$Q1$	$Q0$	$Stanje$	$T1$	$T0$	$Y1$	$Y0$	$Q1$	$Q0$	$Stanje$
0	0	0	S0	0	1	0	0	0	1	S1
0	0	1	S1	1	0	1	0	1	1	S3
0	1	0	S2	1	0	0	1	0	0	S0
0	1	1	S3	0	1	1	1	1	0	S2
1	0	0	S0	1	0	0	0	1	0	S2
1	0	1	S1	0	1	1	0	0	0	S0
1	1	0	S2	0	1	0	1	1	1	S3
1	1	1	S3	1	0	1	1	0	1	S1

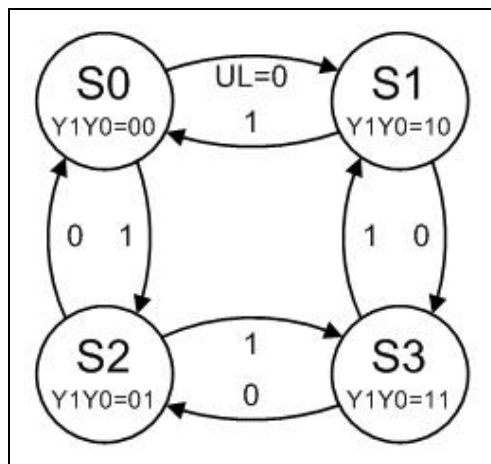
Izdvajanjem kolona stanja sistema ($Stanje$ za $t=t_n$ i $t=t_{n+1}$), signala na ulazu (UL) i izlazu ($Y1$ i $Y0$) iz tabele stanja (tabele br. 14.3) i odgovarajućim grupisanjem vrednosti u cilju dobijanja preciznog opisa prelaska sistema iz stanja u stanje formira se tabela prelaza (tabela br. 14.4).

Tabela 14.4: Tabela prelaza

$t=t_n$				$t=t_{n+1}$
UL	$Stanje$	$Y1$	$Y0$	$Stanje$
0	S0	0	0	S1
1	S0	0	0	S2
0	S1	1	0	S3
1	S1	1	0	S0
0	S2	0	1	S0
1	S2	0	1	S3
0	S3	1	1	S2
1	S3	1	1	S1

Dijagram stanja (slika br. 14.2) formira se na osnovu podataka koji su predstavljeni tabelom prelaza (tabela br. 14.3). Kako je reč o mreži Murovog tipa, svakom stanju sistema

odgovara određena kombinacija vrednosti signala na izlazu ($Y1$, $Y0$) bez direktnog uticaja vrednosti signala na ulazu (UL).



Slika 14.2: Dijagram stanja

Na bazi dijagrama stanja i tabele prelaza generiše se AHDL opis sekvencijalne mreže. AHDL opis dat u nastavku.

SUBDESIGN zadatak

```
(
  CLK, RESET, UL : INPUT;
  Y1, Y0          : OUTPUT;
)
```

VARIABLE

```
Stanje: MACHINE OF BITS ( Q1, Q0 )
        WITH STATES ( S0, S1, S2, S3 );
```

BEGIN

```
Stanje.clk = CLK;
Stanje.reset = RESET;
```

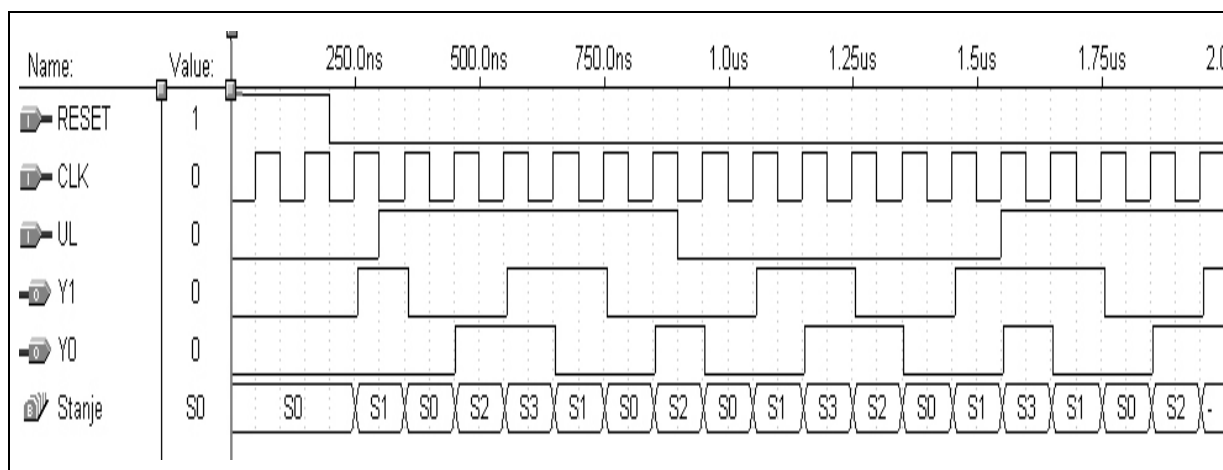
TABLE

Stanje,	UL	=>	Stanje,	Y1,	Y0;
S0,	0	=>	S1,	0,	0;
S0,	1	=>	S2,	0,	0;
S1,	0	=>	S3,	1,	0;
S1,	1	=>	S0,	1,	0;
S2,	0	=>	S0,	0,	1;
S2,	1	=>	S3,	0,	1;
S3,	0	=>	S2,	1,	1;
S3,	1	=>	S1,	1,	1;

```
END TABLE;
```

```
END;
```

Simulacioni dijagram kojim se verifikuje rad opisanog sistema dat je na slici br. 14.3.



Slika 14.3: Simulacioni dijagram

ZADATAK 15 – Opis sekvencijalne mreže zadate preko eksitacionih jednačina

Sekvencijalna mreža sadrži dva binarna memorijska elementa tipa JK-FF sa oznakom FF-A i FF-B. J i K signali na ulazu FF-A su J_A i K_A dok je izlaz FF-A signal Q_A . Analogno važi za drugi FF element, FF-B. Mreža sadrži jedan ulazni signal UL i dva izlaza sa oznakom signala Y i Z . Sistem je opisan preko jednačina tj. eksitacionih jednačina ulaza FF i izlaza:

$$\begin{aligned} J_A &= Q_B & J_B &= UL \cdot \overline{Q_A} + \overline{UL} \cdot Q_A & Y &= \overline{Q_A} + Q_B \\ K_A &= UL \cdot Q_B & K_B &= \overline{UL} & Z &= \overline{UL} \cdot Q_B + UL \cdot \overline{Q_A} \cdot \overline{Q_B} \end{aligned}$$

Za sekvencijalnu mrežu koja je opisana jednačinama:

- odrediti stanja sistema i tabelu stanja i
- formirati dijagram stanja i tabelu prelaza.

Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati mrežu AHDL opisom.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

Sekvencijalna mreža specificirana eksitacionim jednačinama i jednačinama izlaznih signala sadrži dva memorijska elementa tipa JK-FF (FF-A i FF-B). Mreže sa dva flip-flopa mogu da podrže rad sa najviše $2^2 = 4$ različita stanja. U tabeli br. 15.1 data je specifikacija stanja sistema u funkciji izlaznih signala FF-A i FF-B, odnosno signala Q_A i Q_B .

Tabela 15.1: Specifikacija stanja sekvencijalne mreže

Stanje	Q_A	Q_B
S0	0	0
S1	0	1
S2	1	0
S3	1	1

Za sve kombinacije vrednosti signala trenutnog stanja sistema (Q_A i Q_B) i ulaza (UL) popunjavaju se kolone J_A , K_A , J_B , K_B , Y i Z u okviru tabele stanja (tabela br. 15.2). Vrednosti signala izlaza FF-A i FF-B za sledeći trenutak vremena ($t=t_{n+1}$) određuju se na osnovu signala Q_A , J_A , K_A , za FF-A odnosno Q_B , J_B , K_B za FF-B i funkcionalne jednačine za JK-FF:

$$Q_{JK}(t_{n+1}) = J \cdot \overline{Q_{JK}}(t_n) + \overline{K} \cdot Q_{JK}(t_n).$$

Tabela 15.2: Tabela stanja

$t=t_n$										$t=t_{n+1}$		
UL	Q_A	Q_B	<i>Stanje</i>	J_A	K_A	J_B	K_B	Y	Z	Q_A	Q_B	<i>Stanje</i>
0	0	0	S0	0	0	0	1	1	0	0	0	S0
0	0	1	S1	1	0	0	1	1	1	1	0	S2
0	1	0	S2	0	0	1	1	0	0	1	1	S3
0	1	1	S3	1	0	1	1	1	1	1	0	S2
1	0	0	S0	0	0	1	0	1	1	0	1	S1
1	0	1	S1	1	1	1	0	1	0	1	1	S3
1	1	0	S2	0	0	0	0	0	0	1	0	S2
1	1	1	S3	1	1	0	0	1	0	0	1	S1

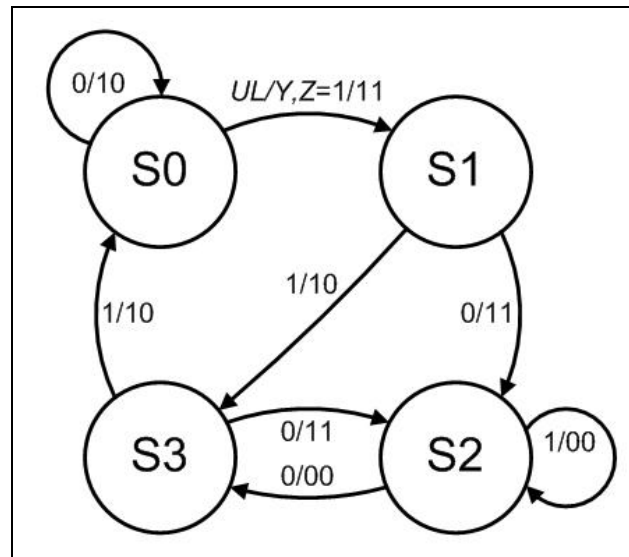
Izdvajanjem kolona stanja sistema za $t=t_n$ i $t=t_{n+1}$, signala na ulazu (UL) i izlazu (Y i Z) iz tabele stanja (tabele br. 15.2) i odgovarajućim grupisanjem vrednosti u cilju dobijanja preciznog opisa prelaska sistema iz stanja u stanje formira se tabela prelaza (tabela br. 15.3).

Tabela 15.3: Tabela prelaza

$t=t_n$				$t=t_{n+1}$
UL	<i>Stanje</i>	Y	Z	<i>Stanje</i>
0	S0	1	0	S0
1	S0	1	1	S1
0	S1	1	1	S2
1	S1	1	0	S3
0	S2	0	0	S3
1	S2	0	0	S2
0	S3	1	1	S2
1	S3	1	0	S1

Dijagram stanja (slika br. 15.1) formira se na osnovu podataka koji su predstavljani tabelom prelaza (tabela br. 15.3). Iz jednačina koje opisuju izlaze kao i iz tabele prelaza uočava se da je reč o sekvencijalnoj mreži Milijeveg tipa.

U okviru dijagrama stanja na linijama prelaza, tj. strelicama prelaza iz stanja u stanje, naznačeni su uslovi prelaska i vrednost na izlazu za određenu kombinaciju stanja i vrednosti na ulazu. Uslov prelaska iz stanja u stanje je vrednost signala UL . Na linijama prelaska nalaze se podaci u formi $UL/Y,Z$



Slika 15.1: Dijagram stanja

Iz dijagrama stanja (slika br. 15.1) i tabele prelaza (tabela br. 15.3) formira se AHDL opis sekvencijalne mreže.

SUBDESIGN zadatak

```
(
  CLK, RESET, UL : INPUT;
  Y, Z           : OUTPUT;
)
```

VARIABLE

```
  Stanje: MACHINE OF BITS ( q1, q0 )
          WITH STATES ( S0, S1, S2, S3 );
```

BEGIN

```
  Stanje.clk    = CLK;
  Stanje.reset = RESET;
```

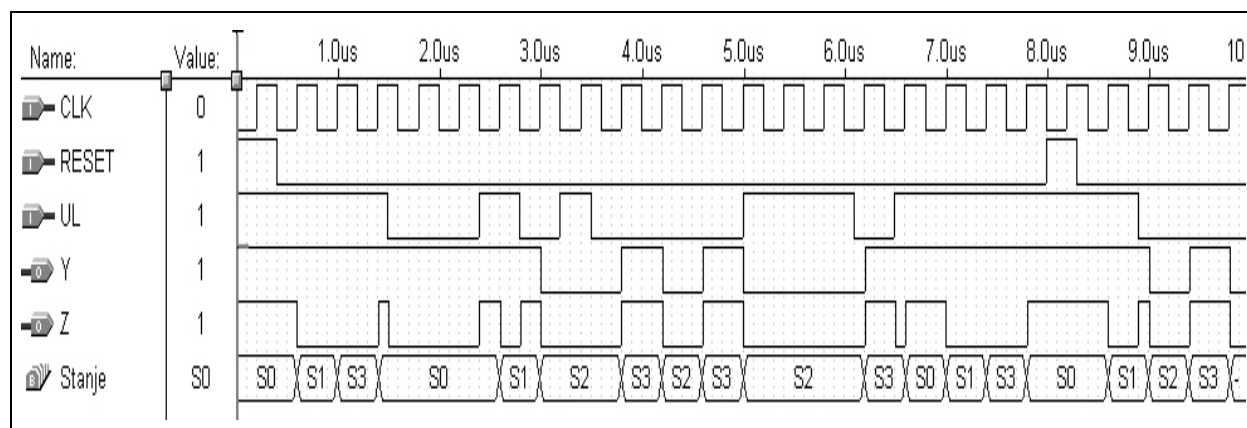
TABLE

Stanje, UL	=>	Stanje,	Y,	Z;
S0, 0	=>	S0,	1,	0;
S0, 1	=>	S1,	1,	1;
S1, 0	=>	S2,	1,	1;
S1, 1	=>	S3,	1,	0;
S2, 0	=>	S3,	0,	0;
S2, 1	=>	S2,	0,	0;
S3, 0	=>	S2,	1,	1;
S3, 1	=>	S0,	1,	0;

END TABLE;

END;

Simulacioni dijagram kojim se verifikuje rad opisanog sistema dat je na slici br. 15.2.

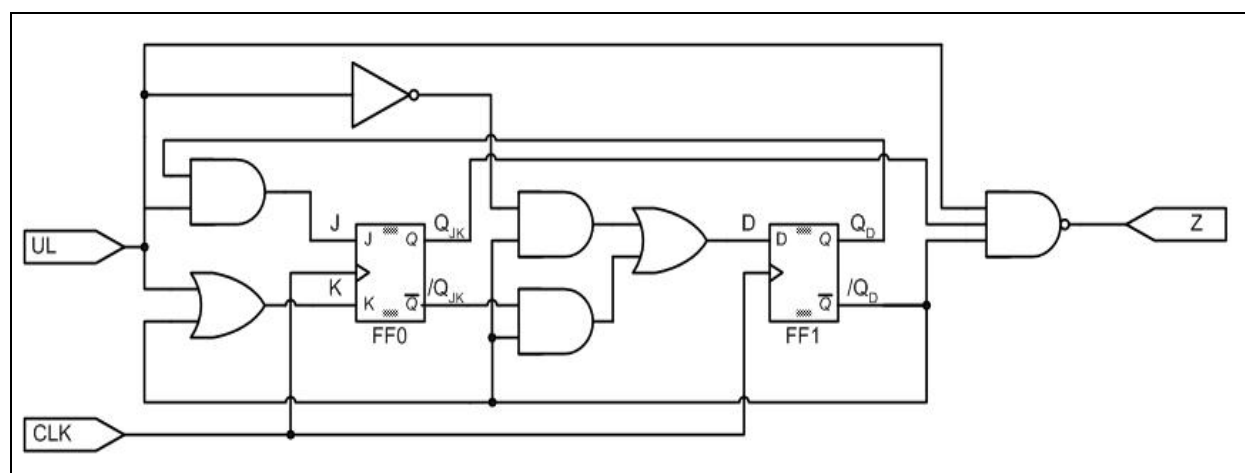


Slika 15.2: Simulacioni dijagram

ZADATAK 16 - Analiza rada sekvencijalne mreže na osnovu električne šeme

Za električnu šemu sekvencijalne mreže koja je prikazana na slici br. 16.1 treba:

- odrediti eksitacione jednačine, tj. izraze za logičke funkcije ulaznih signala memorijskih elementata - flip-flopova FF0 i FF1;
- odrediti funkciju signala na izlazu sekvencijalne mreže (Z);
- odrediti stanja sistema i tabelu stanja i
- formirati dijagram stanja i tabelu prelaza stanja.



Slika 16.1: Električna šema sekvencijalne mreže

Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati mrežu AHDL opisom.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

Eksitacione jednačine signala na ulazu FF elemenata (*J*, *K* i *D*) i izraz za funkciju signala na izlazu sekvencijalne mreže dati su u nastavku:

$$J = UL \cdot Q_D$$

$$K = UL + \overline{Q_D}$$

$$D = (\overline{UL} \cdot \overline{Q_D}) + (\overline{Q_{JK}} \cdot \overline{Q_D})$$

$$Z = \overline{(UL \cdot Q_{JK} \cdot \overline{Q_D})}$$

Sekvencijalna mreža sa dva binarna memorijska elementa (FF) može da podrži rad sa najviše $2^2 = 4$ različita stanja. U tabeli br. 16.1 data je specifikacija stanja sistema u funkciji sadržaja FF0 i FF1, tj. signala Q_{JK} i Q_D .

Tabela 16.1: Specifikacija stanja sekvencijalne mreže sa slike br. 16.1

<i>Stanje</i>	Q_{JK}	Q_D
S0	0	0
S1	0	1
S2	1	0
S3	1	1

Izrazi koji opisuju rad JK-FF i D-FF:

$$Q_{JK}(t_{n+1}) = J \cdot \overline{Q_{JK}(t_n)} + \overline{K} \cdot Q_{JK}(t_n)$$

$$Q_D(t_{n+1}) = D$$

Za sve kombinacije vrednosti signala trenutnog stanja sistema (Q_{JK} i Q_D) i ulaza (UL) popunjavaju se kolona J , K , D i Z u tabeli stanja (tabela br. 16.2). Vrednosti signala na izlazu flip-flopa za sledeći trenutak vremena ($t=t_{n+1}$) određuju se na osnovu signala na priključcima FF0 (Q_{JK} , J i K), odnosno FF1 (Q_D i D) i izraza koji opisuju rad FF elemenata.

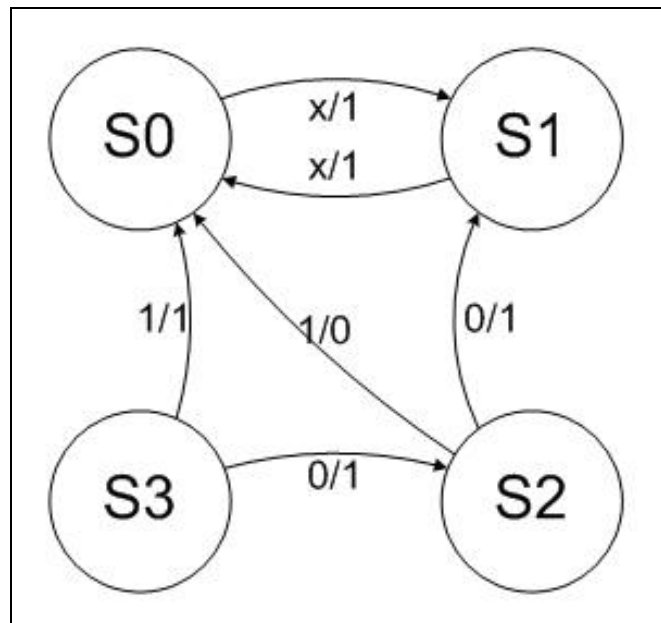
Tabela 16.2: Tabela stanja

$t=t_n$								$t=t_{n+1}$		
UL	Q_{JK}	Q_D	<i>Stanje</i>	J	K	D	Z	Q_{JK}	Q_D	<i>Stanje</i>
0	0	0	S0	0	1	1	1	0	1	S1
0	0	1	S1	0	0	0	1	0	0	S0
0	1	0	S2	0	1	1	1	0	1	S1
0	1	1	S3	0	0	0	1	1	0	S2
1	0	0	S0	0	1	1	1	0	1	S1
1	0	1	S1	0	1	0	1	0	0	S0
1	1	0	S2	1	1	0	0	0	0	S0
1	1	1	S3	1	1	0	1	0	0	S0

Izdvajanjem kolona stanja sistema (*Stanje* za $t=t_n$ i $t=t_{n+1}$), signala na ulazu (UL) i izlaza (Z) iz tabele stanja (tabele br. 16.2) i odgovarajućim grupisanjem vrednosti u cilju dobijanja preciznog opisa prelaska sistema iz stanja u stanje formira se tabela prelaza (tabela br. 16.3).

Tabela 16.3: Tabela prelaza stanja

$t=t_n$			$t=t_{n+1}$
UL	Stanje	Z	Stanje
0	S0	1	S1
1	S0	1	S1
0	S1	1	S0
1	S1	1	S0
0	S2	1	S1
1	S2	0	S0
0	S3	1	S2
1	S3	1	S0

**Slika 16.2:** Dijagram stanja

Iz dijagrama stanja i tabele prelaza stanja generiše se AHDL opis sekvencijalne mreže.

SUBDESIGN zadatak

```
(
  CLK, RESET, UL      : INPUT;
  Z                    : OUTPUT;
)
```

VARIABLE

```
Stanje: MACHINE OF BITS ( q1, q0 )
        WITH STATES ( S3, S2, S1, S0 );
```

BEGIN

```
Stanje.clk = CLK;
Stanje.reset = RESET;
```

TABLE

```
Stanje, UL => Stanje, Z;
% ----- %
S0, 0 => S1, 1;
S0, 1 => S1, 1;

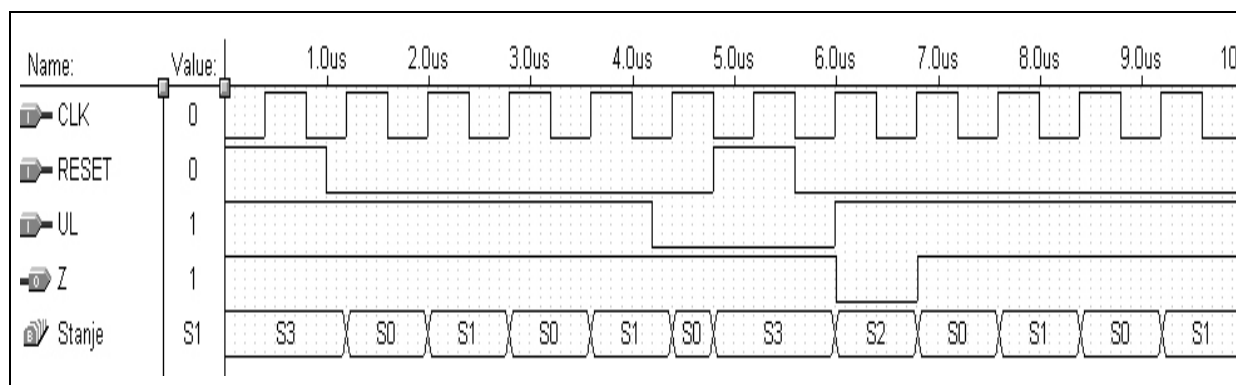
S1, 0 => S0, 1;
S1, 1 => S0, 1;

S2, 0 => S1, 1;
S2, 1 => S0, 0;

S3, 0 => S2, 1;
S3, 1 => S0, 1;
END TABLE;
```

END;

Simulacioni dijagram kojim se verifikuje rad opisanog sistema dat je na slici br. 16.3.



Slika 16.3: Simulacioni dijagram

ZADATAK 17 – Obostrani brojač zadate sekvence brojanja

Realizovati kao mašinu stanja obostrani sinhroni brojač čija je sekvenca brojanja u smeru na gore sledeća: **1→3→5→7→1→3→...** Brojevi su predstavljeni u dekadnom brojnom sistemu. Pored sinhronizacionog signala *CLK* brojač sadrži sledeće kontrolne signale na ulazu:

- *RESET* – signal asinhronog reseta;
- *DOZVOLA* – signal dozvole brojanja i
- *SMER* – signal kojim se određuje smer brojanja (nagore / nadole).

Vrednost brojača prisutna je na izlazu preko trobitne linije signala *BROJ*. Simbol brojača prikazan je na slici br. 17.1.



Slika 17.1: Simbol brojača

Svi kontrolni signali aktiviraju pridružene funkcije pri visokom logičkom nivou tj. pri vrednosti '1'. Ukoliko je *DOZVOLA*='1' sistem obavlja funkciju brojača, dok pri vrednosti *DOZVOLA*='0' brojač ostaje u zatečenom stanju. Brojač broji na gore ukoliko je *SMER*='1', odnosno na dole za *SMER*='0' pod uslovom da je *DOZVOLA*='1'. Asinhroni reset prevodi sistem u stanje u kojem je na izlazu vrednost 3 (*BROJ*[2..0]='011'). Funkcija asinhronog reseta je najvećeg prioriteta.

Primenom softverskog paketa za rad sa proizvodima firme ALTERA realizovati opisanu mrežu brojača u AHDL jeziku.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

Poznato je da sinhrona mreža, pa samim tim i sinhroni brojači, menjaju svoje stanje samo pri aktivnoj ivici sinhronizacionog signala tj. signala takta. Sinhroni brojači imaju tu specifičnost da na svom izlazu daju vrednost iz skupa vrednosti sekvence brojanja.

Kod mreža Murovog tipa vrednost na izlazu sistema je funkcija isključivo trenutnog stanja sistema pri čemu se promena stanja obavlja sinhrono sa sinhronizacionim signalom u skladu sa sekvencom brojanja i kontrolnim signalima na ulazu. Zbog prethodno navedenih specifičnosti, mreže Murovog tipa pogodne su za realizaciju sinhronih brojača.

Analizom sekvence brojanja obostranog sinhronog brojača $1 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 1 \rightarrow 3 \rightarrow \dots$ uočava se da sistem generiše na izlazu četiri različite vrednosti i to: $1_{(10)}$, $3_{(10)}$, $5_{(10)}$ i $7_{(10)}$. Za četiri različite vrednosti na izlazu potrebno je da sistem ima četiri različita stanja. Kako je u AHDL-u dozvoljeno imenovati stanja sistema proizvoljnim imenom, uvek treba težiti da su imena stanja prepoznatljiva i u skladu sa funkcijom sistema u tom stanju. Linije signala vrednosti sinhronog brojača su $BROJ[2..0]$. $BROJ$ je trobitni signal zbog potrebe da se na izlazu generiše maksimalna vrednost $7_{10}=111_2$. Za sinhroni obostrani brojač specifikacija stanja i vrednost koja je svakom stanju pridružena na izlazu data je u tabeli br. 17.1.

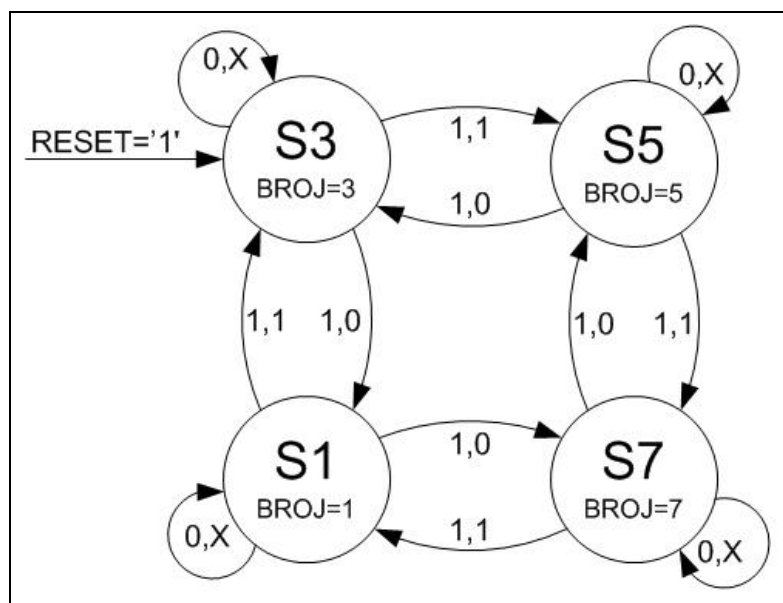
Tabela 17.1: Specifikacija stanja sinhronog obostranog brojača

Stanje	vrednost = $BROJ[2..0]$
S1	$1_{10}=001_2$
S3	$3_{10}=011_2$
S5	$5_{10}=101_2$
S7	$7_{10}=111_2$

Na osnovu specifikacije, sekvenca brojanja sinhronog obostranog brojača kada je $DOZVOLA='1'$ je:

$$\begin{array}{ll} \underline{1 \rightarrow 3 \rightarrow 5 \rightarrow 7} \rightarrow 1 \rightarrow 3 \rightarrow \dots & \text{za } SMER='1' \text{ i} \\ \underline{7 \rightarrow 5 \rightarrow 3 \rightarrow 1} \rightarrow 7 \rightarrow 5 \rightarrow \dots & \text{za } SMER='0'. \end{array}$$

Dijagram stanja prikazan je na slici br. 17.2, dok je tabela prelaza stanja data u tabeli br. 17.2.



Slika 17.2: Dijagram stanja obostranog sinhronog brojača

U okviru dijagrama stanja (slika br. 17.2) i simbola stanja (simbol stanja je naznačen krugom) pored naziva stanja naznačena je vrednost izlaza $BROJ$ koja odgovara tom stanju (npr. za stanje S3 $BROJ=3$). Ovakav način obeležavanja karakterističan je za mreže Murovog tipa.

Signali koji određuju uslov prelaska iz stanja u stanje naznačeni su na strelici prelaska u poretku *DOZVOLA*, *SMER*.

Tabela 17.2: Tabela prelaza stanja

$t=t_n$				$t=t_{n+1}$
<i>DOZVOLA</i>	<i>SMER</i>	<i>Stanje</i>	<i>BROJ</i>	<i>Stanje</i>
0	0	S1	1	S1
0	1			S1
1	0			S7
1	1			S3
0	0	S3	3	S3
0	1			S3
1	0			S1
1	1			S5
0	0	S5	5	S5
0	1			S5
1	0			S3
1	1			S7
0	0	S7	7	S7
0	1			S7
1	0			S5
1	1			S1

Na osnovu dijagrama stanja (slika br. 17.2) i tabele prelaza stanja (tabela br. 17.2) formiran je AHDL kod opisa sistema obostranog brojača:

SUBDESIGN zadatak

```
(
  DOZVOLA, SMER, CLK, RESET: INPUT;
  BROJ[2..0]                : OUTPUT;
)
```

VARIABLE

```
  Stanje: MACHINE
          OF BITS ( q1, q0 )
          WITH STATES ( S3, S1, S5, S7 );
```

BEGIN

```
  Stanje.clk      = CLK;
  Stanje.reset    = RESET;
```

TABLE

```
  Stanje, DOZVOLA, SMER    => Stanje, BROJ[2..0];
% ----- %
  S1,      0,      0      => S1,      1;
  S1,      0,      1      => S1,      1;
  S1,      1,      0      => S7,      1;
  S1,      1,      1      => S3,      1;

  S3,      0,      0      => S3,      3;
  S3,      0,      1      => S3,      3;
```

```

S3,      1,      0      =>  S1,      3;
S3,      1,      1      =>  S5,      3;

S5,      0,      0      =>  S5,      5;
S5,      0,      1      =>  S5,      5;
S5,      1,      0      =>  S3,      5;
S5,      1,      1      =>  S7,      5;

S7,      0,      0      =>  S7,      7;
S7,      0,      1      =>  S7,      7;
S7,      1,      0      =>  S5,      7;
S7,      1,      1      =>  S1,      7;

```

```

END TABLE;
END;

```

Zahtev da se obezbedi da sistem pri asinhronom resetu pređe u stanje S3 postiže se odgovarajućim redosledom stanja u okviru definisanja mašine stanja:

VARIABLE

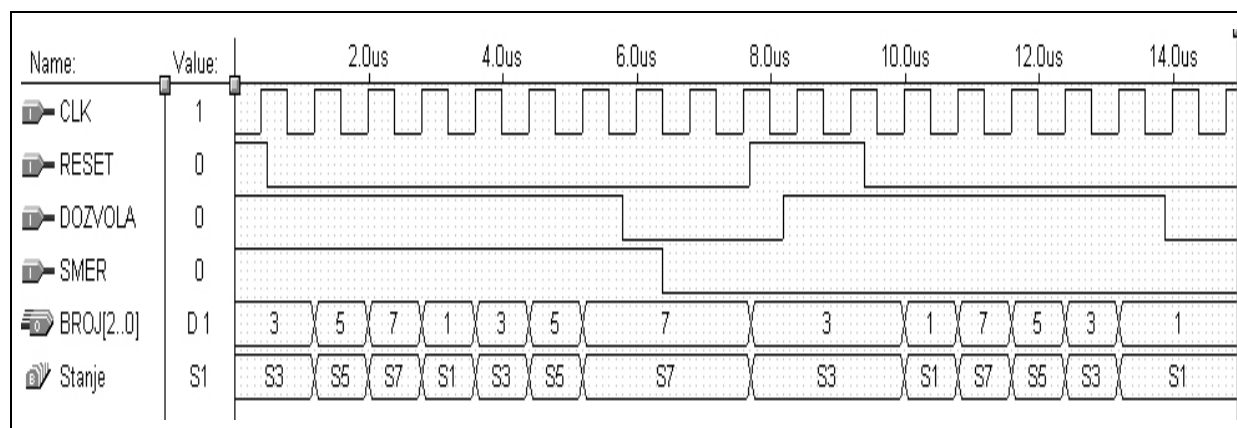
```

Stanje: MACHINE
OF BITS ( q1, q0 )
WITH STATES ( S3, S1, S5, S7 );

```

Prvo stanje koje se navede u okviru spiska stanja (S3, S1, S5, S7) predstavlja stanje u koje će sistem preći pri asinhronom resetu sistema, tj. kada je *RESET*='1'. Redosled nabrajanja ostalih stanja nije bitan.

Jadan od simulacionih dijagrama kojim se verifikuje rad opisanog sistema sinhronog obostranog brojača dat je na slici br. 17.3. Prilikom verifikacije rada brojača potrebno je proveriti brojanje po sekvenci nagore i nadole, asinhroni reset sistema i zamrzavanje vrednosti brojača što je prikazano na ovom simulacionom dijagramu.



Slika 17.3: Simulacioni dijagram provere rada sinhronog obostranog brojača

ZADATAK 18 – Obostrani brojač zadate sekvence brojanja

Realizovati sinhroni brojač čija je sekvenca brojanja za smer na gore sledeća: 1→2→3→4→5→6→1→2→... Brojevi iz sekvence predstavljeni su u dekadnom brojnom sistemu.

Brojač sadrži sledeće kontrolne signale:

- *DOZVOLA* – signal dozvole brojanja;
- *SMER* – signal smera brojanja i
- *RESET* – signal asinhronog reseta.

Vrednost brojača prisutan je na izlazu preko trobitne linije signala *VREDNOST*. Simbol brojača prikazan je na slici br. 18.1.



Slika 18.1: Simbol sinhronog brojača

Svi kontrolni signali aktiviraju svoje funkcije pri visokom logičkom nivou ('1'). Ukoliko je *DOZVOLA*='1' brojač obavlja sinhrono brojanje u skladu sa signalom *SMER*, dok u suprotnom, kada je *DOZVOLA*='0', ostaje u zatečenom stanju. Ukoliko je ispunjen uslov za brojanje (*DOZVOLA*='1'), brojač broji po sledećim pravilima:

- ukoliko je signal *SMER*='1' brojač broji nagore po specificiranoj sekvenci 1→2→3→4→5→6→1→2→...
- ukoliko je signal *SMER*='0' brojač broji nadole po neparnim odnosno parnim brojevima iz specificirane sekvence i to u zavisnosti od toga da li je u trenutku aktiviranja ovog moda brojanja brojač ukazivao na neparan odnosno paran broj. Primera radi, ako je u trenutku aktiviranja *SMER*='0' (*DOZVOLA*='1', *RESET*='0') stanje brojača bilo 3, brojač će sinhrono nastaviti da broji po sekvenci 3→1→5→3→1→...

Funkcija asinhronog reseta je najvećeg prioriteta.

Odrediti stanja, dijagram prelaza i tabelu prelaza stanja specificiranog sistema sinhronog brojača. Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati mrežu AHDL opisom.

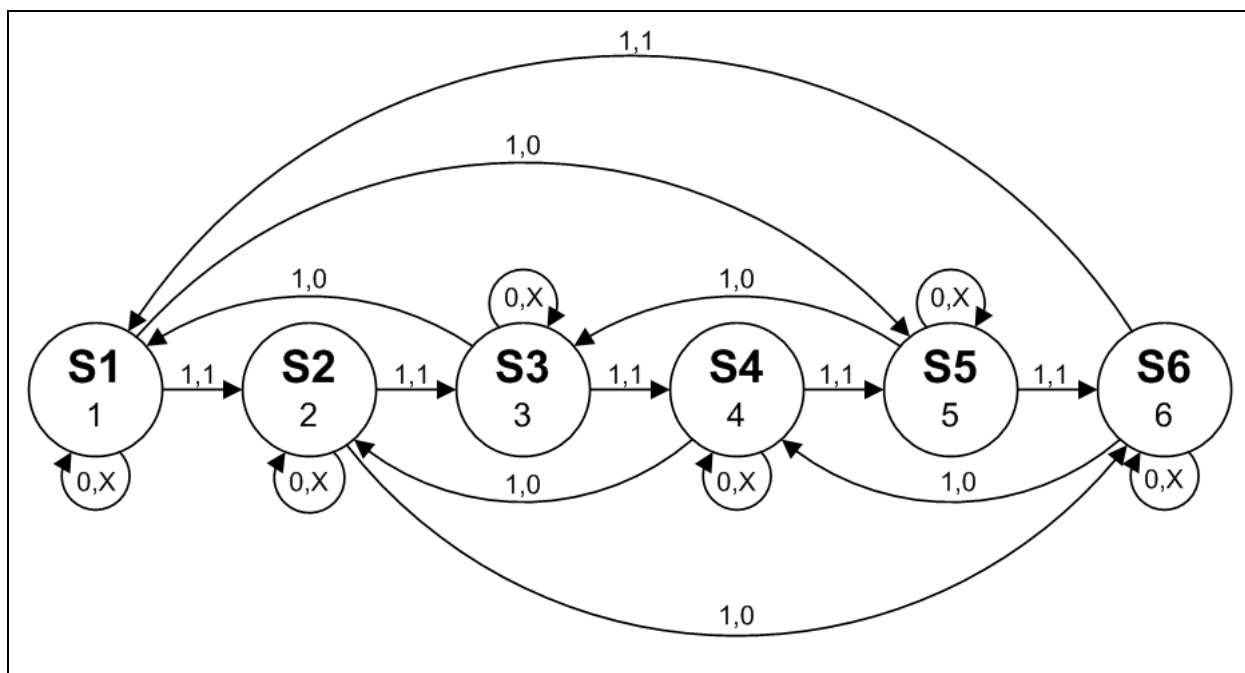
Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

Formirati simbol opisanog brojača. Osmisliti i opisati način verifikacije sistema na UP2 razvojnoj ploči i EPM7128LC84-7 čipu iz familije kola MAX7000S.

REŠENJE

Kao što je u prethodnom zadatku već pomenuto, sinhroni brojači se realizuju preko mreža Murovog tipa. Određivanje stanja sistema postiže se utvrđivanjem broja različitih vrednosti koje brojač generiše, što je u ovom slučaju šest. Svako od ovih šest vrednosti pridružuje se jedno stanje, odnosno S1 za vrednost 1, S2 za vrednost 2, itd. Sistem sadrži šest stanja: S1, S2, S3, S4, S5 i S6.

Na osnovu specifikacije brojača, utvrđenih stanja kao i činjenice da je mreža Murovog tipa (videti zadatak br. 17), formira se dijagram stanja koji je prikazan na slici br. 18.2.



Slika 18.2: Dijagram stanja

Na osnovu dijagrama stanja formira se tabela prelaza stanja koja predstavlja najbolju osnovu za opis sistema u AHDL jeziku. Prisustvo oznake X u dijagramu stanja i tabeli prelaza stanja predstavlja zamenu za vrednosti '0' i '1', tj. ima značenje da vrednost pridruženog signala u tom slučaju nije bitna.

Tabela 18.1: Tabela prelaza

stanje	$t=t_n$			$t=t_{n+1}$
	DOZVOLA	SMER	VREDNOST[3..0]	stanje
S1	0	X	1	S1
	1	0		S5
	1	1		S2
S2	0	X	2	S2
	1	0		S6
	1	1		S3
S3	0	X	3	S3
	1	0		S1
	1	1		S4
S4	0	X	4	S4
	1	0		S2
	1	1		S5
S5	0	X	5	S5
	1	0		S3
	1	1		S6
S6	0	X	6	S6
	1	0		S4
	1	1		S1

Na osnovu podataka iz tabele prelaza i opisanog principa realizacije mašine stanja u AHDL jeziku, formira se opis sinhronog brojača koji je dat u nastavku:

```

SUBDESIGN brojac
(
  CLK, RESET      : INPUT;
  DOZVOLA, SMER   : INPUT;
  VREDNOST[3..0]  : OUTPUT;
)

VARIABLE
  Stanje: MACHINE
           OF BITS ( q2, q1, q0 )
           WITH STATES ( S1, S2, S3, S4, S5, S6 );

BEGIN
  Stanje.clk      = CLK;
  Stanje.reset    = RESET;

  TABLE
  Stanje, DOZVOLA, SMER => Stanje, VREDNOST[3..0];
% ----- %
  S1,      0,      0 => S1,      1;
  S1,      0,      1 => S1,      1;
  S1,      1,      0 => S5,      1;
  S1,      1,      1 => S2,      1;

  S2,      0,      0 => S2,      2;
  S2,      0,      1 => S2,      2;
  S2,      1,      0 => S6,      2;
  S2,      1,      1 => S3,      2;

  S3,      0,      0 => S3,      3;
  S3,      0,      1 => S3,      3;
  S3,      1,      0 => S1,      3;
  S3,      1,      1 => S4,      3;

  S4,      0,      0 => S4,      4;
  S4,      0,      1 => S4,      4;
  S4,      1,      0 => S2,      4;
  S4,      1,      1 => S5,      4;

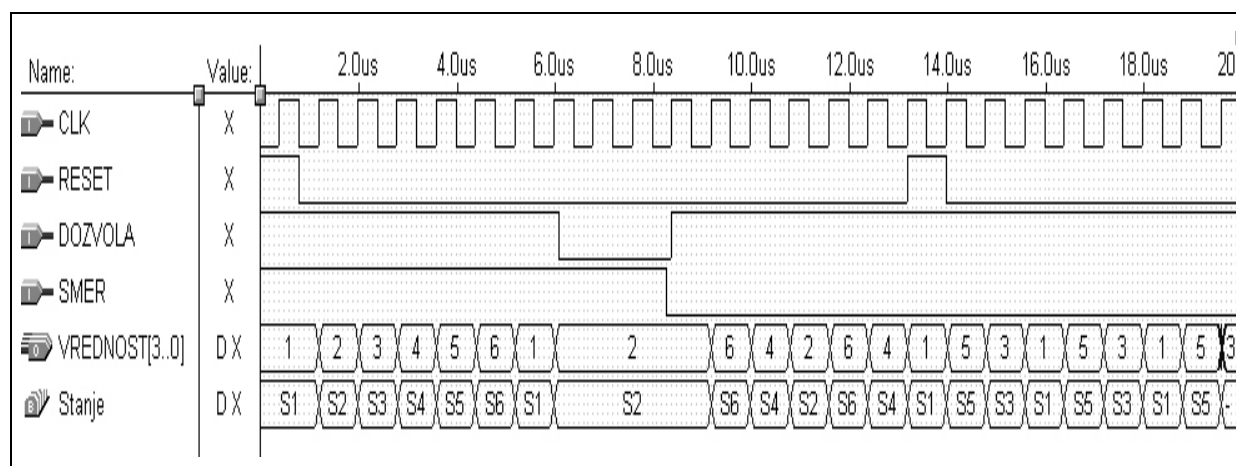
  S5,      0,      0 => S5,      5;
  S5,      0,      1 => S5,      5;
  S5,      1,      0 => S3,      5;
  S5,      1,      1 => S6,      5;

  S6,      0,      0 => S6,      6;
  S6,      0,      1 => S6,      6;
  S6,      1,      0 => S4,      6;
  S6,      1,      1 => S1,      6;
END TABLE;
END;

```

Redosled specificiranja trenutne vrednosti signala na izlazu *VREDNOST* i stanje u koje će sistem preći *Stanje*, tj. signala desno od znaka => u tabeli opisa mašine stanja, nije bitan. Važno je pridržavati se opisa mašine stanja u skladu sa redosledom koji je dat u zaglavlju tabele.

Jadan od simulacionih dijagrama kojim se verifikuje rad opisanog sistema sinhronog brojača dat je na slici br. 18.3. Prilikom verifikacije rada brojača potrebno je proveriti brojanje po sekvenci na gore i na dole, asinhroni reset sistema i zamrzavanje vrednosti brojača što je prikazano na ovom simulacionom dijagramu.



Slika 18.3: Simulacioni dijagram sinhronog brojača

Verifikacija sinhronog brojača na UP2 ploči sastoji se u formiranju sistema preko koga se opisani brojač prilagođava okruženju koje je prisutno na UP2 ploči. Ponuđena test šema predviđena je za implementaciju dizajna i proveru ispravnosti rada na CPLD EPM7128LC84.

Na samom početku treba formirati simbol sinhronog brojača koji predstavlja osnovu test sistema. Aktiviranjem opcije za formiranje simbola opisanog dizajna dobija se element *brojac* čiji je simbol prikazan na slici br. 18.4.

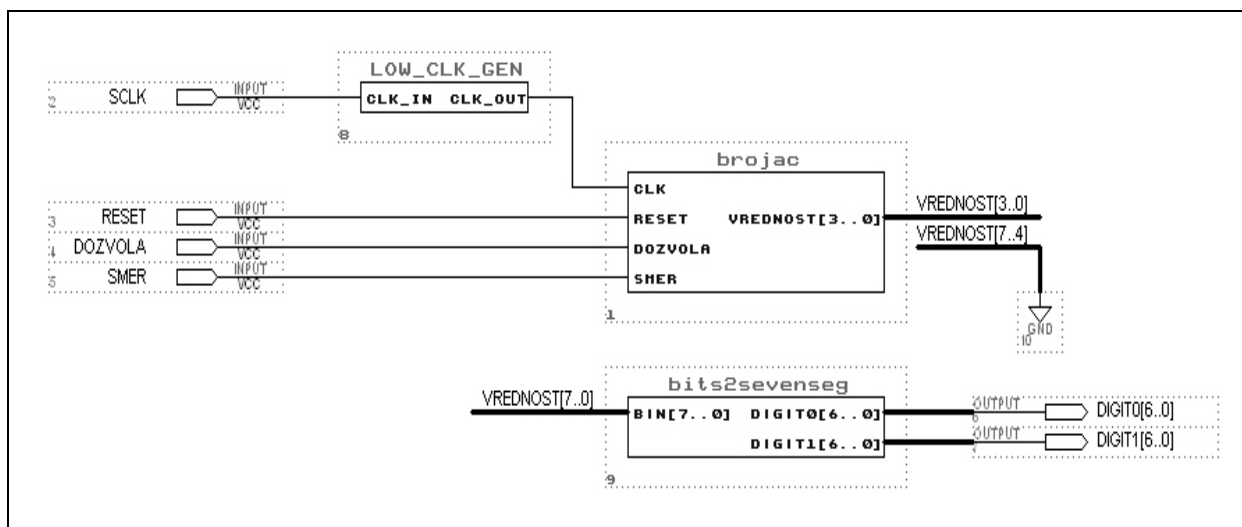


Slika 18.4: Simbol sinhronog brojača

Verifikacija ispravnosti rada realizovanog sinhronog brojača na UP2 razvojnoj ploči može se obaviti preko test šeme koja je prikazana na slici br. 18.5. U tabeli br. 18.2 data je specifikacija signala koji se koriste u okviru test šema kao i opis njihovog korišćenja.

