# POGLAVLJE 6.
# SLOŽENI DIGITALNI SISTEMI

U ovom poglavlju je prikazan jedan način projektovanja složenih digitalnih sitema pomoću VHDL jezika za opis fizičke arhitekture (eng. *Very high speed integrated circuit HDL*). Većina digitalnih sistema, čije se projektovanje prikazuje u ovom poglavlju, čine sistemi koji se sastoje od jedne sekvencijalne i jedne kombinacione mreže. Prikazana rešenja predstavljaju samo uvod u projektovanje složenih digitalnih sistema, radi jednostavnijeg razumevanja rešenja složenih sistema koja se javljaju u narednim poglavljima.

Definicija složenog digitalnog sistema preko kombinacionih tabela i tabela prelaza i izlaza nije praktična zbog činjenice da složeni digitalni sistem sadrži veliki broj stanja tako da predloženi metodi sinteze ne daju upotrebljiv rezultat ili su previše dugotrajni i podložni greškama. Stoga se kao osnovni problem postavlja problem izgradnje složenog digitalnog sistema kao celine iz jasno definisanih i raspoloživih komponenti (kombinacionih i sekvencijalnih mreža).

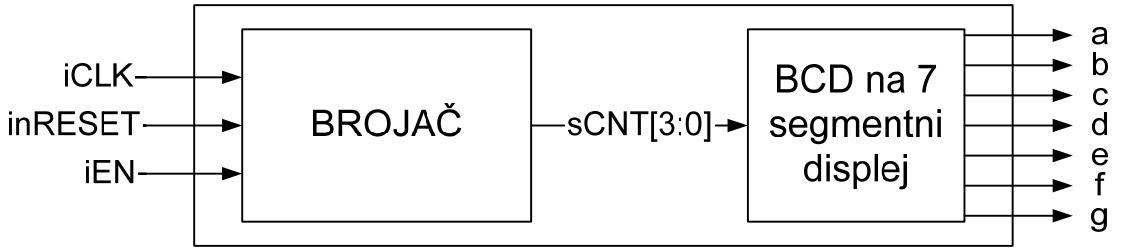
Drugim rečima, spajanje podsistema za realizaciju digitalnih funkcija u cilju formiranja digitalnog sistema ne može se opisati sredstvima kojima su opisivane kombinacione i sekvencijalne mreže. Bulova algebra i teorija automata, iako pogodni za nivo logičkih elemenata i elementarnih automata, nisu pogodni za opis na nivou pojedinih funkcionalnih blokova sistema.

Projektovanje složenih digitalnih sistema započinje se specifikacijom složenog digitalnog sistema kao celine i njegovom podelom na podsisteme, sve do elementarnih realizovanih komponenti. Za svaku komponentu, odnosno podsistem, se na jasan i jednoznačan način definiše arhitektura i njeno ponašanje.

Za opis arhitekture i ponašanja komponenti digitalnog sistema i njihovih veza, čime se formiraju složeni digitalni sistemi, koriste se jezici za opis fizičke arhitekture (eng. *Hardware Description Languages – HDL*). Ovi jezici su razvijeni radi opisa izuzetno složenih digitalnih sistema, gde se umesto rukovanja sa prenosnim funkcijama kombinacionih i sekvencijalnih mreža koje čine složeni digitalni sistem, pomoću HDL jezika opisuje njihova funkcija, a generisanje odgovarajućih prenosnih funkcija se prepušta HDL prevodiocu. U ovoj knjizi se koristi jezik za opis fizičke arhitekture pod imenom VHDL (eng. *Very high speed integrated circuit HDL*).

## 6.1 ZADATAK:

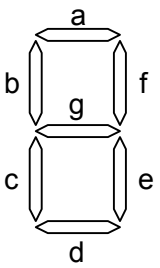
Pomoću VHDL jezika za opis fizičke arhitekture isprojektovati digitalni sistem koji prikazuje Slika 6.1.