Kako bi se verifikovao rad brojača u realnom vremenu vizuelnom indikacijom, potrebno je dovesti sinhronizacioni signal niske učestanosti na *CLK* ulaz brojača. Iz ovog razloga koristi se pomoćno kolo *LOW_CLK_GEN* čiji je opis dat u okviru Priloga B Priručnika za laboratorijske vežbe iz programabilnih logičkih kola. Sistemski takt od 25,175MHz dovodi se na ulaz kola *LOW_CLK_GEN* dok se na izlazu dobija sinhronizacioni signal za rad brojača od oko 1Hz. Prezentacija vrednosti brojača ostvaruje se preko dva sedmosegmentna displeja i pomoćnog podsistema *bits2sevenseg*. Preko podsistema *bits2sevenseg* vrši se dekodiranje

osmobitne vrednosti u niz od 14 signala za pobuđivanje dva sedmosegmentna displeja MAX_DIGIT1 i MAX_DIGIT0. Kako se na sedmosegmentnim displejima prikazuje samo vrednost od 1 do 6, potrebno je četiri linije signala na ulazu kola *bits2sevenseg* koji su najveće težine povezati na *GND*. Time se potpuno definiše ulaz podsistema *bits2sevenseg* i sužava opseg vrednosti koje se prikazuju na niz vrednosti od 0 do 15.



Slika 18.5: Test šema za verifikaciju ispravnosti rada sinhronog brojača na UP2 ploči

Tabela 18.2: Tabela povezanosti signala za verifikaciju rada sistema na UP2 razvojnoj ploči

<i>Signal</i>	<i>Element na UP2 ploči</i>	<i>Opis</i>
<i>SCLK</i>	pin 83	Veza sa sistemskim taktom od 25,175MHz.
<i>RESET</i>	MAX_SW1	Preko DIP prekidača obezbeđuje se signal asinhronog reseta brojača.
<i>DOZVOLA</i>	MAX_PB1	Preko tastera PB1 daje se signal kojim se dozvoljava i stopira rad brojača.
<i>SMER</i>	MAX_PB2	Preko tastera PB2 izdaje se informacija o smeru brojanja brojača.
<i>DIGIT0[6..0]</i>	MAX_DIGIT2	Prikaz nižeg nibla vrednosti brojača na desnom sedmosegmentnom displeju.
<i>DIGIT1[6..0]</i>	MAX_DIGIT1	Prikaz višeg nibla vrednosti brojača na levom sedmosegmentnom displeju. Ovaj displej će uvek prikazivati vrednost 0.

ZADATAK 19 – Detektor zadate sekvence

Realizovati sinhroni sistem (sinhronu digitalnu mrežu) za detekciju i indicaciju pojavljivanja fiksne sekvence binarnih podataka koja se dovodi na jednobitni serijski ulaz. Simbol i raspored priključaka sistema za detekciju i indicaciju (DETEKTOR) prikazan je na slici br. 19.1.



Slika 19.1: Simbol sistema za detekciju fiksne sekvence podataka

Ulazni priključci (signali) detektora sekvence su:

- *clk* – sinhronizacioni signal periode T_{clk} ;
- *rst* – signal asinhronog reseta i
- *ulaz* – jednobitna linija ulaznog signala.

Na izlazu je jednobitni signal *ind* preko kojeg se vrši indicacija detektovane sekvence.

Sekvenca podataka čije pojavljivanje sistem treba detektovati na ulazu *ulaz* je: **010**. Nakon detekcije, sistem treba da izvrši indicaciju detektovane sekvence aktiviranjem signala *ind*, tj. postavljanjem vrednosti signala *ind* na visok logički nivo ($ind='1'$) u trajanju od jedne periode sinhronizacionog *clk* signala.

Problem rešiti implementacijom sistema projektovanjem mašine stanje. Izvršiti analizu problema definisanjem stanja, formiranjem dijagrama stanja i tabele prelaza stanja. Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati mrežu AHDL opisom.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

Veoma čest zahtev pri realizaciji digitalnih sistema je implementacija podsistema za detekciju sekvence podataka fiksne ili promenjive sadržine. Ovaj problem sreće se kod implementacije različitih oblika serijske komunikacije tipa: RS232, I2C, SPI, ... Najpogodniji način rešavanja problema detekcije fiksne sekvence podataka je primenom mašine stanja odnosno konačnog automata. Rešavanje problema u digitalnoj elektronici primenom obrasca mašine stanja podrazumeva:

- analizu i razumevanje ponašanja sistema kao proces koji prolazi kroz konačan broj stanja;
- utvrđivanje stanja sistema i određivanje tipa sekvencijalne mreže na osnovu veze između trenutnog stanja sistema, ulaznih i izlaznih signala (tj. da li je mreža Murovog ili Milijevog tipa);
- utvrđivanje uslova prelaska iz stanja u stanje;
- formiranje dijagrama stanja i tabele prelaza stanja;
- implementaciju sistema postojećim obrascima za realizaciju mašine stanja u AHDL, VHDL, Verilog ...programskom jeziku.

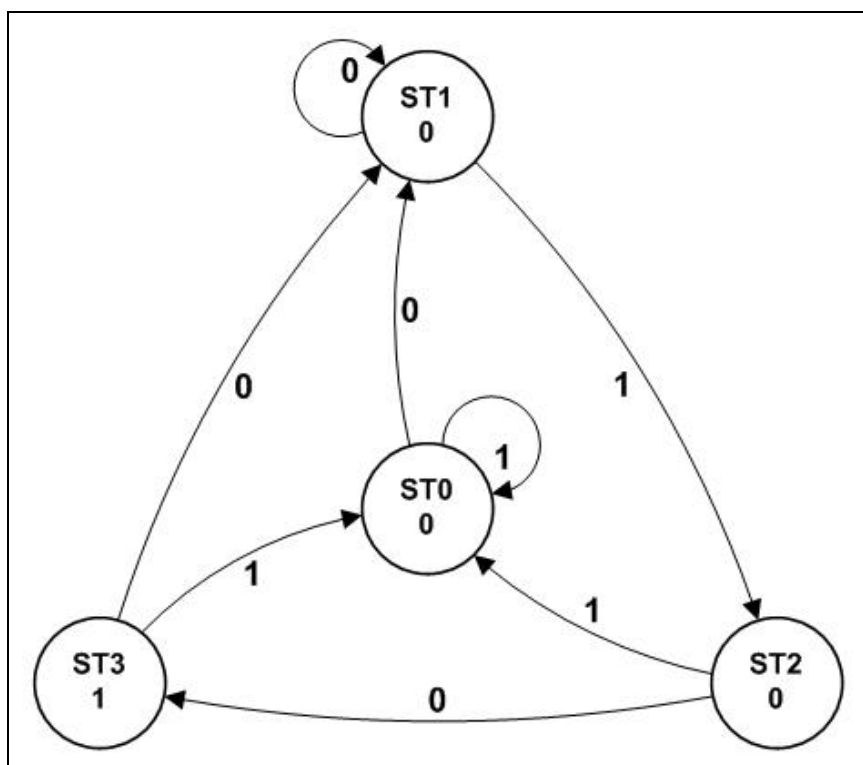
Zahtevana mreža detektora sekvence treba da prepozna tri susedna odbirka u vremenu koji zadovoljavaju uslove: da prvi i poslednji odbirak imaju vrednost **0** i drugi ima vrednost **1**. Jedan od načina detekcije fiksne sekvence je pamćenje susednih odbiraka posmatranog signala preko memorijskog elementa (najčešće pomeračkog registra) dužine koja je jednaka dužini fiksne sekvence i upoređivanje sa fiksnom sekvencom. Ukoliko se uoči podudarnost, generiše se signal indikacije. Ovaj pristup za detekciju fiksne sekvence dužine N , zahteva N memorijskih elemenata i jako kompleksnu logiku za upoređivanje zapamćenih podataka i fiksne sekvence. Mnogo jednostavnije rešenje, koje zahteva znatno manji broj memorijskih elemenata, predstavlja realizacija detektora sekvence preko mašine stanja.

Rešavanje zadatka detekcije fiksne sekvence preko mašine stanja započinje razumevanjem postupka detekcije kroz proces od $N+1$ stanja, gde je N dužina fiksne sekvence koju sistem treba detektovati. Svakom od N stanja odgovara situacija delimično i potpuno detektovane željene fiksne sekvence. Poslednje, $N+1$, stanje predstavlja stanje u kojem se sistem nalazi kada nije detektovao nijedan deo željene fiksne sekvence.

Za zadatak detekcije sekvence 010 sistem treba da sadrži 4 stanja (ST0, ST1, ST2 i ST3). Neka je ST0 stanje u kojem sistem nije detektovao nijedan deo sekvence 010. Opis stanja sistema detektora sekvence 010 dat je u tabeli br. 19.1.

Tabela 19.1: Opis stanja sistema detektora sekvence 010

<i>stanje</i>	<i>Opis stanja</i>
ST0	nije detektovan ni jedan deo sekvence 010;
ST1	detektovana podsekvenca 0;
ST2	detektovana podsekvenca 01;
ST3	stanje detekcije sekvence u celini 010.



Slika 19.2: Dijagram stanja

Preko signala *ind* vrši se indikacija detekcije fiksne sekvence. U stanju ST3 sistem je detektovao željenu fiksnu sekvencu u celini i ova činjenica se može iskoristiti za formiranje

signala indikacije izlaza *ind*. Vezivanje vrednosti izlaza za trenutno stanje sistema bez učešća vrednosti ulaznih signala, upućuje na implementaciju detektora sekvence preko mašine stanja Murovog tipa.

Mreže Murovog tipa koriste se za realizaciju detektora sekvence u najvećem broju slučajeva. U stanjima ST0, ST1 i ST2 sistem vrši indikaciju (*ind*='0') da nije detektovao, dok u stanju ST3 sistem obavlja indikaciju (*ind*='1') da je detektovao sekvencu 010. Obezbeđivanje trajanja indikacije detekcije tačno jednu periodu *clk* signala postiže se na taj način što se sistem ne zadržava u stanju ST3 duže od T_{clk} , tj. prelazi u neko drugo stanje pri sledećoj aktivnoj ivici *clk* signala.

Analizu prelaska iz stanja u stanje, u slučaju detektora sekvence, pogodno je obaviti preko dijagrama stanja. Uzimajući u obzir opis svakog stanja sistema (tabela br. 19.1) kao i analizu svih situacija pri detekciji sekvence 010, formira se dijagram stanja (slika br. 19.2). Na bazi dijagram stanja (slika br. 19.2) formira se tabela prelaza stanja (tabela br. 19.2).

Tabela 19.2: Tabela prelaza
detektora sekvence **010**

$t=t_n$			$t=t_{n+1}$
<i>stanje</i>	<i>ulaz</i>	<i>ind</i>	<i>stanje</i>
ST0	0	0	ST1
ST0	1	0	ST0
ST1	0	0	ST1
ST1	1	0	ST2
ST2	0	0	ST3
ST2	1	0	ST0
ST3	0	1	ST1
ST3	1	1	ST0

Tabela prelaza pogodna je za brz i efikasan opis sistema sa malom verovatnoćom greške pri unosu u toku opisa mašine stanja u AHDL-u. Na osnovu dijagrama stanja i tabele prelaza formiran je AHDL opis sistema za detekciju fiksne sekvence **010**.

```

SUBDESIGN detektor
(
  clk, rst, ulaz : INPUT;
  ind           : OUTPUT;
)

VARIABLE
  stanje: MACHINE
          OF BITS ( q1, q0 )
          WITH STATES ( ST0, ST1, ST2, ST3 );

BEGIN
  stanje.clk           = clk;
  stanje.reset        = rst;

  TABLE
  stanje, ulaz => stanje, ind;
% ----- %
  ST0,      0  =>  ST1,      0;
  ST0,      1  =>  ST0,      0;

```


ZADATAK 20 – Detektor zadate sekvence i mehanizam indikacije detekcije

Realizovati kolo za sinhronu detekciju sekvence bita, odnosno detektor sekvence, koji na izlazu daje signal za indikaciju nakon detekcije. Ulazni signali detektora su:

- *clk* – sinhronizacioni signal;
- *rst* – signal asinhronog reseta i
- *ulaz* – jednobitni ulazni signal podatka.

Indikacija detektovane sekvence ostvaruje se jednobitnim izlaznim signalom *ind*. Blok šema kola za detekciju sekvence prikazana je na slici br. 20.1.



Slika 20.1: Blok šema detektora sekvence

Sekvencu podataka koju detektor treba detektovati na ulazu *ulaz* je: **1010111** pri čemu se podaci iz sekvence pojavljuju čitano s leva na desno. Nakon detekcije, detektor treba da izvrši indikaciju detekcije sekvence aktiviranjem signala *ind*, tj. postavljanjem vrednosti signala *ind* na nizak logički nivo (*ind*='0') u trajanju od 2 takta sinhronizacionog *clk* signala. U toku indikacije detektovane sekvence, sistem ne treba da vrši detekciju nove sekvence.

Izvršiti analizu problema izolovanjem stanja i formiranjem dijagrama stanja. Voditi računa da postoji stanje u kojem se sistem nalazi ako nije detektovao ni jedan deo sekvence kao i to da se nakon detektovanja sledećeg bita iz sekvence prelazi u novo stanje.

Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati mrežu AHDL opisom. Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

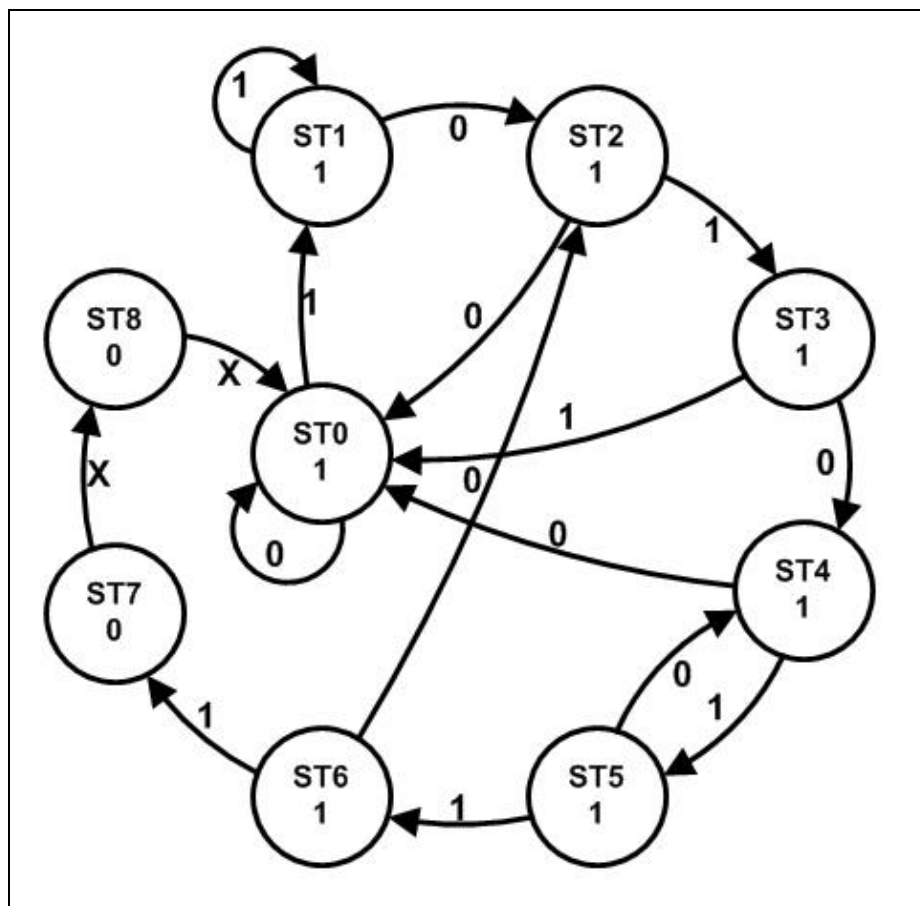
REŠENJE

Analizom zahteva utvrđuju se stanja sistema i kretanje sistema kroz stanja u toku detekcije i indikacije detektovane sekvence. Indikacija detektovane sekvence u trajanju od dva takta sinhronizacionog signala najjednostavnije se može realizovati preko dva stanja sistema. U toku ova dva stanja sistem ne vrši detekciju nove sekvence. Činjenice da sekvenca koju sistem treba da detektuje sadrži 7 bita i to da se indikacija obavlja kroz dva stanja, lako navode na zaključak da sistem treba da ima 9 stanja (npr. neka su stanja ST0, ST1 ... ST8).

Tabela 20.1: Opis stanja sistema detektora

stanje	Opis stanja
ST0	situacija kada nije detektovan ni jedan deo sekvence;
ST1	stanje detekcije podsekvence 1;
ST2	stanje detekcije podsekvence 10;
ST3	stanje detekcije podsekvence 101;
ST4	stanje detekcije podsekvence 1010;
ST5	stanje detekcije podsekvence 10101;
ST6	stanje detekcije podsekvence 101011;
ST7	stanje detekcije sekvence 1010111 i indikacije 1/2;
ST8	stanje detekcije sekvence 1010111 i indikacije 2/2;

Opis stanja sistema detektora sekvence 1010111 dat je u tabeli br. 20.1.



Slika 20.2: Dijagram stanja

Analiza prelaska iz stanja u stanje, u slučaju detektora sekvence, pogodno je obaviti preko dijagrama stanja. Uzimajući u obzir opis svakog stanja sistema (tabela br. 20.1), kao i na osnovu analize svih situacija pri detekciji sekvence 1010111 formiran je dijagram stanja (slika br. 20.2).

Na bazi dijagram stanja (slici br. 20.2) formira se tabela prelaza stanja (tabela br. 20.2).

Tabela 20.2: Tabela prelaza detektora sekvence 1010111

stanje	$t=t_n$		$t=t_{n+1}$
	ulaz	ind	stanje
ST0	0	1	ST0
ST0	1	1	ST1
ST1	0	1	ST2
ST1	1	1	ST1
ST2	0	1	ST0
ST2	1	1	ST3
ST3	0	1	ST4
ST3	1	1	ST0
ST4	0	1	ST0
ST4	1	1	ST5
ST5	0	1	ST4
ST5	1	1	ST6
ST6	0	1	ST2
ST6	1	1	ST7
ST7	0	0	ST8
ST7	1	0	ST8
ST8	0	0	ST0
ST8	1	0	ST0

Na osnovu dijagrama stanja i tabele prelaza stanja formiran je AHDL opis sistema za detekciju fiksne sekvence 1010111.

SUBDESIGN detektor

```
(
    clk, rst, ulaz  : INPUT;
    ind             : OUTPUT;
)
```

VARIABLE

```
    stanje: MACHINE
        OF BITS ( q3, q2, q1, q0 )
        WITH STATES ( ST0, ST1, ST2, ST3, ST4,
                      ST5, ST6, ST7, ST8 );
```

BEGIN

```
    stanje.clk      = clk;
    stanje.reset    = rst;
```

TABLE

stanje,	ulaz	=>	stanje,	ind;
%				%
ST0,	0	=>	ST0,	1;
ST0,	1	=>	ST1,	1;
ST1,	0	=>	ST2,	1;
ST1,	1	=>	ST1,	1;
ST2,	0	=>	ST0,	1;
ST2,	1	=>	ST3,	1;
ST3,	0	=>	ST4,	1;
ST3,	1	=>	ST0,	1;
ST4,	0	=>	ST0,	1;
ST4,	1	=>	ST5,	1;
ST5,	0	=>	ST4,	1;
ST5,	1	=>	ST6,	1;
ST6,	0	=>	ST2,	1;
ST6,	1	=>	ST7,	1;
ST7,	0	=>	ST8,	0;
ST7,	1	=>	ST8,	0;
ST8,	0	=>	ST0,	0;
ST8,	1	=>	ST0,	0;

END TABLE;

END;

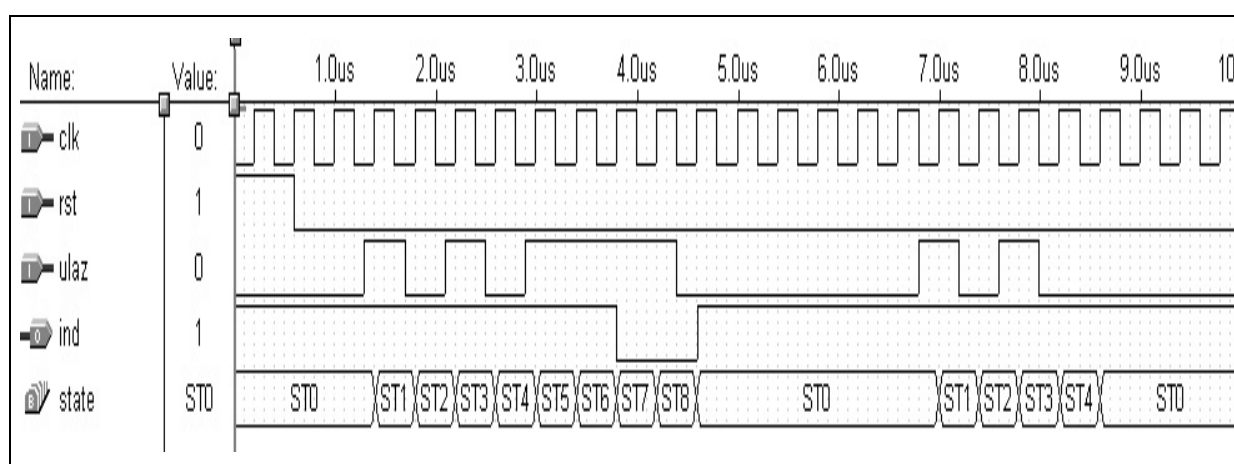
Na dijagramu stanja (iz tabele prelaza stanja ili AHDL opisa sistema) treba obratiti pažnju na sledeće prelaze:

- iz stanja S1 u S1 kada je $ulaz = '1'$;
- iz stanja S5 u S4 kada je $ulaz = '0'$ i
- iz stanja S6 u S2 kada je $ulaz = '0'$.

Prva situacija predstavlja pojavljivanje više uzastopnih jedinica ('1') na ulazu. Ova pojava može da bude situacija početka sekvence koju treba detektovati, tj. ispred pojave sekvence 1010111 može se naći proizvoljan broj jedinica.

Drugi naznačeni primer je slučaj pojave većeg broja ponovljenih podsekvenci 10 pre pojave podsekvence 111. Primer ove situacije je slučaj pojave sekvence 10101010**1010111**.

Treći slučaj obrađuje situaciju detekcije vrednosti 0 na mestu sedmog bita. Ovim se proces detekcije vraća skoro na početak, tj. u stanje S2. Primer ove pojave je povorka 10101**1010111**.



Slika 20.3: Simulacioni dijagram detektora sekvence 1010111

Jedan od simulacionih dijagrama kojim se vrši verifikacija rada sistema detektora dat je na slici br. 20.3. Od trenutka 1,4 μ sec do 3,8 μ sec pojavljuje se sekvenca 1010111 na ulaznom priključku *ulaz*. U periodu od 3,8 μ sec do 4,6 μ sec sistem vrši indicaciju detektovane sekvence. Pojava delimične sekvence 1010 javlja se od 7 μ sec do 8,6 μ sec. Zbog nepojavljivanja podsekvence 111, detektor u sledećoj periodi sinhronizacionog signala (8,6 μ sec) prelazi u stanje ST0 tj. stanje nedetektovanja ni delimične sekvence.

ZADATAK 21 – Detektor zadate sekvence i mehanizam indicacije detekcije

Realizovati detektor sekvence, tj. sinhrono kolo za detekciju fiksno definisane sekvence bita i indicaciju nakon detekcije. Od ulaznih signala prisutni su:

- *clk* – sinhronizacioni signal (perioda *clk* signala je T_{clk});
- *rst* – signal asinhronog resetiranja i
- *podatak* – jednobitni ulazni signal.

Na izlazu sistema detekcije je jednobitni signal *ind* kojim se vrši indicacija detektovane sekvence.

Sekvenca na ulazu *podatak* koju sistem treba detektovati je **01011011** pri čemu se podaci iz sekvence pojavljuju redom čitano s leva na desno. Nakon detekcije, sistem treba da obavi indicaciju da je detektovao sekvencu aktiviranjem signala *ind*, tj. postavljanjem vrednosti signala *ind* na visok logički nivo ($ind = '1'$) u trajanju od pet perioda *clk* signala. U toku indicacije

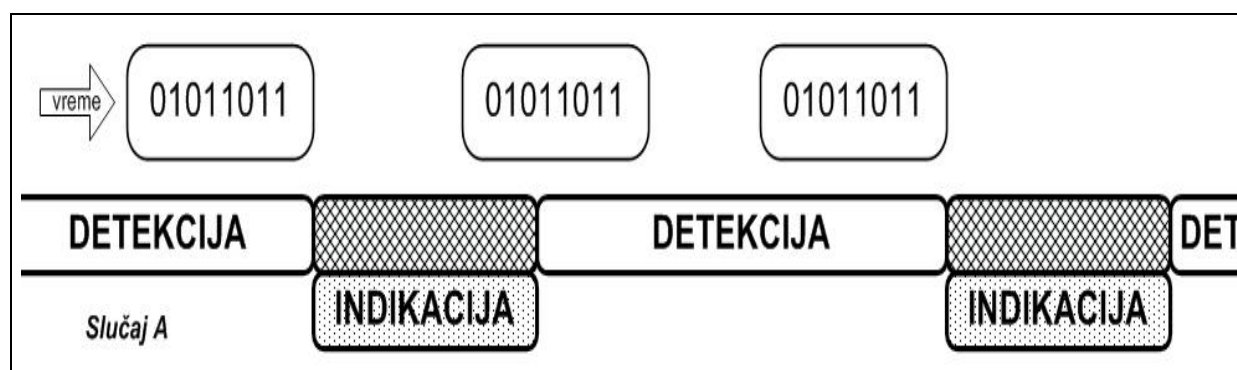
detektovane sekvence, detektor treba da nastavi proces detekcije, tj. ne sme postojati vreme kada sistem ne obavlja detekciju.

Izvršiti analizu problema definisanjem stanja, formiranjem dijagrama stanja i tabele prelaza stanja.

Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati mrežu AHDL opisom. Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

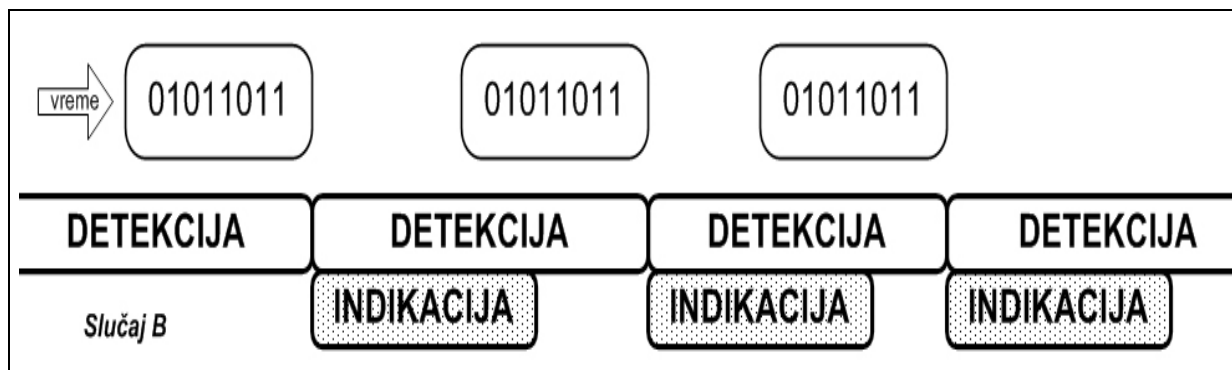
Osnovna razlika između zahteva koje treba da ispuni detektor sekvence iz ovog i prethodnog zadatka (zadatak br. 20) je u tome što se u ovim zadatkom traži da sistem obavlja detekciju sekvence 01011011 sve vreme, tj. i u toku procesa indikacije prethodno detektovane sekvence. Razlika u realizacijama sistema predstavljena je preko vremenskih dijagrama rada sistema u oba slučaja koji su prikazani na slici br. 21.1 za zahtev iz zadatka br. 20 i na slici br.21.2 za zahtev iz ovog zadatka.



Slika 21.1: Vremenski dijagram procesa detekcije i indikacije sekvence 01011011 za slučaj realizacije sistema na način koji je zahtevan u zadatku br. 20

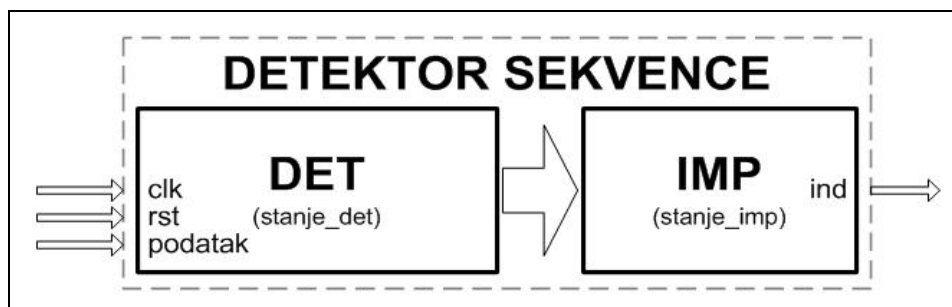
Na slici br.21.1 dat je vremenski dijagram rada sistema detekcije realizovan na način koji je zahtevan u zadatku br 20, tj. proces detekcije sekvence pri čemu sistem u toku indikacije ne obavlja proveru podataka na ulazu i samim tim eventualnu detekciju nove sekvence. Ovakav način realizacije detektora sekvence može dovesti do toga da ne bude detektovano pojavljivanje sekvenci koje se javljaju jako blisko u vremenu, jedna iza druge. Na slici br. 20.1, iako je naznačeno da se pojavlju 3 sekvence podataka 01011011 jedna iz druge, sistem uspeva da detektuje samo dva pojavljivanja, tj. prvo i treće. Drugo pojavljivanje sekvence 01011011 ostaje neprimećeno zato što se početak druge sekvence poklapa sa procesom indikacije detekcije prve sekvence.

Bolji način realizacije detektora sekvence predstavljen je u okviru ovog zadatka. Skica vremenskog dijagrama procesa detekcije i indikacije sekvence na ovaj način demonstriran je na slici br. 21.2. Pojavljivanje sekvenci 01011011 u vremenu isto je kao u prethodno analiziranom slučaju. Na dijagramu se može uočiti da sistem detektuje i vrši indikaciju sva tri pojavljivanja sekvence i da se detekcija obavlja sve vreme rada sistema. Takođe se može zaključiti da se procesi detekcije i indikacije obavljaju konkurentno, tj. u toku indikacije oba procesa se obavljaju paralelno.



Slika 21.2: Vremenski dijagram procesa detekcije i indikacije sekvence 01011011 za slučaj realizacije sistema na način zahtevan ovim zadatkom (paralelna detekcija i indikacija)

Jedan od načina realizacije postavljenog problema je implementacija sistema detekcije sekvence preko dva podsistema DET i IMP. Organizacija i blok dijagram detektora sekvence prikazan je na slici br. 21.3. Prvi deo tj. podsistem DET obavlja funkciju neprekidne detekcije zadate sekvence. Drugi deo, podsistem IMP, je sekvencijalna mreža koja generiše specifičirani signal na izlazu nakon detekcije. U konkretnom slučaju IMP generiše impuls trajanja $5T_{clk}$ nakon konstatovanja detektovane sekvence od strane podsistema DEK.

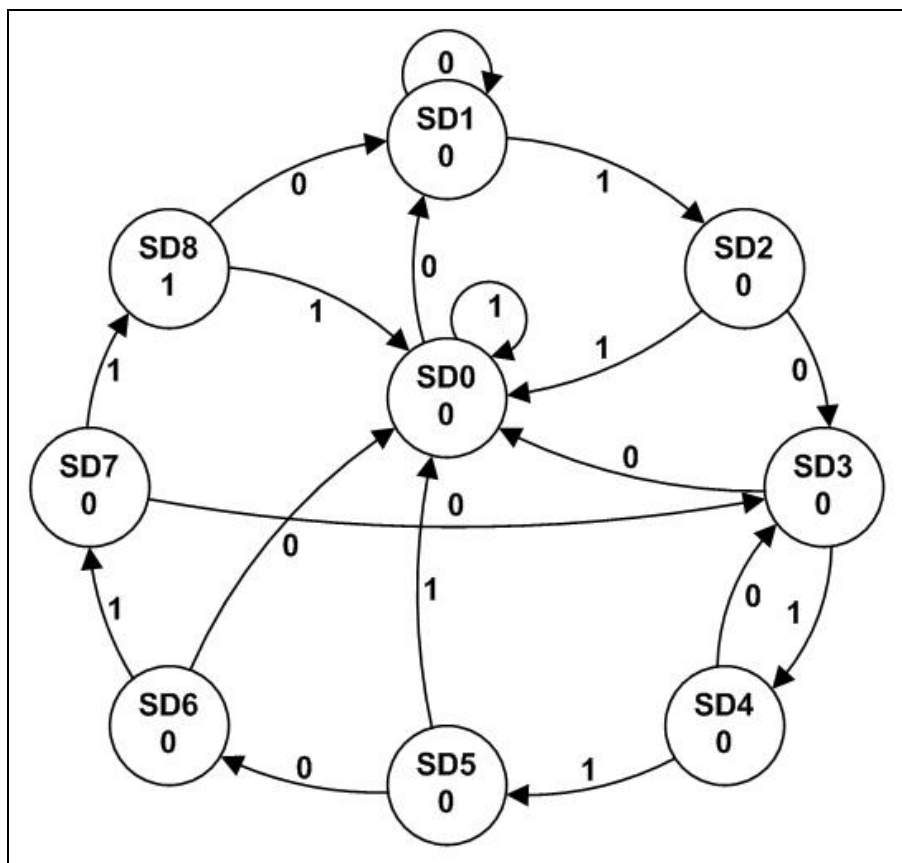


Slika 21.3: Organizacija sistema konkurentne detekcije i indikacije pojavljivanja zadate sekvence bez pauze u procesu detekcije

Implementacija sistema preko konkurentnog izvršavanja detekcije i indikacije obavlja se realizacijom dve sekvencijalne mreže, tj. dve mašine stanja, DET i IMP.

Sekvencijalna mreža DET, tj. mašina stanja detekcije, obavlja detekciju pojavljivanja podataka 01011011 na serijskom ulazu *podatak*. Analiza zahteva i stanja podsistema DET obavlja se na sličan način koji je opisan u zadatku br. 19 i 20.

U procesu detekcije sekvence postoje osam stanja u okviru kojih je izvršena parcijalna i potpuna detekcija sekvence i stanje kada sistem nije detektovao ni jedan deo sekvence. Ovih devet stanja su: SD0, SD1 ... SD8. Stanju SD0 odgovara situacija kada nije detektovan ni jedan deo sekvence 01011011. Stanja SD1, SD2 do SD7 predstavljaju stanja detekcije delimične sekvence. Na primer SD4 je stanje u kojem je detektovan deo sekvence 0101. Ukoliko se zadata sekvenca pojavljuje na ulazu *podatak* sistem se kreće od stanja SD0 preko stanja SD1, SD2, ... SD7 do stanja indikacije SD8 tj. detekcije poslednjeg bita iz sekvence. Iz stanja SD8 sistem prelazi u stanje SD0 ili SD1 i vrši indikaciju pojavljivanja sekvence. Ova indikacija traje jednu periodu *clk* signala i služi za aktiviranje podsistema IMP koji nastavlja proces indikacije u trajanju od $5T_{clk}$ preko serijskog izlaza *ind*.



Slika 21.4: Dijagram stanja podsistema DET – podsistem detekcije sekvence 01011011

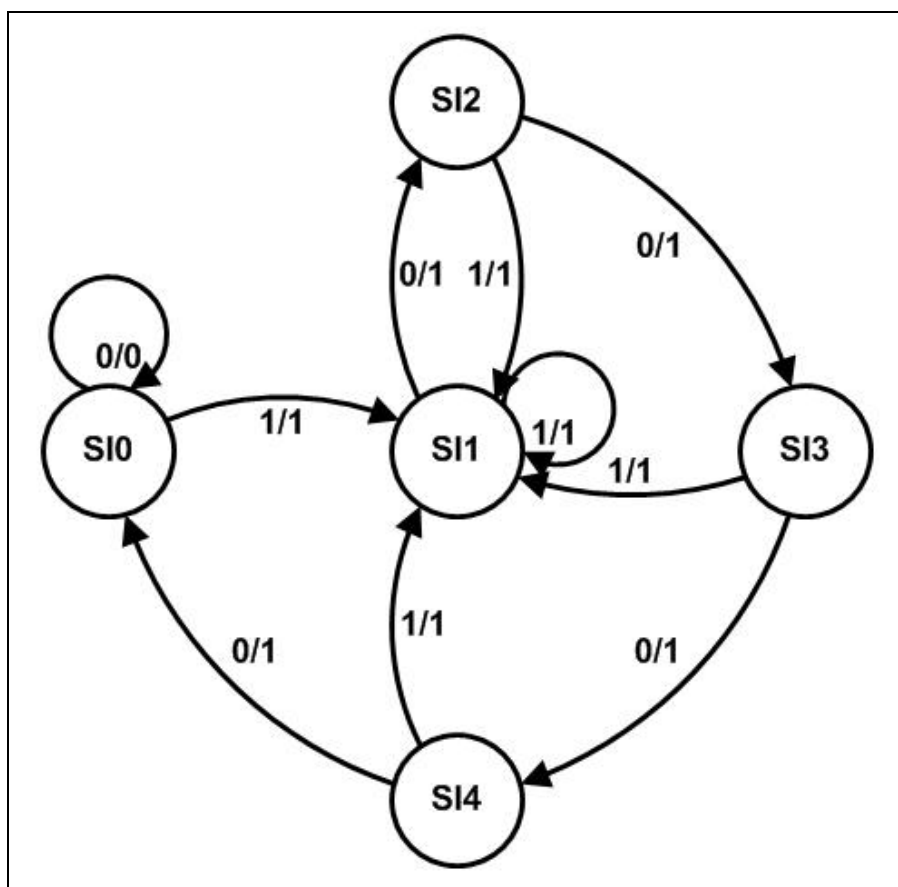
Dijagram stanja podsistema DET, tj. mašine stanja detektora sekvence, prikazan je na slici br. 21.4. Izlaz podsistema DET nazvan je *ind_imp* i predstavlja unutrašnji signal sistema detektora u celini.

Tabela 21.4: Tabela prelaza podsistema DET

<i>state_det</i>	<i>t=t_n</i>		<i>t=t_{n+1}</i>
	<i>podatak</i>	<i>ind_imp</i>	<i>state_det</i>
SD0	0	0	SD1
SD0	1	0	SD0
SD1	0	0	SD1
SD1	1	0	SD2
SD2	0	0	SD3
SD2	1	0	SD0
SD3	0	0	SD0
SD3	1	0	SD4
SD4	0	0	SD3
SD4	1	0	SD5
SD5	0	0	SD6
SD5	1	0	SD0
SD6	0	0	SD0
SD6	1	0	SD7
SD7	0	0	SD3
SD7	1	0	SD8
SD8	0	1	SD1
SD8	1	1	SD0

Podsistem DET analiziran je kao sekvencijalna mreža Murovog tipa pri čemu se vrednost signala na izlazu podsistema, signal *ind_imp*, naznačava u polju simbola stanja. Impuls indikacije detektovane sekvence od strane podsistema DET ostvaruje se preko signala *ind_imp* u trajanju od jedne periode *clk* signala. Na dijagramu stanja uočavaju se prelazi iz stanja SD4 u SD3 kao i iz SD7 u SD3. Ovi prelazi obrađuju pojavljivanje lažnog početka sekvence tj. situacija 010101011011 i 0101101011011. Tabela prelaza podsistema DET koja odgovara dijagramu stanja sa slike br. 21.4 prikazana je u tabeli br. 21.1.

Podsistem generisanja signala indikacije detektovane sekvence 01011011 naznačen je blokom IMP koji je, takođe, realizovan preko sekvencijalne mreže tj. mašine stanja. U cilju obezbeđivanja trenutne indikacije sistema nakon detekcije sekvence od strane podsistema DET, IMP je analiziran i realizovan kao sekvencijalna mreža Milijevog tipa. Sekvencijalne mreže Milijevog tipa karakterišu se time da vrednost signala na izlazu zavisi, osim od trenutnog stanja sistema zavisi i od vrednosti signala na ulazu. Kod mašina stanja Milijevog tipa signal na izlazu može se promeniti asinhrono od *clk* signala. Ova osobina mreža Milijevog tipa iskorišćena je u cilju obezbeđivanja trenutne reakcije podsistema IMP nakon detekcije sekvence od strane podsistema DET. Pojavom visokog nivoa na liniji signal *ind_imp*, podsistem IMP odmah (asinhrono) generiše visok nivo signala na izlazu *ind* i nastavlja sinhrono da se kreće kroz stanja podsistema. IMP je realizovan kao sekvencijalna mreža sa pet stanja SI0, SI1, SI2, SI3, SI4 i SI5. U stanju SI0 IMP drži neaktivan signal izlaza *ind* ukoliko je vrednost signala na ulazu *ind_imp*='0'. U svim drugim stanjima podsistema IMP i svim situacijama vrednosti signala na ulazu, podsistem IMP generiše *ind*='1'. Dijagram stanja podsistema IMP prikazan je na slici br. 21.5.



Slika 21.5: Dijagram stanja podsistema IMP – podsistem generisanja signala indikacije *ind*

Podsistem IMP obavlja funkciju generatora impulsa $ind='1'$ trajanja najkraće četiri periode clk signala. Specifičnost ovog podsistema je u tome što ako se signal generisanja impulsa ind_imp pojavi sinhrono sa clk signalom i traje jednu periodu clk signala, generator će generisati impuls trajanja $5T_{clk}$ odmah nakon što ind_imp dobije vrednost 1. Ukoliko signal iniciranja rada generatora impulsa IMP (ind_imp) traje duže od jedne periode clk signala, impuls na izlazu ind trajaće četiri periode T_{clk} i nakon prelasla ind_imp na neaktivan nivo. Tabela prelaza podsistema IMP koja odgovara dijagramu stanja sa slike br. 21.5 prikazana je u tabeli br. 21.2.

Tabela 21.2: Tabela prelaza podsistema IMP

$t=t_n$			$t=t_{n+1}$
<i>state_imp</i>	<i>ind_imp</i>	<i>ind</i>	<i>state_imp</i>
SI0	0	0	SI0
SI0	1	1	SI1
SI1	0	1	SI2
SI1	1	1	SI1
SI2	0	1	SI3
SI2	1	1	SI1
SI3	0	1	SI4
SI3	1	1	SI1
SI4	0	1	SI0
SI4	1	1	SI1

Na bazi prethodno opisanih dijagrama stanja i prelaza za podsisteme DET i IMP obavljena je implementacija sistema detekcije u celini. AHDL opis sistema detekcije dat je u nastavku:

```

SUBDESIGN detektor
(
  clk, rst : INPUT;
  podatak  : INPUT;
  ind       : OUTPUT;
)

VARIABLE
  stanje_det : MACHINE
    OF BITS ( qd3, qd2, qd1, qd0 )
    WITH STATES ( SD0, SD1, SD2, SD3, SD4,
                  SD5, SD6, SD7, SD8 );

  stanje_imp : MACHINE
    OF BITS ( qi2, qi1, qi0 )
    WITH STATES ( SI0, SI1, SI2, SI3, SI4 );

  ind_imp : WIRE;

BEGIN
  stanje_det.clk  = clk;
  stanje_det.reset = rst;

TABLE

```

```

    stanje_det,podatak => ind_imp, stanje_det;
% ----- %
SD0, 0 => 0, SD1;
SD0, 1 => 0, SD0;
SD1, 0 => 0, SD1;
SD1, 1 => 0, SD2;
SD2, 0 => 0, SD3;
SD2, 1 => 0, SD0;
SD3, 0 => 0, SD0;
SD3, 1 => 0, SD4;
SD4, 0 => 0, SD3;
SD4, 1 => 0, SD5;
SD5, 0 => 0, SD6;
SD5, 1 => 0, SD0;
SD6, 0 => 0, SD0;
SD6, 1 => 0, SD7;
SD7, 0 => 0, SD3;
SD7, 1 => 0, SD8;
SD8, 0 => 1, SD1;
SD8, 1 => 1, SD0;
END TABLE;

stanje_imp.clk = clk;
stanje_imp.reset = rst;
TABLE
stanje_imp,ind_imp => ind, stanje_imp;
% ----- %
SI0, 0 => 0, SI0;
SI0, 1 => 1, SI1;
SI1, 0 => 1, SI2;
SI1, 1 => 1, SI1;
SI2, 0 => 1, SI3;
SI2, 1 => 1, SI1;
SI3, 0 => 1, SI4;
SI3, 1 => 1, SI1;
SI4, 0 => 1, SI0;
SI4, 1 => 1, SI1;
END TABLE;

END;

```

AHDL jezik dozvoljava implementaciju više sekvencijalnih sklopova tj. mašina stanja u okviru jednog AHDL opisa, tj. TDF datoteke. Mašina stanja podsistema DET opisan je preko obrasca mašine stanja *stanje_det* a podsistem IMP preko *stanje_imp*. Izlaz podsistema DET, tj. signal *ind_imp*, predstavlja ulaz za podsistem IMP. Signal sprege podsistema DET i IMP definisan je ključnom reči **WIRE** u okviru sekcije **VARIABLE**:

```

...
VARIABLE
...
    ind_imp : WIRE;
...

```

Za obe mašine stanja (*stanje_det* i *stanje_imp*) potrebno je specificirati sinhronizacioni signal i asinhroni reset obrasca mašine stanja.

```

...
stanje_det : MACHINE
  OF BITS ( qd3, qd2, qd1, qd0 )
  WITH STATES ( SD0, SD1, SD2, SD3, SD4,
               SD5, SD6, SD7, SD8 );

stanje_imp : MACHINE
  OF BITS ( qi2, qi1, qi0 )
  WITH STATES ( SI0, SI1, SI2, SI3, SI4 );

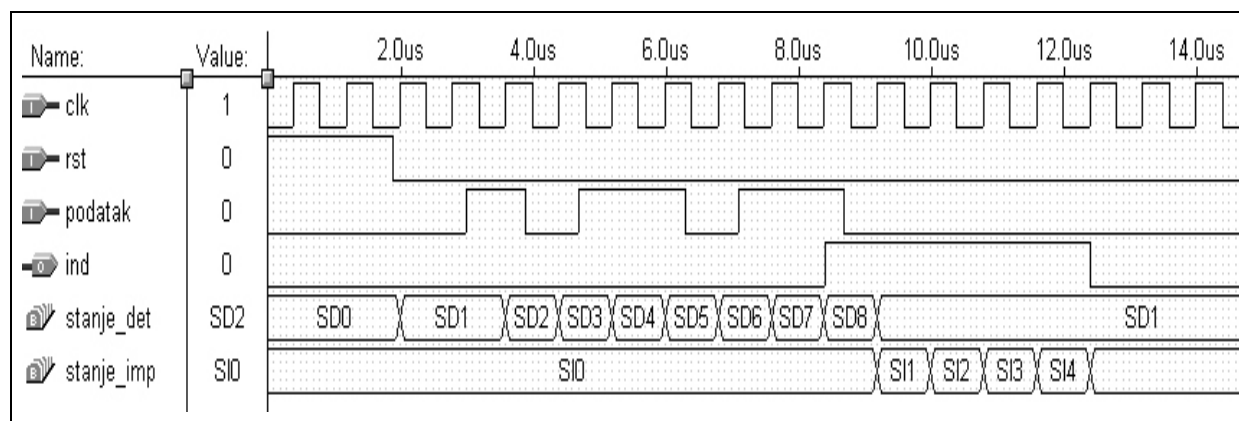
...
stanje_imp.clk = clk;
stanje_imp.reset = rst;

...
stanje_imp.clk = clk;
stanje_imp.reset = rst;

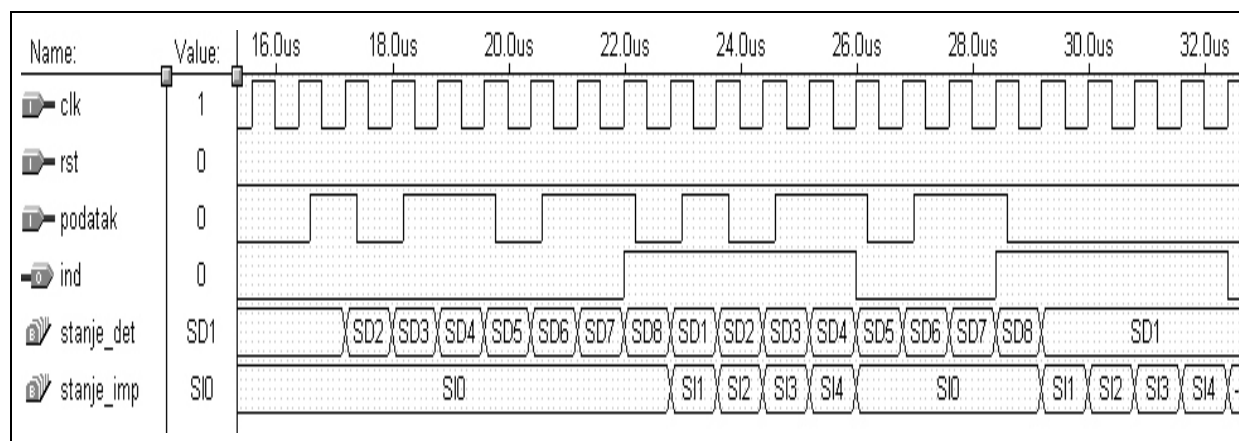
...

```

Na slici br. 21.6 dat je simulacioni dijagram provere rada detektora sekvence u slučaju pojavljivanja podataka 01011011 na ulazu *podatak*. Na simulacionom dijagramu može se pratiti kretanje sistema kroz stanja obe sekvencijalne mreže mašina stanja DET i IMP.



Slika 21.6: Simulacioni dijagram detektora sekvence 01011011



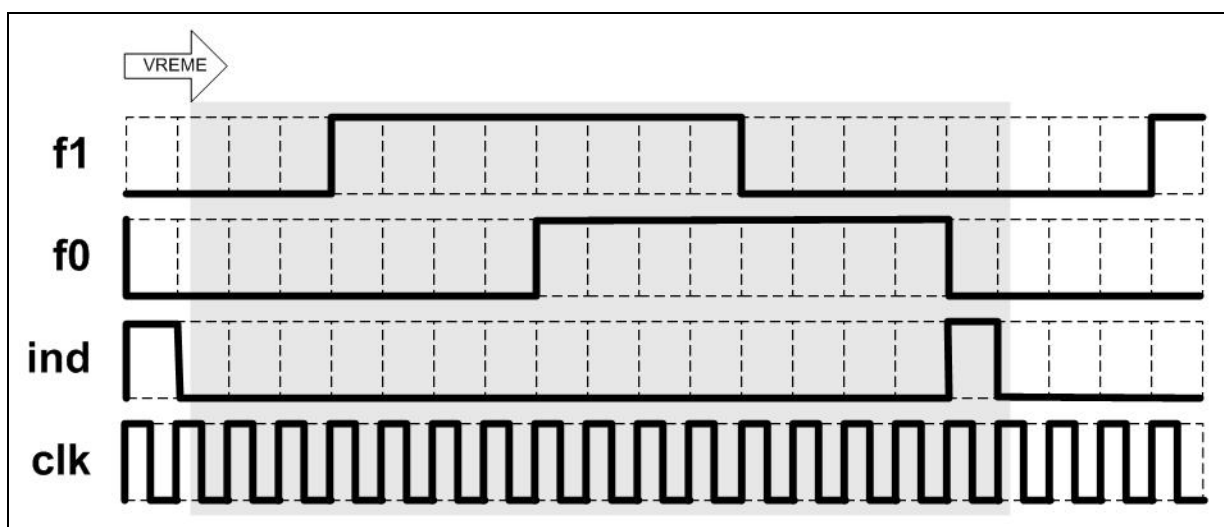
Slika 21.7: Simulacioni dijagram detektora sekvence u situaciji detekcije dve uzastopne i vezane sekvence 01011011

Na slici br. 21.7 prikazan je simulacioni dijagram detekcije dve vezane sekvence 01011011 iz niza podataka na ulazu ...0101101101011011... Na vremenskom dijagramu uočava se konkurentan rad obe mašine stanja (*stanje_det* i *stanje_imp*) i generisanje impulsa *ind* već pri ulasku podsistema DET u stanje SD8, tj. nakon detekcije poslednjeg bita iz sekvence. Trajanje impulsa *ind* traje $5T_{clk}$.

ZADATAK 22 – Sistem za obradu signala sa optičkog enkodera

Realizovati sistem sprege sa inkrementalnim optičkim enkoderom. Sistem na osnovu signala dobijenih sa enkodera treba da vrši detekciju kretanja osovine optičkog enkodera samo pri jednom smeru rotacije.

U aplikaciji se koristi inkrementalni optički enkoder sa dva para fotoelemenata A i B i pridruženim digitalnim signalima *f0* i *f1*. Pri konstantnom okretanju osovine optičkog enkodera vremenski dijagram digitalnih signala *f1* i *f0* predstavljaju simetrične povorke impulsa koje su pomerene u vremenu za 45° . Inkrementalni optički enkoder karakteriše se rezolucijom diska od 400 ppr. Maksimalna brzina okretanja osovine koju sistem treba da procesira je 1500 obr/min. Smer okretanja osovine optičkog enkodera koji sistem treba da detektuje karakteriše se time da impulsi signala *f0* kasne za *f1*.



Slika 22.1: Vremenski dijagram detekcije i indikacije okretanja osovine inkrementalnog optičkog enkodera u smeru koji se treba detektovati

Potrebno je obezbediti da sistem sprege za svaku kombinaciju vrednosti signala *f1* i *f0* pri maksimalnoj brzini okretanja osovine izvrši odabiranje i procesiranje na bazi najmanje dva odbirka. Na slici br. 22.1 dat je vremenski dijagram obrade sistema pri okretanju osovine enkodera u smeru koji treba detektovati. Pored signala sa fotodetektora *f1* i *f0*, sinhronizacionog signala *clk*, na dijagramu je naznačen i signal indikacije *ind*.

Sistem treba da podržava zahteve koji su prethodno navedeni u pogledu rezulucije, broja odabiranja i maksimalne brzine okretanja osovine optičkog enkodera.

Ulazni signali detektora su:

- *clk* – sinhronizacioni signal;
- *rst* – signal asinhronog reseta i
- *f0* i *f1* – dva jednobitna ulazna signala sa fotodetektora A i B inkrementalnog optičkog enkodera.



Slika 22.2: Simbol sistema sprege sa inkrementalnoim optičkim enkoderom

Izlazni signal sistema je jednobitni signal *ind* kojim se vrši indikacija detekcije zahtevanog smera okretanja osovine optičkog enkodera. Simbol sistema prikazana je na slici br. 22.2.

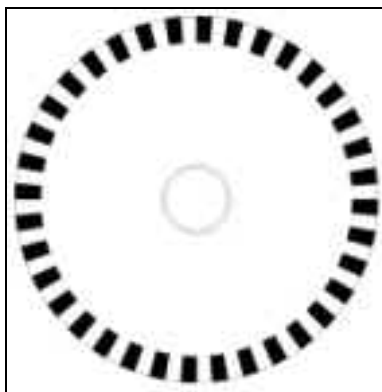
Na osnovu postavljenih zahteva:

- Izvršiti procenu minimalne učestanosti signala *clk* pri kojoj sistem obavlja pravilan rad. Projektovati sistem koji će raditi pri minimalnoj učestanosti sinhronizacionog signala *clk*.
- Obaviti indikaciju zahtevanog smera okretanja osovine optičkog enkodera preko signala *ind* u trajanju od jedne periode sinhronizacionog signala *clk* neposredno nakon detekcije pravilnog oblika signala *f0* i *f1* za zahtevani smer okretanja.
- Izvršiti analizu problema definisanjem stanja i formiranjem dijagrama stanja.
- Primenom softverskog paketa za rad sa programabilnim logičkim kolima firme ALTERA realizovati mrežu AHDL opisom.
- Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

Inkrementalni optički enkoder – teorijski uvod

Optički enkoder je elektromehanički uređaj koji se primenjuje za merenje pozicije i brzine kretanja pri rotacionom ili translatornom kretanju objekta. Najzastupljenija varijanta optičkog enkodera služi za merenje pozicije i brzine okretanja osovine. Takođe, postoji varijanta uređaja kojim se meri translatorno pomeranje objekta. Optički enkoder se odlikuje dobrom preciznošću, visokom pouzdanošću, i relativno niskom cenom, zbog čega nalazi primenu u različitim sistemima.

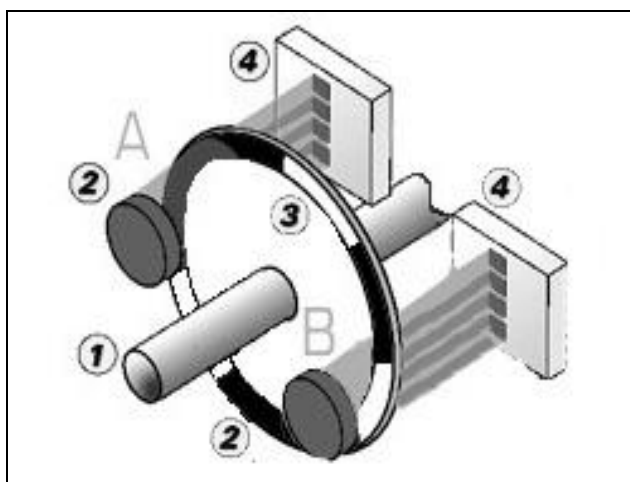


Slika 22.3: Raspored prozirnih i neprozirnih segmenata na obrtnom disku inkrementalnog optičkog enkodera

Najjednostavniji tip optičkog enkodera je inkrementalni enkoder koji služi za određivanje pozicije, brzine i smeru obrtanja osovine. Obrtni disk kod ovih enkodera (slika br. 22.3) sastoji se od jedne trake koja je izdeljena na prozirne i neprozirne delove. Broj prozirnih i neprozirnih segmenata određuje rezoluciju diska ili broj impulsa po jednom obrtaju (**pulses per revolution = ppr**). Na primer, ako je rezolucija diska 200 ppr tada će na njemu postojati 200 odvojenih linija.

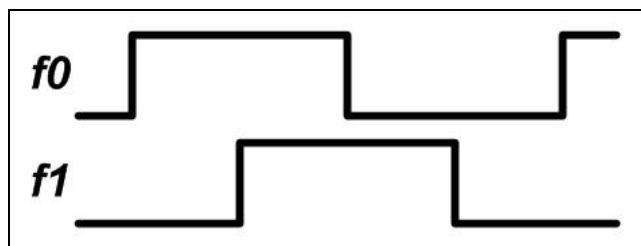
Sa jednom trakom na obrtnom disku i jednim izvorom svetlosti, inkrementalni enkoder na električnom izlazu može dati logičku jedinicu ili logičku nulu signala. Određivanje brzine je direktno proporcionalno promeni logičkog stanja na izlazu enkodera.

Određivanje smeru okretanja osovine postiže se preko dva izvora svetlosti i dva detektora (fotoelementi A i B na slici br. 22.4). Sistem sprege sa optičkim enkoderom prati promene oba signala f_0 i f_1 sa fotoelementa A i B i detektuje koji se signal prvi pojavljuje (slika br. 22.4) tj. u kakvom su odnosi signali f_0 i f_1 .

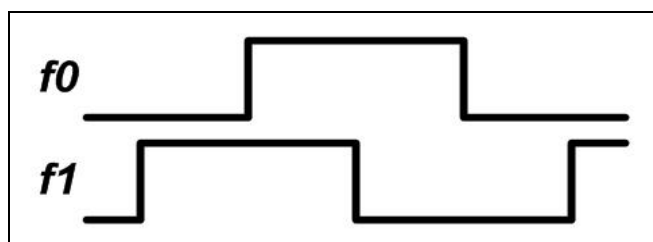


Slika 22.4: Inkrementalni optički enkoder sa dva para svetlosnih izvora i detektora (1 - osovina, 2 A i B - svetlosni izvori, 3 – obrtni disk, 4 – detektori A i B)

Kao što se uočava na slici br. 22.4, prisutna su dva signala f_0 i f_1 koji potiču sa senzora na naznačenim pozicijama A i B. Na slici br. 22.5 prikazan je oblik signala na izlazu inkrementalnog enkodera koji ima dva para fotodetektora. U ovom primeru (slika br. 22.5) uočava se da signal f_1 kasni za signalom f_0 . Okretanje osovine u suprotnom smeru prouzrokovalo bi generisanje signala f_0 i f_1 pri čemu f_0 kasni za signalom f_1 (slika br. 22.6).



Slika 22.5: Sekvenca signala f_0 i f_1 sa fotodetektora A i B pri čemu f_0 prednjači f_1



Slika 22.6: Sekvenca signala $f0$ i $f1$ sa fotodetektora A i B pri čemu $f0$ kasni za $f1$

Iako je konfiguracija inkrementalnog enkodera veoma jednostavna, postoje i nedostaci, kao što su javljanje gličeva kod signala $f0$ i $f1$ i mogućnost propuštanja detekcije impulsa.

Rešenje zadatka

a) Najpre treba odrediti učestanost rada sistema, tj. minimalni takt signala clk pri kojem sistem sprege radi ispravno.

Minimalna čestanost rada sistema ($F_{clk,MIN}$) određuje se za slučaj kada sistem radi pri maksimalnoj periodi impulsa (F_{MAX}) ulaznih signala $f0$ i $f1$ tj. pri najvećoj brzini okretanja osovine optičkog enkodera.

Maksimalna učestanost signala $f0$ i $f1$ (F_{MAX}) proračunava se na osnovu podataka o maksimalnoj ugaonoj brzini (ω_{MAX} [obr/sec]) osovine optičkog enkodera koju sistem treba da obradi kao i rezoluciji inkrementalnog optičkog enkodera (μ [imp/obr]).

Formula koja određuje maksimalnu učestanost (F_{MAX}) signala $f0$ i $f1$ je:

$$F_{MAX} = \omega_{MAX} [\text{obr/sec}] \cdot \mu [\text{imp/obr}].$$

Za podatke iz zadatka F_{MAX} se određuje iz relacija:

$$F_{MAX} = \omega_{MAX} [\text{obr/sec}] \cdot \mu [\text{imp/obr}] = \frac{\omega_{MAX} [\text{obr/min}]}{60 [\text{sec/min}]} \cdot \mu [\text{imp/obr}] =$$

$$\frac{1500 [\text{obr/min}]}{60 [\text{sec/min}]} \cdot 400 [\text{imp/obr}] = 10 \text{kHz}$$

Na osnovu proračunate vrednosti $F_{MAX} = 10 \text{kHz}$ i ako imamo u vidu:

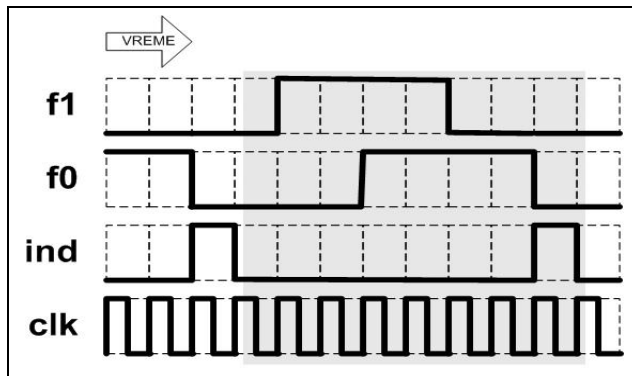
- da se u toku jedne kombinacije vrednosti $f0$ i $f1$ treba vršiti odabiranje najmanje dva puta kao i to
- da postoji 4 različite kombinacije vrednosti signala $f0$ i $f1$ (00, 01, 10 i 11),

$F_{clk,MIN}$ se računa preko formule:

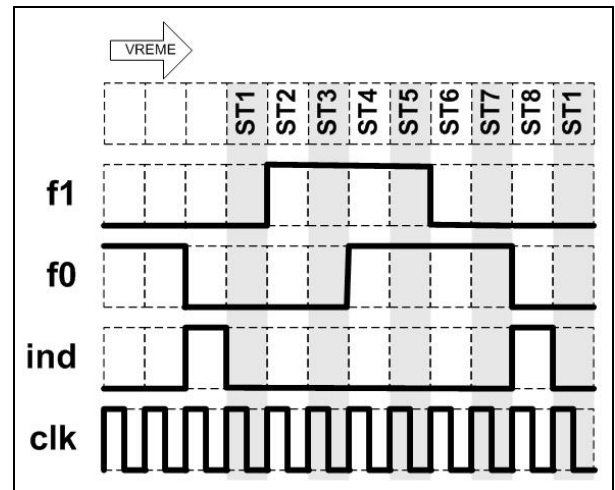
$$F_{clk,MIN} = 4 \cdot 2 \cdot F_{MAX} = 8 \cdot F_{MAX}$$

i za postavljeni zadatak ima vrednost $F_{clk,MIN} = 8 \cdot F_{MAX} = 8 \cdot 10 \text{kHz} = 80 \text{kHz}$.

b) Da bi se izvršila detekcija smera okretanja osovine optičkog enkodera, posmatra se vremenski dijagram na slici br. 22.7. Na ovoj slici prikazan je vremenski oblik signala $f1$, $f0$, ind i clk za slučaj okretanja osovine optičkog enkodera maksimalnom dozvoljenom brzinom $\omega_{MAX} = 1500$ obr/min pri učestanosti rada sistema $F_{clk,MIN}$. Za detekciju zahtevanog smera okretanja osovine optičkog enkodera potrebno je da se na ulazima $f1$ i $f0$ pojave parovi signala $f1f0$ u sledećem redosledu $00 \rightarrow 10 \rightarrow 11 \rightarrow 01$. Svaka kombinacija signala $f1f0$ treba da traje najmanje dve periode clk signala. Pored stanja za praćenje sekvence pojavljivanja i dužine trajanja parova signala $f1f0$, sistem treba da sadrži i stanje u kojem će se sistem naći ukoliko utvrdi da je došlo do neke nelogičnosti u procesu detekcije. Ovakva analiza dovodi do zaključka da sistem mora da ima 9 stanja.



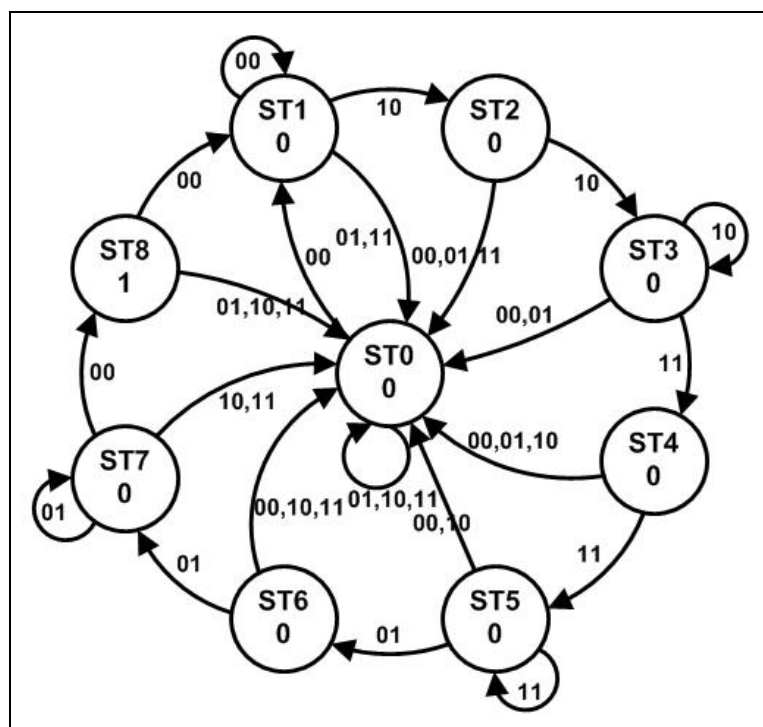
Slika 22.7: Vremenski dijagram detekcije zahtevanog smera optičkog enkodera pri maksimalnoj brzini okretanja osovine (1500 obr/min)



Slika 22.8: Specifikacija stanja pri detekciji smera okretanja osovine optičkog enkodera u slučaju maksimalne brzine okretanja (1500 obr/min)

Tabela 22.1: Opis stanja sistema detektora smera okretanja osovine optičkog enkodera

stanje	Opis stanja
ST0	stanje neregularne detekcija;
ST1	drugo i svako sledeće pojavljivanje kombinacije $f1f0=00$;
ST2	prvo pojavljivanje kombinacije $f1f0=10$;
ST3	drugo i svako sledeće pojavljivanje kombinacije $f1f0=10$;
ST4	prvo pojavljivanje kombinacije $f1f0=11$;
ST5	drugo i svako sledeće pojavljivanje kombinacije $f1f0=11$;
ST6	prvo pojavljivanje kombinacije $f1f0=01$;
ST7	drugo i svako sledeće pojavljivanje kombinacije $f1f0=01$;
ST8	prvo pojavljivanje kombinacije $f1f0=00$ i detekcija smera.



Slika 22.9: Dijagram stanja

Spisak i opis stanja dat je u tabeli br. 22.1 a njihova interpretacija je predstavljena na slici br. 22.8.

c) Analizu prelaska iz stanja u stanje, u slučaju detektora smera okretanja osovine optičkog enkodera, pogodno je analizirati preko dijagrama stanja. Uzimajući u obzir opis svakog stanja sistema (tabela br. 22.1) kao i analizu svih situacija formira se dijagram stanja (slika br. 22.9).

Tabela 22.2: Tabela prelaza detektora smera okretanja osovine optičkog enkodera

$t=t_n$				$t=t_{n+1}$
stanje	f1	f0	ind	stanje
ST0	0	0	0	ST1
	0	1	0	ST0
	1	0	0	ST0
	1	1	0	ST0
ST1	0	0	0	ST1
	0	1	0	ST0
	1	0	0	ST2
	1	1	0	ST0
ST2	0	0	0	ST0
	0	1	0	ST0
	1	0	0	ST3
	1	1	0	ST0
ST3	0	0	0	ST0
	0	1	0	ST0
	1	0	0	ST3
	1	1	0	ST4

$t=t_n$				$t=t_{n+1}$
stanje	f1	f0	ind	stanje
ST4	0	0	0	ST0
	0	1	0	ST0
	1	0	0	ST0
	1	1	0	ST5
ST5	0	0	0	ST0
	0	1	0	ST6
	1	0	0	ST0
	1	1	0	ST5
ST6	0	0	0	ST0
	0	1	0	ST7
	1	0	0	ST0
	1	1	0	ST0
ST7	0	0	0	ST8
	0	1	0	ST7
	1	0	0	ST0
	1	1	0	ST0
ST8	0	0	1	ST1
	0	1	1	ST0
	1	0	1	ST0
	1	1	1	ST0

Na bazi dijagram stanja (slici br. 22.9) formira se tabela prelaza stanja (tabela br. 22.2). Na osnovu dijagrama stanja i tabele prelaza stanja formiran je AHDL opis sistema.

SUBDESIGN detektor

```
(
  clk, rst : INPUT;
  f1, f0   : INPUT;
  ind      : OUTPUT;
)
```

VARIABLE

```
  stanje : MACHINE
           OF BITS (q3, q2, q1, q0)
           WITH STATES (ST0, ST1, ST2, ST3, ST4, ST5, ST6, ST7, ST8);
```

BEGIN

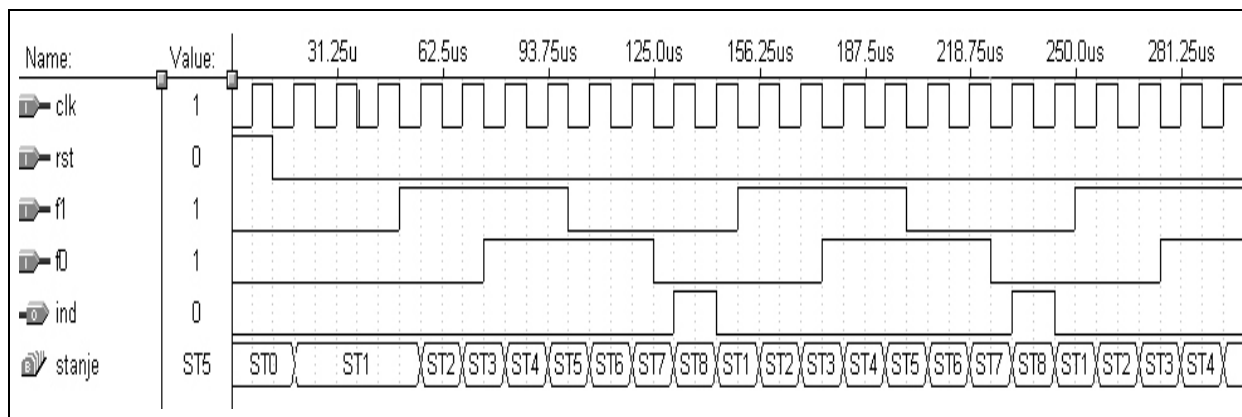
```
  stanje.clk = clk;
  stanje.reset = rst;
```

TABLE

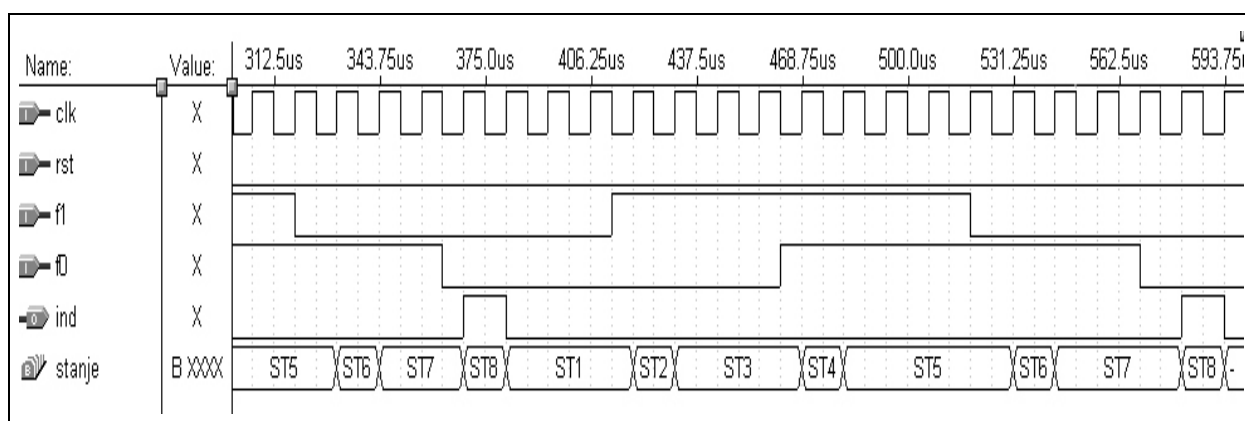
stanje,	f1,	f0	=>	ind,	stanje;	%
%	-----					%
ST0,	0,	0	=>	0,	ST1;	
ST0,	0,	1	=>	0,	ST0;	
ST0,	1,	0	=>	0,	ST0;	
ST0,	1,	1	=>	0,	ST0;	
ST1,	0,	0	=>	0,	ST1;	
ST1,	0,	1	=>	0,	ST0;	
ST1,	1,	0	=>	0,	ST2;	
ST1,	1,	1	=>	0,	ST0;	
ST2,	0,	0	=>	0,	ST0;	
ST2,	0,	1	=>	0,	ST0;	
ST2,	1,	0	=>	0,	ST3;	
ST2,	1,	1	=>	0,	ST0;	
ST3,	0,	0	=>	0,	ST0;	
ST3,	0,	1	=>	0,	ST0;	
ST3,	1,	0	=>	0,	ST3;	
ST3,	1,	1	=>	0,	ST4;	
ST4,	0,	0	=>	0,	ST0;	
ST4,	0,	1	=>	0,	ST0;	
ST4,	1,	0	=>	0,	ST0;	
ST4,	1,	1	=>	0,	ST5;	
ST5,	0,	0	=>	0,	ST0;	
ST5,	0,	1	=>	0,	ST6;	
ST5,	1,	0	=>	0,	ST0;	
ST5,	1,	1	=>	0,	ST5;	
ST6,	0,	0	=>	0,	ST0;	
ST6,	0,	1	=>	0,	ST7;	
ST6,	1,	0	=>	0,	ST0;	
ST6,	1,	1	=>	0,	ST0;	
ST7,	0,	0	=>	0,	ST8;	
ST7,	0,	1	=>	0,	ST7;	
ST7,	1,	0	=>	0,	ST0;	
ST7,	1,	1	=>	0,	ST0;	
ST8,	0,	0	=>	1,	ST1;	
ST8,	0,	1	=>	1,	ST0;	
ST8,	1,	0	=>	1,	ST0;	
ST8,	1,	1	=>	1,	ST0;	

END TABLE;**END;**

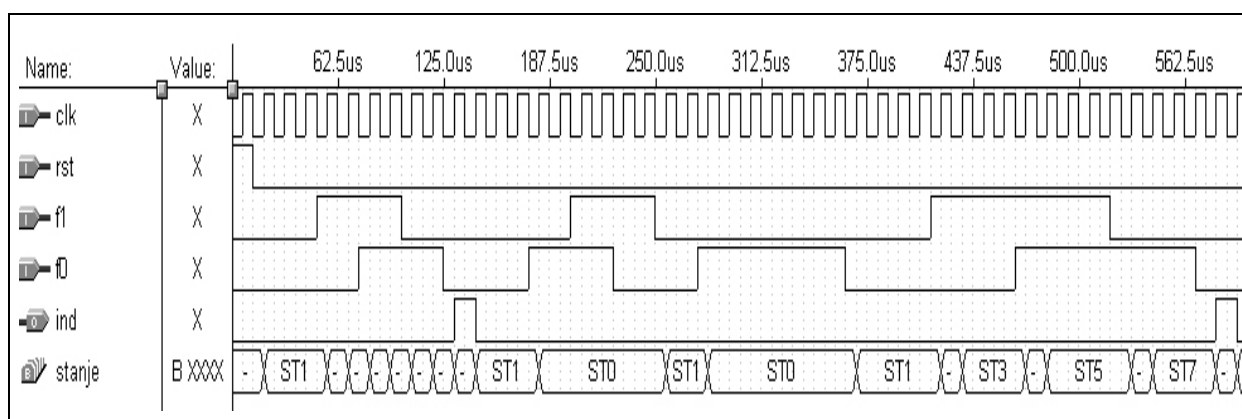
Nekoliko karakterističnih simulacionih dijagrama dato je na slikama br. 22.10, 22.11 i 22.12.



Slika 22.10: Simulacioni dijagram povorke impulsa $f1$, $f0$ pri maksimalnoj brzini okretanja osovine optičkog enkodera u smeru koji se treba detektovati



Slika 22.11: Simulacioni dijagram povorke impulsa $f1$, $f0$ pri brzini okretanja osovine optičkog enkodera u smeru koji se treba detektovati



Slika 22.12: Simulacioni dijagram neregularnih pojavljivanja sekvenci signala $f1$, $f0$

Opis sistema primenom VHDL jezika

ZADATAK 23 – Programabilan sinhroni brojač nadole sa definisanom granicom brojanja

Realizovati programabilni osmobaritni sinhroni brojač nadole sa funkcijom definisanja gornje granice brojanja. Ovaj brojač sadrži ulazni registar preko koga se spolja definiše vrednost gornje granice brojanja. Kada dostigne vrednost 0 (0x00), pri sledećoj uzlaznoj ivici signala takta sinhroni brojač prelazi u stanje u kojem ima vrednost koja se nalazi u ulaznom registru sistema. Sistem sadrži sledeće kontrolne signale:

- sinhronizacioni signal *CLK* pri čemu se sinhronne promene dešavaju pri usponskoj ivici *CLK* signala;
- signal asinhronog reseta *RST* koji je aktivan pri *RST*= '1';
- signal dozvole brojanja *DOZVOLA* (dozvoljava se brojanje kada je *DOZVOLA*= '1')
- signal dozvole upisa u ulazni registar *UPIS* (dozvola upisa pri *UPIS*= '1').

Pored kontrolnih signala, sistem sadrži osmobaritne signale podataka:

- *ULAZ* –ulazni podatak koji nosi vrednost koja se upisuje u ulazni registar (vrednost gornje granice brojanja) i
- *IZLAZ* –izlazni podatak trenutne vrednosti brojača.

Podatak na ulazu (*ULAZ*) i izlazu (*IZLAZ*) predstavljen je kao ceo pozitivan broj. Funkcija asinhronog reseta ima najveći prioritet izvršavanja. Prilikom asinhronog reseta vrednosti brojača i ulaznog registra postavljaju se na 0 (0x00). Upis vrednosti gornje granice brojanja izvršava se paralelno i nezavisno od funkcije brojanja brojača.

Primenom softverskog paketa za rad sa programabilnim kolima firme ALTERA realizovati mrežu VHDL opisom.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

Na osnovu specificiranog zadatka lako se uočava da sistem sadrži dva podsistema i to:

- sinhroni registar na ulazu, tj. bafer koji sadrži vrednost gornje granice brojanja i čiji se sadržaj može promeniti spolja preko signala *UPIS* i *ULAZ* i
- sinhroni brojač nadole od vrednosti koja se nalazi u ulaznom registru do 0. Kada brojač dostigne vrednost 0 aktivira se novi ciklus brojanja od vrednosti koja je specificirana u ulaznom registru.

Kompletan VHDL opis sistema dat je u nastavku.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
```

```
ENTITY zadatak IS PORT
```

```
(
```

```
CLK      : IN  STD_LOGIC;
RST      : IN  STD_LOGIC;
DOZVOLA  : IN  STD_LOGIC;;
```

```

    UPIS      : IN   STD_LOGIC;;
    ULAZ      : IN   STD_LOGIC_VECTOR( 7 DOWNT0 0 );
    IZLAZ     : OUT  STD_LOGIC_VECTOR( 7 DOWNT0 0 )
);
END zadatak;

ARCHITECTURE opis OF zadatak IS
    SIGNAL tmpGranica      : STD_LOGIC_VECTOR( 7 DOWNT0 0 );
    SIGNAL tmpVrBrojaca     : UNSIGNED( 7 DOWNT0 0 );
BEGIN
    PROCESS ( CLK, RST ) BEGIN
        IF ( RST = '1' ) THEN
            -- Asinhrona promena - asinhroni reset
            tmpGranica      <= ( OTHERS => '0' );
            tmpVrBrojaca     <= ( OTHERS => '0' );
        ELSIF ( rising_edge( CLK ) ) THEN
            -- Sinhrona promena
            IF ( UPIS = '1' ) THEN
                -- Baferisanje vrednosti gornje granice brojanja
                tmpGranica <= ULAZ;
            END IF;
            IF ( DOZVOLA = '1' ) THEN
                -- Funkcija sinhronog brojaca na dole
                IF ( tmpVrBrojaca = 0 ) THEN
                    tmpVrBrojaca <= UNSIGNED( tmpGranica );
                ELSE
                    tmpVrBrojaca <= tmpVrBrojaca - 1;
                END IF;
            ELSE
                tmpVrBrojaca <= tmpVrBrojaca;
            END IF;
        END IF;
    END PROCESS;
    IZLAZ <= STD_LOGIC_VECTOR( tmpVrBrojaca( 7 DOWNT0 0 ) );
END opis;

```

Za realizaciju sistema u kojima se koriste operacije sabiranja i oduzimanja pogodno je koristiti *UNSIGNED* tip podatka. Kako bi se koristio *UNSIGNED* tip potrebno je uključiti pakete *std_logic_arith* iz biblioteke *ieee* što je učinjeno na samom početku VHDL koda.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

```

Memorisanje vrednosti gornje granice brojanja, tj. realizacija ulaznog registra, obezbeđuje se preko pomoćnog signala *tmpGranica*. Realizacija brojača postiže se unutrašnjim osmobaritnim registrom i *UNSIGNED* tipom podatka kojim je definisan preko signala *tmpVrBrojaca*. Oba ova tipa podatka su dužine osam bita.

```

SIGNAL tmpGranica      : STD_LOGIC_VECTOR( 7 DOWNT0 0 );
SIGNAL tmpVrBrojaca     : UNSIGNED( 7 DOWNT0 0 );

```

Nad *UNSIGNED* tipom signala, *tmpVrBrojaca*, definisana je operacija oduzimanja što se koristi za implementaciju funkcije brojanja nadole.

Sve funkcije sistema realizovane su preko procesa koji reaguje na promene signala *RST* i *CLK* čime se obezbeđuju asinhroni reset i sinhronne promene sistema. Asinhroni reset sistema, tj. reset stanja ulaznog registra i registra brojača, realizuje se preko *IF* uslova strukture *IF...ELSIF...END IF*. Na ovaj način obezbeđuje se veći prioritet asinhronog reseta nad sinhronim promenama u sistemu. Sinhronne promene, koje su nižeg prioriteta od asinhronog reseta, realizuju se unutar *ELSIF* bloka iste strukture.

```

PROCESS ( CLK, RST ) BEGIN
  IF ( RST = '1' ) THEN
    -- Asinhrona promena - asinhroni reset
    tmpGranica      <= ( OTHERS => '0' );
    tmpVrBrojaca    <= ( OTHERS => '0' );
  ELSIF ( rising_edge( CLK ) ) THEN
    -- Sinhronne promene sistema
    -- ...
  END IF;
END PROCESS;

```

Sinhronne promene sistema, tj. funkcija baferisanja gornje granice brojanja i mehanizam brojanja na dole, implementirane su u sekciji procesa *Sinhronne promene sistema*. Deo VHDL koda kojim se baferiše gornja granica brojanja je:

```

-- Baferisanje vrednosti gornje granice brojanja
IF ( UPIS = '1' ) THEN
  tmpGranica <= ULAZ;
END IF;

```

dok se u delu koda:

```

-- Sinhroni brojac na dole
IF ( DOZVOLA = '1' ) THEN
  IF ( tmpVrBrojaca = 0 ) THEN
    tmpVrBrojaca <= UNSIGNED( tmpGranica );
  ELSE
    tmpVrBrojaca <= tmpVrBrojaca - 1;
  END IF;
ELSE
  tmpVrBrojaca <= tmpVrBrojaca;
END IF;

```

realizuje mehanizam brojača nadole. Ove dve sekcije, unutar bloka koji se dešava sinhrono sa uzlaznom ivicom *CLK* signala (*rising_edge(CLK)*), obavljaju se paralelno.

Formiranje signala na izlazu *IZLAZ* postiže se konverzijom *UNSIGNED* u *STD_LOGIC_VECTOR* tip podatka na sledeći način:

```

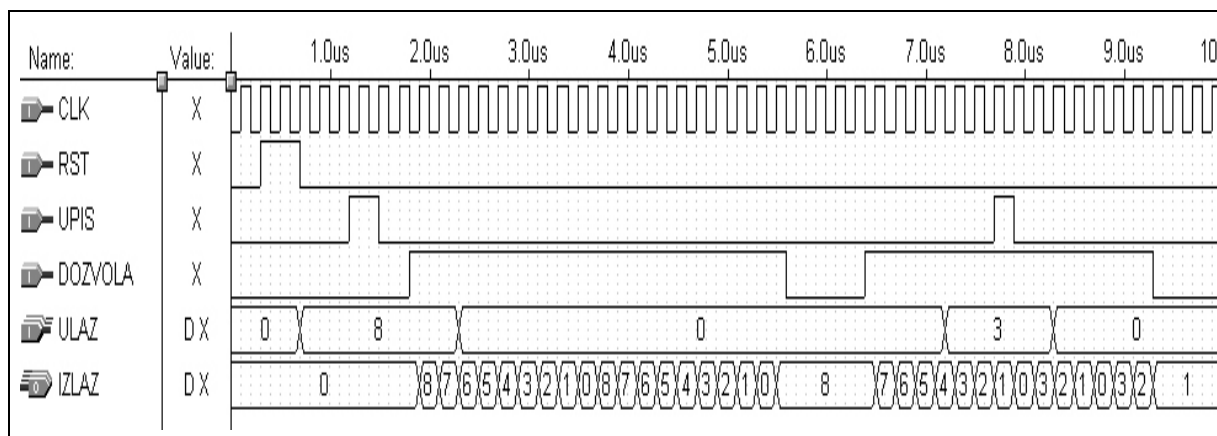
IZLAZ <= STD_LOGIC_VECTOR( tmpVrBrojaca( 7 DOWNTO 0 ) );

```

Na slici br. 23.1 prikazan je vremenski dijagram jedne kombinacije test signala kojim se delom verifikuje rad sistema. Vremenski dijagram dobijen je unosom željenih oblika signala za ulazne priključke i aktiviranjem simulatora. Nakon simulacije, vremenski dijagram se popunjava

oblikom signala na izlazu u skladu sa ulaznim signalima i funkcijom sistema koji je opisan VHDL kodom.

Uočava se da brojač postavlja vrednost brojanja 0x08 u trenucima 1.9, 3.7 i 5.5usec, tj. broji od 0x08 do 0x00 pošto je u trenutku 1.4usec postavljena gornja granica brojanja na 0x08. Nakon upisu nove granice brojanja 0x03 u ulazni registar u trenutku 7.7usec, brojač broji na dole od 0x03 do 0x00.



Slika 23.1: Vremenski oblik signala za verifikaciju rada sistema

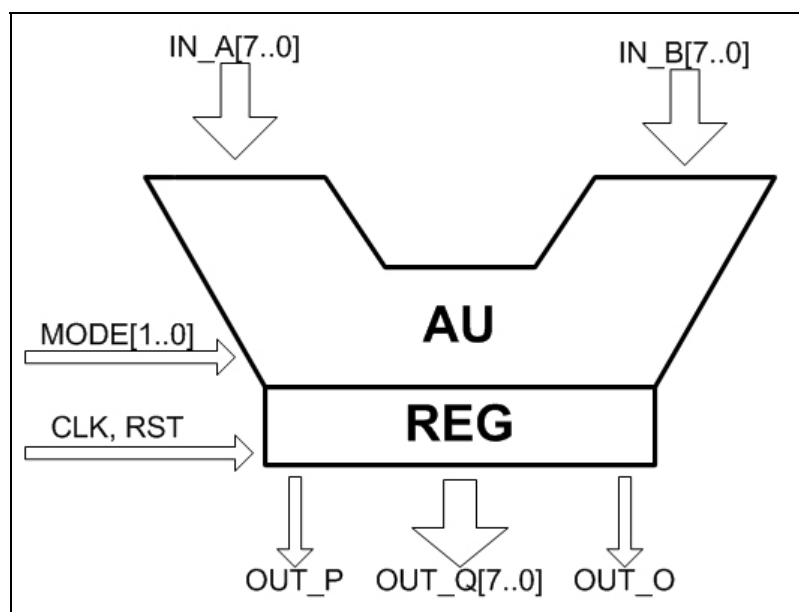
ZADATAK 24 – Aritmetička jedinica sa prihvatnim registrom

Realizovati sistem koji predstavlja spregu jednostavne aritmetičke jedinice (AU) i osmобitnog sinhronog prihvatnog registra (REG). AU jedinica može obaviti sledeće operacije:

- da pihvati podatak,
- da sabere vrednosti dva ulazna podatka i
- da udvostruči vrednost podatka.

Rezultat treba upisati u prihvatni registar REG.

Blok šema sistema data je na slici br. 24.1.



Slika 24.1: Blok šema sistema

Kontrolni signali sistema su:

- *CLK* – sinhronizacioni signal;;
- *RST* – signal sinhronog reseta prihvatnog registra REG i
- *MODE* – dvobitni ulazni signal koji određuje način rada AU podistema.

Sinhronne promene u sistemu dešavaju se pri usponskoj ivici *CLK* signala. Sinhroni reset prihvatnog registra je aktivan kada je *RST*= '1'.

Pored kontrolnih signala, sistem sadrži i signale podataka i to:

- *IN_A* i *IN_B* – linije signala 8-bitnih ulaznih podataka;
- *OUT_Q* – linije signala 8-bitnog izlaznog podatka, tj. izlaza prihvatnog registra REG;
- *OUT_P* – izlazni signal indikacije prenosa pri sabiranju i
- *OUT_O* – izlazni signal indikacije prekoračenja pri množenju.

Rad AU podistema, i vrednost koja se upisuje u prihvatni registar REG, zavise od vrednosti *MODE* signala na sledeći način:

- *MODE* = "00" prihvatni registar ostaje nepromenjen;
- *MODE* = "01" u registar se upisuje vrednost podatka *IN_A*,
- *MODE* = "10" u registar se upisuje rezultat sabiranja podataka sa ulaza *IN_A* i *IN_B* i
- *MODE* = "11" u registar se upisuje dvostruka vrednost podatka *IN_A*.

U svim slučajevima rada sistema, osim kada je *MODE*= "10", izlazni signal *OUT_P* ima vrednost '0'. Pri sabiranju, tj. kada je *MODE*= "10", signal *OUT_P* ima funkciju signala prenosa. U svim slučajevima rada, osim kada je *MODE*= "11", izlazni signal *OUT_O* ima vrednost '0'. Pri množenju sa dva, tj. kada je *MODE* = "11", signal *OUT_O* ima funkciju signala indikacije prekoračenja. Signal *RST*, tj. funkcija sinhronog reseta, je najvećeg prioriteta. Ulazni i izlazni podaci *IN_A*, *IN_B* i *OUT_Q* predstavljaju binarne osmobarne celobrojne vrednosti.