Slika 6.1: Blok dijagram digitalnog sistema

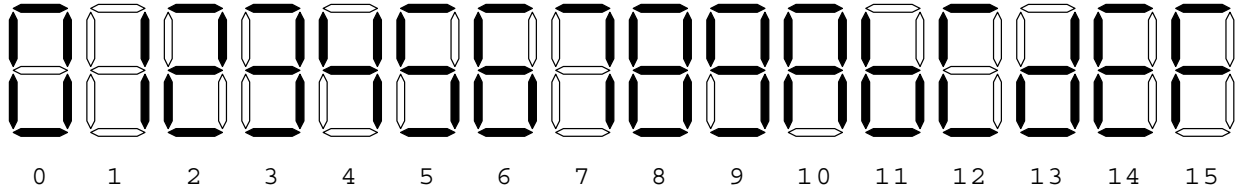
Sistem treba da obezbedi sledeće funkcije:

- sinhroni reset brojača signalom koji je aktivan na niskom naponskom nivou (`inRESET`),
- kontrolu brojanja pomoću signala koji je aktivan na viskom naponskom nivou (`iEN`),
- stanje četvorobitnog brojača se menja na rastuću ivicu takt signala `iCLK`,
- stanje brojača prikazuje signal `sCNT[3:0]`,
- mogućnost prikazivanja stanja brojača na 7 segmentnom displeju, Slika 6.2.




Slika 6.2: Izgled 7 segmentnog displeja

Slika 6.3 prikazuje stanje 7 segementnog displeja u zavisnosti od stanja brojača. Segment na 7 segmentnom displeju je aktivan, tj. svetli, kada je odgovarajući signal na visokom naponskom nivou.




Slika 6.3: Prikazivanje stanja brojača na 7 segmentnom displeju

**REŠENJE:**

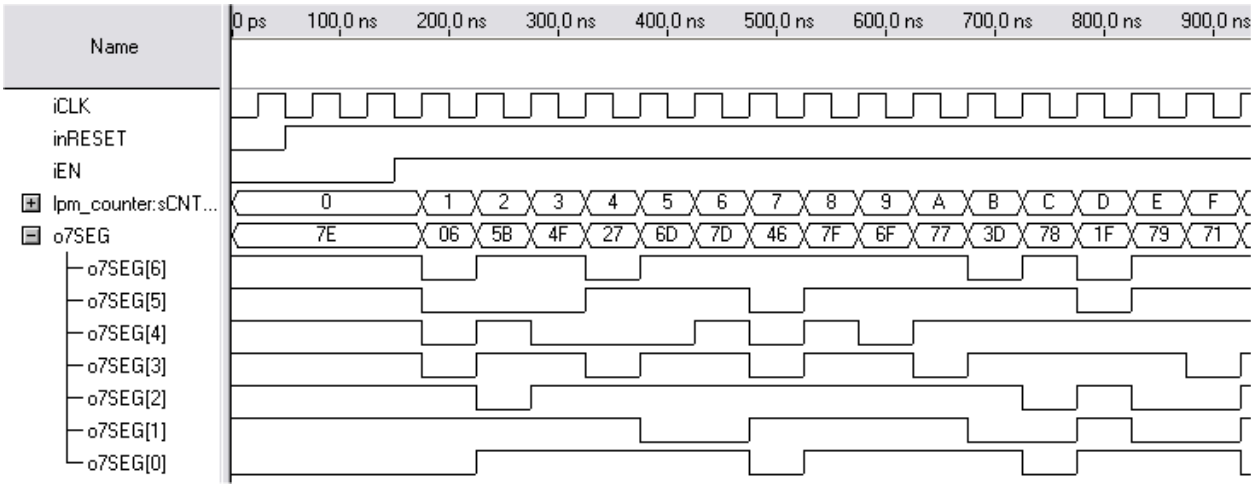
Traženi sistem se sastoji iz dva sastavna dela. Prvi je četvorobitni BCD brojač sa sinhronim postavljanjem u početno stanje i sa signalom dozvole brojanja. Drugi deo je zapravo koder koji trenutno stanje brojača koduje u oblik pogodan za prikazivanje na 7 segmentnom displeju. Vidi se da je prvi deo sistema sekvencijalan, dok drugi deo predstavlja tipičnu kombinacionu mrežu. Stoga je potrebno razdvojiti njihove realizacije. U ovom slučaju je najjednosnostavnije sekvencijalni deo realizovati pomoću jednog procesa, a kombinacionu mrežu drugim procesom. Svi ulazni signali digitalnog sistema direktno utiču na ponašanje prvog procesa, dok drugi proces generiše izlazne signale digitalnog sistema. Pri tome je izlaz iz prvog procesa jedini ulaz u drugi proces. VHDL kod ovakve realizacije traženog digitalnog sistema je prikazan u nastavku teksta.

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

ENTITY ZAD01 IS PORT (
  iCLK    : IN  STD_LOGIC;
  inRESET,
  iEN     : IN  STD_LOGIC;
  o7SEG   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0) );
END ZAD01;

ARCHITECTURE ARH_ZAD01 OF ZAD01 IS
  -- stanje brojaca
  SIGNAL sCNT: UNSIGNED(3 DOWNTO 0);
BEGIN
  -- brojac sa sinhronim resetom
  PROCESS(iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      IF  (inRESET = '0') THEN -- sinhroni reset
        sCNT <= "0000";
      ELSE
        IF (iEN = '1') THEN     -- dozvoljeno brojanje?
          sCNT <= sCNT + 1;     -- brojanje
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- dekoder BCD vrednosti na 7 segmentni displej
  PROCESS (sCNT) BEGIN
    CASE sCNT IS
      --                          ABCDEFG
      WHEN "0000" => o7SEG <= "1111110";
      WHEN "0001" => o7SEG <= "0000110";
      WHEN "0010" => o7SEG <= "1011011";
      WHEN "0011" => o7SEG <= "1001111";
      WHEN "0100" => o7SEG <= "0100111";
      WHEN "0101" => o7SEG <= "1101101";
      WHEN "0110" => o7SEG <= "1111101";
      WHEN "0111" => o7SEG <= "1000110";
```