Primenom softverskog paketa za rad sa programabilnim kolima firme ALTERA realizovati opisani sistem u VHDL jeziku.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

Formirati simbol opisanog sistema u VHDL-u.

Osmisliti i opisati način verifikacije sistema na UP2 razvojnoj ploči i EPM7128LC84-7 čipu iz familije kola MAX7000S.

REŠENJE

VHDL opis zahtevanog sistema, koji obuhvata AU jedinicu i sinhroni prihvatni registar, dat je u nastavku.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY au_reg IS PORT
(
    CLK      : IN    STD_LOGIC;
    RST      : IN    STD_LOGIC;
    MODE     : IN    STD_LOGIC_VECTOR( 1 DOWNT0 0 );
    IN_A     : IN    STD_LOGIC_VECTOR( 7 DOWNT0 0 );
    IN_B     : IN    STD_LOGIC_VECTOR( 7 DOWNT0 0 );
    OUT_Q    : OUT   STD_LOGIC_VECTOR( 7 DOWNT0 0 );
    OUT_P    : OUT   STD_LOGIC;
    OUT_O    : OUT   STD_LOGIC

```

```

);
END au_reg ;

ARCHITECTURE behav OF au_reg IS
    -- Pomocni signali
    SIGNAL tmp          : UNSIGNED( 8 DOWNT0 0 );
    SIGNAL tmp_out_o     : STD_LOGIC;
BEGIN
    PROCESS ( CLK, RST ) BEGIN
        IF ( rising_edge( CLK ) ) THEN
            -- Sinhrona logika sistema
            IF ( RST = '1' ) THEN
                -- Realizacija sinhronog reseta
                tmp          <= ( OTHERS => '0' );
                tmp_out_o    <= '0';
            ELSE
                -- Opis podataka u sinhroni prihvatni registar REG
                IF ( MODE = "01" ) THEN
                    -- Prihvatanje podatka
                    tmp( 7 DOWNT0 0 ) <= UNSIGNED( IN_A );
                    tmp( 8 ) <= '0';
                    tmp_out_o <= '0';

                    ELSIF ( MODE = "10" ) THEN
                        -- Sabiranje
                        tmp          <= UNSIGNED( IN_A ) +
                                      UNSIGNED( IN_B );
                        tmp_out_o <= '0';
                    ELSIF ( MODE = "11" ) THEN
                        -- Mnozenje
                        tmp( 7 downto 1 ) <=
                            UNSIGNED( IN_A( 6 DOWNT0 0 ) );
                        tmp( 0 ) <= '0';
                        tmp_out_o <= IN_A( 7 );
                        tmp( 8 ) <= '0';
                    ELSE
                        tmp_out_o <= '0';
                    END IF;
                END IF;
            END IF;
        END PROCESS;
    -- Povezivanje izlaza registra REG i izlaza sistema
    OUT_Q <= STD_LOGIC_VECTOR( tmp( 7 DOWNT0 0 ) );
    OUT_P <= tmp( 8 );
    OUT_0 <= tmp_out_o;
END behav;

```

Za opis sistema u VHDL jeziku koriste se dva standardna paketa *ieee* biblioteke koje se uključuju u projekat navođenjem na početku VHDL koda:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

```

Paket koji podržava rad sa *UNSIGNED* tipom podatka je *ieee.std_logic_arith.all*.

U nastavku je specificiran interfejs sistema, koji obuhvata imena i tipove ulaznih i izlaznih priključaka sistema.

```

ENTITY au_reg IS PORT
(
  CLK      : IN   STD_LOGIC;
  RST      : IN   STD_LOGIC;
  MODE     : IN   STD_LOGIC_VECTOR( 1 DOWNTO 0 );
  IN_A     : IN   STD_LOGIC_VECTOR( 7 DOWNTO 0 );
  IN_B     : IN   STD_LOGIC_VECTOR( 7 DOWNTO 0 );
  OUT_Q    : OUT  STD_LOGIC_VECTOR( 7 DOWNTO 0 );
  OUT_P    : OUT  STD_LOGIC;
  OUT_O    : OUT  STD_LOGIC
);
END au_reg ;

```

Ime interfejsa (u ovom slučaju *au_reg*) je ujedno i ime sistema. Da bi se dizajn sistema regularno preveo od strane prevodioca potrebno je da se ime sistema poklapa sa imenom datoteke dizajna.

Sprega između podsistema AU i REG ostvaruje se preko pomoćnih signala *tmp[8..0]* i *tmp_out_o*.

```

-- Pomocni signali
SIGNAL tmp      : UNSIGNED( 8 DOWNTO 0 );
SIGNAL tmp_out_o : STD_LOGIC;

```

Funkcije sistema u celini ostvaruju se preko procesa koji je osetljiv na promene signala *CLK* i *RST*. Obezbeđivanje najvećeg prioriteta funkcije sinhronog reseta postiže se strukturom *IF...ELSIF...END IF* gde je uslov sinhronog reseta prvi po važnosti.

```

PROCESS ( CLK, RST ) BEGIN
  IF ( rising_edge( CLK ) ) THEN
    -- Sinhrona logika sistema
    IF ( RST = '1' ) THEN
      -- Realizacija sinhronog reseta
      tmp      <= ( OTHERS => '0' );
      tmp_out_o <= '0';
    ELSE
      -- Opis podataka u sinhroni prihvatni registar REG
      -- ...
    END IF;
  END IF;
END PROCESS;

```

tmp je devetobitni signal koji sadrži objedinjenu vrednost rezultata operacije AU podsistema (*tmp[7..0]*) i signal indikacije prekoračenja pri sabiranju (*tmp8*). Pri sabiranju dva osmобitna celobrojna pozitivna broja moguće je dobiti devetobitnu vrednost rezultata istog tipa. Bit najveće težine devetobitne vrednosti rezultata tumači se kao prekoračenje pri sabiranju. Sabiranje vrednosti *IN_A* i *IN_B* ostvaruje se konverzijom ovih podataka tipa *STD_LOGIC_VECTOR* u *UNSIGNED* tip podatka i korišćenjem operacije sabiranja.

```

tmp      <= UNSIGNED( IN_A ) + UNSIGNED( IN_B );

```

Da bi se obezbedio tip `STD_LOGIC_VECTOR` za podatak na izlazu iz sistema (`OUT_Q`) koristi se funkcija konverzije podataka iz `UNSIGNED` u `STD_LOGIC_VECTOR`.

```
OUT_Q <= STD_LOGIC_VECTOR( tmp( 7 DOWNT0 0 ) );
OUT_P <= tmp( 8 );
```

Kod implementacije operacije izračunavanja dvostruke vrednosti podatka `IN_A` koristi se mogućnost realizacije množenja sa brojem dva preko operacije pomeranja binarnog sadržaja za jedno mesto ulevo.

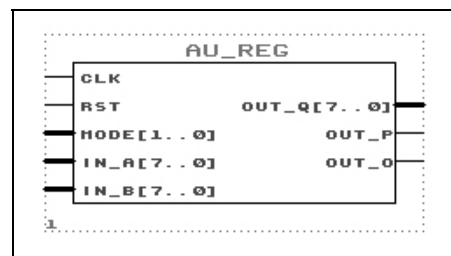
```
tmp( 7 downto 1 ) <= UNSIGNED( IN_A( 6 DOWNT0 0 ) );
tmp( 0 ) <= '0';
tmp_out_o <= IN_A( 7 );
tmp( 8 ) <= '0';
```

Prekoračenje pri množenju sa dva određuje bit najveće težine podatka `IN_A`.

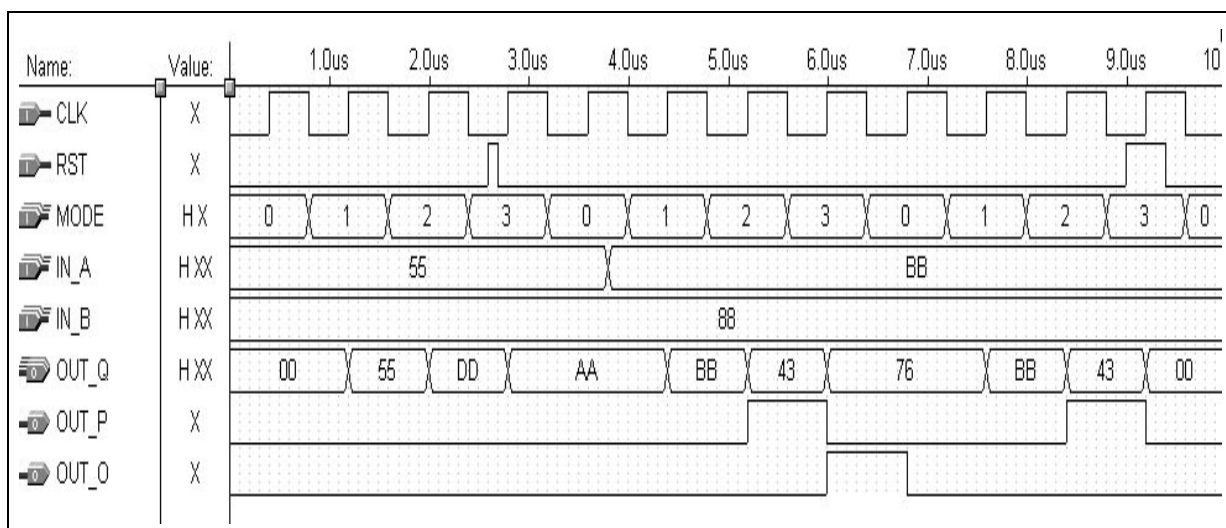
Važno je napomenuti da je neophodno za svaku situaciju, tj. za svaki mod rada AU, definisati sve signale koji se pojavljuju na izlazu tj. `OUT_Q[7..0]`, `OUT_O` i `OUT_P`. Definisanje ovih signala ostvaruje se preko definisanja pomoćnih signala koji se povezuju sa signalima na izlazu.

```
OUT_Q <= STD_LOGIC_VECTOR( tmp( 7 DOWNT0 0 ) );
OUT_P <= tmp( 8 );
OUT_O <= tmp_out_o;
```

Simbol realizovanog sistema dobijenog posle prevođenja dizajna prikazan je na slici br. 24.2.



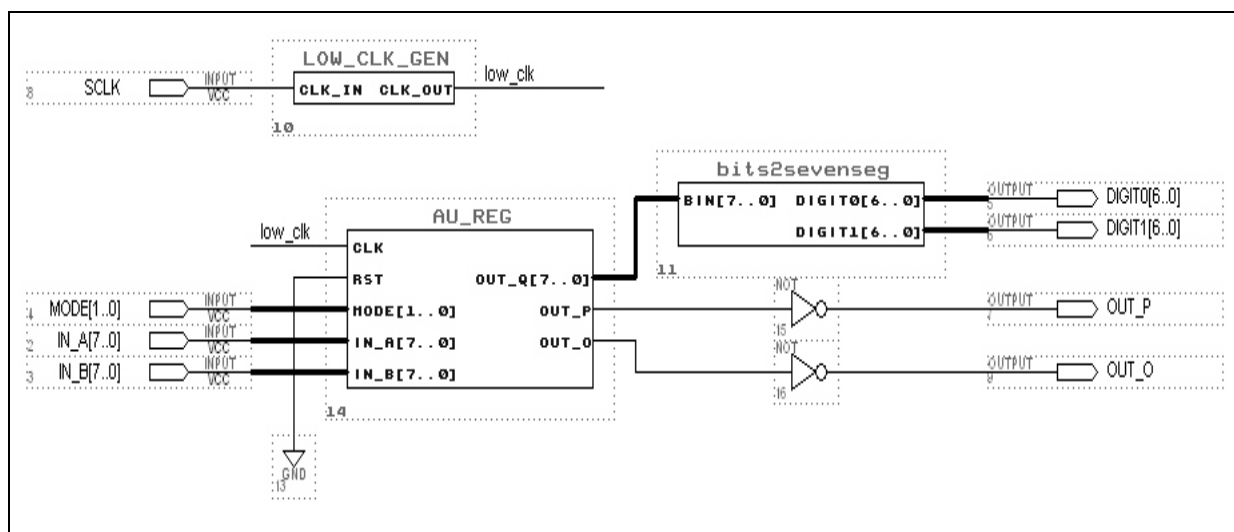
Slika 24.2: Simbol sistema



Slika 24.3: Vremenski dijagram simulacije rada sistema

Na slici br. 24.3 prikazana je jedna kombinacija signala na ulazu kao i rezultat simulacije opisanog sistema u VHDL-u.

Verifikacija realizovanog sistema na UP2 razvojnoj ploči može se obaviti preko test šeme koja je prikazana na slici br. 24.3. Na UP2 razvojnoj ploči, EPM7128LC84 čipu pridruženo je 16 prekidača preko dve grupe od po 8 DIP prekidača i dva tastera. Iz tih razloga, jedan kontrolni signal projektovanog sistema (*AU_REG*) nije moguće kontrolisati spolja. Na slici br. 24.4, signal *RST* vezan je za GND i time onemogućeno sinhrono resetovanje sistema. Povezanost signala sa test šeme na slici br. 24.4 i pinova EPM7128SLC84-7 čipa na UP2 ploči dat je u tabeli br. 24.1.



Slika 24.4: Test šema za verifikaciju rada sistema na UP2 razvojnoj ploči

Tabela 24.1: Tabela povezanosti signala za verifikaciju rada sistema na UP2 razvojnoj ploči

<i>Signal</i>	<i>Element na UP2 ploči</i>	<i>Opis</i>
<i>SCLK</i>	pin 83	Veza sa sistemskim klokom od 25.175MHz.
<i>MODE[1..0]</i>	MAX_PB1, MAX_PB2	Zadavanje moda rada AU preko tastera.
<i>IN_A[7..0]</i>	MAX_SW1	Zadavanje vrednosti IN_A preko DIP prekidača MAX_SW1.
<i>IN_B[7..0]</i>	MAX_SW2	Zadavanje vrednosti IN_A preko DIP prekidača MAX_SW2.
<i>DIGIT0[6..0]</i>	MAX_DIGIT2	Prikaz nižeg nibla vrednosti prihvatnog registra REG na desni sedmosegmentni displej.
<i>DIGIT1[6..0]</i>	MAX_DIGIT1	Prikaz višeg nibla vrednosti prihvatnog registra REG na desni sedmosegmentni displej.
<i>OUT_P</i>	LED D1	Prikaz prenosa pri sabiranju na LED D1.
<i>OUT_O</i>	LED D2	Prikaz prekoračenja pri množenju na LED D2.

Na slici br. 24.4 uočavaju se dva bloka *LOW_CLK_GEN* i *bits2sevenseg* čije su funkcije pomoćne pri testiranju sistema. Prvi blok, *LOW_CLK_GEN*, služi da obezbedi takt od oko 1Hz za potrebe testiranja sistema preko spoljašnjih prekidača. Preko bloka *bits2sevenseg* vrši se

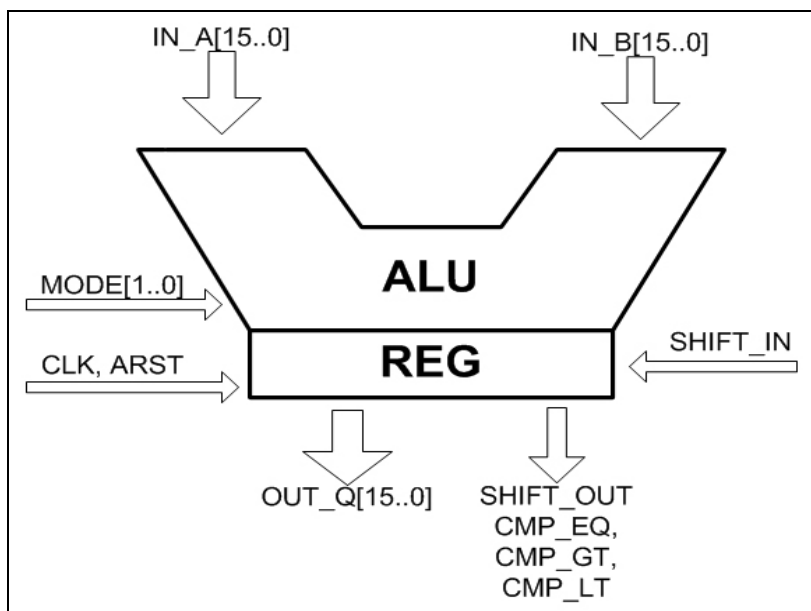
dekodovanje osmootne vrednosti u niz od 14 signala za pobuđivanje dva sedmosegmentna displeja MAX_DIGIT1 i MAX_DIGIT0. Invertori na linijama signala *OUT_P* i *OUT_O* obezbeđuju paljenje LED dioda u situacijama kada se pojavi prenos pri sabiranju i prekoračenje pri množenju.

ZADATAK 25 – Aritmetičko-logička jedinica sa prihvatnim registrom

Realizovati sistem koji sadrži aritmetičko-logičku jedinicu (ALU) i šesnaestobitni sinhroni prihvatni registar (REG). ALU jedinica može da obavi sledeće operacije nad ulaznim podacima *IN_A* i *IN_B*:

- da prihvati podatak sa ulaza *IN_A*
- da izvrši upoređivanje vrednosti *IN_A* i *IN_B*;
- da prihvati korigovanu vrednost podatka *IN_B* i
- da izvrši pomeranje vrednosti prihvatnog registra za jedno mesto ulevo ili udesno.

Rezultat operacije se upisuje u prihvatni registar REG. Blok šema sistema data je na slici br. 25.1.



Slika 25.1: Blok šema sistema

Kontrolni signali sistema su:

- *CLK* – sinhronizacioni signal;
- *ARST* – signal asinhronog reseta prihvatnog registra REG;
- *MODE* – dvobitni ulazni signal koji određuje način rada ALU podsistema.

Sinhronne promene u sistemu dešavaju se pri opadajućoj ivici *CLK* signala. Asinhroni reset prihvatnog registra REG dešava se pri *ARST* = '0'.

Pored kontrolnih signala sistem sadrži i signale podataka i to:

- *IN_A* i *IN_B* – linije signala 16-bitnih ulaznih podataka;
- *SHIFT_IN* – linija ulaznog signala koja ima ulogu pridodatog bita pri pomeranju vrednosti registra;
- *OUT_Q* – linije signala 16-bitnog izlaznog podatka tj. stanja prihvatnog registra REG;
- *SHIFT_OUT* – linija signala koja ima funkciju baferisanja bita koji se potiskuje iz registra pri pomeranju vrednosti i
- *CMP_EQ*, *CMP_GT* i *CMP_LT* – tri jednobitna signala preko kojih se daje indikacija komparacije ulaznih vrednosti *IN_A* i *IN_B*.

Rad ALU podsistema, tj. rezultat operacije koju obavlja ALU jedinica i vrednost koja se dovodi na ulaz prihvatnog registra REG odnosno upisuje u prihvatni registar, određen je na bazi vrednosti *MODE* signala i to:

- kada je *MODE* = "00" u prihvatni registar se upisuje vrednost podatka *IN_A* i na izlazu se obezbeđuje indikacija upoređivanja podataka *IN_A* i *IN_B* u skladu sa tablicom br. 1;
- kada je *MODE* = "01" u prihvatni registar se upisuje korigovana vrednost ulaznog podatka *IN_B* pri čemu se korekcija sastoji u uzajamnoj zameni mesta nižeg i višeg bajta *IN_B*;
- kada je *MODE* = "10" u prihvatni registar se upisuje vrednost koja odgovara pomerenoj vrednosti prihvatnog registra za jedno mesto ulevo i
- kada je *MODE* = "11" u prihvatni registar se upisuje vrednost koja odgovara pomerenoj vrednosti prihvatnog registra za jedno mesto udesno.

Table 25.1: Opis rada funkcije upoređivanja ALU jedinice sistema

	<i>CMP_LT</i>	<i>CMP_EQ</i>	<i>CMP_GT</i>
<i>IN_A > IN_B</i>	'0'	'0'	'1'
<i>IN_A == IN_B</i>	'0'	'1'	'0'
<i>IN_A < IN_B</i>	'1'	'0'	'0'

Signali kojima se vrši indikacija upoređivanja vrednosti *IN_A* i *IN_B* aktivni su samo prilikom rada sistema za *MODE* = "00". U ostalim modovima rada *CMP_EQ* = *CMP_GT* = *CMP_LT* = '0'. Prilikom realizacije funkcije pomeranja vrednosti prihvatnog registra nalevo (*MODE* = "10") ili desno (*MODE* = "11") vrednost bita koji se dodaje na poziciju 0 odnosno 15 je vrednost signala sa ulaza *SHIFT_IN*. Bit koji se potiskuje iz prihvatnog registra prilikom pomeranja vrednostu predstavlja se linijom signala *SHIFT_OUT*. Mehanizam upoređivanja vrednosti *IN_A* i *IN_B* takođe treba realizovati u sinhronoj logici.

Primenom softverskog paketa za rad sa programabilnim kolima firme ALTERA realizovati opisani sistem u VHDL jeziku.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

U nastavku je dat VHDL opis sistema ALU jedinice i prihvatnog registra REG.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY alu_reg IS PORT
(
    CLK           : IN  STD_LOGIC;
    ARST          : IN  STD_LOGIC;
    MODE          : IN  STD_LOGIC_VECTOR( 1 DOWNT0 0 );
    IN_A          : IN  STD_LOGIC_VECTOR( 15 DOWNT0 0 );
    IN_B          : IN  STD_LOGIC_VECTOR( 15 DOWNT0 0 );
    SHIFT_IN      : IN  STD_LOGIC;
    OUT_Q         : OUT STD_LOGIC_VECTOR( 15 DOWNT0 0 );
    SHIFT_OUT     : OUT STD_LOGIC;
    CMP_EQ        : OUT STD_LOGIC;
    CMP_GT        : OUT STD_LOGIC;
    CMP_LT        : OUT STD_LOGIC
);

```

```

END alu_reg ;

ARCHITECTURE behav OF alu_reg IS
    SIGNAL tmp          : STD_LOGIC_VECTOR( 15 DOWNT0 0 );
    SIGNAL tmp_cmp_eq   : STD_LOGIC;
    SIGNAL tmp_cmp_gt   : STD_LOGIC;
BEGIN
    tmp_cmp_eq <= '1' WHEN IN_A = IN_B ELSE '0';
    tmp_cmp_gt <= '1' WHEN IN_A > IN_B ELSE '0';
    PROCESS ( CLK, ARST ) BEGIN
        IF ( ARST = '0' ) THEN
            -- Asinhroni reset
            tmp <= ( OTHERS => '0' );
            SHIFT_OUT <= '0';
            CMP_EQ <= '0';
            CMP_GT <= '0';
            CMP_LT <= '0';
        ELSIF ( falling_edge( CLK ) ) THEN
            -- Sinhrona promene
            IF ( MODE = "00" ) THEN
                -- Uporedjivanje vrednosti IN_A i IN_B
                CMP_EQ <= tmp_cmp_eq;
                CMP_GT <= tmp_cmp_gt;
                CMP_LT <= ( NOT tmp_cmp_eq ) AND ( NOT tmp_cmp_gt );
                SHIFT_OUT <= '0';
                tmp <= IN_A;
            ELSE
                IF ( MODE = "01" ) THEN
                    -- Upis korigovane vrednosti IN_B
                    SHIFT_OUT <= '0';
                    tmp <= IN_B( 7 DOWNT0 0 ) & IN_B( 15 DOWNT0 8 );
                ELSIF ( MODE = "10" ) THEN
                    -- Pomeranje u levo
                    SHIFT_OUT <= tmp( 15 );
                    tmp <= tmp( 14 DOWNT0 0 ) & SHIFT_IN;
                ELSIF ( MODE = "11" ) THEN
                    -- Pomeranje u desno
                    SHIFT_OUT <= tmp( 0 );
                    tmp <= SHIFT_IN & tmp( 15 DOWNT0 1 );
                ELSE
                    SHIFT_OUT <= '0';
                    tmp <= tmp;
                END IF;
                -- Neaktivni signali uporedjivanja vrednosti
                CMP_EQ <= '0';
                CMP_GT <= '0';
                CMP_LT <= '0';
            END IF;
        END IF;
    END PROCESS;
    OUT_Q <= tmp( 15 DOWNT0 0 );
END behav;

```

Zbog nepostajanja potrebe korišćenja operacija sabiranja i oduzimanja nad podacima u okviru sistema, nije neophodno uključiti paket za rad sa podacima tipa UNSIGNED. Svi podaci i signali u sistemu su tipa STD_LOGIC_VECTOR ili STD_LOGIC. Zbog postojanja više izvora za memorijske elemente prihvatnog registra i prihvatnih bafera signala na izlazu potrebno je uvesti pomoćne signale *tmp*, *tmp_cmp_eq* i *tmp_cmp_gt*.

```
SIGNAL tmp           : STD_LOGIC_VECTOR( 15 DOWNTO 0 );
SIGNAL tmp_cmp_eq    : STD_LOGIC;
SIGNAL tmp_cmp_gt    : STD_LOGIC;
```

Za upoređivanje vrednosti *IN_A* i *IN_B* koriste se dva pomoćna signala i to *tmp_cmp_eq* (za određivanje *IN_A == IN_B*) i *tmp_cmp_gt* (za određivanje *IN_A > IN_B*).

```
tmp_cmp_eq <= '1' WHEN IN_A = IN_B ELSE '0';
tmp_cmp_gt <= '1' WHEN IN_A > IN_B ELSE '0';
```

Ova pomoćna logika za upoređivanje vrednosti *IN_A* i *IN_B*, predstavlja čistu kombinacionu mrežu koja se nalazi van glavnog procesa sistema.

Osnovne funkcije sistema sprovode se preko procesa koji je osetljiv na promene vrednosti signala *CLK* i *ARST*. Asinhrona i sinhrona promene realizovane su u okviru blokova IF...ELSIF...END IF strukture koja je realizovana unutar procesa.

```
PROCESS ( CLK, ARST ) BEGIN
    IF ( ARST = '0' ) THEN
        -- Asinhroni reset
        -- ....
    ELSIF ( falling_edge( CLK ) ) THEN
        -- Sinhrona promene
        -- ....
    END IF;
END PROCESS;
```

Sinhronim resetom sistema anuliraju se vrednosti memorijskih elementata, tj. sadržaj prihvatnog registra *REG* (*tmp*) i pomoćnih jednobitnih bafera koji memorišu signale indikacije pri upoređivanju (*CMP_EQ*, *CMP_GT* i *CMP_LT*) i pomeranju (*SHIFT_OUT*).

```
tmp           <= ( OTHERS => '0' );
CMP_EQ        <= '0';
CMP_GT        <= '0';
CMP_LT        <= '0';
SHIFT_OUT     <= '0';
```

Mod rada u kojem ALU vrši upoređivanje vrednosti *IN_A* i *IN_B* obezbeđuje sinhroni upis signala *CMP_EQ*, *CMP_LT* i *CMP_GT* u pomoćne jednobitne registre. Za generisanje vrednosti koriste se pomoćni signali logike za upoređivanje koja je implementirana čistom kombinacionom mrežom *tmp_cmp_eq* i *tmp_cmp_gt*.

```
CMP_EQ <= tmp_cmp_eq;
CMP_GT <= tmp_cmp_gt;
CMP_LT <= ( NOT tmp_cmp_eq ) AND ( NOT tmp_cmp_gt );
```

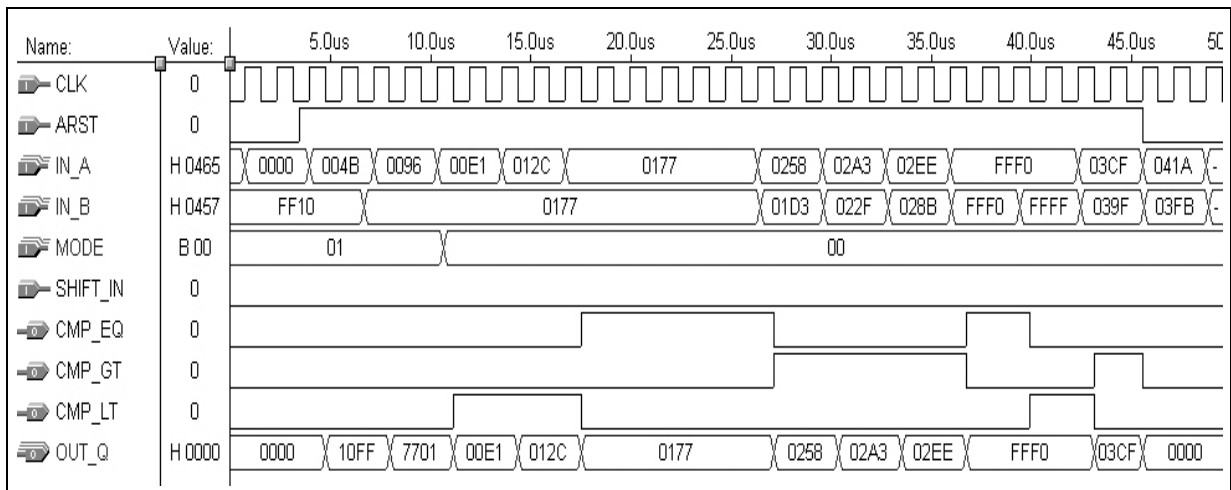
Operacije upisa modifikovane vrednosti *IN_B* kao i pomeranja vrednosti prihvatnog registra ostvaruju se preko VHDL operacije spajanja signala *&*. Upis modifikovane vrednosti podatka *IN_B* u smislu zamenjenog mesta višeg i nižeg bajta ostvaruje se na sledeći način:

```
tmp <= IN_B( 7 DOWNT0 0 ) & IN_B( 15 DOWNT0 8 );
```

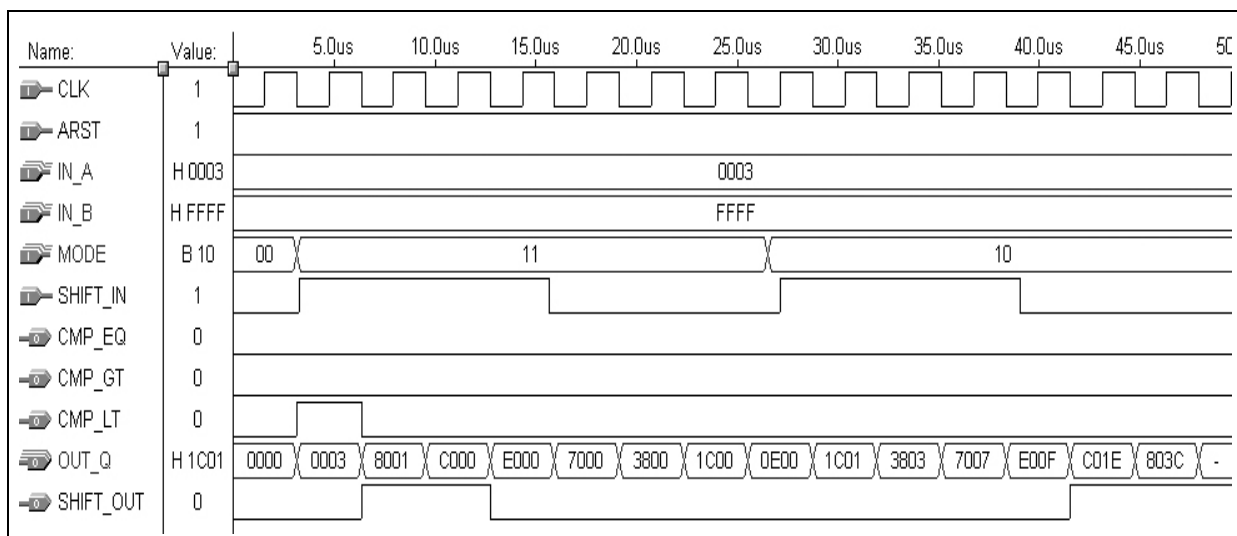
Pomeranje sadržaja prihvatnog registra za jedno mesto ulevo i udesno, generisanje signala *SHIFT_OUT* ostvaruje se na sledeći način:

```
-- Pomeranje ulevo
SHIFT_OUT <= tmp( 15 );
tmp <= tmp( 14 DOWNT0 0 ) & SHIFT_IN;
```

```
-- Pomeranje udesno
SHIFT_OUT <= tmp( 0 );
tmp <= SHIFT_IN & tmp( 15 DOWNT0 1 );
```



Slika 25.2: Simulacioni dijagram verifikacije rada sistema za *MODE*="01" i *MODE*="00"



Slika 25.3: Simulacioni dijagram verifikacije rada sistema za *MODE*="11" i *MODE*="10"

Vremenski dijagrami verifikacije rada sistema u sva četiri moda rada ALU kao i pri asinhronom resetu prikazani su na slikama br. 25.2 i 25.3.

ZADATAK 26 – Sistem za usrednjavanje četiri osmobarne celobrojne binarne vrednosti

Realizovati sekvencijalnu digitalnu mrežu koja obavlja funkciju usrednjavanja četiri sukcesivno zadate osmobarne vrednosti. Ulazni signali koji čine interfejs sistema su:

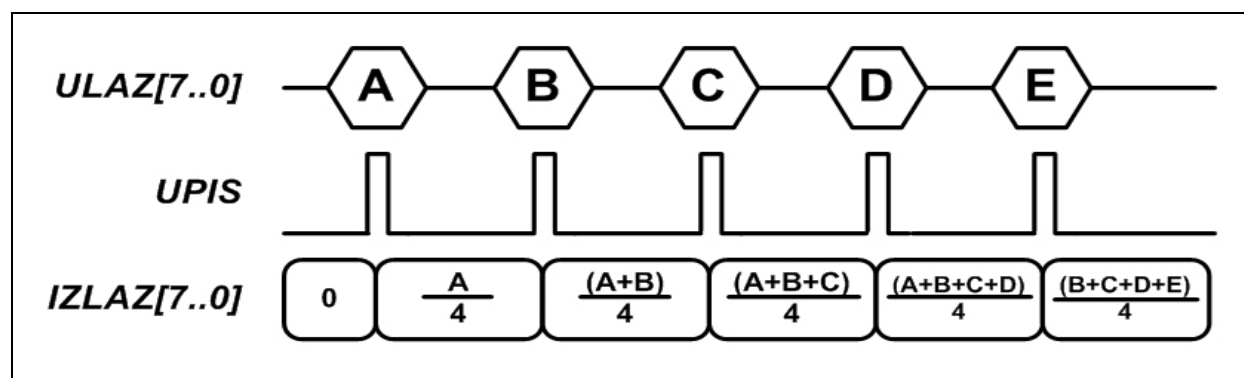
- sinhronizacioni signal *CLK* (usponska ivica je aktivna promena signala *CLK*);
- signal asinhronog reseta *RESET* (asinhroni reset aktivan je pri visokom nivou signala, tj. pri $RESET = '1'$);
- osmobarne ulazne vrednosti *ULAZ*;
- signal *UPIS* kojim se daje dozvola prihvatanja vrednosti i uključivanje u proces usrednjavanja (prihvatanje vrednosti sa ulaza obavlja se pri usponskoj ivici signala *UPIS*).

Na izlazu sistema prisutna je osmobarne vrednost preko signala *IZLAZ*.

Funkcija sistema može se predstaviti preko formule:

$$IZLAZ(t) = \frac{ULAZ(t-3) + ULAZ(t-2) + ULAZ(t-1) + ULAZ(t)}{4},$$

pri čemu oznaka t unutar zagrade: (t) , $(t-1)$, $(t-2)$ i $(t-3)$, predstavlja vrednost pridruženog signala pomoću koga se označavaju sukcesivni podaci koji se dobijaju za četiri uzastopne usponske ivice signala *UPIS*. Opis rada sistema prikazan je uprošćenim vremenskim dijagramom na slici br. 26.1. Dijagram na slici br. 26.1 pokriva početak rada sistema kao i dva ciklusa ustaljenog rada.



Slika 26.1: Vremenski dijagram opisa rada sistema za usrednjavanje četiri susedne vrednosti

Preko osmobarne ulaznih (*ULAZ[7..0]*) i izlaznih (*IZLAZ[7..0]*) signala prenose se celobrojni pozitivni brojevi ili nula.

Primenom softverskog paketa za rad sa programabilnim kolima firme ALTERA realizovati mrežu VHDL opisom.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

Za obezbeđivanje pravilnog rada opisanog sistema za usrednjavanje četiri sukcesivno zadate vrednosti, treba realizovati:

- mehanizam memorisanja vrednosti koje su potrebne za proračun zahtevane srednje vrednosti,
- sabiranje četiri vrednosti bez prekoračenja pri sabiranju i
- deljenje zbira četiri sukcesivno zadate vrednosti sa konstantom 4.

Prvi zahtev, tj. memorisanje potrebnih vrednosti, rešava se implementacijom četiri osmobaritna registra u FIFO konfiguraciji, čiji se sadržaj ažurira pri svakoj aktivnoj promeni signala *UPIS*. Registri u kojima se memorišu četiri uzastopne vrednosti sa ulazu su: *BROJ0[7..0]*, *BROJ1[7..0]*, *BROJ2[7..0]* i *BROJ3[7..0]*. Kompletan VHDL opis sistema dat je u nastavku.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY zadatak IS PORT
(
  CLK      : IN  STD_LOGIC;
  RESET    : IN  STD_LOGIC;
  UPIS     : IN  STD_LOGIC;
  ULAZ     : IN  STD_LOGIC_VECTOR( 7 DOWNTO 0 );
  IZLAZ    : OUT STD_LOGIC_VECTOR( 7 DOWNTO 0 )
);
END zadatak ;

ARCHITECTURE opis OF zadatak IS
  SIGNAL UPIS1 : STD_LOGIC;
  SIGNAL BROJ0 : UNSIGNED ( 7 DOWNTO 0 );
  SIGNAL BROJ1 : UNSIGNED ( 7 DOWNTO 0 );
  SIGNAL BROJ2 : UNSIGNED ( 7 DOWNTO 0 );
  SIGNAL BROJ3 : UNSIGNED ( 7 DOWNTO 0 );
  SIGNAL ZBIR  : UNSIGNED ( 9 DOWNTO 0 );
BEGIN
  PROCESS ( CLK, RESET) BEGIN
    IF ( RESET = '1' ) THEN
      -- Asinhrona promena - asinhroni reset
      BROJ0 <= ( OTHERS => '0' );
      BROJ1 <= ( OTHERS => '0' );
      BROJ2 <= ( OTHERS => '0' );
      BROJ3 <= ( OTHERS => '0' );
      UPIS1 <= '0';
    ELSIF ( rising_edge( CLK ) ) THEN
      -- Sinhrona promena sistema
      IF ( UPIS1 = '0' AND UPIS = '1' ) THEN
        -- Akcija pri usponskoj ivici signala UPIS
        BROJ0 <= BROJ1;
        BROJ1 <= BROJ2;
        BROJ2 <= BROJ3;
        BROJ3 <= UNSIGNED( ULAZ );
      END IF;
      UPIS1 <= UPIS;
      IZLAZ <= STD_LOGIC_VECTOR( ZBIR( 9 DOWNTO 2 ) );
    END IF;
  END PROCESS;
  ZBIR <= BROJ0 + BROJ1 + BROJ2 + BROJ3;
END opis;

```

Detekcija usponske ivice signala *UPIS*, tj. situacije pri kojoj treba prihvatiti novu vrednost sa ulaza *ULAZ* i ažurirati ostale memorisane vrednosti prethodno prihvaćene, obezbeđuje se metodom diferenciranja signala *UPIS*. Diferenciranje *UPIS* signala realizuje se korišćenjem jednog pomoćnog memorijskog elementa, tj. signala *UPIS1* i dela koda koji je izdvojen:

```

PROCESS ( CLK, RESET) BEGIN
  IF ( RESET = '1' ) THEN
    -- Asinhrona promena - asinhroni reset
    -- ...
  ELSIF ( rising_edge( CLK ) ) THEN
    -- Sinhrona promena sistema
    IF ( UPIS1 = '0' AND UPIS = '1' ) THEN
      -- Akcija pri usponskoj ivici signala UPIS
      BROJ0 <= BROJ1;
      BROJ1 <= BROJ2;
      BROJ2 <= BROJ3;
      BROJ3 <= UNSIGNED( ULAZ );
    END IF;
    UPIS1 <= UPIS;
    -- ...
  END IF;
END PROCESS;

```

Pri svakoj usponskoj ivici sinhronizacionog signala *CLK* dolazi do memorisanja vrednosti signala *UPIS* u pomoćni registar (čiji je signal na izlazu *UPIS1*). Takođe, pri svakoj aktivnoj ivici signala *CLK* ispituje se da li postoji promena signala *UPIS* sa vrednosti 0 na 1, tj. detektuje se usponska ivica signala *UPIS*. Ovaj uslov ispituje se preko trenutne vrednosti signala *UPIS* i vrednosti signala *UPIS* pri prethodnoj usponskoj ivici *CLK* signala.

Pomoćna vrednost koja prihvata rezultat sabiranja četiri osmobarbitna broja je *ZBIR*. Sabiranjem četiri osmobarbitna broja najveće moguće vrednosti 0xFF dobija se desetobarbitni broj koji ima vrednost 0x3FC. Kako bi se obezbedilo da ne dođe do prekoračenja pri sabiranju definisano je da *ZBIR* bude desetobarbitni broj. Za jednostavan opis sabirača u VHDL-u koristi se *UNSIGNED* tip podatka za signale *BROJ0*, *BROJ1*, *BROJ2*, *BROJ3* i *ZBIR*. Sabiranje četiri sukcesivno zadate vrednosti, tj. vrednosti *BROJ0*, *BROJ1*, *BROJ2* i *BROJ3*, obavlja se izvan procesa, tj. preko čiste kombinacione mreže sabirača i podrške za rad sa celobrojnim brojevima (tip *UNSIGNED*).

```

ZBIR <= BROJ0 + BROJ1 + BROJ2 + BROJ3;

```

Usrednjavanja, tj. implementacija deljenja zbira četiri sukcesivno zadate vrednosti brojem četiri, ostvaruje se odbacivanjem dva bita najmanje težine iz desetobarbitnog broja *ZBIR*.

```

IZLAZ <= STD_LOGIC_VECTOR( ZBIR( 9 DOWNTO 2 ) );

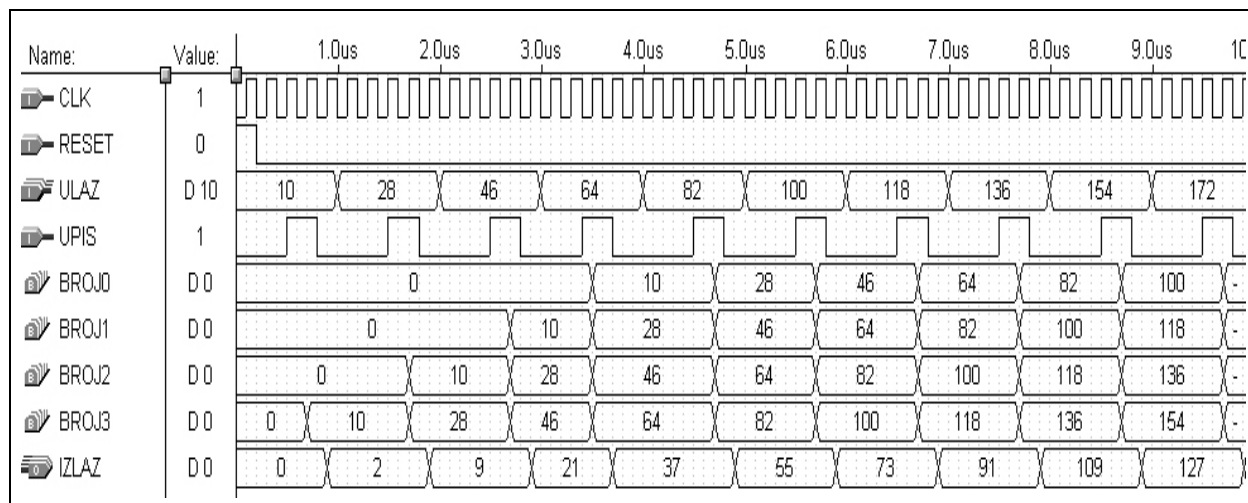
```

Baferisanje usrednjene vrednosti i prikazivanje preko signala *IZLAZ* obavlja se unutar procesa.

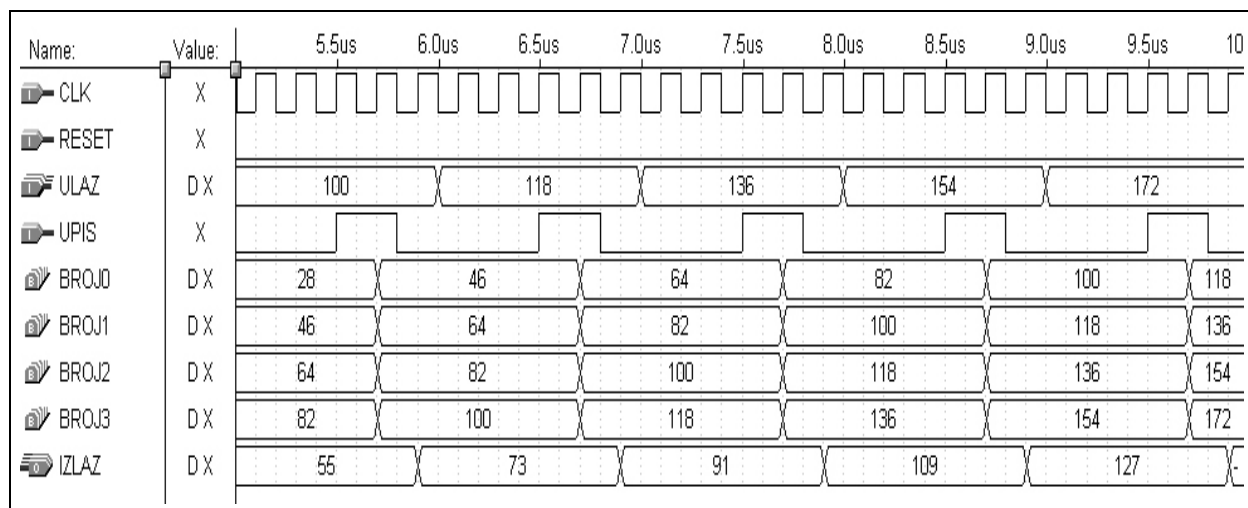
```

PROCESS ( CLK, RESET) BEGIN
  IF ( RESET = '1' ) THEN
    -- ...
  ELSIF ( rising_edge( CLK ) ) THEN
    -- ...
    IZLAZ <= STD_LOGIC_VECTOR( ZBIR( 9 DOWNTO 2 ) );
  END IF;
END PROCESS;

```



Slika 26.2: Simulacioni dijagram na početku rada sistema



Slika 26.3: Simulacioni dijagram u ustaljenom režimu rada sistema

Na slikama br. 26.2 i 26.3 dati su vremenski dijagrami iz procesa verifikacije sistema u okviru simulatora. Na slici br. 26.2 prikazan je početak rada sistema, dok je na slici br. 26.3 izdvojen rad sistema u toku ustaljenog režima, tj. kada sistem procesira četiri prethodno zadate vrednosti.