```
        WHEN "1000" => o7SEG <= "1111111";
        WHEN "1001" => o7SEG <= "1101111";
        WHEN "1010" => o7SEG <= "1110111";
        WHEN "1011" => o7SEG <= "0111101";
        WHEN "1100" => o7SEG <= "1111000";
        WHEN "1101" => o7SEG <= "0011111";
        WHEN "1110" => o7SEG <= "1111001";
        WHEN OTHERS => o7SEG <= "1110001";
      END CASE;
    END PROCESS;
  END ARH_ZAD01;
```

Prikazani VHDL kod odgovara blok šemi traženog digitalnog sistema, Slika 6.1. Pri tome su svi izlazni signali (`a`, `b`, `c`, `d`, `e`, `f`, `g`) objedinjeni jednim izlaznim vektorom `o7SEG`. Bit najveće važnosti ovog vektora odgovara izlaznom signalu koji upravlja segmentom `a`. Redom slede signali koji upravljaju ostalim segmentima sve do bita najniže važnosti kojim upravlja segmentom `g`.

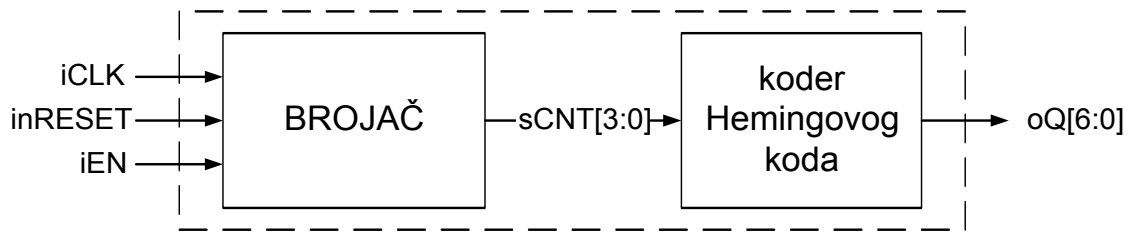
Vremenski dijagram simulacije rada realizovanog digitalnog sistema prikazuje Slika 6.4.




Slika 6.4: Simulacija rada realizovanog sistema

## 6.2 ZADATAK:

Pomoću VHDL jezika za opis fizičke arhitekture isprojektovati digitalni sistem koji prikazuje Slika 6.5.




Slika 6.5: Blok dijagram digitalnog sistema

Sistem treba da obezbedi sledeće funkcije:

- sinhroni reset brojača signalom koji je aktivan na niskom naponskom nivou (`inRESET`),
- kontrolu brojanja pomoću signala koji je aktivan na visokom naponskom nivou (`iEN`),
- stanje četvorobitnog brojača se menja na rastuću ivicu takt signala `iCLK`,
- stanje brojača prikazuje signal `sCNT[3:0]`,
- kodovanje stanja brojača pomoću Hemingovog (eng. *Hamming*) koda.

Pomoću Hemingovog koda je moguće ispraviti jednobitne greške prilikom prenosa podataka i detektovati pojavu dvobitne greške. Hemingov (n,k) kod spada u grupu linearnih blok kodova, gde je sa n predstavljen broj elemenata (bita) kodne reči, a sa k broj informacijskih bita. U ovom slučaju reč je o (7,4) Hemingovom kodu, gde se pored 4 informacijska bita generišu i 7-4=3 kontrolna bita. Ako se sa H označi kodovana reč, a sa B informacijski sadržaj, tada je raspored bita u kodovanoj reči sledeći:

| $H_7$ | $H_6$ | $H_5$ | $H_4$ | $H_3$ | $H_2$ | $H_1$ |
|---|---|---|---|---|---|---|
| $B_3$ | $B_2$ | $B_1$ | | $B_0$ | | |

Kontrolni biti na pozicijama $H_1$, $H_2$ i $H_4$ se generišu prema sledećim jednačinama:

$$H_4 = H_5 \oplus H_6 \oplus H_7 = B_1 \oplus B_2 \oplus B_3$$

$$H_2 = H_3 \oplus H_6 \oplus H_7 = B_0 \oplus B_2 \oplus B_3$$

$$H_1 = H_3 \oplus H_5 \oplus H_7 = B_0 \oplus B_1 \oplus B_3$$

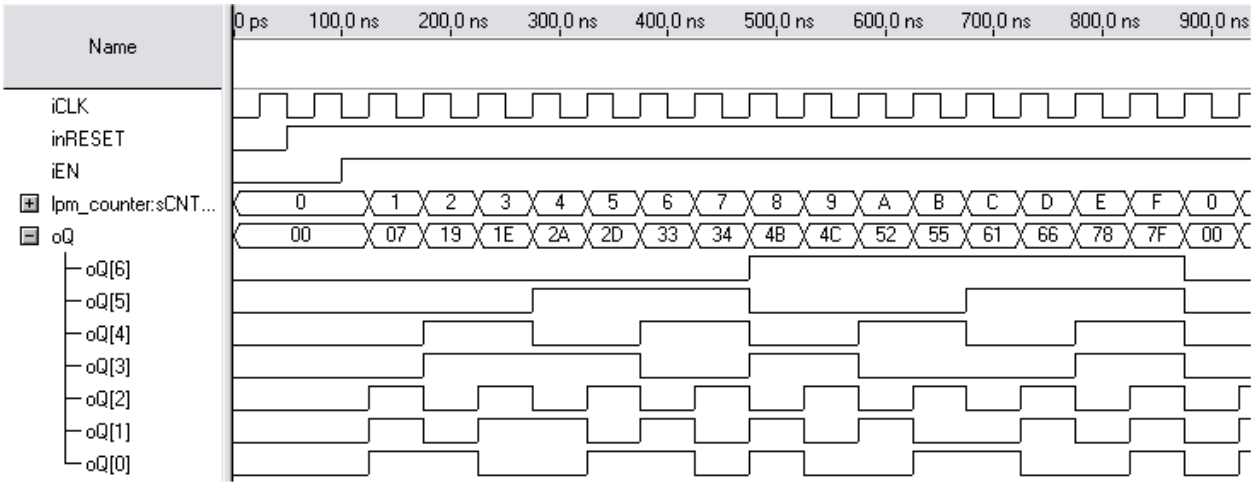
**REŠENJE:**

Slično kao i u prethodnom zadatku i u ovom slučaju je reč o sistemu sa jednim sekvencijalnim delom (brojač) i sa odgovarajućom kombinacionom mrežom. Brojač je isti kao i u prethodnom zadatku, dok kombinaciona mreža sada služi za generisanje Hemingovih kontrolnih bita na osnovu stanja brojača. Izlaz iz digitalnog sistema je sedmobitna kodna reč generisana na osnovu Hemingovog koda. Pošto je potrebna kombinaciona mreža precizno definisana odgovarajućim jednačinama, za realizaciju iste nije potrebno uvoditi zaseban proces, već je dovoljno napisati odgovarajuće konkurentne iskaze koji definišu koder Hemingovih bita.

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

```
ENTITY ZAD02 IS PORT (
  iCLK    : IN  STD_LOGIC;
  inRESET,
  iEN     : IN  STD_LOGIC;
  oQ      : OUT STD_LOGIC_VECTOR(6 DOWNTO 0) );
END ZAD02;

ARCHITECTURE ARH_ZAD02 OF ZAD02 IS
  -- stanje brojaca
  SIGNAL sCNT: UNSIGNED(3 DOWNTO 0);
  -- hemingovi biti
  SIGNAL sH0, sH1, sH2: STD_LOGIC;
BEGIN

  -- brojac sa sinhronim resetom
  PROCESS(iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      IF  (inRESET = '0') THEN -- sinhroni reset
        sCNT <= "0000";
      ELSE
        IF (iEN = '1') THEN     -- dozvoljeno brojanje?
          sCNT <= sCNT + 1;     -- brojanje
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- koder hemingovog koda
  sH0 <= sCNT(0) XOR sCNT(1) XOR sCNT(3);
  sH1 <= sCNT(0) XOR sCNT(2) XOR sCNT(3);
  sH2 <= sCNT(1) XOR sCNT(2) XOR sCNT(3);

  -- formiranje izlaznog vektora
  oQ <= (sCNT(3) & sCNT(2) & sCNT(1) & sH2 & sCNT(0) & sH1 & sH0);