ZADATAK 27 – Generator impulsno-širinski modulisanog signala (PWM generator)

Realizovati generator impulsno-širinski modulisanog signala – PWM generator. PWM generator treba da radi na fiksnoj učestanosti i da ima mogućnost zadavanja širine impulsa signala preko spoljašnjeg signala koji se dovodi na osmobaritni ulaz *PULSE*. Na izlazu PWM generatora se dobija jednobitni signal *PWM_OUT*, čiji vrednost direktno zavisi od vrednosti dovedene na ulaz *PULSE* na sledeći način:

- *PULSE* = 0x00 $\Rightarrow$ na izlazu *PWM_OUT* se generiše logička nula (*PWM_OUT* = '0');
- *PULSE* = 0x80 $\Rightarrow$ na ulazu *PWM_OUT* će se dobiti povorka impulsa podjednake širine impulsa i pauze.
- *PULSE* = 0xFF $\Rightarrow$ na ulazu *PWM_OUT* će biti generisana logička jedinica (*PWM_OUT* = '1').

Izvršiti analizu PWM generatora i primenom softverskog paketa za rad sa programabilnim kolima firme ALTERA realizovati sistem VHDL opisom.

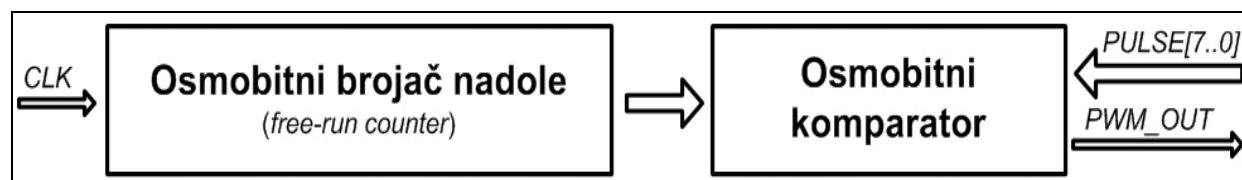
Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

Najjednostavniji način realizacije PWM generatora koji je specificiran u zadatku je preko:

- osmobaritnog brojača nadole i
- osmobaritnog komparatora.

U cilju pojednostavljenja dizajna, vrednost podatka *PULSE* direktno se dovodi na jedan od ulaza komparatora. Blok šema sistema data je na slici br. 27.1.



Slika 27.1: Blok šema PWM generatora

Osmobaritni brojač nadole je podsistem PWM generatora koji obezbeđuje neprekidno brojanje po sekvenci: 0xFF→0xFE→0xFD→...→0x02→0x01→0x00→0xFF→0xFE→... Vrednost na izlazu iz brojača upoređuje se sa podatkom na ulazu *PULSE*. U zavisnosti od odnosa ove dve vrednosti generiše se signal na izlazu *PWM_OUT*.

Kompletan VHDL opis sistema na bazi blok šeme sa slike br. 27.1 dat je u nastavku.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY pwm_gen IS PORT
(
  CLK      : IN   STD_LOGIC;
  PULSE    : IN   STD_LOGIC_VECTOR( 7 DOWNTO 0 );

```

```

    PWM_OUT : OUT STD_LOGIC
);
END pwm_gen;

ARCHITECTURE behav OF pwm_gen IS
    SIGNAL tmp : UNSIGNED( 7 DOWNT0 0 );
BEGIN
    PROCESS ( CLK ) BEGIN
        IF ( rising_edge( CLK ) ) THEN
            tmp <= tmp - 1;
            IF ( tmp < UNSIGNED( PULSE ) ) THEN
                PWM_OUT <= '1';
            ELSE
                PWM_OUT <= '0';
            END IF;
        END IF;
    END PROCESS;
END behav;

```

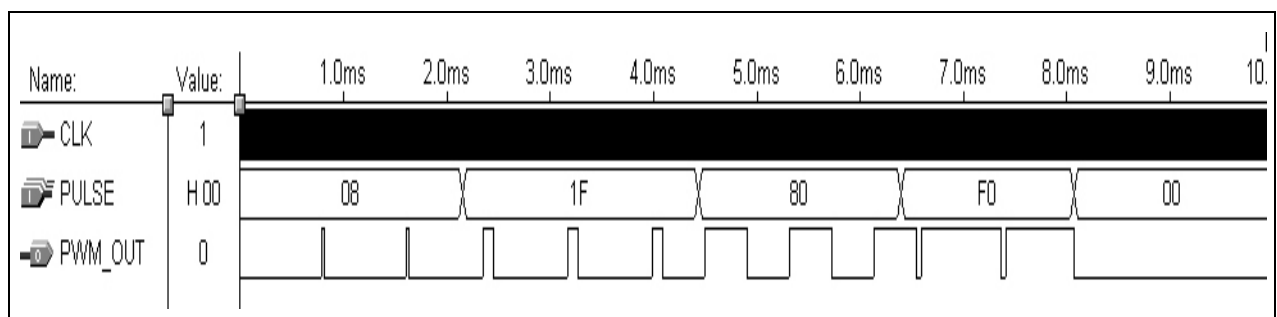
Kompletan sistem realizovan je preko procesa koji implementira sinhronu logiku. Proces je osetljiv na promenu signala *CLK*. Pri svakoj usponskoj ivici signala *CLK* dekrementira se vrednost brojača *tmp*. Generisanje signala na izlazu (*PWM_OUT*) obezbeđuje se pomoću komparatora koji se nalazi kao uslov unutar IF...ELSE...END IF strukture.

```

    IF ( tmp < UNSIGNED( PULSE ) ) THEN
        PWM_OUT <= '1';
    ELSE
        PWM_OUT <= '0';
    END IF;

```

Simulacioni dijagram kojim se verifikuje rad sistema prikazan je na slici br. 27.2. Rad sistema je testiran za vrednosti signala na ulazu 0x08, 0x1F, 0x80, 0xF0 i 0x00.



Slika 27.2: Simulacioni dijagram jednostavnog PWM generatora

ZADATAK 28 – Četvorobitni obostrani pomerački registar na bazi „barrel shifter“ kola

Realizovati sistem koji predstavlja četvorobitni programabilni pomerački registar, čiji sadržaj može da se pomera za zadati broj mesta u jednom taktnom intervalu (registar sa *barrel shifter* funkcijom). Sistem treba da sadrži signale kontrole dozvole upisa podatka u registar i dozvole pomeranja vrednosti registra u skladu sa funkcijom *barrel shifter* - a. Takođe, pomeranje vrednosti registra može da se obavlja nalevo i nadesno od jedne do tri pozicije u zavisnosti od vrednosti odgovarajućih kontrolnih signala *LEFT_RIGHT* i *CNT*.

Sistem sadrži sledeće signale na ulazu i izlazu:

- sinhronizacioni signal *CLK* (promena kada je aktivna usponska ivica);
- 4-bitni ulazni podatak *DATA_IN[3..0]*;
- 4-bitni izlazni podatak *DATA_OUT[3..0]*;
- jednobitni kontrolni signal *LD* kojim se aktivira funkcija upisa podataka *DATA_IN* u registar sistema (za nivo signala *LD*= '1' vrši se upis);
- jednobitni kontrolni signal *EN* kojim se dozvoljava pomeranje vrednosti registra u skladu sa osobinom *barrel shifter* kola (pri nivou signala *EN*= '1' dozvoljava se pomeranje vrednosti registra);
- jednobitna ulazna vrednost smeru pomeraja *LEFT_RIGHT* (za *LEFT_RIGHT*= '1' vrši se pomeranje ulevo tj. za *LEFT_RIGHT*= '0' pomeranje udesno) i
- dvobitni ulazni podatak *CNT[1..0]* koji definiše broj pozicija pomeraja vrednosti registra.

Sadržaj registra prisutan je na izlazu preko signala *DATA_OUT[7..0]*. Funkcija baferisanja, tj. paralelnog upisa vrednosti u registar, je većeg prioriteta od pomeranja vrednosti registra.

Realizovati sistem primenom softverskog paketa za rad sa programabilnim kolima firme ALTERA u VHDL jeziku.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

Formirati simbol opisanog sistema u VHDL-u.

Osmisliti i opisati način verifikacije sistema na UP2 razvojnoj ploči i EPM7128LC84-7 čipu iz familije kola MAX7000S.

REŠENJE

Barrel shifter registar je ustvari složenija varijanta kružnog pomeračkog registra. Kod klasičnih kružnih pomeračkih registara u jednom taktnom intervalu, ukoliko je izabran smer pomeranja udesno, bit najmanje težine napušta svoju poziciju, dolazi na mesto bita najveće težine, dok se istovremeno celokupan sadržaj registra pomera za jedno mesto udesno. Slično, kod pomeranja ulevo, bit najveće težine napušta svoju poziciju, dolazi na mesto bita najmanje težine, dok se istovremeno celokupan sadržaj registra pomera za jednu poziciju ulevo. Po istom principu radi i *barrel shifter*, s tim što broj pomeranja bita rotacijom u jednom taktnom intervalu, kao i smer rotacije mogu da se zadaju spoljašnjim kontrolnim signalima, zbog čega se ovaj tip registra zove i obostrani programabilni pomerački registar.

Realizacija *barrel shifter* registra, koja je data VHDL opisom, izneta je u nastavku:

```

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY barrel_shifter IS PORT
(
  CLK, LD, EN      : IN  STD_LOGIC;
  LEFT_RIGHT       : IN  STD_LOGIC;
  CNT              : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
  DATA_IN         : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
  DATA_OUT        : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
);
END barrel_shifter;

ARCHITECTURE behav OF barrel_shifter IS
  SIGNAL tmp_reg      : STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL tmp_shifter  : STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL tmp_cnt      : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
  -- Funkcija Barrel Shifter kola - kombinaciona mreza
  tmp_cnt <= LEFT_RIGHT & CNT;
  WITH tmp_cnt SELECT
    tmp_shifter(3 DOWNTO 0)
      <= tmp_reg(3 DOWNTO 0) WHEN "000", -- nema pomeranja udesno
        tmp_reg(3 DOWNTO 0) WHEN "100", -- nema pomeranja ulevo
  -- pomeranje za jedno mesto udesno/ulevo
    tmp_reg(0) & tmp_reg(3 DOWNTO 1) WHEN "001",
    tmp_reg(2 DOWNTO 0) & tmp_reg(3) WHEN "101",
  -- pomeranje za dva mesta udesno/ulevo
    tmp_reg(1 DOWNTO 0) & tmp_reg(3 DOWNTO 2) WHEN "010",
    tmp_reg(1 DOWNTO 0) & tmp_reg(3 DOWNTO 2) WHEN "110",
  -- pomeranje za tri mesta udesno/ulevo
    tmp_reg(2 DOWNTO 0) & tmp_reg(3) WHEN "011",
    tmp_reg(0) & tmp_reg(3 DOWNTO 1) WHEN "111",
    "00000000" WHEN OTHERS;

  -- Registar - sekvencijalna mreza
  PROCESS ( CLK ) BEGIN
    -- Registarska funkcija sistema
    IF ( rising_edge( CLK ) ) THEN
      IF ( LD = '1' ) THEN
        tmp_reg <= DATA_IN;
      ELSIF ( EN = '1' ) THEN
        tmp_reg <= tmp_shifter;
      END IF;
    END IF;
    DATA_OUT <= tmp_reg;
  END PROCESS;
END behav;

```

Realizacija sistema obavlja se kroz dve celine VHDL koda i to:

- kombinacionog kola kojim se realizuje funkcija *barrel shifter* pomerača i

- sekvencijalne mreže kojom se realizuju prioritetne funkcije sistema i funkcija prihvatnog registra.

Kombinaciona mreža koja implementira način rada *barrel shifter* pomerača realizovana je preko *WITH...SELECT* strukture.

```
tmp_cnt <= LEFT_RIGHT & CNT;
WITH tmp_cnt SELECT
    tmp_shifter(3 DOWNT0 0)
    <= tmp_reg(3 DOWNT0 0) WHEN "000", -- nema pomeranja udesno
    tmp_reg(3 DOWNT0 0) WHEN "100", -- nema pomeranja ulevo
    -- pomeranje za jedno mesto udesno/ulevo
    tmp_reg(0) & tmp_reg(3 DOWNT0 1) WHEN "001",
    tmp_reg(2 DOWNT0 0) & tmp_reg(3) WHEN "101",
    -- pomeranje za dva mesta udesno/ulevo
    tmp_reg(1 DOWNT0 0) & tmp_reg(3 DOWNT0 2) WHEN "010",
    tmp_reg(1 DOWNT0 0) & tmp_reg(3 DOWNT0 2) WHEN "110",
    -- pomeranje za tri mesta udesno/ulevo
    tmp_reg(2 DOWNT0 0) & tmp_reg(3) WHEN "011",
    tmp_reg(0) & tmp_reg(3 DOWNT0 1) WHEN "111",
    "00000000" WHEN OTHERS;
```

WITH...SELECT struktura karakteristična je za realizaciju kombinacionih mreža. Radi jednostavnije analize pomeranja vrednosti do tri pozicije na levu ili desnu stranu, formira se pomoćni signal *tmp_cnt* koji objedinjuje kontrolne signale pomeranja *LEFT_RIGHT* i *CNT*. Vrednost na izlazu kombinacione mreže (*tmp_shifter*) određuje se za sve kombinacije novoformiranog signala *tmp_cnt*.

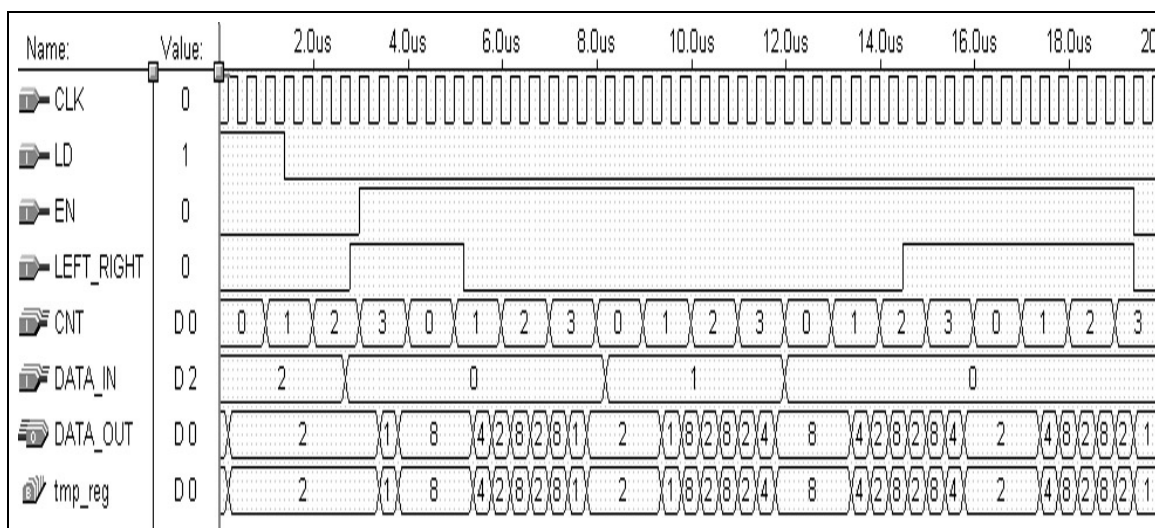
Sinhroni deo sistema realizuje se preko procesa koji reaguje na promenu vrednosti signala *CLK*.

```
PROCESS ( CLK ) BEGIN
    -- Registarska funkcija sistema
    IF ( rising_edge( CLK ) ) THEN
        IF ( LD = '1' ) THEN
            tmp_reg <= DATA_IN;
        ELSIF ( EN = '1' ) THEN
            tmp_reg <= tmp_shifter;
        END IF;
    END IF;
    DATA_OUT <= tmp_reg;
END PROCESS;
```

Sve akcije u sistemu dešavaju se pri usponskoj ivici *CLK* signala što određuje prva *IF...END IF* struktura i uslov *rising_edge(CLK)*. Druga ugnježdjena *IF...ELSIF...END IF* struktura određuje prioritet operacija u sistemu na bazi kontrolnih signala *LD* i *EN*. Kada je ispunjen uslov rada sistema u funkciji *barrel shifter* pomerača u registar se upisuje rezultat kombinacionog dela sistema koji je prethodno opisan.

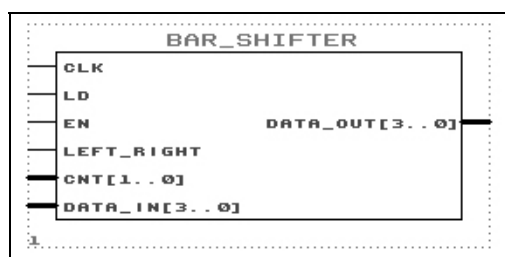
```
tmp_reg <= tmp_shifter;
```

Na slici br. 28.1 dat je jedan od simulacionih dijagrama kojim se proverava ispravnost rada projektovanog sistema sa funkcijom *barrel shifter* pomerača.

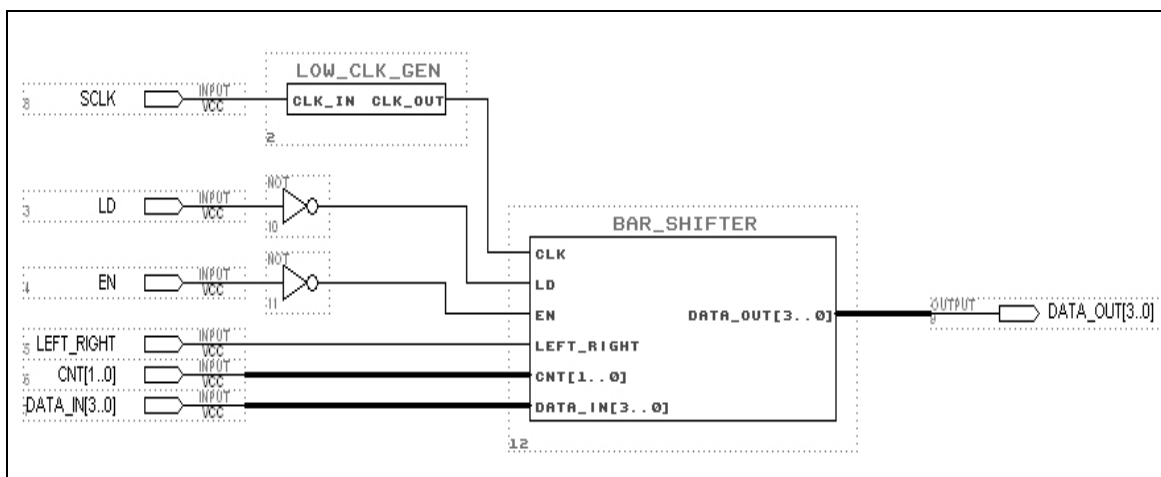


Slika 28.1: Vremenski dijagram simulacije

Simbol realizovanog sistema prikazan je na slici br. 28.2. Verifikacija realizovanog sistema na UP2 razvojnoj ploči može se obaviti preko test šeme koja je prikazana na slici br. 28.3.

Slika 28.2: Simbol opisanog sistema *barrel shifter* pomerača

Povezanost signala sa test šeme na slici br. 28.3 i pinova EPM7128SLC84-6 čipa na UP2 ploči dat je u tabeli br. 28.1.



Slika 28.3: Test šema za verifikaciju realizovanog sistema na UP2 ploči

Na slici br. 28.3 uočava se bloka *LOW_CLK_GEN* koji služi da obezbedi takt od oko 1Hz za potrebe testiranja sistema preko eksternih prekidača.

Tabela 28.1: Tabela povezanosti signala za verifikaciju rada sistema na UP2 razvojnoj ploči

<i>Signal</i>	<i>Element na UP2 ploči</i>	<i>Opis</i>
<i>SCLK</i>	pin 83	Veza sa sistemskim taktom od 25,175MHz.
<i>LD</i>	MAX_PB1	Izdavanje zahteva za upis vrednosti u registar preko tastera PB1.
<i>EN</i>	MAX_PB2 prekidač 7	Izdavanje zahteva za pomeranje vrednosti preko tastera PB2.
<i>LEFT_RIGHT</i>	MAX_SW1 pinovi 5 i 4	Zadavanje smeru pomeranja preko DIP prekidača.
<i>CNT[1..0]</i>	MAX_SW1 prekidač 5 i 4	Zadavanje broja pomeranja pozicija preko DIP prekidača.
<i>DATA_IN[1..0]</i>	MAX_SW1 prekidač 3, 2, 1 i 0	Unos vrednosti za pomeranje preko DIP prekidača.
<i>DATA_OUT[1..0]</i>	LED D1, LED D2, LED D3, LED D4	Prikaz sadržaja registra preko LED indikatora.

ZADATAK 29 – Jednostavan sistem za upravljanje radom motora

Realizovati sistem koji generiše kontrolne signale za upravljanje radom motora. Motor se koristi za pokretanje kabine lifta pri čemu može da ostvari kretanje kabine nagore, nadole ili da zadrži kabinu u zatečenom stanju (da je ne pomera).

Kontrolni signal na ulazu sistema je jednobitni signal *ULAZ*. Preko ovog kontrolnog signala zadaje se smer okretanja osovine motora i smer kretanja kabine lifta. Na izlazu iz sistema prisutna su dva jednobitna signala *GORE* i *DOLE* kojima se aktivira pojačavački stepen motora u cilju pokretanja motora. U tabeli br. 29.1. dat je opis generisanja signala na izlazu u zavisnosti od signala na ulazu kao i specifikacija određenih kombinacija signala.

Tabela 29.1: Opis generisanja signala na izlazu u funkciji signala na ulazu

<i>Signal</i>			<i>opis</i>
<i>ULAZ</i>	<i>GORE</i>	<i>DOLE</i>	
0	0	1	Motor pokreće kabinu lifta na dole.
1	1	0	Motor pokreće kabinu lifta na gore.
nedefinisano	0	0	Motor ne pokreće kabinu lifta. Kabina ostaje u zatečenom stanju.
nedefinisano	1	1	Stanje koje nije dozvoljeno

Pored osnovne funkcije koju sistem treba da obezbedi u pogledu generisanja signala na izlazu (tabela br. 29.1), potrebno je realizovati funkciju blokade oba kontrolna signala *GORE* i *DOLE* nakon promene signala na ulazu. Nakon promene vrednosti signala na ulazu, sistem treba da obezbedi blokadu signala na izlazu *GORE* i *DOLE* u trajanju od 1msec, tj. nakon promene vrednosti *ULAZ* 0→1 ili 1→0, sistem generiše stanje u kojem je *GORE*=*DOLE*=’0’ u trajanju od 1msec nakon čega nastavlja rad u skladu sa opisom funkcionalnosti sistema iz tablice br. 29.1. Na izlazu ne smeju istovremeno biti aktivni upravljački signali *GORE* i *DOLE*, tj. ne sme se ni u jednom trenutku javiti stanje u kojem je *GORE*=*DOLE*=’1’.

Sistem treba realizovati kao sinhronu mrežu pri čemu je sinhronizacioni signal *CLK*. Kontrolni signal *ULAZ* ne mora biti sinhron sa *CLK* signalom. Treba obezbediti da signali na

izlazu *GORE* i *DOLE* budu sinhroni sa *CLK* signalom. Sinhronizacioni takt je učestanosti 100kHz.

Primenom softverskog paketa za rad sa programabilnim kolima firme ALTERA realizovati mrežu VHDL opisom.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

Centralno mesto u opisanom sistemu za generisanje signala za upravljanje radom lifta je retriggerabilni tajmer koji stavlja blokadu i daje dozvolu za generisanje kontrolnih signala na izlazu. Blokada signala *GORE* i *DOLE* treba da traje najmanje 1msec i to tačno 1msec od poslednje promene vrednosti signala na ulazu.

Za merenje vremenskog intervala od 1msec koristi se sinhronizacioni signal *CLK* učestanosti 100kHz i trajanja periode 10 µsec. U toku vremenskog intervala od 1msec potrebno je izbrojati 100 perioda sinhronizacionog signala *CLK*. Tajmer za ovu primenu realizuje se preko internog brojača nadole. Iz tih razloga, interni brojač koji obavlja funkciju tajmera treba da bude realizovan kao sedmobitni brojač koji broji od vrednosti 99_{10} (1100011_2) do 0. Aktiviranje rada brojača obavlja se detekcijom promene vrednosti ulaznog signala.

VHDL opis zahtevanog sistema dat je u nastavku.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY zadatak IS PORT
(
    CLK : IN STD_LOGIC;
    ULAZ : IN STD_LOGIC;
    GORE : OUT STD_LOGIC;
    DOLE : OUT STD_LOGIC
);
END zadatak ;

ARCHITECTURE opis OF zadatak IS
    SIGNAL tmpTajmer : UNSIGNED( 6 DOWNTO 0 );
    SIGNAL tmpUlazSinh : STD_LOGIC;
    SIGNAL tmpPromenaUlaza : STD_LOGIC;
    SIGNAL tmpBlokadaTajmera : STD_LOGIC;

BEGIN
    PROCESS ( CLK ) BEGIN
        IF ( rising_edge( CLK ) ) THEN
            tmpUlazSinh <= ULAZ;
            tmpPromenaUlaza <= tmpUlazSinh XOR ulaz;

            IF ( ( tmpPromenaUlaza ) = '1' ) THEN
                tmpTajmer <= "1100011";
                tmpBlokadaTajmera <= '0';
            ELSE
                IF ( tmpTajmer = 0 ) THEN
                    tmpTajmer <= tmpTajmer;
                    tmpBlokadaTajmera <= '1';
                ELSE
                    tmpTajmer <= tmpTajmer - 1;
                END IF
            END IF
        END IF
    END PROCESS

```

```

        ELSE
            tmpTajmer <= tmpTajmer - 1;
            tmpBlokadaTajmera <= '0';
        END IF;
    END IF;
    GORE <= tmpBlokadaTajmera AND NOT ( tmpPromenaUlaza ) AND
            tmpUlazSinh;
    DOLE <= tmpBlokadaTajmera AND NOT ( tmpPromenaUlaza ) AND
            NOT ( tmpUlazSinh );
    END IF;
END PROCESS;
END opis;

```

Za realizaciju zahtevanog sistema potrebno je definisati nekoliko pomoćnih signala:

```

SIGNAL    tmpTajmer           : UNSIGNED( 6 DOWNTO 0 );
SIGNAL    tmpUlazSinh        : STD_LOGIC;
SIGNAL    tmpPromenaUlaza    : STD_LOGIC;
SIGNAL    tmpBlokadaTajmera  : STD_LOGIC;

```

Svi unutrašnji registri, brojači i pomoćni signali definišu se kao unutrašnji signali sistema. Izbor tipa i dužine signala zavisi od njihove primene. Na samom početku realizacije zadatka nije moguće odrediti tačan broj pomoćnih signala tako da se njihov izbor i definisanje obavlja u toku procesa realizacije prema stvorenim potrebama.

Za regularan rad sinhronne logike, tj. sistema u celini, potrebno je obaviti sinhronizaciju ulaznih signala sa *CLK* signalom takta. To se postiže u okviru procesa na sledeći način:

```

PROCESS ( CLK ) BEGIN
    IF ( rising_edge( CLK ) ) THEN
        tmpUlazSinh      <= ULAZ;
        tmpPromenaUlaza <= tmpUlazSinh XOR ulaz;
        -- ...
    END PROCESS;

```

Pomoćni signal *tmpUlazSinh* predstavlja baferisan i sinhronisan ulazni signal *ULAZ* preko jednog D-FF. Drugi pomoćni signal *tmpPromenaUlaza* je takođe sinhron sa *CLK* signalom i nosi informaciju o promeni vrednosti signala na ulazu. Ukoliko se detektuje promena signala na ulazu u odnosu na dve aktivne ivice *CLK* signala, *tmpPromenaUlaza* dobija vrednost 1 u trajanju od jedne periode *CLK* signala. Ovaj signal se koristi za aktiviranje rada tajmera u cilju merenja vremena.

Preko sedmobitnog signala *tmpTajmer* koji je tipa *UNSIGNED* i dela koda koji je dat u nastavku:

```

IF ( ( tmpPromenaUlaza ) = '1' ) THEN
    tmpTajmer <= "1100011";
    tmpBlokadaTajmera <= '0';
ELSE
    IF ( tmpTajmer = 0 ) THEN
        tmpTajmer <= tmpTajmer;
        tmpBlokadaTajmera <= '1';
    ELSE
        tmpTajmer <= tmpTajmer - 1;
    END IF;
END IF;

```

```

        tmpBlokadaTajmera <= '0';
    END IF;
END IF;

```

realizovan je brojač koji obavlja funkciju tajmera. Tajmer daje zabranu i dozvolu generisanja signala na izlazu preko pomoćnog jednobitnog signala *tmpBlokadaTajmera*. Aktiviranje rada tajmera, tj. aktiviranje mehanizma merenja vremena dužine 1msec, obavlja se preko signala *tmpPromenaUlaza*.

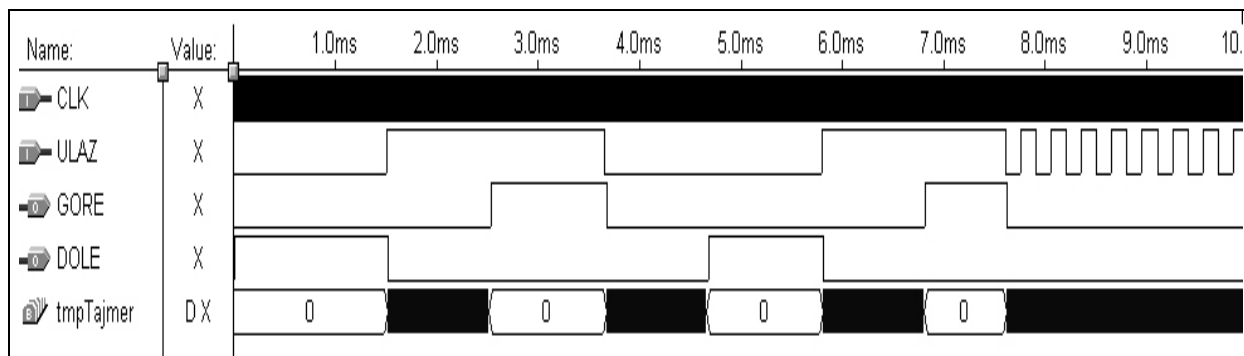
Kako bi signali na izlazu bili sinhroni sa *CLK* signalom, blok koji generiše signale *GORE* i *DOLE*:

```

GORE <= tmpBlokadaTajmera AND NOT(tmpPromenaUlaza) AND
        tmpUlazSinh;
DOLE <= tmpBlokadaTajmera AND NOT(tmpPromenaUlaza) AND
        NOT ( tmpUlazSinh );

```

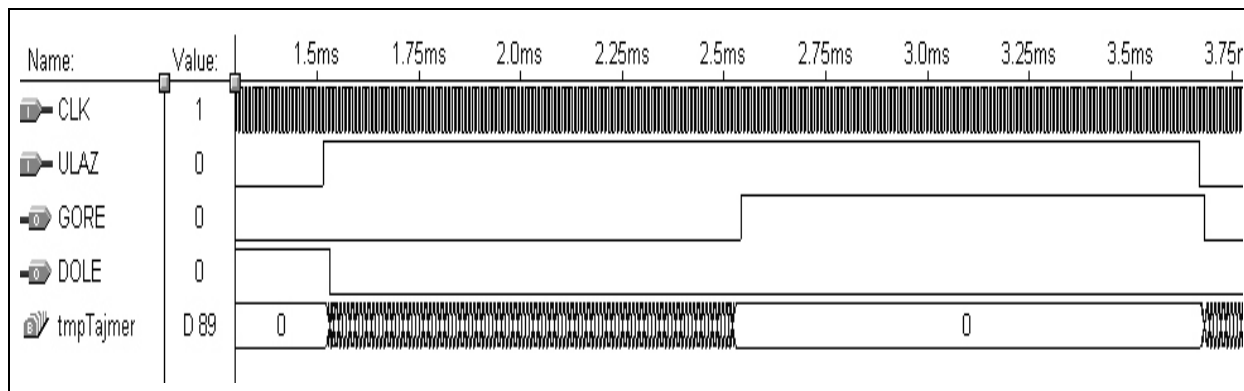
nalazi se unutar procesa i dela koda koji se izvršava sinhrono sa *CLK* signalom.



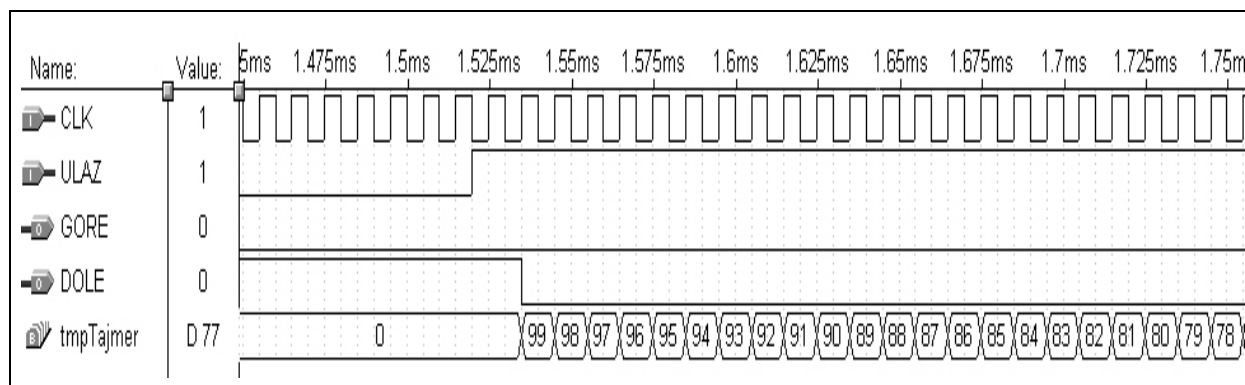
Slika 29.1: Simulacioni dijagram

Na slikama br. 29.1, 29.2 i 29.3 dati su simulacioni dijagrami kojima se verifikuje rad opisanog sistema.

Na slici br. 29.1 uočava se mogućnost blokade rada motora adekvatnim izborom signala na ulazu. Ukoliko se na *ULAZ* dovede signal učestanosti veće od 500Hz doći će do blokade rada motora (*GORE*=*DOLE*='0'). Ovo se dešava zato što je tajmer realizovan preko retrigirabilnog brojača i pri detekciji svake promene signala na ulazu dolazi do ponovnog aktiviranja mehanizma merenja vremena od 1msec. Ovaj efekat se uočava od 7.5msec do 10msec na vremenskom dijagramu na slici br. 29.1.



Slika 29.2: Simulacioni dijagram



Slika 29.3: Simulacioni dijagram

Na slici br. 29.2 i 29.3 pokazan je mehanizam blokade signala *GORE* i *DOLE* nakon detekcije promene vrednosti na ulazu kao i rad brojača.

ZADATAK 30 – RAM memorija

Realizovati RAM memoriju koja je specificirana sledećim parametrima:

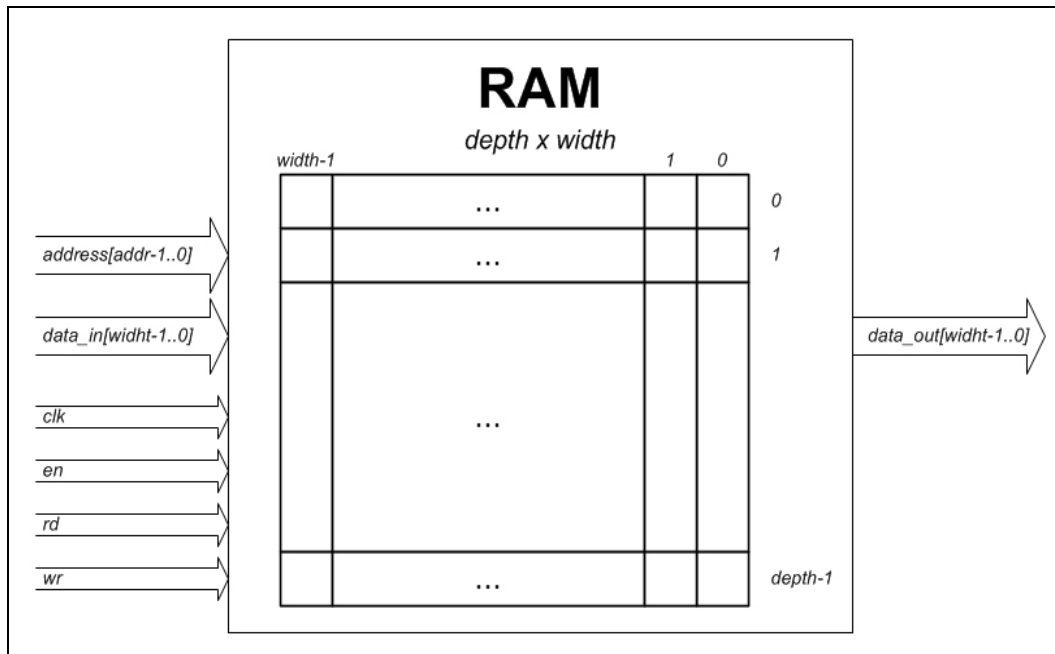
- *width* – širina podatka;
- *addr* – broj adresnih linija memorije i
- *depth* – broj podataka u memoriji.

Simbol RAM memorije sa spoljašnjim priključcima prikazan je na slici br. 30.1. RAM memorija sadrži sledeće priključke:

- *clk* – sinhronizacioni signal (usponska ivica je aktivna promena signala *clk*);
- *en* – signal dozvole pristupa memoriji (dozvola pristupa pri *en*= '1');
- *rd* – signal zahteva za čitanjem podataka iz memorije (čitanje pri *rd*= '1');
- *wd* – signal zahteva za upisom podatka u memoriju (upis pri *wr*= '1');
- *address[addr-1..0]* – signali adresiranja memorije;
- *data_in[width-1..0]* – signali podataka na ulazu (podaci za upis u memoriju) i
- *data_out[width-1..0]* – signali podataka na izlazu (podaci koji su pročitani iz memorije).

Primenom softverskog paketa za rad sa programabilnim kolima firme ALTERA realizovati RAM memoriju VHDL opisom. VHDL realizacija RAM memorije treba da ima mogućnost jednostavne i brze konfiguracije, tj. mogućnost promene konfiguracije samo izmenom odgovarajućih parametara. Na ovaj način može se formirati memorija željenog kapaciteta, npr. 10x8 bita ili 16x7 bita.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema. Za potrebe verifikacije formirati RAM memoriju kapaciteta 4x4 bita tj. RAM koji ima parametre *width*=4, *addr*=2 i *depth*=4.



Slika 30.1: Blok šema RAM memorije

REŠENJE

Realizacija zahtevane RAM memorije u VHDL data je u nastavku.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY RAM IS
GENERIC
(
    width      : INTEGER:=4;
    depth      : INTEGER:=4;
    addr       : INTEGER:=2
);
PORT
(
    clk, en    : IN STD_LOGIC;
    rd, wr     : IN STD_LOGIC;
    address    : IN STD_LOGIC_VECTOR( addr-1 DOWNTO 0 );
    data_in    : IN STD_LOGIC_VECTOR( width-1 DOWNTO 0 );
    data_out   : OUT STD_LOGIC_VECTOR( width-1 DOWNTO 0 );
);
END RAM;

ARCHITECTURE behav OF RAM IS
    TYPE ram_type is ARRAY( 0 to depth-1 ) OF
        STD_LOGIC_VECTOR( width-1 DOWNTO 0 );
    SIGNAL tmp_ram: ram_type;
BEGIN
    PROCESS(clk, rd)
    BEGIN
        IF( rising_edge( clk ) ) THEN

```

```

        IF( en='1' )THEN
            IF( rd='1' )THEN
                data_out <= tmp_ram( conv_integer( address ) );
            ELSE
                data_out <= ( data_out'range => '0' );
            END IF;
        ELSE
            data_out <= ( data_out'range => '0' );
        END IF;
    END IF;
END PROCESS;

PROCESS(clk, wr)
BEGIN
    IF( rising_edge( clk ) ) THEN
        IF( en='1' )THEN
            IF( wr='1' )THEN
                tmp_ram( conv_integer( address ) ) <= data_in;
            END IF;
        END IF;
    END IF;
END PROCESS;
END behav;

```

Implementacija lako promenljivog VHDL koda u pogledu širine podataka sa kojima se manipuliše postiže se korišćenjem parametrizovanog opisa. Ključna reč koja karakteriše parametrizovani VHDL opis je *GENERIC*. Parametrizovani VHDL opis karakteriše se time što se na početku koda, u okviru opisa interfejsa, definišu parametri koji se u daljem kodu koriste kao podaci o veličini podataka. Izmena širine reči, tj. veličine podataka, postiže se izmenom samo vrednosti parametara na jednom mestu u okviru opisa. Interfejs RAM memorije počinje kodom koji specificira parametre sistema:

```

ENTITY RAM IS
GENERIC
(
    width      : INTEGER:=4;
    depth      : INTEGER:=4;
    addr       : INTEGER:=2
);
PORT
(
    -- ...
);
END RAM;

```

Ovim se definišu parametri *width*, *depth* i *addr* tj. širina podatka, broj podataka i broj adresnih linija. Za definisanje širine linija signala na ulazu, izlazu kao i u ostatku VHDL koda specificirani parametri se koriste na isti način kao i konstante. Sam interfejs RAM memorije opisan je sledećim kodom:

```

PORT
(
  clk, en  : IN STD_LOGIC;
  rd, wr   : IN STD_LOGIC;
  address  : IN STD_LOGIC_VECTOR( addr-1 DOWNTO 0 );
  data_in  : IN STD_LOGIC_VECTOR( width-1 DOWNTO 0 );
  data_out : OUT STD_LOGIC_VECTOR( width-1 DOWNTO 0 )
);
END RAM;

```

Definisanje jezgra RAM memorije (memorijskih ćelija), postiže se korišćenjem matrice određene veličine i širine podatka.

```

TYPE ram_type is ARRAY( 0 to depth-1 ) OF
  STD_LOGIC_VECTOR( width-1 DOWNTO 0 );
SIGNAL tmp_ram: ram_type;

```

Za ovu potrebu definiše se novi tip podatka *ram_type* koji se koristi za specifikaciju jezgra memorije *tmp_ram*. Memorisjko jezgro (*tmp_ram*) definisano je kao matrica kapaciteta *depth* x *width* biti.

Definisanje mehanizma kontrole upisa i čitanja podataka RAM memorije realizuje se preko dva nezavisna procesa. Prvi proces obezbeđuje čitanje podataka iz RAM memorije i osetljiv je na promenu vrednosti signala *clk* i *rd*. Adresiranje podataka u okviru matrice *tmp_ram* postiže se na osnovu informacije koju nosi adresa, odnosno signali *address*. Kako bi se signali *address* prilagodili da adresiraju podatke u okviru matrice *tmp_ram* koristi se pomoćna funkcija *conv_integer*.

```

data_out <= tmp_ram( conv_integer( address ) );

```

Pomoćna funkcija *conv_integer* definisana je u okviru paketa *ieee.std_logic_unsigned.all* koji se poziva u bloku specifikacije biblioteka tj. na početku koda.

Drugi proces realizuje upis u RAM memoriju i osetljiv je na promenu signala *clk* i *wr*. Pri upisu, adresiranje memorije obezbeđuje se na isti način kao i pri čitanju.

```

tmp_ram( conv_integer( address ) ) <= data_in;

```

Dodeljivanje fiksne vrednosti podatka (sve nule ili sve jedinice) čija je širina reči određena parametrom, postiže se pozivanjem operacije *range*.