END ARH_ZAD02;
```

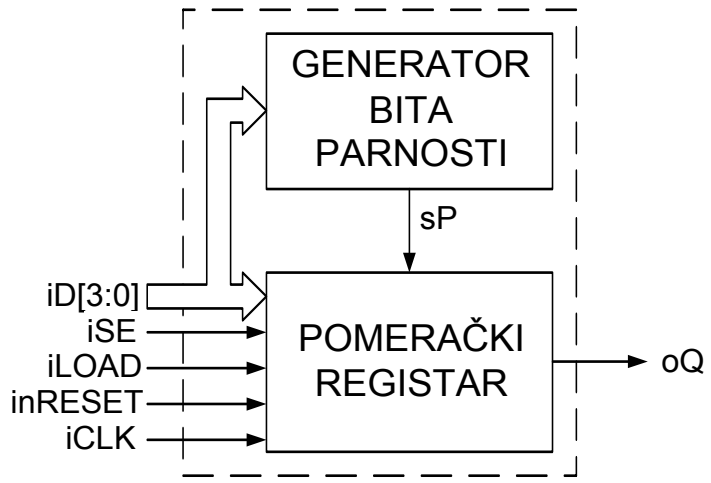
Vremenski dijagram simulacije rada realizovanog digitalnog sistema prikazuje Slika 6.6.




Slika 6.6: Simulacija rada realizovanog sistema

## 6.3 ZADATAK:

Pomoću VHDL jezika za opis fizičke arhitekture isprojektovati digitalni sistem koji prikazuje Slika 6.7.




Slika 6.7: Blok dijagram digitalnog sistema

Sistem treba da obezbedi sledeće funkcije:

- generisanje bita parne parnosti, signal `sP`, za četvorobitni ulazni vektor podataka `iD[3:0]`,
- paralelni upis podataka u petobitni registar, signalom aktivnim na visokom nivou (`iLOAD`), tako da generisani bit parnosti bude smešten na poziciji bita najveće važnosti (MSB),
- serijski izlaz registra gde bit sa pozicije sa najmanjom važnošću (LSB) izlazi prvi,
- kontrolu pomeranja pomoću signala koji je aktivan na visokom naponskom nivou (`iSE`),
- sinhroni reset registra signalom koji je aktivan na niskom naponskom nivou (`inRESET`),
- stanje registra, prilikom upisa i pomeranja, se menja na rastuću ivicu takt signala `iCLK`.

### REŠENJE:

Traženi sistem se sastoji od petobitnog registra sa paralelnim ulazom i serijskim izlazom, i od kombinacione mreže za generisanje bita parne parnosti.

Registar poseduje sinhroni reset signal aktivan na niskom nivou. Paralelni upis podataka u registar je takođe sinhron sa takt signalom. Bit najmanje važnosti predstavlja serijski izlaz registra, koji se menja na svaku rastuću ivicu takt signala pod uslovom da je aktivan signal dozvole pomeranja. U suprotnom slučaju stanje registra se ne menja. Ovakav registar se u VHDL-u opisuje sa jednim procesom gde u listi osetljivosti postoji samo identifikator takt signala, pošto su sve promene stanja registra sinhrone.

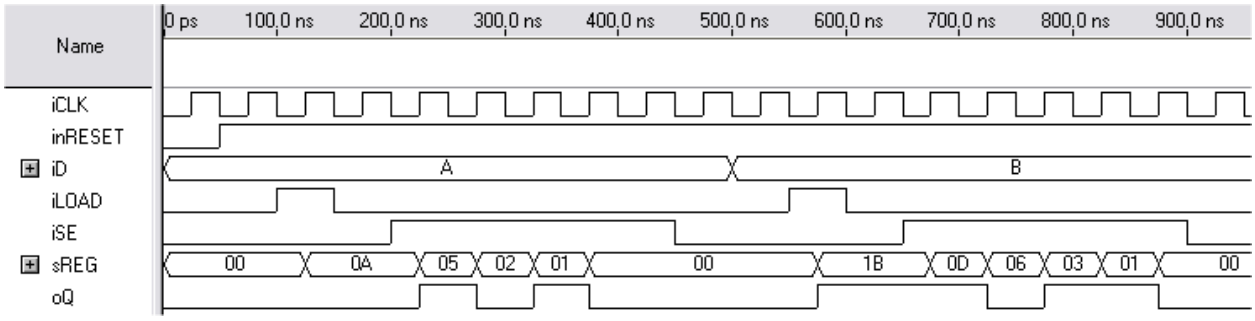
Bit parne parnosti se generiše pomoću EXILI kola, gde se na ulaze dovode svi biti za koje se generiše bit parnosti koji se pojavljuje na izlazu EXILI kola. U ovom slučaju bit parnosti se definiše prema sledećoj jednačini:

$$P = D_3 \oplus D_2 \oplus D_1 \oplus D_0$$

Na osnovu ovih informacija se kreće sa sintezom odgovarajućeg VHDL koda, koji je prikazan u nastavku.

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

ENTITY ZAD03 IS PORT (
  iCLK    : IN  STD_LOGIC;
  inRESET,
  iSE,
  iLOAD   : IN  STD_LOGIC;
  iD      : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
  oQ      : OUT STD_LOGIC );
END ZAD03;

ARCHITECTURE ARH_ZAD03 OF ZAD03 IS
  -- stanje registra
  SIGNAL sREG: STD_LOGIC_VECTOR(4 DOWNTO 0);
  -- bit parnosti
  SIGNAL sP: STD_LOGIC;
BEGIN

  -- registar sa sinhronim resetom
  PROCESS(iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      IF  (inRESET = '0') THEN -- sinhroni reset
        sREG <= "00000";
      ELSE
        IF (iLOAD = '1') THEN  -- paralelni upis?
          sREG <= (sP & iD);
        ELSE
          IF (iSE = '1') THEN  -- dozvoljeno pomeranje?
            sREG <= ('0' & sREG(4) & sREG(3) & sREG(2) & sREG(1));
          ELSE
            sREG <= sREG;
          END IF;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- generator bita parnosti
  sP <= iD(0) XOR iD(1) XOR iD(2) XOR iD(3);

  -- formiranje izlaznog signala
  oQ <= sREG(0); -- LSB bit izlazi prvi

END ARH_ZAD03;
```