```

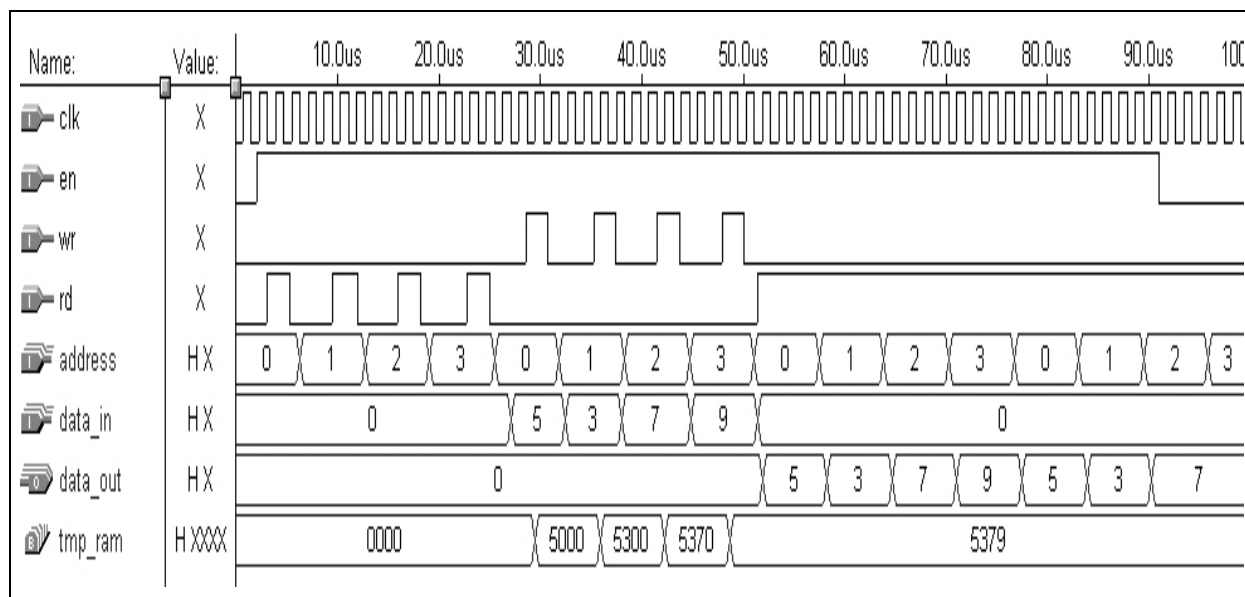
data_out <= ( data_out'range => '0' );

```

Parametrizovani VHDL opis treba shvatiti kao opis sistema preko konstanti koje se specificiraju samo na jednom mestu u okviru dizajna i da se na dalje koriste kao i bilo koja druga fiksna vrednost. Svakako treba razumeti da prevodilac VHDL koda raspolaže sa informacijom o tačnim zahtevima sistema u pogledu kapaciteta, tj. da se konfiguracija sistema ne može dinamički menjati u toku rada sistema.

Na slici br. 30.2. prikazan je vremenski dijagram simulacije rada opisane RAM memorije u VHDL jeziku. Od početka simulacije u trajanju od 15 μ sec vrši se testiranje sadržaja memorije sukcesivnim čitanjem sadržaja sve četiri adrese RAM memorije. Nakon čitanja, od 25 do 50 μ sec upisuje se sadržaj 5, 3, 7 i 9 redom u lokacije RAM memorije: 0x0,

0x1, 0x2 i 0x3. Od 50 μ sec do 90 μ sec obavlja se provera prethodno obavljenog upisa ponovnim čitanjem sadržaja memorije.

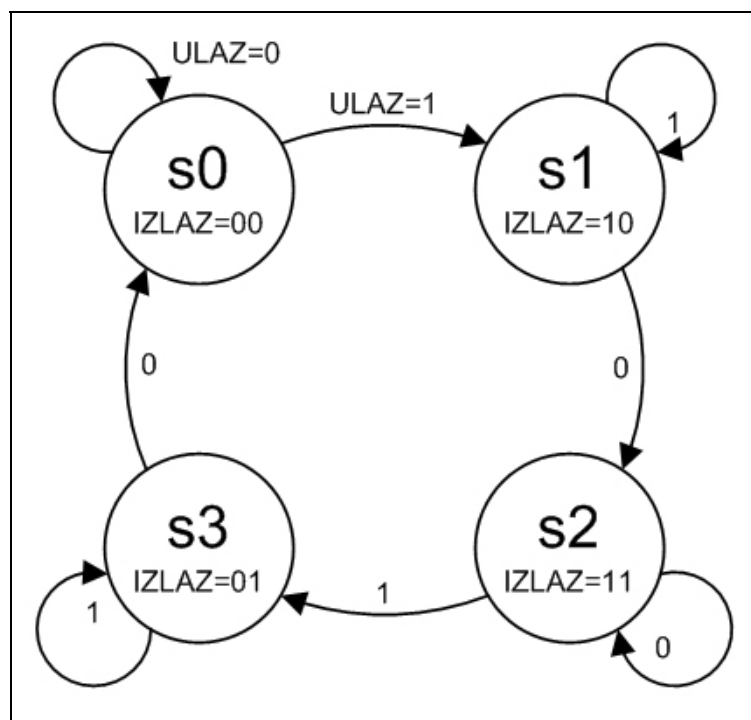


Slika 30.2: Vremenski dijagram simulacije

ZADATAK 31 – Realizacija sistema specificiranog preko dijagrama stanja Murovog tipa

Realizovati digitalnu sekvencijalnu mrežu Murovog tipa koja je specificirana preko dijagrama mašine stanja. Kontrolni i upravljački signali interfejsa sistema su:

- jednobitni ulazni signal *ULAZ* i
- dvobitni signal na izlazu *IZLAZ*.



Slika 31.1: Dijagram stanja

Pored signala *ULAZ* i *IZLAZ*, interfejs sistema sadrži i standardne signale digitalne sekvencijalne mreže: *CLK* i *RESET*, tj. sinhronizacioni signal i signal asinhronog reseta. Na slici br. 31.1 prikazan je dijagram stanja koji opisuje sistem.

Rad sistema opisan je kroz četiri stanja: *s0*, *s1*, *s2* i *s3*. Sistem menja stanje pri usponskoj ivici *CLK* signala u zavisnosti od vrednosti signala na ulazu *ULAZ*. Asinhroni reset, koji ima najveći prioritet izvršavanja, aktivan je pri vrednosti signala *RESET*='1'. Prilikom asinhronog reseta sistem prelazi u stanje *s2*.

Primenom softverskog paketa za rad sa programabilnim kolima firme ALTERA realizovati mrežu VHDL opisom.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

Pored dela VHDL opisa sistema koji se odnosi na specifikaciju interfejsa, realizacija sadrži tri karakteristične celine koda i to:

- definisanje mehanizma rada i signala kojim se specificira mašina stanja sistema;
- definisanje procesa promene stanja sistema i
- definisanje signala na izlazu.

Realizacije prve dve faze opisa sistema na bazi dijagrama stanja iste su za mreže Murovog i Milijevog tipa. Treća faza, tj. definisanje signala na izlazu, neznatno se razlikuje kod mreža ova dva tipa. Realizacije mašine stanja u VHDL jeziku omogućava veliku fleksibilnost u implementaciji i mogućnost realizacije različitih specifičnosti.

U nastavku je dat VHDL opis zahtevanog sistema koji je specificiran preko dijagrama stanja na slici br. 31.1.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY zadatak IS PORT
(
    CLK    : IN  STD_LOGIC;
    RESET  : IN  STD_LOGIC;
    ULAZ   : IN  STD_LOGIC;
    IZLAZ  : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
);
END zadatak ;
ARCHITECTURE moore OF zadatak IS
    -- Definisanje mehanizma masine stanja
    TYPE stanje IS (s0, s1, s2, s3);
    SIGNAL masina_stanja : stanje;
BEGIN
    -- Definisanje procesa promene stanja
    PROCESS ( CLK, RESET ) BEGIN
        IF ( RESET = '1' ) THEN
            -- Asinhrona promena- asinhroni reset
            masina_stanja <= s2;
        ELSIF ( rising_edge( CLK ) ) THEN
            -- Sinhrona promena - promena stanja sistema
            CASE masina_stanja IS
                WHEN s0 =>
                    IF ( ULAZ='1' ) THEN
                        masina_stanja <= s1;
                    END IF;

```

```

        WHEN s1 =>      IF ( ULAZ='0' ) THEN
                        masina_stanja <= s2;
                        END IF;
        WHEN s2 =>      IF ( ULAZ='1' ) THEN
                        masina_stanja <= s3;
                        END IF;
        WHEN s3 =>      IF ( ULAZ='0' ) THEN
                        masina_stanja <= s0;
                        END IF;

        END CASE;
    END IF;
END PROCESS;
-- Definisanje signala na izlazu
PROCESS ( masina_stanja ) BEGIN
    CASE masina_stanja IS
        WHEN s0 => IZLAZ <= "00";
        WHEN s1 => IZLAZ <= "10";
        WHEN s2 => IZLAZ <= "11";
        WHEN s3 => IZLAZ <= "01";
    END CASE;
END PROCESS;
END moore;

```

Definisanje mehanizma rada mašine stanja, tj. signala koji nose informaciju o stanju u kojem se sistem nalazi, obavlja se na sledeći način:

```

TYPE stanje IS (s0, s1, s2, s3);
SIGNAL masina_stanja : stanje;

```

Tip *stanje* karakteriše signal koji može da bude u jednom četiri različita stanja: *s0*, *s1*, *s2* i *s3*. Signal *masina_stanja* je tipa *stanje* i opisuje centralni mehanizam sekvencijalne mreže, tj. registar stanja.

Druga faza u toku realizacije sistema na bazi dijagrama stanja je specifikacija promene stanja sistema. Definisanje prelaska sistema iz stanja u stanje sprovodi se preko procesa koji je osetljiv na promenu signala *CLK* i *RESET*. Aktivna vrednost signala *RESET* (*RESET*='1'), asinhrono prevodi sistem u stanje *s3*. Ukoliko *RESET* nije aktivan (*RESET*='0') sistem se sinhrono kreće kroz stanja u zavisnosti od trenutnog stanja sistema i vrednosti signala na ulazu *ULAZ*. Za ovu primenu najpogodnije je koristiti *CASE...WHEN* i *IF...ELSE...END IF* strukture.

```

PROCESS ( CLK, RESET ) BEGIN
    IF ( RESET = '1' ) THEN
        -- Asinhrona promena- asinhroni reset
        masina_stanja <= s2;
    ELSIF ( rising_edge( CLK ) ) THEN
        -- Sinhrona promena - promena stanja sistema
        CASE masina_stanja IS
            WHEN s0 =>      IF ( ULAZ='1' ) THEN
                            masina_stanja <= s1;
                            END IF;
            WHEN s1 =>      IF ( ULAZ='0' ) THEN
                            masina_stanja <= s2;
                            END IF;
            WHEN s2 =>      IF ( ULAZ='1' ) THEN
                            masina_stanja <= s3;

```

```

                END IF;
        WHEN s3 =>
            IF ( ULAZ='0' ) THEN
                masina_stanja <= s0;
            END IF;
        END CASE;
    END IF;
END PROCESS;

```

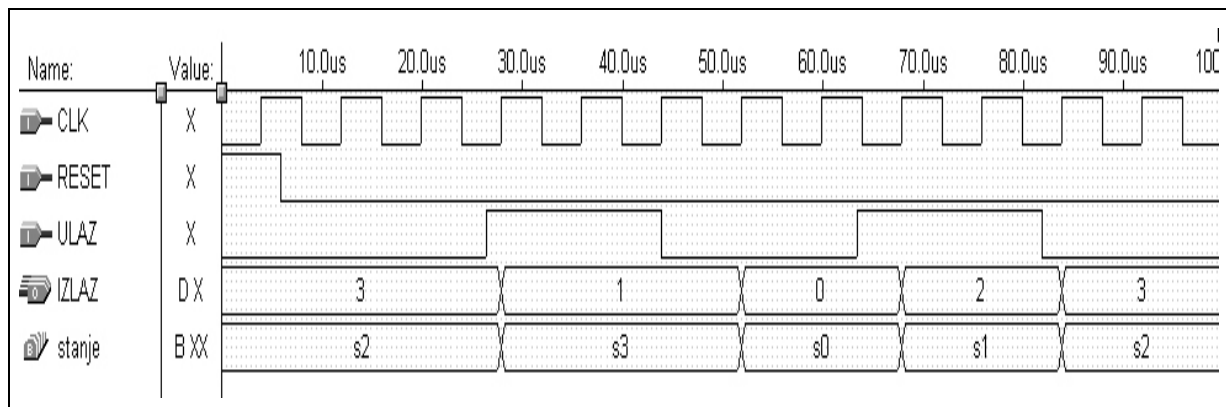
Definisanje signala na izlazu realizuje se kroz drugi proces, tj. proces nezavisan od procesa definisanje promene stanja. Ovaj proces opisuje kombinacionu mrežu za generisanje izlaznih signala *IZLAZ* i to samo na osnovu trenutnog stanja sistema. Deo VHDL koda kojim se definišu signali izlaza na bazi trenutnog stanja sistema dat je u nastavku.

```

PROCESS ( masina_stanja ) BEGIN
    CASE masina_stanja IS
        WHEN s0 => IZLAZ <= "00";
        WHEN s1 => IZLAZ <= "10";
        WHEN s2 => IZLAZ <= "11";
        WHEN s3 => IZLAZ <= "01";
    END CASE;
END PROCESS;

```

Proces se izvršava pri promeni vrednosti signala *masina_stanja* što je naznačeno argumentom unutar ključne reči *PROCESS* i *CASE*. Kako ovaj proces ne zavisi direktno od sinhronizacionog signala *CLK*, VHDL prevodilac će ga realizovati kao kombinacionu mrežu čiji su ulazni signali signali mehanizma *masina_stanja*, a izlazni signali su *IZLAZ*.

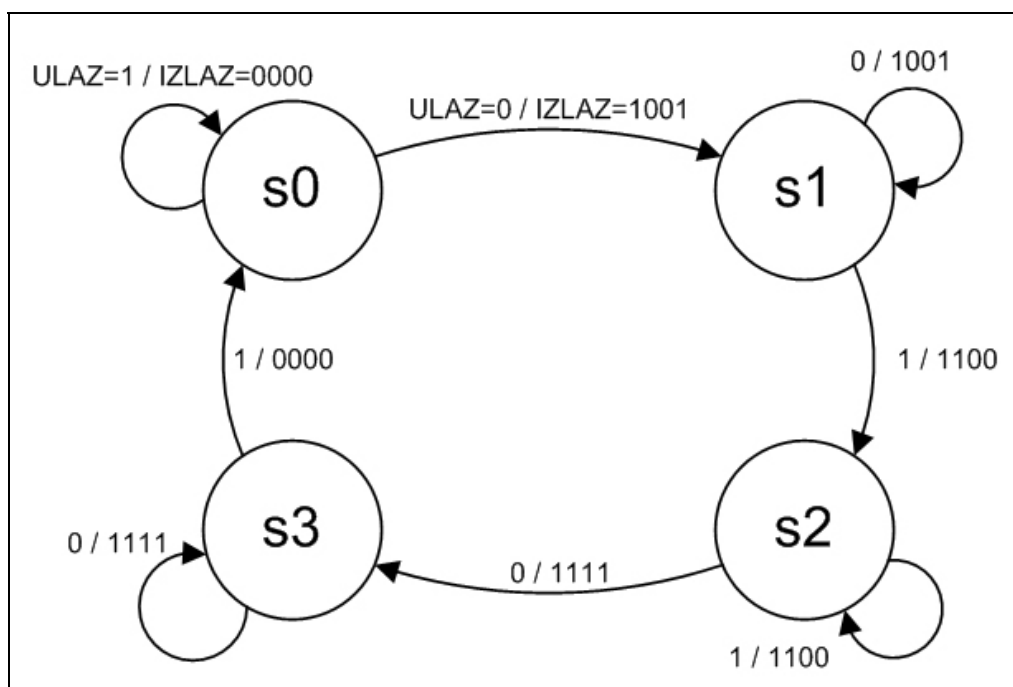


Slika 31.2: Simulacioni dijagram verifikacije sistema

Jedan od vremenskih dijagrama procesa simulacije opisanog sistema dat je na slici br. 31.2. Na vremenskom dijagramu uočljivo je da su promene stanja sistema (*masina_stanja*) i vrednost signala na izlazu *IZLAZ* dešavaju sinhrono sa *CLK* signalom. Promena stanja i izlaza obavljaju se u skladu sa postavljenim zahtevom zadatog dijagrama stanja.

ZADATAK 32 – Realizacija sistema specificiranog preko dijagrama stanja Milijevog tipa

Realizovati sekvencijalnu digitalnu mrežu koja predstavlja mašinu stanja Milijevog tipa. Sistem sadrži jednobitni kontrolni signal *ULAZ* i četvorobitni izlazni signal *IZLAZ*. Pored signala *ULAZ* i *IZLAZ* prisutni su i signali *i*, koji predstavljaju sinhronizacioni signal (*CLK*) i signal asinhronog reseta (*RESET*). Na slici br. 32.1 prikazan je dijagram stanja sistema.



Slika 32.1: Dijagram stanja sistema

Rad sistema opisan je kroz četiri stanja: *s0*, *s1*, *s2* i *s3*. Promena stanja sistema dešava se pri usponskoj ivici *CLK* signala. Asinhroni reset aktivan je pri *RESET*='1' i ima najveći prioritet izvršavanja. Prilikom asinhronog reseta sistem prelazi u stanje *s0*.

Primenom softverskog paketa za rad sa programabilnim kolima firme ALTERA realizovati mrežu VHDL opisom.

Izvršiti simulaciju rada sistema valjanim izborom vremenskog inkrementa (*Grid Size*), vremena trajanja simulacije (*End Time*) i test signala ulaznih priključaka sistema.

REŠENJE

U skladu sa objašnjenjem iz prethodnog zadatka koji opisuje realizaciju sistema koji je specificiran preko dijagrama stanja, realizovana je digitalna sekvencijalna mreža Milijevog tipa. Osnovna razlika između realizacije mreže Milijevog tipa i Murovog tipa je način definisanja signala na izlazu.

Kompletan VHDL opis zahtevanog sistema dat je u nastavku.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY zadatak IS PORT
(
  CLK      : IN  STD_LOGIC;
  RESET    : IN  STD_LOGIC;

```

```

    ULAZ      : IN STD_LOGIC;
    IZLAZ     : OUT STD_LOGIC_VECTOR(3 DOWNT0 0)
);
END zadatak ;
ARCHITECTURE mealy OF zadatak IS
    -- Definisanje masine stanja
    TYPE tip_stanje IS (s0, s1, s2, s3);
    SIGNAL stanje : tip_stanje;
BEGIN
    -- Definisanje procesa promene stanja
    PROCESS ( CLK, RESET ) BEGIN
        IF ( RESET = '1' ) THEN
            -- Asinhrona promena- asinhroni reset
            stanje <= s0;
        ELSIF ( rising_edge( CLK ) ) THEN
            -- Sinhrona promena - promena stanja sistema
            CASE stanje IS
                WHEN s0 =>
                    IF ( ULAZ='0' ) THEN
                        stanje <= s1;
                    END IF;
                WHEN s1 =>
                    IF ( ULAZ='1' ) THEN
                        stanje <= s2;
                    END IF;
                WHEN s2 =>
                    IF ( ULAZ='0' ) THEN
                        stanje <= s3;
                    END IF;
                WHEN s3 =>
                    IF ( ULAZ='1' ) THEN
                        stanje <= s0;
                    END IF;
            END CASE;
        END IF;
    END PROCESS;

    -- Definisanje signala na izlazu
    PROCESS ( stanje, ULAZ ) BEGIN
        CASE stanje IS
            WHEN s0 =>
                IF ( ULAZ='1' ) THEN IZLAZ <= "0000";
                ELSE IZLAZ <= "1001";
                END IF;
            WHEN s1 =>
                IF ( ULAZ='1' ) THEN IZLAZ <= "1100";
                ELSE IZLAZ <= "1001";
                END IF;
            WHEN s2 =>
                IF ( ULAZ='1' ) THEN IZLAZ <= "1100";
                ELSE IZLAZ <= "1111";
                END IF;
            WHEN s3 =>
                IF ( ULAZ='1' ) THEN IZLAZ <= "0000";
                ELSE IZLAZ <= "1111";
                END IF;
        END CASE;
    END PROCESS;
END mealy ;

```

Definisanje tipa mehanizma mašine stanja (*tip_stanje*) i signala (*stanje*) koji nosi informaciju o stanju sistema ostvareno je sledećim instrukcijama VHDL koda:

```

TYPE tip_stanje IS (s0, s1, s2, s3);
SIGNAL stanje : tip_stanje;

```

Definisanje prelaska sistema iz stanja u stanje sprovodi se preko procesa koji je osetljiv na promenu signala *CLK* i *RESET*. Aktivna vrednost signala *RESET* (*RESET*='1'), asinhrono prevodi sistem u stanje *s0*. Ukoliko *RESET* nije aktivan (*RESET*='0'), sistem se sinhrono kreće kroz stanja u zavisnosti od trenutnog stanja sistema i vrednosti signala na ulazu *ULAZ*.

```

PROCESS ( CLK, RESET ) BEGIN
  IF ( RESET = '1' ) THEN
    -- Asinhrona promena- asinhroni reset
    stanje <= s0;
  ELSIF ( rising_edge( CLK ) ) THEN
    -- Sinhrona promena - promena stanja sistema
    CASE stanje IS
      WHEN s0 =>      IF ( ULAZ='0' ) THEN
                        stanje <= s1;
                        END IF;
      WHEN s1 =>      IF ( ULAZ='1' ) THEN
                        stanje <= s2;
                        END IF;
      WHEN s2 =>      IF ( ULAZ='0' ) THEN
                        stanje <= s3;
                        END IF;
      WHEN s3 =>      IF ( ULAZ='1' ) THEN
                        stanje <= s0;
                        END IF;
    END CASE;
  END IF;
END PROCESS;

```

Ove dve faze realizacije sistema Milijevog tipa iste su kao i u slučaju realizacije mašine stanja Murovog tipa.

Proces koji definiše podsistem generisanja signala na izlazu za mreže Milijevog tipa, saglasno dijagramu stanja datom u ovom zadatku opisan je na sledeći način:

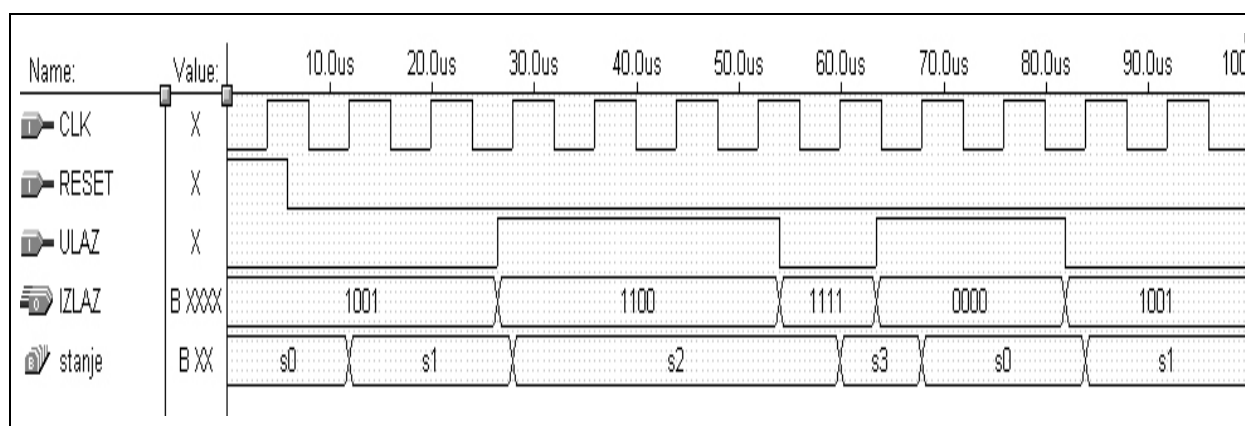
```

PROCESS ( stanje, ULAZ ) BEGIN
  CASE stanje IS
    WHEN s0 =>      IF ( ULAZ='1' ) THEN IZLAZ <= "0000";
                        ELSE                IZLAZ <= "1001";
                        END IF;
    WHEN s1 =>      IF ( ULAZ='1' ) THEN IZLAZ <= "1100";
                        ELSE                IZLAZ <= "1001";
                        END IF;
    WHEN s2 =>      IF ( ULAZ='1' ) THEN IZLAZ <= "1100";
                        ELSE                IZLAZ <= "1111";
                        END IF;
    WHEN s3 =>      IF ( ULAZ='1' ) THEN IZLAZ <= "0000";
                        ELSE                IZLAZ <= "1111";
                        END IF;
  END CASE;
END PROCESS;

```

Reč je o procesu koji opisuje kombinacionu mrežu čiji su signali na ulazu: signal ulaza *ULAZ* i signali stanja *stanje*. Kako u aktivnosti procesa direktno ne učestvuje sinhronizacioni signal *CLK*, ovaj proces opisuje kombinacionu mrežu. Pošto postoji direktan uticaj signala *ULAZ* na generisanje signala na izlazu *IZLAZ*, to znači da se radi o digitalnoj mreži Milijevog tipa.

Jedan od vremenskih dijagrama procesa simulacije opisanog sistema dat je na slici br. 32.2. Na vremenskom dijagramu može se uočiti je da su promene stanja sistema (*stanje*) sinhronne sa *CLK* signalom dok se na izlazu promene javljaju asinhrono u odnosu na *CLK* signal ali u skladu sa ulaznim signalom *ULAZ*. Promena stanja i izlaza obavljaju se u skladu sa postavljenim zahtevom zadatog dijagrama stanja.



Slika 32.2: Simulacioni dijagram

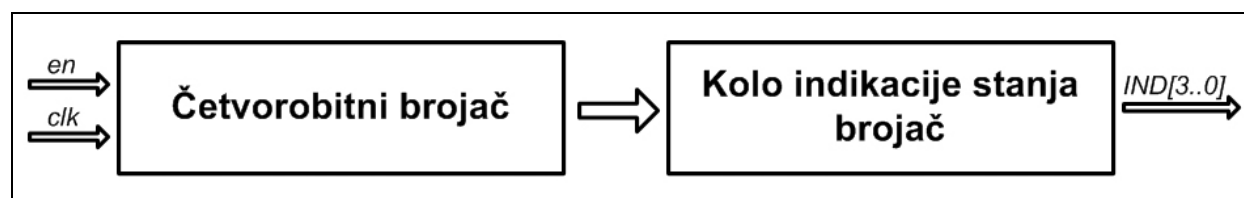
Opis sistema preko mega-funkcija i kombinovanom metodom

ZADATAK 33 – Brojač i indikator stanja brojača primenom megafunkcija

Realizovati sistem koji predstavlja spregu brojača i kola za indicaciju stanja brojača. Organizacija sistema prikazana je na slici br. 33.1. Brojač u okviru sistema je četvorobitni sa funkcijom brojanja nagore po sekvenci: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow \dots \rightarrow 14 \rightarrow 15 \rightarrow 0 \rightarrow 1 \rightarrow \dots$. Brojač sadrži kontrolne signale:

- *en* – signal dozvole brojanja koji je aktivan pri visokom nivou, tj. za $en = '1'$ i
- *clk* – sinhronizacioni signal rada brojača (sinhrone promene dešavaju se pri usponskoj ivici *clk* signala).

Vrednost, tj. izlaz, brojača dovodi se na ulaz dekodera stanja brojača.



Slika 33.1: Organizacija sistema sprege četvorobitnog brojača i kola za indicaciju stanja brojača

Dekoder stanja brojača je kombinaciona mreža koja na bazi ulaznih signala vrši postavljanje signala indicacije *IND[3..0]*. U tabeli br. 33.1 dat je opis rada dekodera stanja brojača u zavisnosti od vrednosti signala na ulazu.

Table 33.1: Specifikacija rada dekodera stanja brojača

Izlaz	Vrednost na ulazu dekodera indicacije															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
<i>IND0</i>		L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
<i>IND1</i>					L	L	L	L	L	L	L	L	L	L	L	L
<i>IND2</i>												L	L	L	L	L
<i>IND3</i>																L

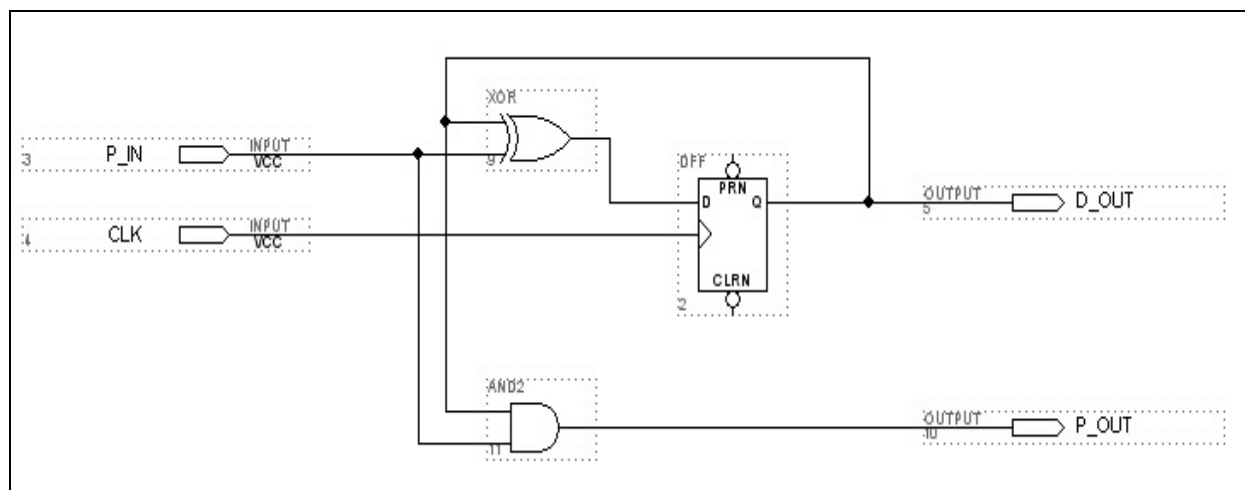
Ukoliko polje u tabeli br. 33.1 sadrži simbol L za određenu kombinaciju ulaza i izlaza *INDx*, na taj izlaz *INDx*, dekode stanja brojača treba da postavi nizak logički nivo $INDx = '0'$. U suprotnom, treba da se pojavi visok logički nivo, tj. $INDx = '1'$. Primera radi, ako je na ulazu dekodera stanja brojača prisutna vrednost 10 signali indicacije *IND0*, *IND1* imaju nizak ($IND0 = IND1 = '0'$), dok signali *IND2* i *IND3* imaju visok logički nivo ($IND2 = IND3 = '1'$).

Zadatak rešiti opisom četvorobitnog brojača u grafičkom editoru primenom kola za inkrementiranje. Kolo indicacije stanja brojača realizovati preko mega-funkcije *lpm_rom*. Sistem u celini opisati u grafičkom editoru i primeniti hijerarhijsku organizaciju sistema.

Primenom alata za simulaciju izvršiti simulaciju rada sistema pravilnim izborom vremenskih oblika signala, vremena trajanja simulacije i vremenskog inkrementa.

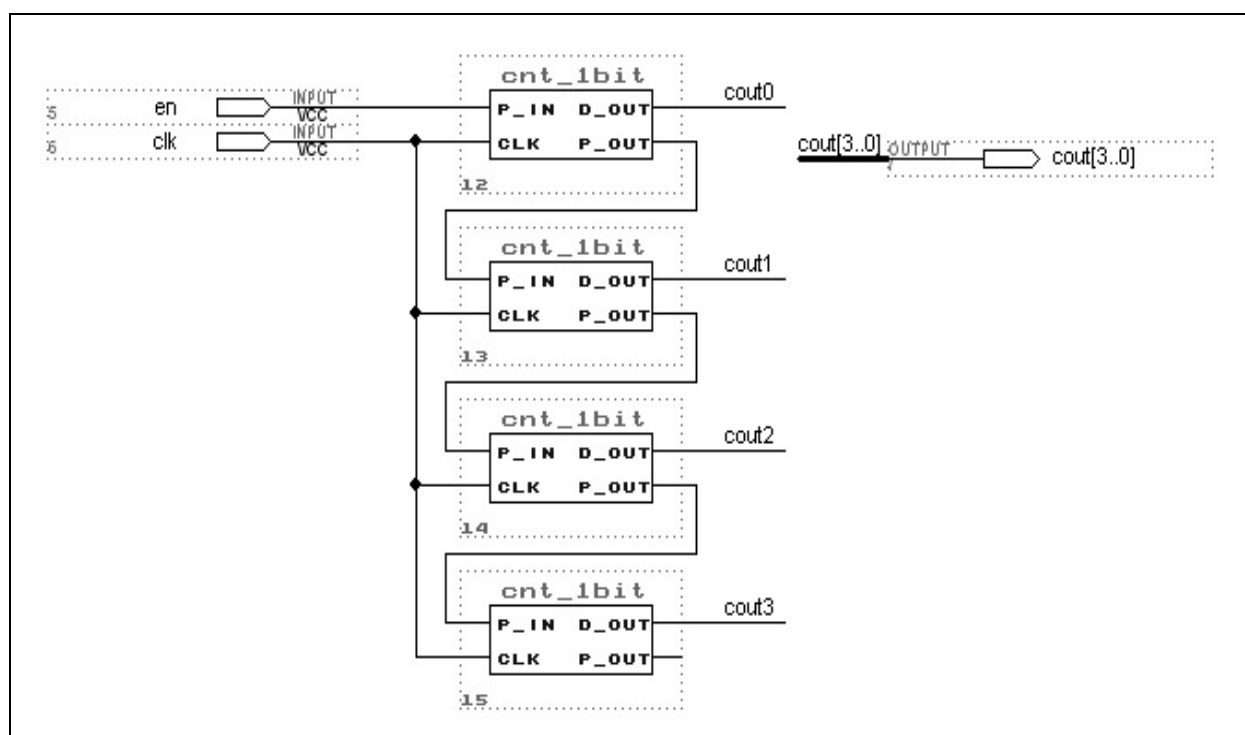
REŠENJE

Kolo za inkrementiranje je kombinaciona mreža jednog razreda u sistemu brojača. Kolo za inkrementiranje obavlja funkciju sabiranja trenutne vrednosti memorijskog elementa sa signalom na ulazu P_IN u cilju formiranja nove vrednosti memorijskog elementa kao i signal prenosa u sledeći razred (P_OUT). Kolo za inkrementiranje predstavlja polusabirač koji u odgovarajućoj konfiguraciji sa memorijskim elementom (D-FF) čini osnovnu ćeliju brojača – kolo jednobitnog razreda brojača. Realizacija četvorobitnog brojača obavlja se preko kola jednobitnog razreda brojača koje je prikazano na slici br. 33.2.



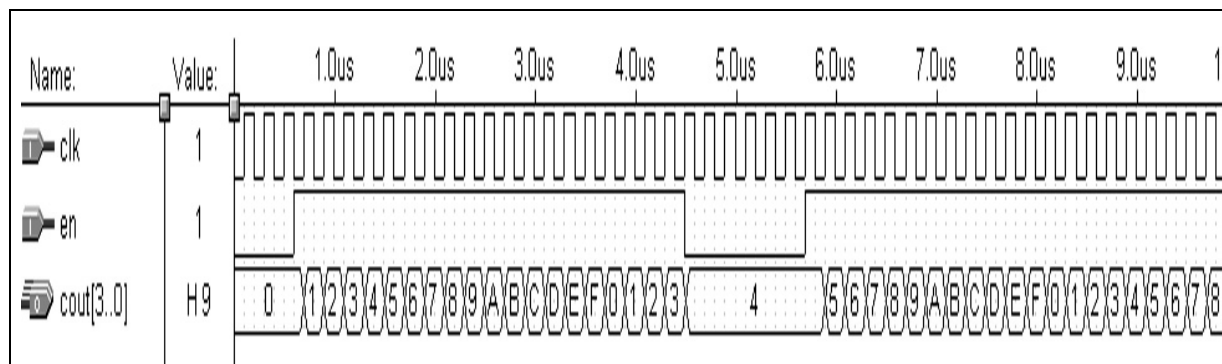
Slika 33.2: Šema ćelije brojača na bazi kola za inkrementiranje

Pravilnim povezivanjem četiri jednobitna razreda brojača (konfiguracija je data na slici br. 33.2, a simbol nazvan **cnt_1bit**) dobija se četvorobitni brojač zasnovan na kolu za inkrementiranje. Konfiguracija četvorobitnog brojača data je na slici br. 33.3.



Slika 33.3: Četvorobitni brojač na bazi kola za inkrementiranje

U cilju pravilne realizacije sistema u celini potrebno je vršiti proveru ispravnosti rada pojedinih podsistema, tj. u konkretnom slučaju četvorobitnog brojača **cnt_4bits**. Na slici br. 33.4 prikazan je simulacioni dijagram koji služi za verifikaciju ispravnosti opisa četvorobitnog brojača na bazi kola za inkrementiranje. Na dijagramu se uočava pravilan rad brojača u režimu brojanja kao i prilikom zamrzavanja vrednosti brojača.



Slika 33.4: Simulacioni dijagram četvorobitnog brojača na bazi kola za inkrementiranje

Specifikacija kola indikacije stanja brojača obavlja se preko megafunkcije ROM memorije tj. *lpm_rom* komponente. Realizacija dekodera preko ROM memorije zasniva se na principu popunjavanja sadržaja ROM memorije kombinacijama vrednosti vektora koje se mogu naći na izlazu dekodera pri čemu vrednost na ulazu vrši adresiranje memorije.

Opis rada *lpm_rom* komponente ostvaruje se preko *MIF* datoteke u kojoj se definiše sadržaj ROM memorije. Sadržaj *MIF* datoteke koja opisuje rad kola indikacije stanja brojača dat je u nastavku:

```

DEPTH = 16;
WIDTH = 4;
ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;
CONTENT
BEGIN
    0 : 1111;
    1 : 1110;
    2 : 1110;
    3 : 1110;
    4 : 1110;
    5 : 1100;
    6 : 1100;
    7 : 1100;
    8 : 1100;
    9 : 1100;
    A : 1100;
    B : 1000;
    C : 1000;
    D : 1000;
    E : 1000;
    F : 0000;
END ;

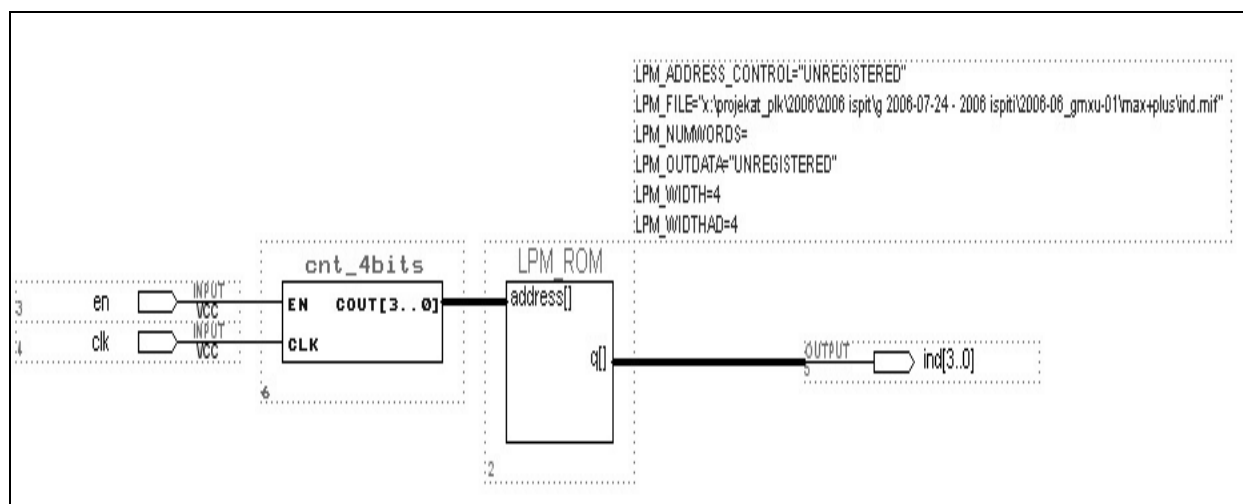
```

Četvorobitna vrednost sadržaja ROM memorije predstavlja vrednost signala na izlazu dekodera indikacije i to čitano sleva na desno: *IND3*, *IND2*, *IND1* i *IND0*.

Nakon formiranja novog dizajna u grafičkom editoru za potrebe realizacije sistema u celini treba izvršiti pozivanje četvorobitnog brojača (*cnt_4bits*) i *lpm_rom* komponente. U skladu sa zahtevima zadatka treba postaviti parametre *lpm_rom* komponente kako bi se dobilo kolo indikacije stanja brojača i to izmenom sledećih parametara:

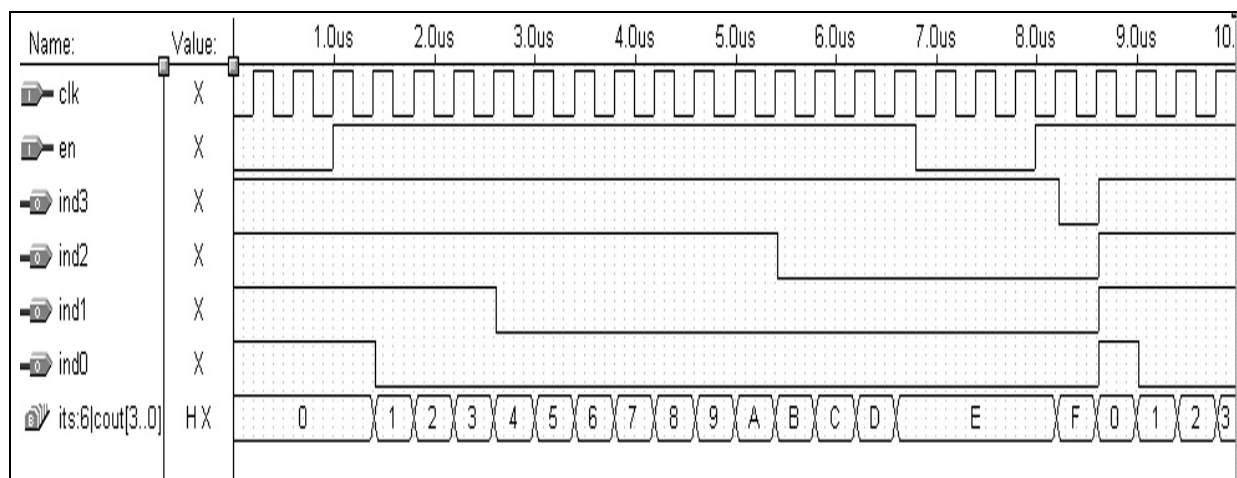
```
LPM_ADDRESS_CONTROL="UNREGISTERED"
LPM_OUTDATA="UNREGISTERED"
LPM_FILE="...\ind.mif"
LPM_WIDTH=4
LPM_WIDTHAD=4
```

Organizacija sistema u celini prikazana je na slici br. 33.5.



Slika 33.5: Blok šema zahtevanog sistema brojača i kola za indikaciju stanja brojača

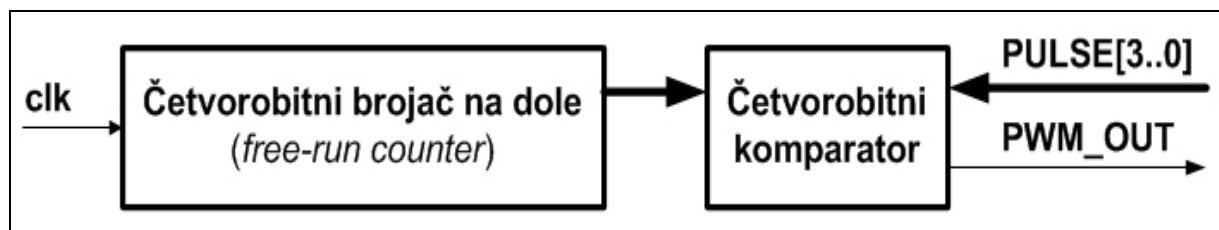
Simulacioni dijagram preko koga se vrši verifikacija sistema u celini dat je na slici br. 33.6.



Slika 33.6: Simulacioni dijagram za verifikaciju sistema u celini

ZADATAK 34 – Generator impulsno-širinski modulisanog signala

Realizovati generator impulsno-širinski modulisanog signala (PWM generator) koji treba da radi sa konstantnom periodom i promenljivom širinom impulsa koja se kontroliše preko vrednosti zadate spolja. Blok šema sistema PWM generatora data je na slici br. 34.1. Širina impulsa na izlazu PWM generatora, tj. signala *PWM_OUT*, zadaje se preko četvorobitnog ulaza *PULSE[3..0]*. PWM generator treba realizovati preko implementacije četvorobitnog brojača na dole (*free-run counter*) i četvorobitnog komparatora. Četvorobitnu vrednost *PULSE[3..0]* koja određuje širinu impulsa nije neophodno baferisati.



Slika 34.1: Blok šema sistema PWM generatora

Četvorobitni brojač na dole specificirati preko mega-funkcije brojača (*lpm_counter*), a četvorobitni komparator preko standardnih logičkih kola u grafičkom editoru.

Izvršiti simulaciju PWM generatora valjanim izborom test signala za simulaciju pri čemu učestanost signala na izlazu treba da bude približno 16kHz.

REŠENJE

Na bazi blok šeme koja je data na slici br. 34.1 generisanje impulsno-širinski modulisanog signala na izlazu PWM generatora (*PWM_OUT* signal) ostvaruje se upoređivanjem vrednosti brojača i zadate vrednosti širine impulsa (*PULSE[3..0]*). Vrednost *PULSE[3..0]* signala određuje širinu impulsa *PWM_OUT* signala. Pod impulsom *PWM_OUT* signala podrazumeva se deo periode signala u okviru koje signal *PWM_OUT* ima vrednost '1'. Ukoliko je vrednost brojača manja od zadate vrednosti *PULSE[3..0]*, PWM generator treba da generiše visok logički nivo na izlazu *PWM_OUT* tj. *PWM_OUT*='1'. U suprotnom PWM generator generiše nizak logički nivo na izlazu tj. *PWM_OUT*='0'. Za implementaciju PWM generatora na bazi blok šeme koja je data na slici br. 34.1 potrebno je realizovati »nepotpuni« četvorobitni komparator tj. četvorobitni komparator sa jednom od dve funkcije upoređivanja vrednosti A i B: $A < B$ (*LT* signal) ili $A > B$ (*GT* signal). Rešenje u ovom zadatku je zasnovano na četvorobitnom komparatoru sa funkcijom upoređivanja $A < B$.

Realizacija višebitnog komparatora bazira se na podsistemu jednobitnog komparatora. Jednačine koje povezuju ulazne (A i B) i izlazne (*GT*, *EQ* i *LT*) signale jednobitnog komparatora date su u nastavku:

$$A < B: LT = A \cdot \overline{B}$$

$$A = B: EQ = \overline{A \cdot \overline{B}} + \overline{A \cdot B} = \overline{A \cdot B} + \overline{A \cdot \overline{B}}$$

$$A > B: GT = \overline{A} \cdot B$$

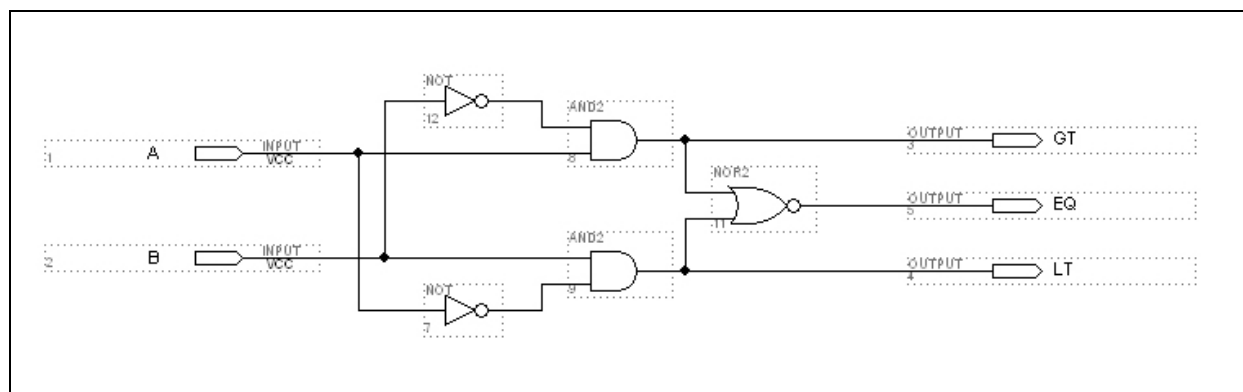
Komparator N-to bitnih vrednosti signala na ulazu dobija se pravilnom organizacijom N jednobitnih komparatora. Za realizaciju PWM generatora koji je specificiran u zadatku koristi se komparator četvorobitnih signala koji na izlazu ima signal upoređivanja $A < B$ (*LT*). Za implementaciju ove funkcije komparatora potrebno je realizovati kombinacionu logiku koja je data jednačinom u nastavku:

$$\begin{aligned}
 A < B &= (A_3 < B_3) + \\
 &\quad (A_3 = B_3)(A_2 < B_2) + \\
 &\quad (A_3 = B_3)(A_2 = B_2)(A_1 < B_1) + \\
 &\quad (A_3 = B_3)(A_2 = B_2)(A_1 = B_1)(A_0 < B_0)
 \end{aligned}$$

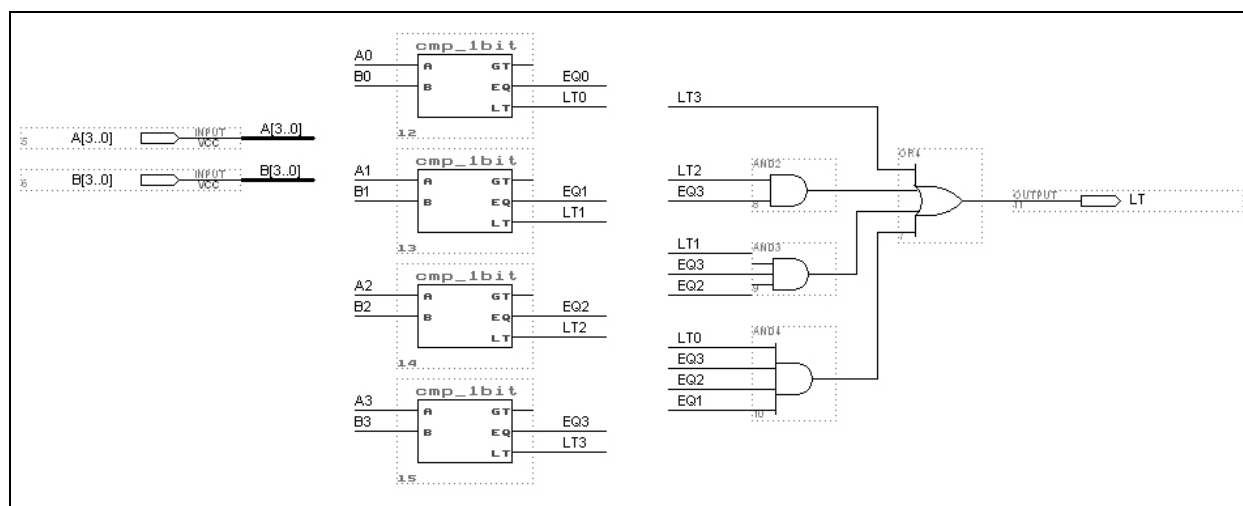
tj. u formi koja koristi izlaze komparatora jednobitnih vrednosti ($LT3, LT2, LT1, LT0, EQ3, EQ2$ i $EQ1$):

$$\begin{aligned}
 LT &= LT3 + \\
 &\quad EQ3 \cdot LT2 + \\
 &\quad EQ3 \cdot EQ2 \cdot LT1 + \\
 &\quad EQ3 \cdot EQ2 \cdot EQ1 \cdot LT0
 \end{aligned}$$

Na osnovu datih jednačina realizuje se kolo jednobitnog komparatora *cmp_1bit* (slika br. 34.2) i na bazi njega kolo za upoređivanje dva četvorobitna pozitivna broja *cmp_4bits* (slika br. 34.3) i generisanje signala upoređivanja LT .



Slika 34.2: Šema jednobitnog komparatora (*cmp_1bit*)



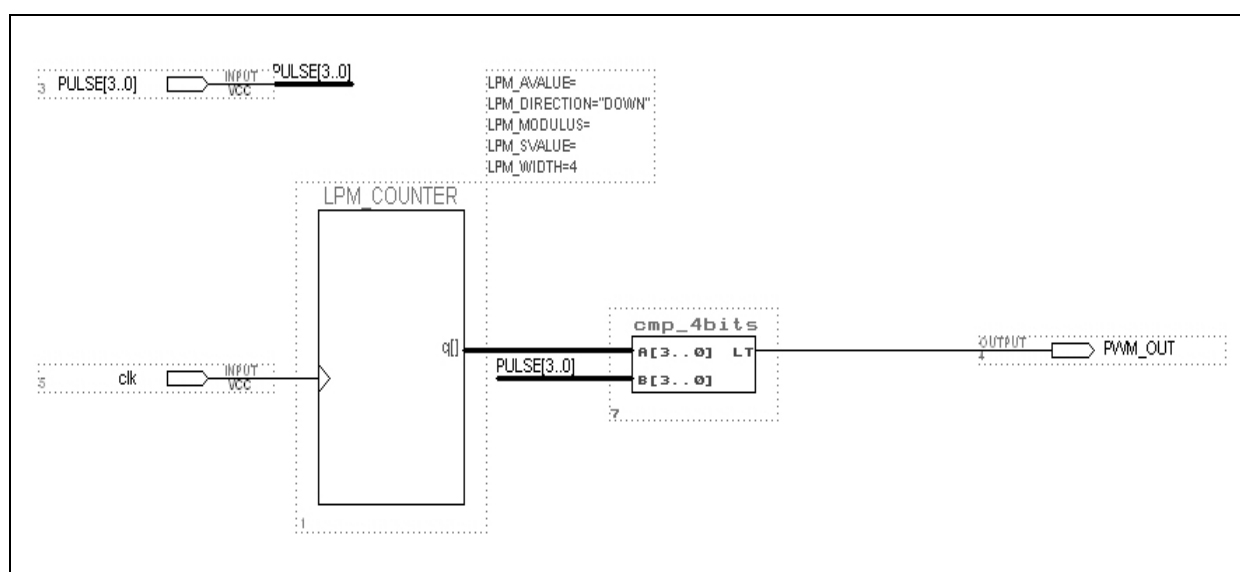
Slika 34.3: Šema četvorobitnog komparatora (*cmp_4bits*)

Realizacija PWM generatora obavlja se grafičkim opisom na višem hijerarhijskom nivou (slika br. 34.4) preko brojača na dole i implementiranog kola komparatora (*cmp_4bits*). Za

realizaciju brojača nadole koristi se mega-funkcija brojača. Da bi se implementirao četvorobitni brojač nadole treba pozvati *lpm_counter* komponente i podesiti sledeće parametre:

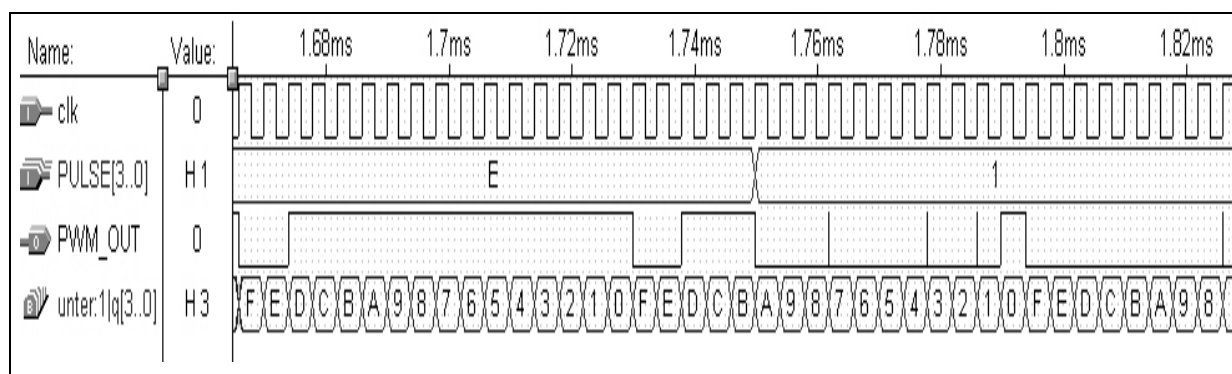
- svi signali (*Ports*), osim *clock* i *q[LPM_WIDTH-1..0]*, treba da budu postavljeni tako da se ne koriste (*Unused*). Za signale *clock* i *q[LPM_WIDTH-1..0]* treba postaviti parametar korišćenja na *Used*;
- parametar *LPM_DIRECTION* treba postaviti na vrednost "DOWN" kako bi se ostvarila funkcija brojanja na dole i
- parametar *LPM_WIDTH* treba postaviti na vrednost 4 kako bi se realizovao četvorobitni brojač.

Povezivanjem realizovane komponente brojača na bazi mega-funkcije *lpm_counter* i četvorobitnog komparataro *cmp_4bits* dobija se sistem PWM generatora (slici br. 34.4).



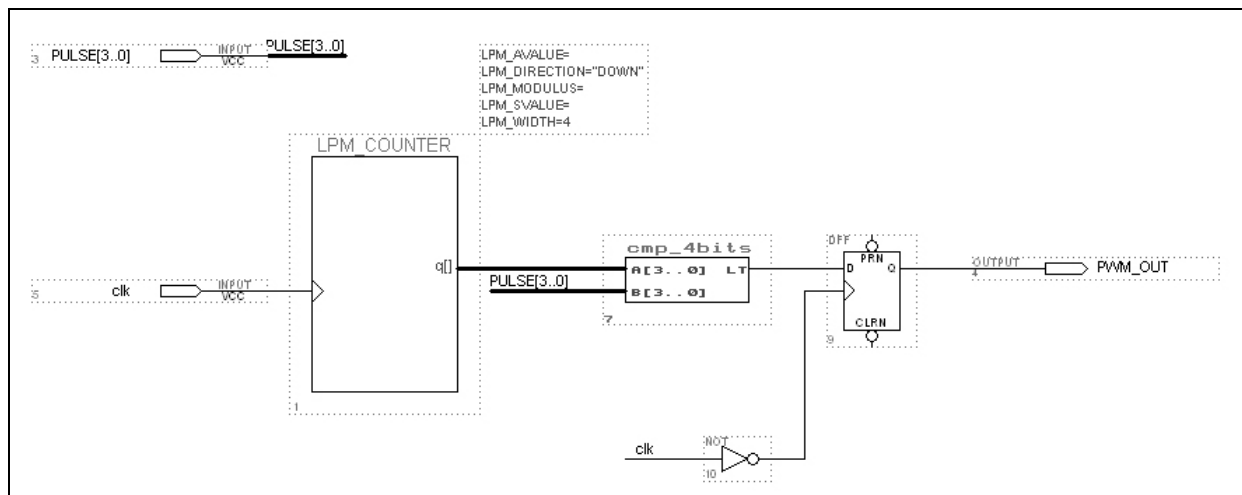
Slika 34.4: Šema PWM generatora

Signal osnovne sinhronizacije učestanosti rada sistema je *clk*. Na bazi signala *clk* i periode brojača formira se periodičan signal izlaza PWM generatora *PWM_OUT*. Kako je reč o četvorobitnom brojaču, signal *PWM_OUT* je 2^4 puta sporiji od signala *clk*, tj. ima 2^4 puta manju učestanost od učestanosti *clk* signala. Za izvođenje simulacije rada PWM generatora pri učestanosti izlaznog signala *PWM_OUT* od 16kHz treba postaviti osnovnu učestanost *clk* signala na 256kHz, tj. periodu *clk* signala na približno 4μsec.



Slika 34.5: Simulacioni dijagram sistema u celini koji je prikazan na slici br. 34.4

Na simulacionom dijagramu PWM generatora koji je prikazan na slici br. 34.4 uočava se pojava gličeva u nekim situacijama rada sistema. Te pojave su naročito uočljive za vrednosti signala kontrole širine impulsa $PULSE=1$. Ova pojava je poznata i potiče od načina realizacije komparatora, tj. zbog različitih vremena propagacija signala kod komparatora. Jedan od načina rešavanja problema ovog tipa je preko baferisanja vrednosti na izlazu komparatora PWM_OUT . Šema sistema PWM generatora sa modifikacijom u cilju otklanjanja pojave gličeva prikazana je na slici br. 34.6. Simulacioni dijagrami verifikacije rada sistema sa slike br. 34.6 prikazani su na slikama br. 34.7 i 34.8

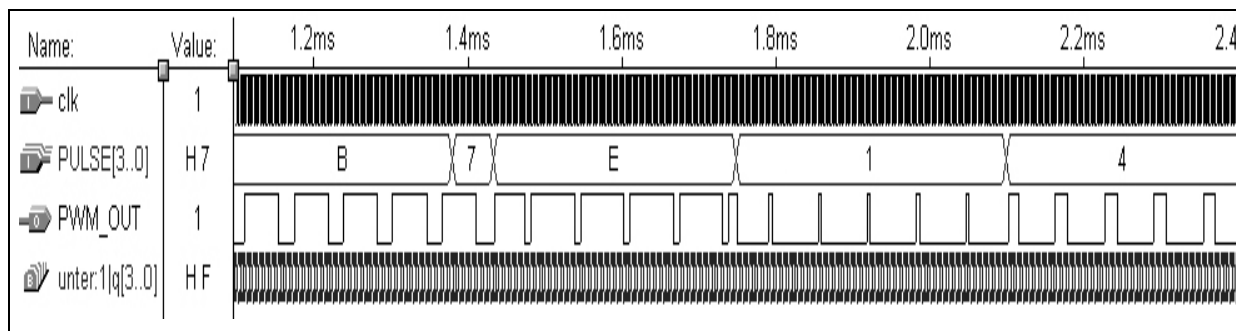


Slika 34.6: Modifikovana šema sistema PWM generatora radi otklanjanja gličeva

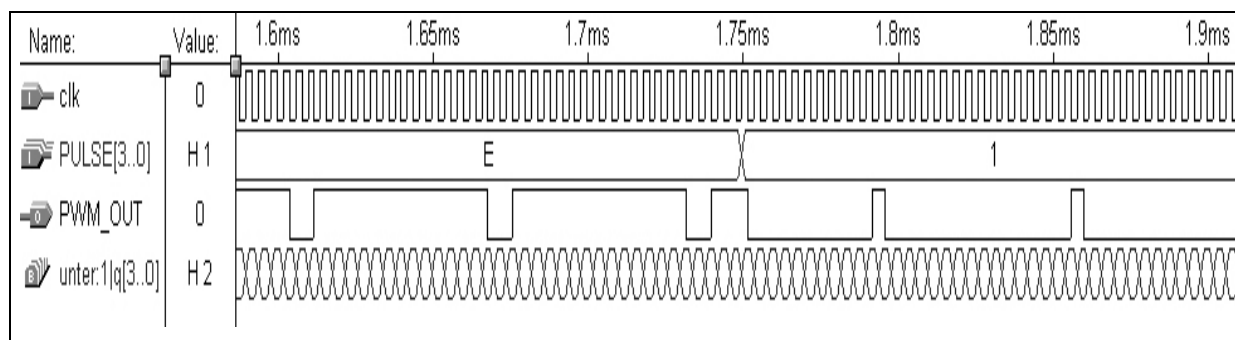
Modifikovana šema PWM generatora karakteristična je u tome što element za baferisanje (D-FF) PWM_OUT signala ima sinhronu promenu sa opadajućom ivicom clk signala. Za pravilan rad ovakvog sistema potrebno je da bude ispunjeno:

$$t_D + t_{cmp_4bits} + t_{ST} < \frac{t_{clk}}{2}$$

pri čemu su: t_D - vreme formiranja vrednosti na izlazu brojača tj. kašnjenje kroz memorijski element za baferisanje (registrarski izlaz $lpm_counter$ komponente), t_{cmp_4bits} - vreme formiranja signala na izlazu cmp_4bits kola, t_{ST} - „setup time“ D-FF komponente na putu signala PWM_OUT i t_{clk} - perioda clk signala.



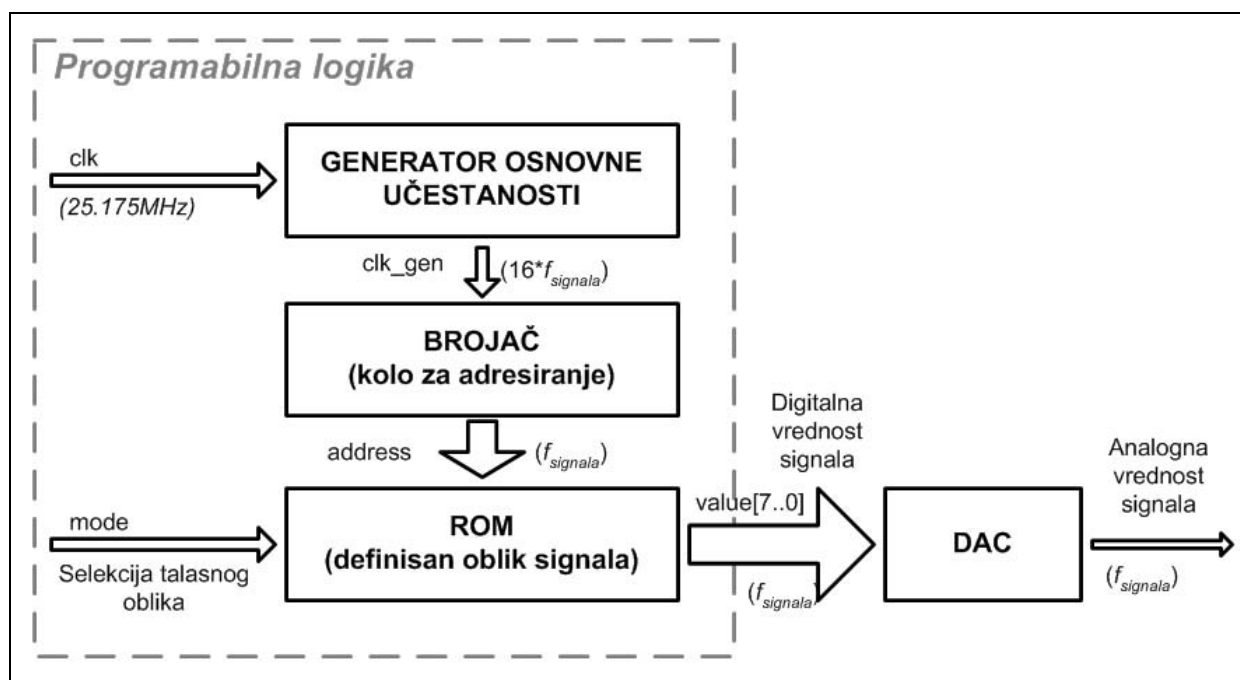
Slika 34.7: Simulacioni dijagram PWM generatora nakon modifikacije



Slika 34.8: Simulacioni dijagram PWM generatora nakon modifikacije

ZADATAK 35 – Generator periodičnog signala

Opisati realizaciju sistema za generisanje periodičnog signala specificiranog oblika i fiksne učestanosti u programabilnoj logici. Organizacija sistema za generisanje periodičnog signala u programabilnoj logici dat je na slici br. 35.1.



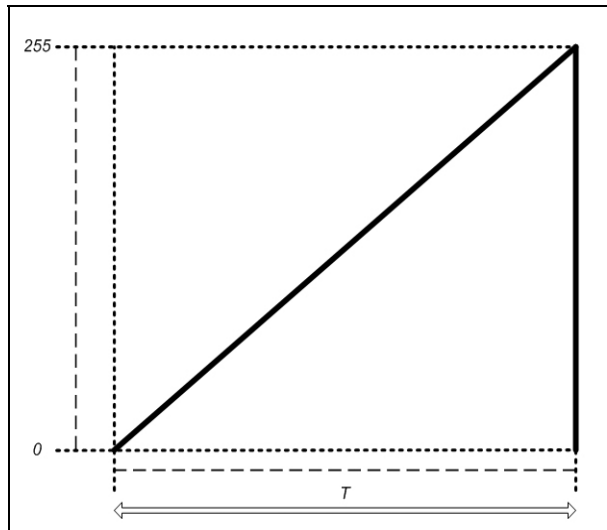
Slika 35.1: Organizacija sistema za generisanje periodičnog signala

Predviđeno je da se periodični signal vodi na ulaz jednostavnog digitalno-analognog konvertora (DAC) koji radi na principu R2R mreže. DAC u sistemu je osmobitni i nalazi se izvan programabilne logike.

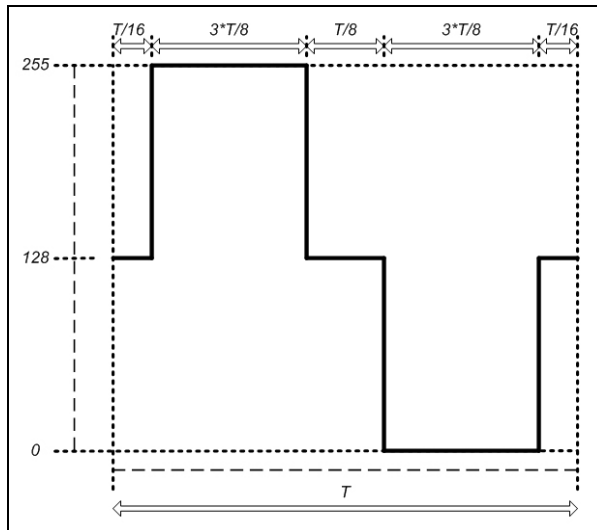
Sistem za generisanje periodičnog oblika signala sadrži sledeće kontrolne i izlazne signale:

- *clk* – sinhronizacioni signal učestanosti $f_{clk}=25,175\text{MHz}$;
- *mode* – signal selekcije oblika izlaznog signala, tj. za $mode=0$ generiše se “rampa” signal (dat na slici br. 35.2) dok se za $mode=1$ dobija impulsni signal (prikazan na slici br. 35.3);

- $value[7..0]$ – digitalna vrednost nivoa generisanog signala na izlazu. Ova vrednost se kreće od 0 do 255 pri čemu je za 0 minimalna a za 255 maksimalna vrednost pridruženog analognog signala (nakon digitalno-analogne konverzije).



Slika 35.2: Oblik signala – “rampa” signal ($mode=0$)



Slika 35.3: Oblik signala – impulsni signal - dvostruki impuls ($mode=1$)

Učestanost generisanog periodičnog signala na izlazu sistema je $f_{signala}=10kHz$. Jedna perioda signala na izlazu definisana je preko 16 tačaka (vrednosti nivoa signala) na ekvidistantnim rastojanjima u okviru periode T ($T=1/f_{signala}$). Sistem za generisanje periodičnog signala sastoji se od:

- ROM memorije u okviru koje su zabeležene vrednosti nivoa signala oba oblika preko 16 osmobitnih vrednosti po jednom obliku signala;
- brojača koji obezbeđuje adresiranje ROM memorije i
- generatora osnovne učestanosti rada sistema $16f_{signala}$ na bazi sistemskog sinhronizacionog signala ($f_{clk}=25.175MHz$);

Sistem za generisanje periodičnog oblika signala opisati grafičkim putem korišćenjem što većeg broja mega-funkcija (*lpm_counter*, *lpm_rom*, ...)

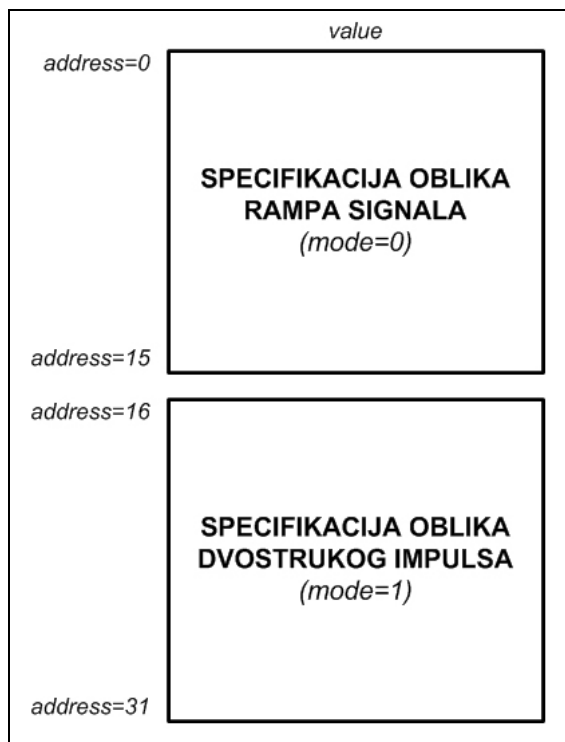
Primenom alata za simulaciju i pravilnim izborom vremenskih oblika signala, vremena trajanja simulacije i vremenskog inkrementa izvršiti verifikaciju rada opisanog sistema.

REŠENJE

Sistem treba da ima mogućnost generisanja dva oblika signala pri čemu se kontrolnim signalom *mode* određuje koji od ova dva signala se generiše. Vrednosti nivoa signala za karakteristične vremenske trenutke u okviru periode T upisuju se u ROM memoriju sistema i to u organizaciji koja je prikazana na slici br. 35.4. ROM memorija je kapaciteta 32x8bita.

Prvih 16 osmobitnih lokacija ROM memorije sadrže 16 tačaka u okviru jedne periode “rampa” signala. Od šesnaeste do trideset i prve lokacije ROM memorije, tj. u okviru 16 narednih lokacija, nalaze se specificirane vrednosti nivoa impulsnog signala. Na osnovu oblika signala (slike br. 35.2 i 35.3) određuju se 16 ekvidistantnih tačaka u okviru perioda oba signala i memorišu se u memoriji na način koji je prethodno opisan. Vrednosti nivoa ovih tačaka koje opisuju oblike periodičnih signala date su u tabeli br. 35.1.

Ovakvom organizacijom memorije obezbeđuje se da kontrolni signal *mode* mapira prvih 16 vrednosti za *mode=0* ili drugih šesnaest vrednost za *mode=1* čime se generiše “rampa” ili impulsni signal na izlazu. Ovakav sistem adresiranja realizuje se vezivanjem *mode* signala za adresnu liniju ROM memorije težine 2^4 .



Slika 35.4: Organizacija ROM memorije sistema u okviru koje se nalaze karakteristične tačke nivoa signala oba oblika

Tabela 35.1: Vrednosti 32 tačke za dva oblika signala (16 po svakom) kao i raspored u memorijskom prostoru ROM memorije

“Rampa” signal		Impulsni signal	
<i>mode=0</i>		<i>mode=1</i>	
<i>address</i>	<i>data</i>	<i>address</i>	<i>data</i>
0	8	16	128
1	24	17	255
2	40	18	255
3	56	19	255
4	72	20	255
5	88	21	255
6	104	22	255
7	120	23	128
8	136	24	128
9	152	25	0
10	168	26	0
11	184	27	0
12	200	28	0
13	216	29	0
14	232	30	0
15	248	31	128

Na osnovu tabele br. 35.1 definiše se MIF datoteka (generator.mif) koja opisuje sadržaj ROM memoriju kapaciteta 32x8bita. Sadržaj generator.mif datoteke dat je u nastavku:

```

DEPTH = 32;
WIDTH = 8;
ADDRESS_RADIX = DEC;
DATA_RADIX = DEC;
CONTENT
BEGIN
    0      :      8;
    1      :      24;
    2      :      40;
    3      :      56;
    4      :      72;
    5      :      88;
    6      :     104;
    7      :     120;
    8      :     136;
    9      :     152;
   10      :     168;
   11      :     184;
   12      :     200;
   13      :     216;
   14      :     232;
   15      :     248;

   16      :     128;
   17      :     255;
   18      :     255;
   19      :     255;
   20      :     255;
   21      :     255;
   22      :     255;
   23      :     128;
   24      :     128;
   25      :      0;
   26      :      0;
   27      :      0;
   28      :      0;
   29      :      0;
   30      :      0;
   31      :     128;
END ;

```

Sistem za generisanje periodičnog oblika signala dat je na slici br. 35.5. ROM memorija sa sadržajem definisanih tačaka oba oblika signala dobija se formiranjem komponente na bazi *lpm_rom* mega-funkcije i podešavanjem parametara na sledeći način:

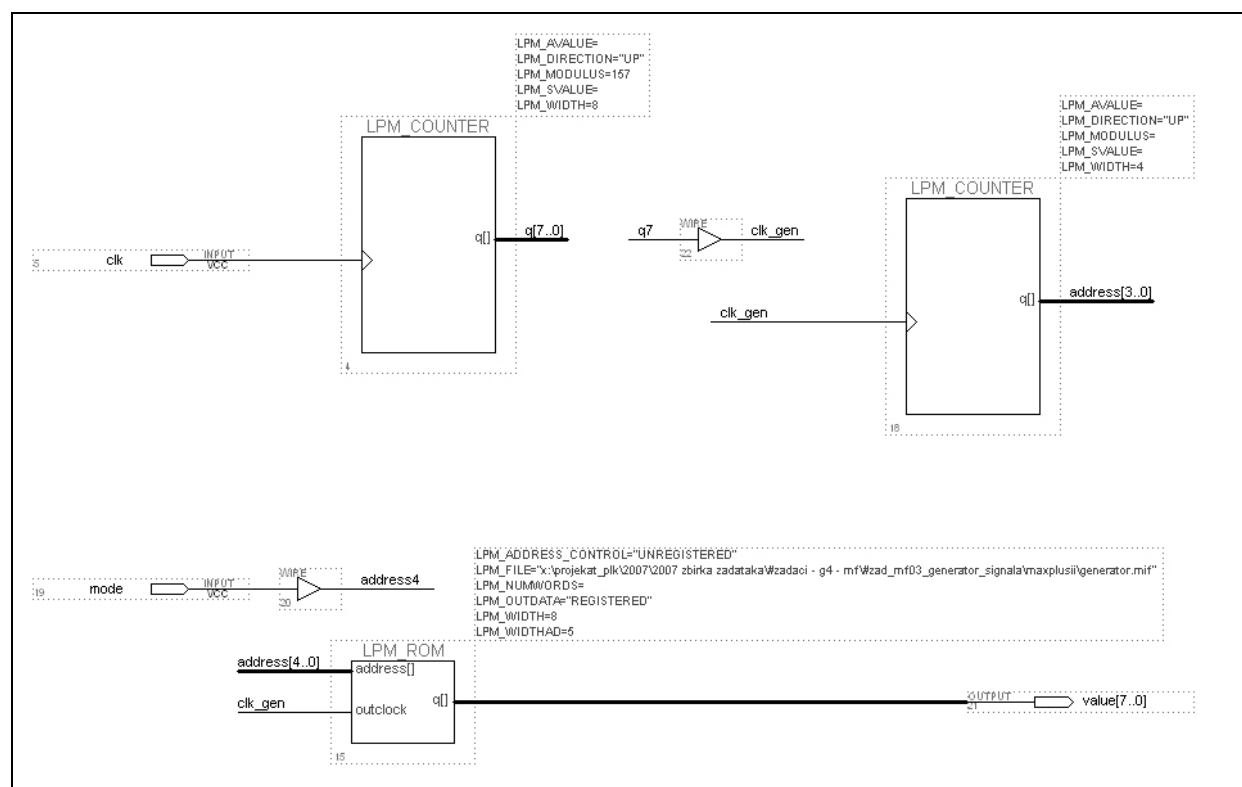
```

LPM_ADDRESS_CONTROL="UNREGISTERED"
LPM_OUTDATA="REGISTERED"
LPM_FILE="...\generator.mif"
LPM_WIDTH=8
LPM_WIDTHAD=5

```

Komponenta treba da ima aktivirane (*Used*) samo sledeće signale: *address*, *q* i *outclock*. Aktiviranjem signala *outclock* i podešavanjem parametra *LPM_OUTDATA*=“REGISTERED” postiže se obezbeđivanje baferisanja signala na izlazu *lpm_rom* memorije. Nakon vezivanja priključka *outclock* za signal *clk_gen* izlazni signal podatka ROM memorije *q* postaje sinhron sa *clk_gen* signalom ($f_{clk_gen}=10kHz$)

U toku jedne periode signala učestanosti $f_{signala}=10kHz$, sistem generiše 16 promena vrednosti signala na izlazu i to na bazi sadržaja ROM memorije. Adresiranje ROM memorije ostvaruje se preko četvorobitnog kružnog brojača na gore i kontrolnog signala *mode*. Četvorobitni kružni brojač nagore adresira kružno vrednosti 0,1,2,...,14,15,0,1... dok kontrolni signal *mode* aktivira opseg memorije od 0 do 15 ili od 16 do 31. Kružni brojač na gore treba da radi na učestanosti koja je 16 puta veća od učestanosti periodičnog signala na izlazu ($f_{signala}$), tj. $f_{clk_gen}=16f_{signala}=160kHz$. Učestanost f_{clk_gen} dobija se deljenjem učestanosti f_{clk} brojem 157. To se postiže preko osmобitnog kružnog brojača nagore po modulu 157. Realizacija kružnog brojača nagore po modulu 157 moguće je obaviti preko *lpm_counter* komponente i pravilnim podešavanjem parametara.



Slika 35.5: Simulacioni dijagram podsistema CLK_GEN

Za realizaciju osmобitnog brojača nagore po modulu 157 koristi se mega-funkcija brojača (*lpm_counter*). Mega-funkciju *lpm_counter* treba podesiti na sledeći način:

- svi signali (*Ports*), osim *clock* i $q[LPM_WIDTH-1..0]$, treba da budu postavljeni tako da se ne koriste (*Unused*). Za signale *clock* i $q[LPM_WIDTH-1..0]$ treba postaviti parametar korišćenja na *Used*;
- parametar *LPM_MODULUS* treba postaviti na vrednost 157 kako bi se dobio brojač po modulu 157;
- parametar *LPM_DIRECTION* treba postaviti na vrednost “UP” kako bi se ostvarila funkcija brojanja nagore i

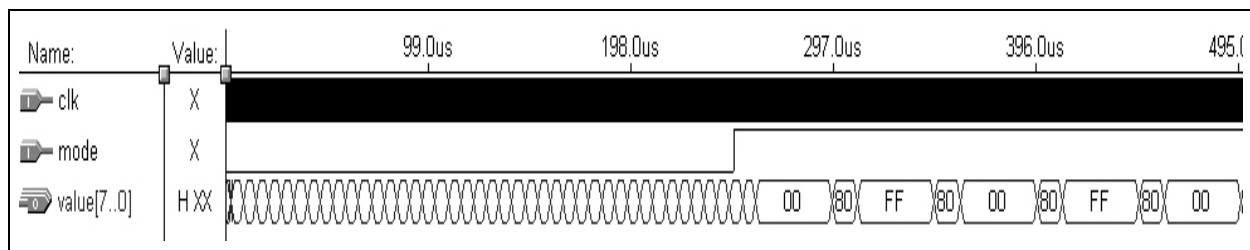
- parametar *LPM_WIDTH* treba postaviti na vrednost 8 kako bi se realizovao osmobitni brojač.

Signal *q7* na izlazu osmobitnog kružnog brojača nagore po modulu 157 u funkciji generatora učestanosti treba preimenovati u signal *clk_gen*. To se postiže korišćenjem komponente *WIRE*.

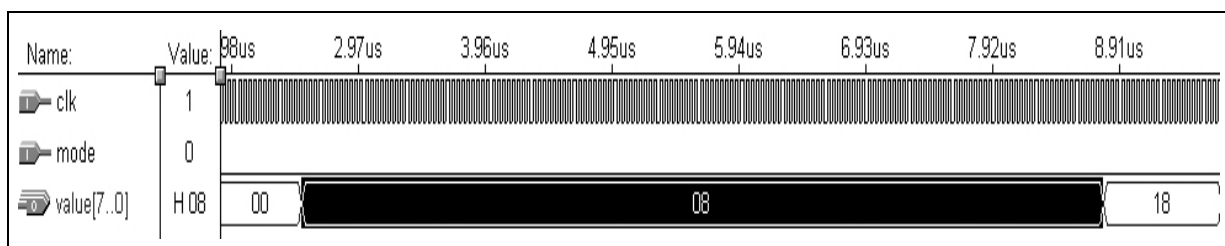
Za realizaciju četvorobitnog kružnog brojača nagore u funkciji kola za adresiranje koristi se mega-funkcija brojača (*lpm_counter*). Mega-funkciju *lpm_counter* treba podesiti na sledeći način:

- svi signali (*Ports*), osim *clock* i *q[LPM_WIDTH-1..0]*, treba da budu postavljeni tako da se ne koriste (*Unused*). Za signale *clock* i *q[LPM_WIDTH-1..0]* treba postaviti parametar korišćenja na *Used*;
- parametar *LPM_DIRECTION* treba postaviti na vrednost “UP” kako bi se ostvarila funkcija brojanja nagore i
- parametar *LPM_WIDTH* treba postaviti na vrednost 4 kako bi se realizovao četvorobitni brojač.

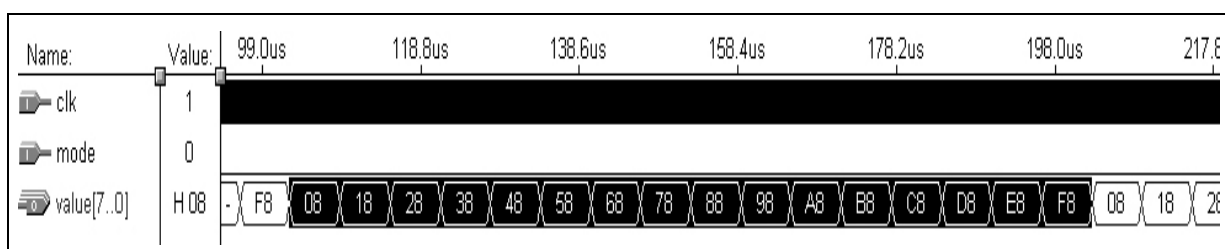
Verifikacija rada sistema postiže se preko serije simulacija pri čemu treba podesiti *grid size* na 19.86μsec i učestanost *clk* signala na 25,175MHz tj. ($T=39,72\mu\text{sec}$). Na slici br. 35.6 prikazan je deo simulacije gde se uočava generisanje oba oblika signala. Na slici br. 7 prikazan je šesnaestina periode signala na izlazu tj. vreme držanja jedne vrednosti signala. Dijagrami na slikama br. 35.8 i 35.9 prikazuju jednu periodu generisanja rampa odnosno implusnog signala. Zacrtnjeni delovi signala *value[7..0]* predstavljaju segmente od interesa.



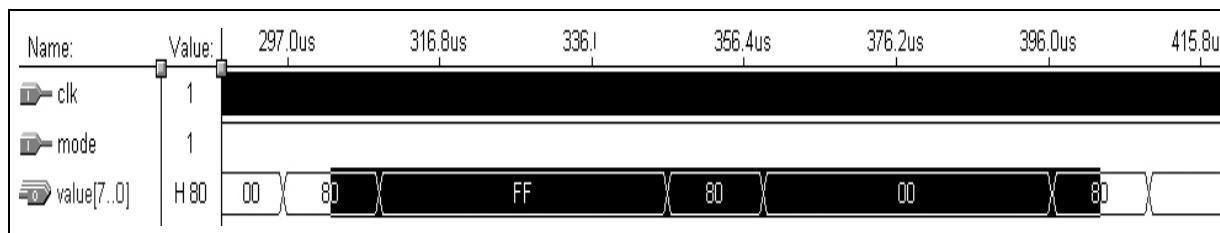
Slika 35.6: Simulacioni dijagram generatora periodičnog signala



Slika 35.7: Simulacioni dijagram generatora periodičnog signala



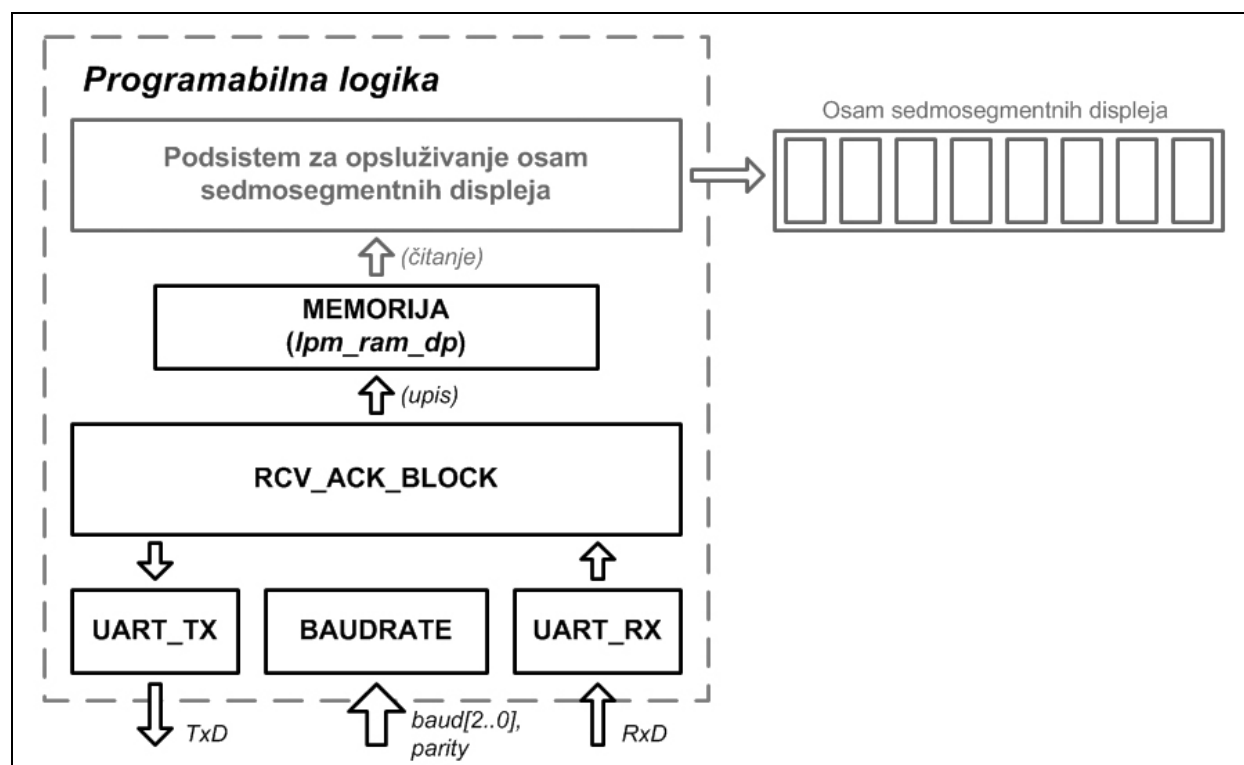
Slika 35.8: Simulacioni dijagram generatora periodičnog signala (“rampa” signal)



Slika 35.9: Simulacioni dijagram genaratora periodičnog signala (impulsni signal)

ZADATAK 36 – Realizacija UART-a u programabilnoj logici

Posmatra se sistem za realizaciju u programabilnoj logici koji obavlja ispis broja preko osam cifara displeja. Broj koji se prikazuje je pozitivan i predstavlja se u heksadecimalnom zapisu. Svaka cifra broja se prikazuje preko sedmosegmentnog displeja koji sadrži tačku sa desne strane broja. Nizak logički nivo kojim se pobuđuje linija signala segmenta aktivira pridruženi segment, tj. čini ga svetlećim.



Slika 36.1: Blok šema sistema – organizacija sistema

Postavljanje vrednosti broja za prikaz obavlja se preko serijske komunikacije (UART podsistema). Nadređeni sistem (PC računar, μ C sistem ili sl.) preko serijske komunikacije prosleđuje sistemu podatak o vrednosti svake cifre pojedinačno. Podatak o pojedinačnoj cifri prosleđuje se protokolom koji je specificiran na sledeći način:

- nadređeni sistem šalje sistemu u programabilnoj logici poruku koja sadrži dva bajta pri čemu prvi bajt nosi informaciju o kodu komande (konstantna vrednost), a drugi bajt informaciju o vrednosti i poziciji cifre;
- kod komande, tj. prvi bajt poruke, je 0x10;
- vrednost i pozicija cifre prenosi se drugim bajtom poruke pri čemu niža četiri bita (B3..B0) poruke sadrže vrednost cifre (od 0x0 do 0xF za vrednosti od 0 do F), bit

težine 4 (*B4*) nosi informaciju o aktivnosti tačke koja je pridružena cifri dok tri bita najveće težine (*B7..B5*) prenose informaciju o poziciji cifre. Za vrednost pozicije 0x0 reč je o cifri najmanje težine (cifra sa desne strane) dok je za vrednost 0x7 reč o cifri najveće težine;

- nakon prihvatanja regularne poruke, sistem odgovara nadređenom centru da je uspešno prihvatio poruku slanjem jednobajtnog podatka vrednosti 0x90.

Blok šema sistema prikazana je na slici br. 36.1. Svaki prihvaćeni podatak o vrednosti cifre upisuje se u memoriju sistema koja se realizuje preko *dual-port ram* memorije (*lpm_ram_dp*). Memorija je kapaciteta 8x8bita pri čemu se na adresi 0 nalazi se podatak koji aktivira cifra najmanje težine na sedmosegmentnom displeju. Bit najmanje težine podatka u memoriji za aktiviranje sedmosegmentnog displeja odgovara segmentu *a*, dok je bit najveće težine pridružen vrednosti koja aktivira segment tačke.

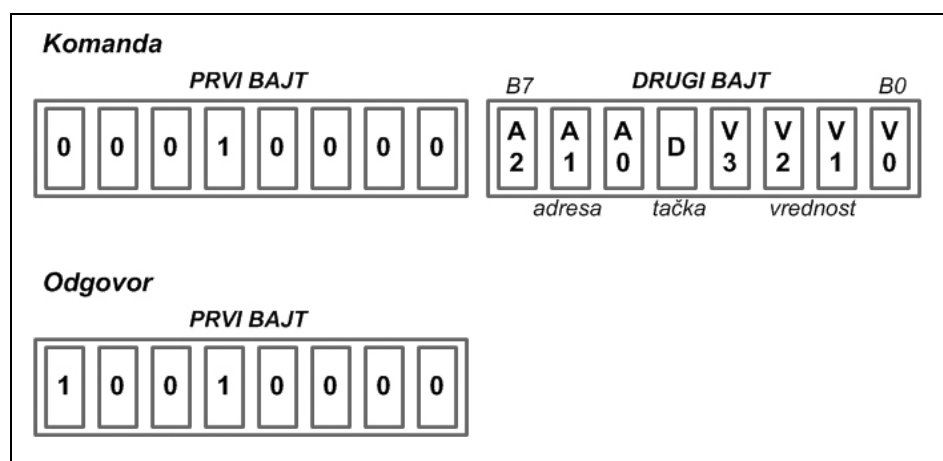
Podsistem *RCV_ACK_BLOCK* obavlja obradu prihvaćenih podataka, realizaciju mehanizma slanja odgovora i upis u memoriju sistema. UART komunikacija ostvaruje se preko podsistema *UART_RX*, *UART_TX* i *BAUDRATE*.

Realizovati deo sistema koji je naznačena na slici br. 36.1 naznačen, tj. komunikacioni deo, *RCV_ACK_BLOCK* i memoriju. Za komunikacione delove sistema koristiti gotove blokove *UART_RX*, *UART_TX* i *BAUDRATE*. UART podsistemi detaljno su obrađeni u okviru dvanaeste laboratorijske vežbe Priručnika iz programabilnih logičkih kola (treće izdanje). Memoriju sistema realizovati kao mega-funkciju *lpm_ram_dp*. Vrednost osnovnog sinhronizacionog takta sistema (*clk* signal) je 25,175MHz. UART podsistem ima mogućnost rada pri standardnim brzinama komunikacije tj. od 1200 do 115200bps.

Primenom alata za simulaciju izvršiti simulaciju rada delova sistema koji su bitni za valjanu verifikaciju opisanog sistema.

REŠENJE

Na osnovu specifikacije protokola iz zadatka formirana je šema rasporeda vrednosti u okviru podataka komande i odgovora i prikazana je na slici br. 36.2.



Slika 36.2: Šematski prikaz protokola

Centralno mesto podsistema *RCV_ACK_BLOCK* čini element *RCV_ACK* koji je opisan u VHDL-u i prikazan u nastavku. Element *RCV_ACK* obavlja kompletan mehanizam opisanog bloka *RCV_ACK_BLOCK* izuzev generisanja impulsa za aktiviranje slanja odgovora preko *tx_data_load* linije signala.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY rcv_ack IS PORT
(
    clk          : IN STD_LOGIC;
    reset        : IN STD_LOGIC;
    rx_ready     : IN STD_LOGIC;
    data_rx      : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    data_tx      : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
    value        : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
    wraddress    : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
    wren         : OUT STD_LOGIC
);
END rcv_ack;

ARCHITECTURE behav OF rcv_ack IS
    TYPE state_type IS (s0, s1, s2, s3);
    SIGNAL state : state_type;
    SIGNAL flag  : STD_LOGIC;
BEGIN

    data_tx      <= "10010000";
    wraddress    <= data_rx( 7 DOWNTO 5 );
    value(7)     <= NOT data_rx(4);

    -- Dekodiranje vrednosti broja za prikaz na sedmogmentni displej
    WITH data_rx( 3 DOWNTO 0 ) SELECT
        value( 6 DOWNTO 0 ) <= "0000001" WHEN "0000",
                                "1001111" WHEN "0001",
                                "0010010" WHEN "0010",
                                "0000110" WHEN "0011",
                                "1001100" WHEN "0100",
                                "0100100" WHEN "0101",
                                "0100000" WHEN "0110",
                                "0001111" WHEN "0111",
                                "0000000" WHEN "1000",
                                "0000100" WHEN "1001",
                                "0001000" WHEN "1010",
                                "1100000" WHEN "1011",
                                "0110001" WHEN "1100",
                                "1000010" WHEN "1101",
                                "0110000" WHEN "1110",
                                "0111000" WHEN OTHERS;

    -- Mehanizam obrade
    PROCESS ( clk, reset ) BEGIN
        IF ( reset = '1' ) THEN
            state <= s0;
            flag  <= '0';
            wren  <= '0';

            ELSIF ( rising_edge( clk ) ) THEN
                CASE state IS
                    WHEN s0 =>

```

```

        IF ( rx_ready = '1' ) THEN
            state <= s1;
        END IF;

    WHEN s1 =>
        IF ( flag = '0' ) THEN
            state <= s3;
            IF ( data_rx = "00010000" ) THEN
                flag <= '1';
            END IF;
        ELSE
            state <= s2;
            flag <= '0';
        END IF;

    WHEN s2 =>
        state <= s3;

    WHEN s3 =>
        IF ( rx_ready = '0' ) THEN
            state <= s0;
        END IF;
    END CASE;

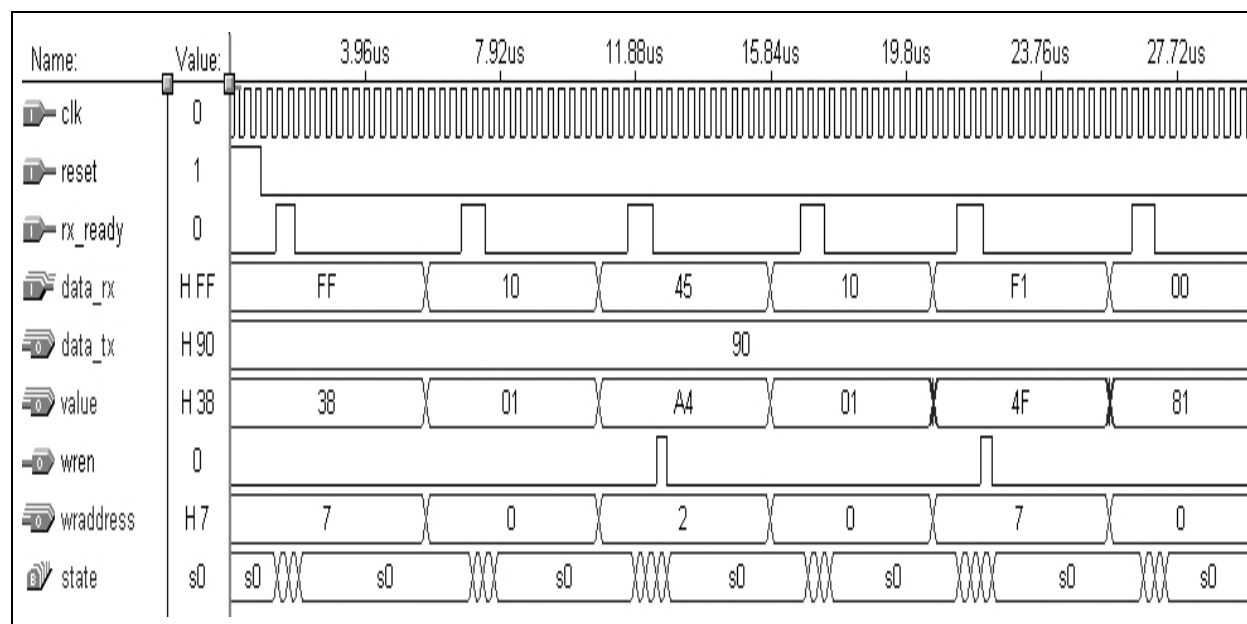
CASE state IS
    WHEN s0 => wren <= '0';
    WHEN s1 => wren <= '0';
    WHEN s2 => wren <= '1';
    WHEN s3 => wren <= '0';
END CASE;

END IF;
END PROCESS;
END behav ;

```

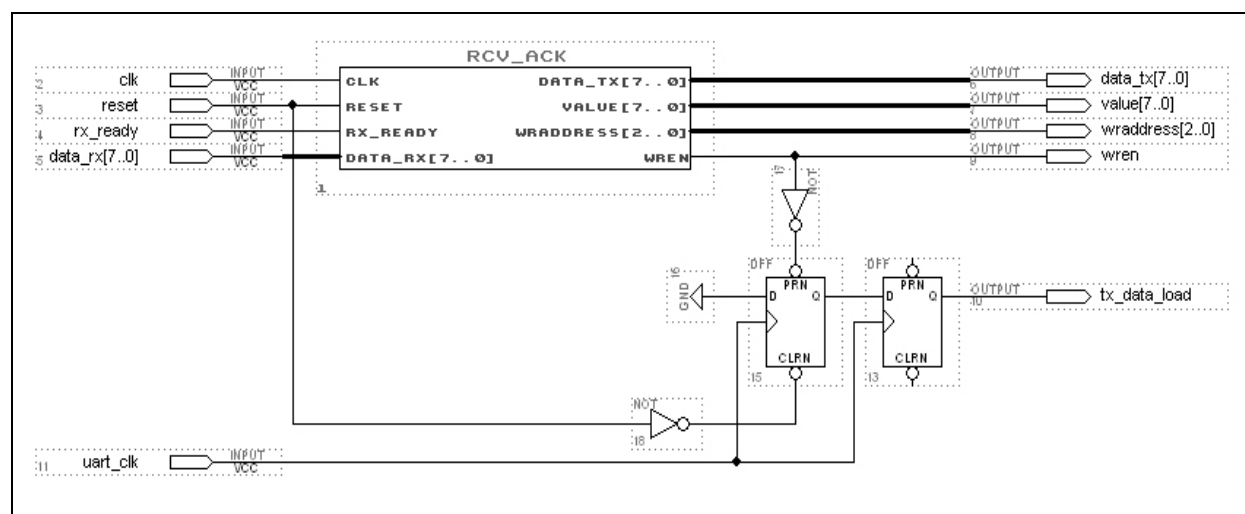
Rad podsistema *RCV_ACK* je realizovan preko mašine stanja (*state*) i četiri stanja *s0*, *s1*, *s2* i *s3*. Ceo sistem radi pri sinhronizacionom taktu *clk* (25,175MHz) koji je mnogo puta veći od takta UART sistema (*uart_clk*). *s0* je stanje u kojem se sistem nalazi kada nije u procesu obrade prihvaćenog podatka. U okviru stanja *s1* obavlja se određivanje da li sistem obrađuje kod komande (prvi bajt) ili vrednost tj. drugi bajt protokola. Signal *flag* nosi informaciju o tome koji bajt protokola se prihvata. U stanju *s2* vrši se aktiviranje mehanizma upisa u memoriju kao i slanje odgovora preko *UART_TX* podsistema nakon prijema drugog bajta protokola. U stanju *s3* sistem stoji zadržan sve dok je vrednost *rx_ready* signala 1.

Kako je širina impulsa *rx_ready* signala (jedna perioda *uart_clk* signala) znatno veća od tri periode *clk* signala sistem se uvek zadržava u stanju *s3* i time čeka sledeći podatak i otpočinjanje novog ciklusa. Na slici br. 36.3 dat je simulacioni dijagram podsistema *RCV_ACK*.



Slika 36.3: Simulacioni dijagram verifikacije rada podsistema *RCV_ACK*

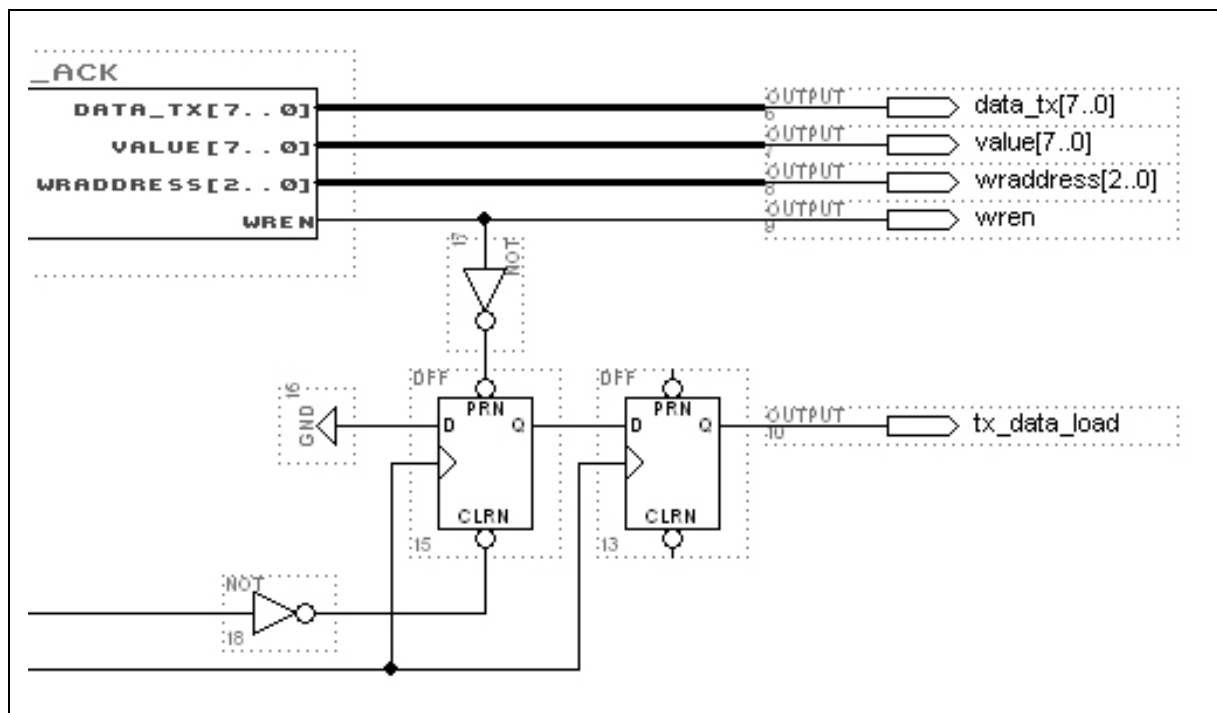
Organizacija podsistema *RCV_ACK_BLOCK* prikazana je na slici br. 36.4. Ovaj podsistem sadrži proširenu funkcionalnost *RCV_ACK* podsistema u pogledu generisanja signala za aktiviranje slanja odgovora (*tx_data_load*).



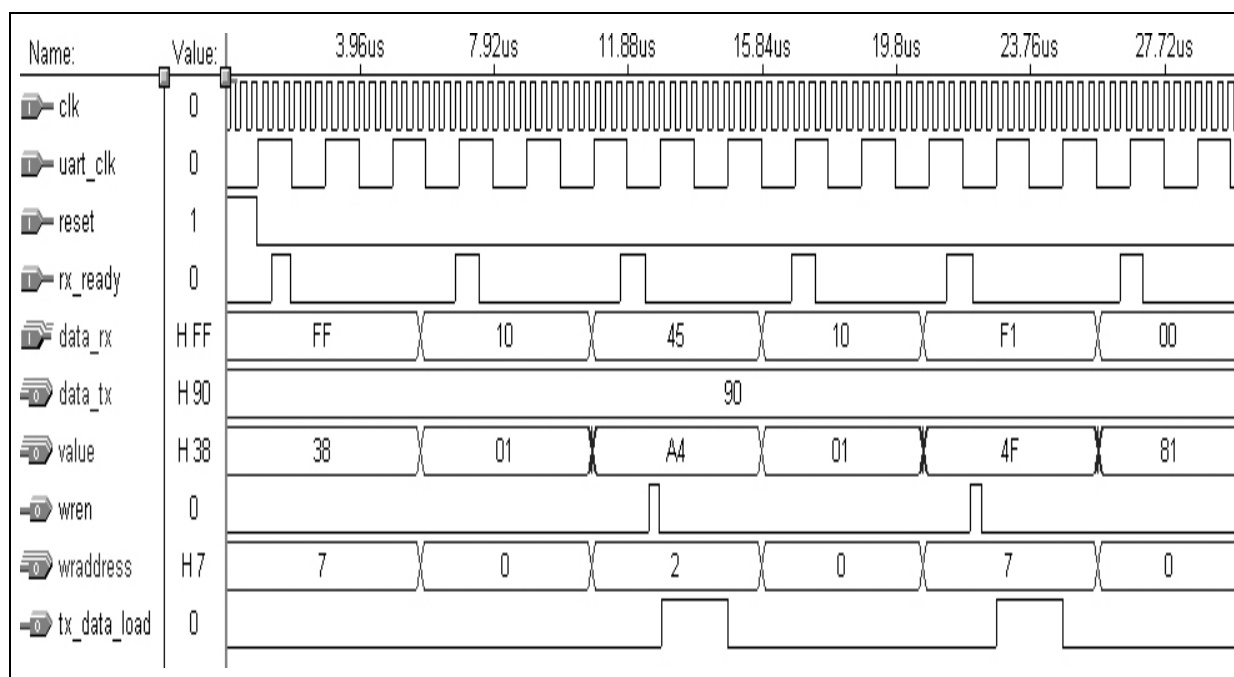
Slika 36.4: Šema ćelije brojača na bazi kola za inkrementiranje

Element za generisanje signala *tx_data_load* trajanja jedne periode *uart_clk* signala takta dat je na slici br. 36.5. Impuls *tx_data_load* se generiše neposredno nakon generisanja impulsa *wren* signala pri čemu je impuls *tx_data_load* signala sinhron sa *uart_clk* taktom.

Na slici br. 36.3 dat je simulacioni dijagram podsistema *RCV_ACK_BLOCK*. Na prethodnom i ovom simulacionom dijagramu vrednosti signala nisu u skladu sa realnim vrednostima. U realnim uslovima *uart_clk* signal je znatno niže učestanosti od sinhronizacionog signala *clk*.



Slika 36.5: Šema za generisanje impulsa signala *tx_data_load* trajanja jedne periode *uart_clk* sinhronizacionog signala



Slika 36.6: Simulacioni dijagram verifikacije rada podsistema *RCV_ACK_BLOCK*

Na slici br. 36.7 data je kompletna šema dela sistema koji se zahteva u zadatku. Preko signala *rdaddress[2..0]*, *rden* i *data_mem[7..0]* vrši se iščitavanje sadržaja memorije u cilju prihvatanja baferisanih vrednosti cifara za aktiviranje sedmosegmentnih displeja. Realizacija dela sistema koji obavlja čitanje memorije i aktiviranje sedmosegmentnih displeja nije deo ovog zadatka.

Povezivanje UART podsistema u skladu je sa opisom koji je obavljen u okviru dvanaeste laboratorijske vežbe Priručnik iz programabilnih logičkih kola (treće izdanje).

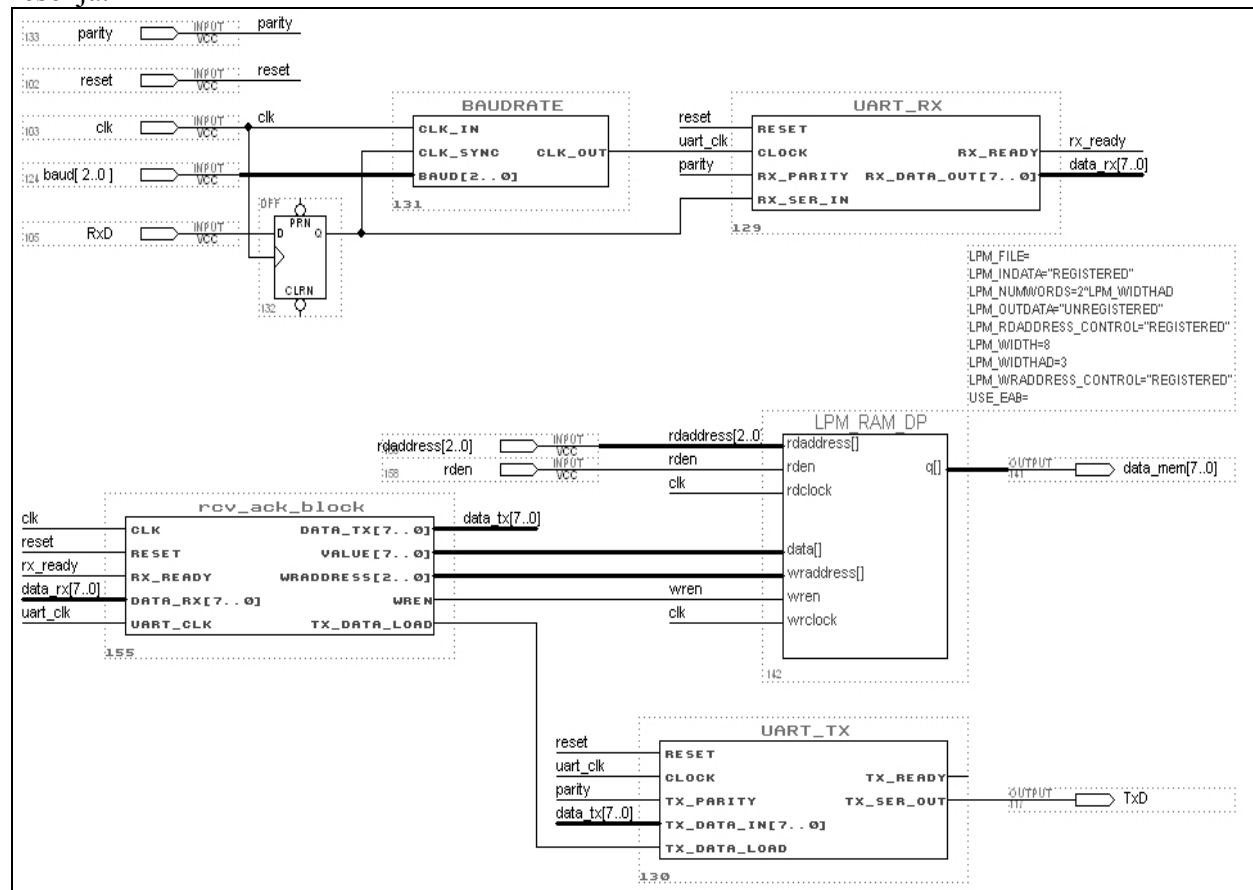
Memorija sistema realizovana je preko mega-funkcije *lpm_ram_dp* i postavljanjem sledećih parametara u cilju realizacije *dual-port* RAM memorije kapaciteta 8x8 bita:

```
LPM_INDATA="REGISTERED"
LPM_OUTDATA="UNREGISTERED"
LPM_WIDTH=8
LPM_WIDTHAD=3
```

Signali *lpm_ram_dp* koji se koriste (*Used*) za realizaciju memorije prikazani su u nastavku dok ostali signali nisu od interesa.

```
rdaddress[], rden, rdclock,
data[],
waddress[], wren, wrclock,
q[]
```

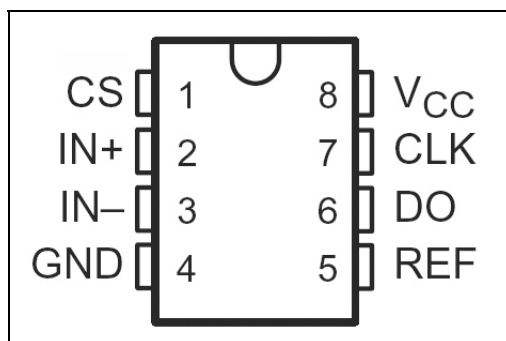
Zbog kompleksnosti i obimnosti rezultata, simulacioni dijagrami nisu prikazani u okviru rešenja.



Slika 36.7: Šema ćelije brojača na bazi kola za inkrementiranje

ZADATAK 37 – Sistem sprege sa AD-konvertorom tipa ADC0831

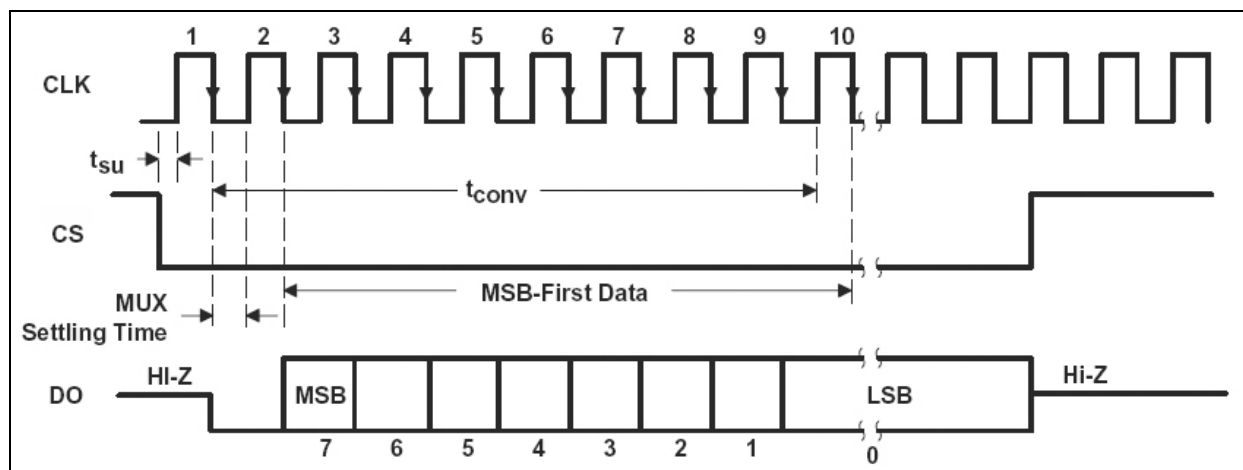
Realizovati u programabilnoj logici sistem koji obezbeđuje prikupljanje podataka analogno digitalne konverzije sa AD konvertora tipa ADC0831. Učestanost odabiranja, tj. prikupljanja podataka sa AD konvertora, treba da bude 25kHz. Za potrebe rešavanja zadatka dat je kratak opis i osnovni parametri kola ADC0831.



Slika 37.1: Simbol AD konvertora ADC0831

ADC0831 (u daljem tekstu ADC) je osmobaritni serijski AD konvertor. Simbol ADC0831 kola dat je na slici br. 37.1. Priključci ADC su:

- *VCC* i *GND* – priključci za napajanje kola;
- *IN+*, *IN-* –diferencijalni priključci analognog ulaza ADC;
- *REF* – referentni napon za potrebe analogno-digitalne konverzije i
- *CS*, *CLK* i *DO* – tri digitalna signala za obezbeđivanje serijske komunikacije sa nadređenim sistemom.



Slika 37.2: Vremenski dijagram komunikacije sa AD konvertorom ADC0831

Za potrebe ostvarivanja sprege ADC i programabilne logike od najvećeg interesa su digitalni signali *CS*, *CLK* i *DO* kao i vremenski oblici ovih signala. Vremenski dijagrami signala *CLK*, *CS* i *DO* pri akviziciji ADC i prenosu podataka konverzije prikazani je na slici br. 37.2. Preko *CLK* signala dovodi se sinhronizacioni takt za rad ADC. Učestanost *CLK* signala određuje vreme konverzije t_{CONV} (slika br. 37.2) kao i brzinu prenosa podataka. ADC0831 može da radi na učestanost *CLK* signala u opsegu od 10 do 400 kHz. Npr. vreme konverzije ADC je 32μsec pri učestanosti *CLK* signala od 250 kHz. Treba obezbediti da promena *CS* signala sa visokog

(neaktivnog) na nizak (aktivan) logički nivo bude najmanje 350nsec pre usponske ivice *CLK* signala tj. treba da bude $t_{SU} \geq 350\text{nsec}$.

Obratiti pažnju da se podatak na liniji *DO* formira pri opadajućoj ivici *CLK* signala i to prvi bit iz povorke od osam bita *AD* konverzije (*MSB*) se generiše se pri drugoj opadajućoj ivici *CLK* signala.

Programabilna logika na kojoj treba realizovati sistem za spregu sa *ADC* radi na učestanosti od 25.175MHz.

Sistema za spregu sa *ADC* realizovati preko implementacije dva podsistema i to:

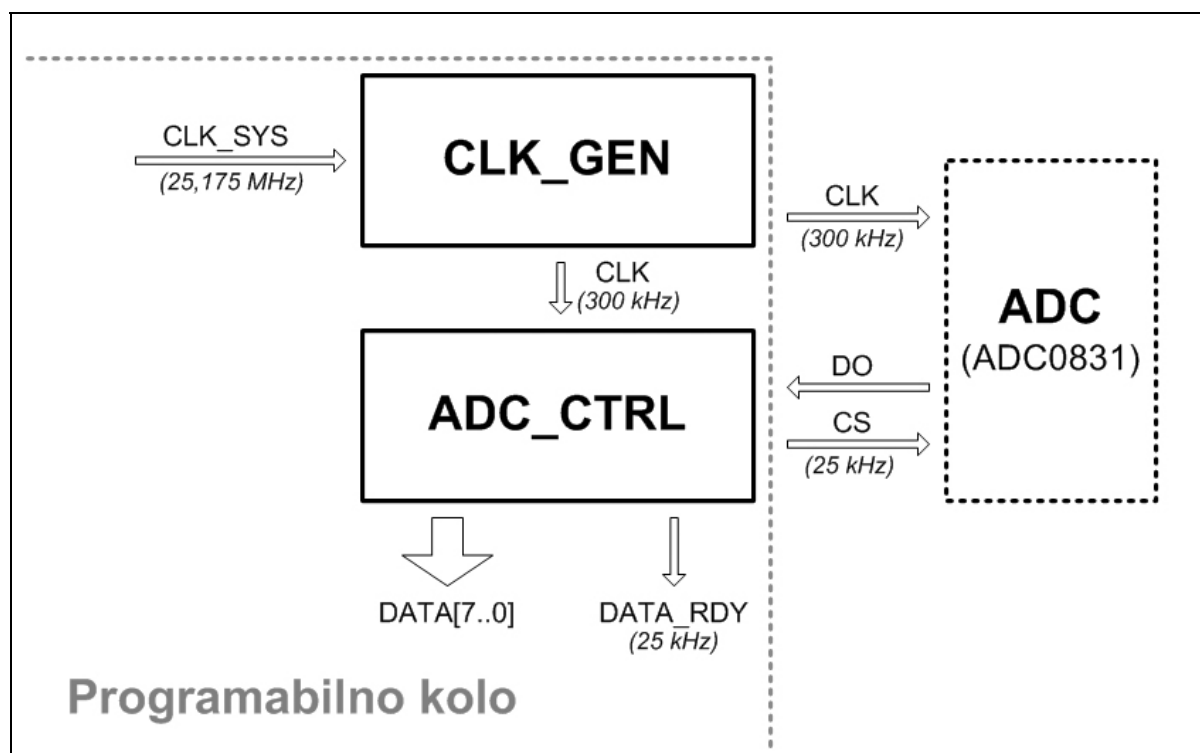
- podsistema za generisanje potrebne učestanosti i
- podsistema za generisanje signala *CLK* i *CS* kao i prihvatanje podataka konverzije od *ADC* preko signala *DO*.

Podsisteme sistema za spregu sa *ADC* realizovati primenom mega-funkcija i preko *VHDL* opisa. Sistem u celini opisati u grafičkom editoru i primeniti hijerarhijsku organizaciju.

Primenom alata za simulaciju i pravilnim izborom vremenskih oblika signala, vremena trajanja simulacije i vremenskog inkrementa izvršiti verifikaciju rada opisanog sistema.

REŠENJE

Na osnovu specifikacije zadatka i opisa rada kola *ADC0831* formira se blok dijagram organizacije sistema za rad sa kolom *ADC0831* (*ADC*) i prikupljanje podataka analogno-digitalne konverzije. Organizacija sistema je prikazana na slici br. 37.3. Komunikacija programabilne logike sa *ADC* ostvaruje se preko linija signala *CLK*, *DO* i *CS*.



Slika 37.3: Organizacija sistema za opsluživanje *ADC* (*ADC0831*)

Da bi se u potpunosti specificirao sistem potrebno je definisati način komunikacije sa ostatkom sistema u okviru programabilne logike, a sve u cilju prenosa vrednosti dobijene analogno-digitalnom konverzijom. Prihvaćena vrednost analogno-digitalne konverzije dostupna je ostatku sistema u okviru programabilne logike preko signala:

- $DATA[7..0]$ – signala vrednosti konverzije (osmobitna vrednost od 0 do 0xFF) i
- $DATA_RDY$ – signala indikacije pristiglog podatka sa ADC.

Impuls visokog logičkog nivoa trajanja jedne periode CLK signala preko linije signala $DATA_RDY$ daje informaciju da je došlo do prihvatanja nove vrednosti analogno digitalne konverzije. Sve vreme trajanja impulsa signala $DATA_RDY$ stabilna je vrednost podatka na linijama signala $DATA[7..0]$.

Na osnovu vremenskog dijagrama rada ADC (slika br. 37.2) uočava se da jedan ciklus konverzije iznosi 10 ili više perioda CLK signala. Signal CLK formira se od sistemskog sinhronizacionog signala koji je učestanosti 25.175MHz. U cilju određivanja broja ciklusa CLK signala (N) po jednoj periodi odabiranja ($1/25kHz$) kao i učestanost CLK signala f_{CLK} formiraju se sledeće jednačine:

$$f_{CLK} = 25kHz \cdot N$$

$$25.175MHz = f_{CLK} \cdot M$$

pri čemu treba da bude zadovoljeno:

$$N \geq 10, M \geq 1, \text{ gde su } M \text{ i } N \text{ celi pozitivni brojevi i}$$

$$10kHz \leq f_{CLK} \leq 400kHz.$$

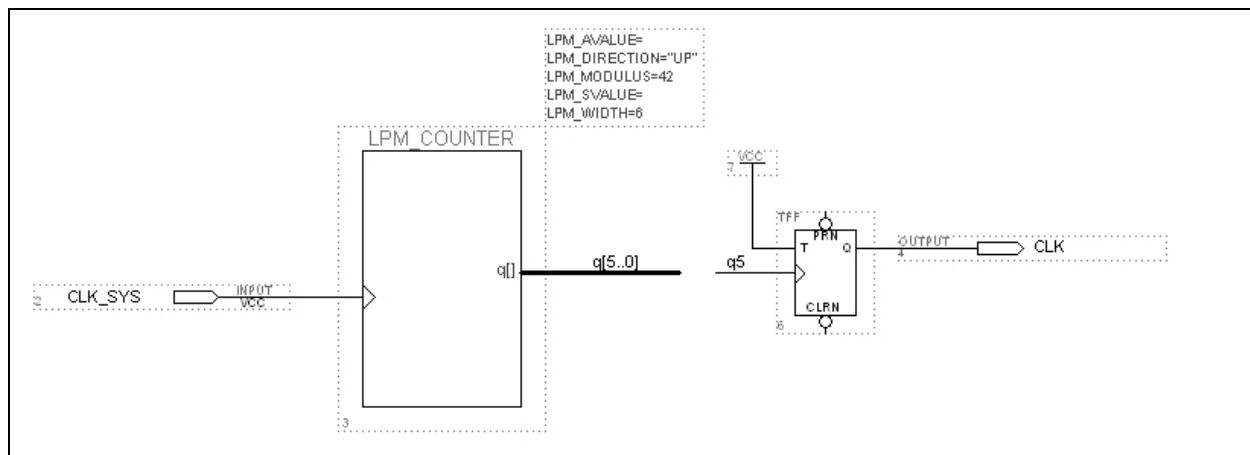
Jedna od kombinacija vrednosti brojeva M i N za koje prethodne jednačine važe i pri čemu su ispunjeni navedeni uslovi su:

$$N = 12 \text{ i } M = 84.$$

Za ove vrednosti brojeva M i N učestanost CLK signala je $f_{CLK} = 300kHz$.

Podsistem CLK_GEN – Generator signala CLK

Generator signala CLK_GEN formira periodičan signal CLK učestanosti $f_{CLK} = 300kHz$ iz sistemskog sinhronizacionog signala CLK_SYS ($f_{CLK_SYS} = 25.175MHz$). Realizacija podistema CLK_GEN data je na slici br. 37.4.

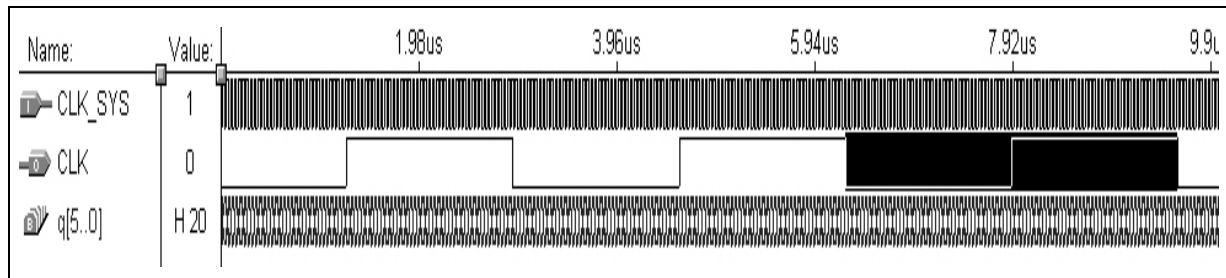


Slika 37.4: Realizacija podistema CLK_GEN

CLK_GEN realizovan je preko brojača i T-FF kola. Brojač je realizovan preko megafunkcije *lpm_counter* i to kao šestobitni brojač nagore po modulu brojanja 42. Bit najveće težine sa izlaza brojača $q5$ dovodi se na clk ulaz T-FF koji radi u konfiguraciji delitelja učestanosti sa koeficijentom 2. Na ovaj način dobija se učestanost od 300kHz deljenjem

učestanosti od 25,175MHz sa brojem 84. Ovakva realizacija obezbeđuje da signal na izlazu, tj. *CLK*, ima simetričnu povorku impulsa i pauza.

Na slici br. 37.5 dat je simulacioni dijagram verifikacije rada sistema *CLK_GEN*. Na simulacionom dijagramu je generisana učestanost *CLK_SYS* signala od 25,175MHz. Jedna perioda *CLK* signala markirana je na istom dijagramu i iznosi tačno 3,33μsec što odgovara učestanosti od 300kHz.



Slika 37.5: Simulacioni dijagram podsistema *CLK_GEN*

ADC_CTRL – podistem za spregu sa ADC (ADC0831)

Drugi podsistem (*ADC_CTRL*) predstavlja kolo za spregu sa ADC. Podsistem *ADC_CTRL* generiše signal *CS*, prihvata podatke sa linije signala *DO* koje generiše ADC i analizira ih u cilju izdvajanja vrednosti analogno-digitalne konverzije. Vrednosti analogno-digitalne konverzije dostupne su ostatku sistema preko signala *DATA[7..0]* i *DATA_RDY*. VHDL kod *ARC_CTRL* podsistema prikazan je u nastavku:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY adc_ctrl IS
PORT
(
    CLK      : IN STD_LOGIC;
    DO       : IN STD_LOGIC;
    CS       : OUT STD_LOGIC;
    DATA_RDY : OUT STD_LOGIC;
    DATA    : OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0 )
);
END adc_ctrl;

ARCHITECTURE behav OF adc_ctrl IS
    CONSTANT max_period      : NATURAL := 11;
    CONSTANT max_cs          : NATURAL := 10;
    SIGNAL    temp_rdy        : STD_LOGIC;
    SIGNAL    tmp_cs          : STD_LOGIC;
    SIGNAL    tmp_data_rdy    : STD_LOGIC;
    SIGNAL    tmp_data        : STD_LOGIC_VECTOR ( 6 DOWNTO 0 );

    BEGIN
        PROCESS ( CLK )

```

```

    VARIABLE cnt : NATURAL RANGE 0 TO max_period;
BEGIN
    -- Aktivnosti na opadajucoj ivici CLK signala
    IF ( CLK'EVENT AND CLK='0' ) THEN
        IF ( cnt >= max_period ) THEN
            cnt := 0;
            tmp_cs <= '0';
        ELSE
            cnt := cnt + 1;
        END IF;

        IF ( cnt = max_cs ) THEN
            tmp_cs <= '1';
            tmp_data_rdy <= '1';
        ELSIF ( cnt = max_cs + 1 ) THEN
            tmp_data_rdy <= '0';
        END IF;
    END IF;

    -- Aktivnoski na usponskoj ivici CLK signala
    IF ( CLK'EVENT AND CLK='1' ) THEN
        IF ( cnt >= 2 AND cnt < 9 ) THEN
            tmp_data <= tmp_data ( 5 DOWNT0 0 ) & DO;
        ELSIF ( cnt = 9 ) THEN
            DATA <= tmp_data ( 6 DOWNT0 0 ) & DO;
        END IF;
    END IF;
END PROCESS;

CS <= tmp_cs;
DATA_RDY <= tmp_data_rdy;
END behav;

```

Treba napomenuti da se uslov $t_{SU} \geq 350\text{nsec}$ realizuje generisanjem opadajuće ivice CS signala sinhrono sa opadajućom ivicom CLK signala. Na ovaj način i pri učestanosti CLK signala od 300kHz obezbeđuje se da bude $t_{SU} = 1.67\mu\text{sec}$ čime je ispunjen uslov. Generisanje CS i DATA_RDY signala obavlja se sinhrono sa opadajućom ivicom CLK signala. Prihvatanje podataka sa DO linije signala obavlja se sinhrono sa usponskom ivicom CLK signala. Ovakva realizacija ostvarena je kroz istu **PROCESS** strukturu i dve nezavisne **IF...END IF** strukture.

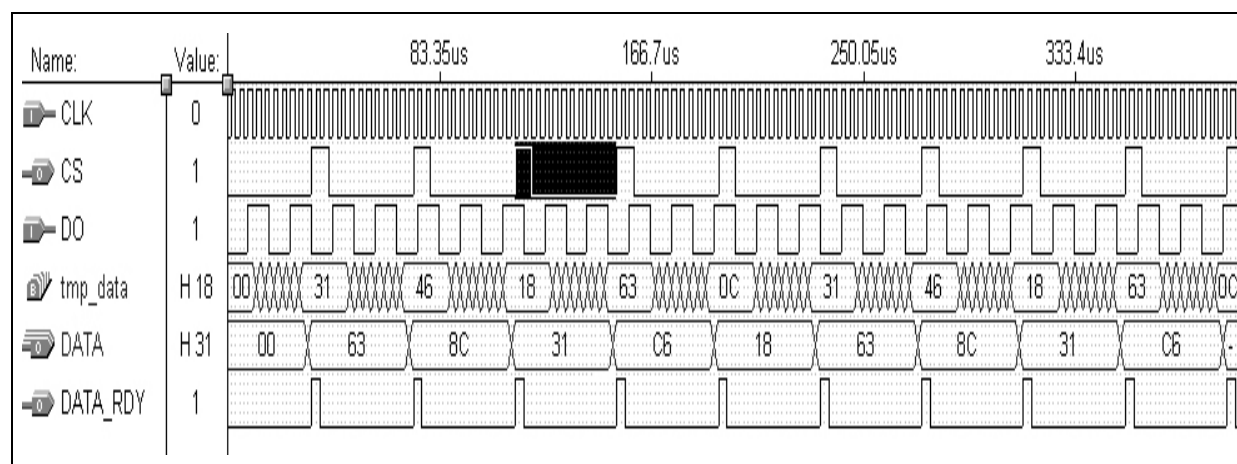
```

PROCESS ( CLK )
BEGIN
    -- Aktivnosti na opadajucoj ivici CLK signala
    IF ( CLK'EVENT AND CLK='0' ) THEN
        ...
    END IF;
    -- Aktivnosti na usponskoj ivici CLK signala
    IF ( CLK'EVENT AND CLK='1' ) THEN
        ...
    END IF;
END PROCESS;

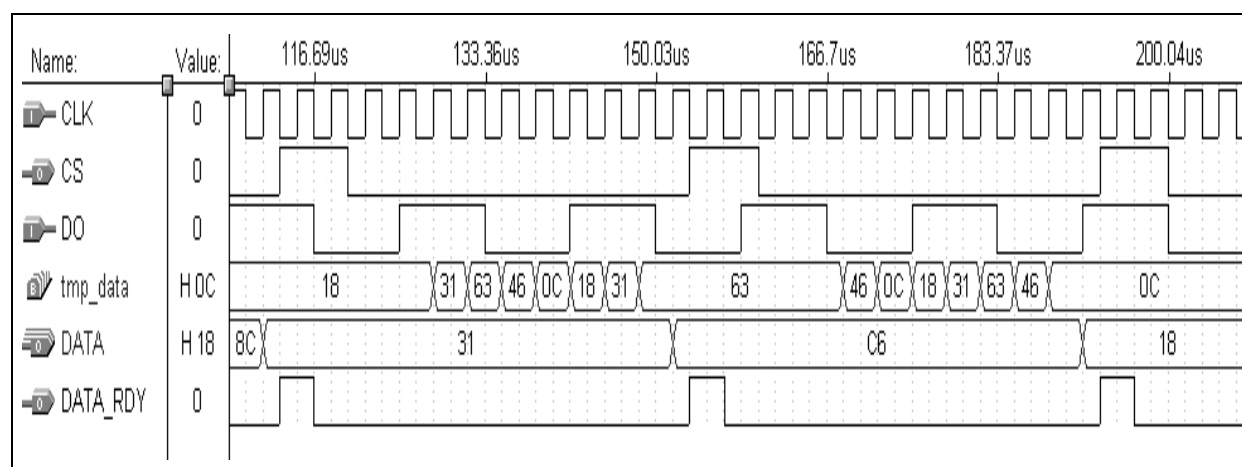
```

Pomoćni brojač *cnt* broji periode *CLK* signala u toku jednog ciklusa konverzije. Na bazi vrednosti *clk* vrši se privremeno baferisanje vrednosti analogno-digitalne konverzije (preko *tmp_data*) i na kraju upis u izlazni registar (*DATA*).

Simulacioni dijagrami za verifikaciju rada ADC_CTRL podsistema prikazani su na slici br. 37.6 i br. 37.7. Radi pravilne analize rada podsistema, učestanost *CLK* signala postavljena je na 300kHz a signal *DO* je proizvoljno generisan. Zatamnjena oblast na liniji signala *CS* predstavlja jedan ciklus analogno-digitalne konverzije koji traje 40μsec.



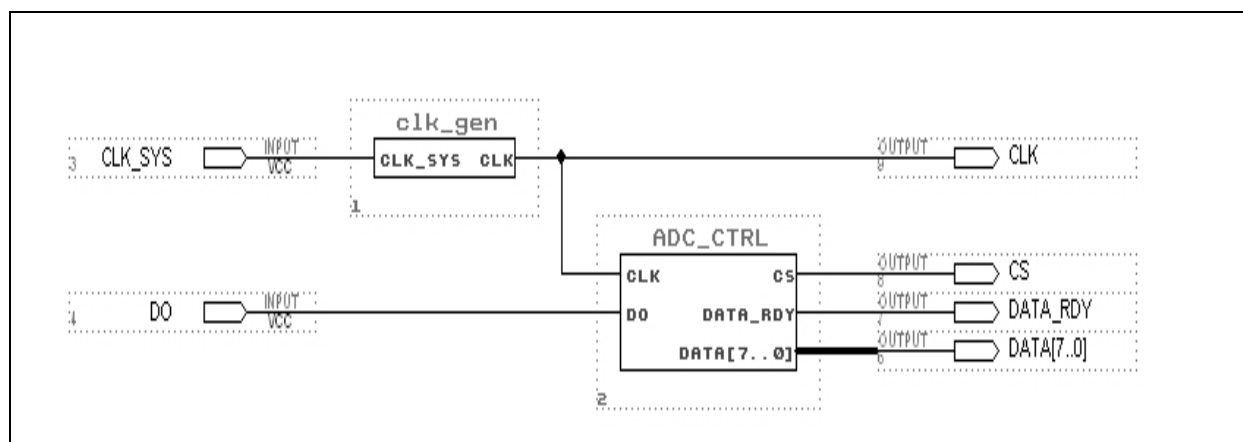
Slika 37.6: Simulacioni dijagram ADC_CTRL podsistema



Slika 37.7: Simulacioni dijagram ADC_CTRL podsistema (detaljno)

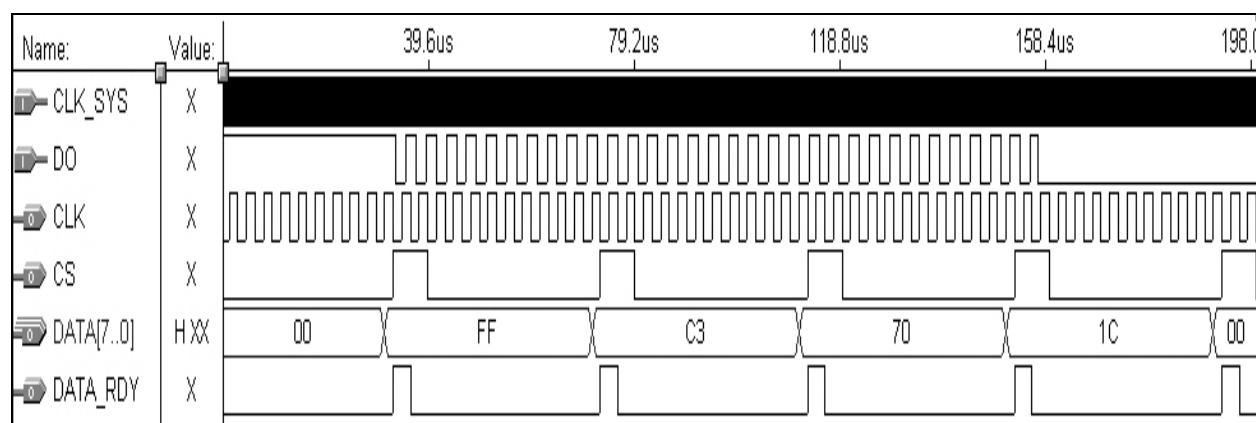
Sistem u celini

Sistem u celini prikazan je na slici br. 37.8.



Slika 37.8: Realizacija sistema u celini na bazi ADC_CTRL i CLK_GEN (clk_gen) podsistema

Zbog korišćenja nemodifikovanog *CLK* signala za ADC_CTRL podistem kao i za rad ADC kola ovaj signal se koristi direktno za obe primene iz CLK_GEN podsistema.



Slika 37.9: Simulacioni dijagram sistema u celini

Simulacioni dijagram za verifikaciju rada sistema u celini prikazan je na slici br. 37.9. Učestanost *CLK_SYS* signala je 25,175MHz što se na dijagramu uočava kao zatamnjena oblast (prva linija signala simulacionog dijagrama). Detaljnom analizom dijagrama utvrđuje se da sistem ispunjava zahteve zadatka.

Literatura

1. Dragana Prokin, Dušan Todović, “Programabilna logička kola, priručnik za laboratorijske vežbe”, Viša elektrotehnička škola u Beogradu, Akademska izdanja, Beograd 2007.
2. Milan Prokin “Računarska elektronika”, Elektrotehnički fakultet u Beogradu, Akademska misao, Beograd, 2006.
3. Dejan Živković, Miodrag Popović, “Impulsna i digitalna elektronika”, Elektrotehnički fakultet u Beogradu, Akademska misao, Beograd, 2004.
4. Vladimir Kovačević “Logičko projektovanje računarskih sistema i projektovanje digitalnih sistema”, Fakultet tehničkih nauka u Novom Sadu, Srem DOO, Novi Sad, 2001.
5. Spasoje Tešić, Dragan Vasiljević, “Zbirka zadataka iz digitalne elektronike”, Naučna knjiga, Beograd 1981.
6. Zainalabedin Navabi “Digital Design and Implementation with Field Programmable Devices”, Kluwer Academic Publishers, Massachusetts, USA, 2005
7. Uwe Meyer-Baese “Digital Signal Processing with Field Programmable Gate Arrays”, Springer-Verlag Berlin, Heidelberg, 2004
8. Thomas L. Floyd, David M. Buchla, “The Science of Electronics – Digital”, Pearson Prentice Hall, New Jersey, 2005
9. Douglas L. Perry, “VHDL”, McGraw-Hill, USA, 1991
10. MAX 7000 – Programmable Logic Device Family, *Data Sheet*, ALTERA, june 2003, ver. 6.6
11. FLEX 10K – Embedded Programmable Logic Device Family, *Data Sheet*, ALTERA, januar 2003, ver. 4.2
12. MAX+PLUS II Programmable Logic Development System & Software, *Data Sheet*, ALTERA, jan. 1998, ver. 8.0
13. Introduction to Quartus II, ALTERA, 2006
14. Dokumenti sa zvanične prezentacije firme ALTERA - www.altera.com



by GER