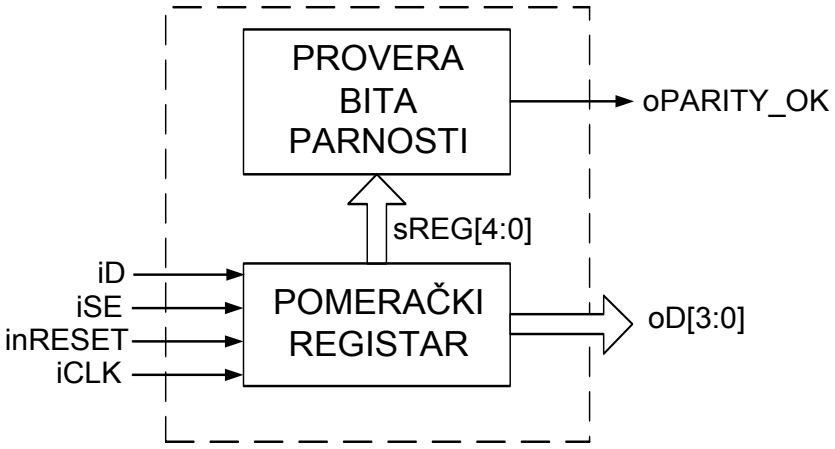
Simulaciju rada realizovanog sistema prikazuje Slika 6.8. Na slici su prikazana dva slučaja. Prvo je prikazana situacija kada ulazna reč, `iD=`$A_{HEX}$, sadrži paran broj jedinica i generiše se bit parnosti koji je jednak nuli. Bit parnosti se vidi kao MSB bit signala koji prikazuje stanje registra, `sREG=0`$A_{HEX}$. Druga situacija prikazuje slučaj kada ulazna reč, `iD=`$B_{HEX}$, sadrži neparan broj jedinica i generiše se bit parnosti koji je jednak jedinici, `sREG=1`$B_{HEX}$.




Slika 6.8: Simulacija rada realizovanog sistema

## 6.4 ZADATAK:

Pomoću VHDL jezika za opis fizičke arhitekture isprojektovati digitalni sistem koji prikazuje Slika 6.9.




Slika 6.9: Blok dijagram digitalnog sistema

Sistem treba da obezbedi sledeće funkcije:

- serijski ulaz podataka u petobitni registar, gde prvi ulazi bit sa pozicije sa najmanjom važnošću (LSB),
- kontrolu pomeranja pomoću signala koji je aktivan na viskom naponskom nivou (`iSE`),
- paralelni izlaz iz registra, primljeni bit parne parnosti je smešten na poziciji bita najveće važnosti (MSB),
- sinhroni reset regista signalom koji je aktivan na niskom naponskom nivou (`inRESET`),

- stanje registra se menja na rastuću ivicu takt signala `iCLK`,
- proveru primljenog bita parne parnosti, i indikaciju da su podaci primljeni bez grešeke parnosti (`oPARITY_OK=1`) ili sa greškom parnosti (`oPARITY_OK=0`).

**REŠENJE:**

Traženi sistem se sastoji od pomeračkog registra sa serijskim prijemom i paralelnim izlazom, i od kombinacione mreže za proveru parnosti primljene reči.

Na serijskom ulazu registra prvo dolazi bit sa najmanjom težinom (LSB). Zbog toga je potrebno realizovati prijem podataka sa bita sa najvećom težinom (MSB):

```
sREG <= (iD & sREG(4) & sREG(3) & sREG(2) & sREG(1));
```

Na ovaj način će se primljeni LSB bit postepeno tokom prijema reči pomerati prema svojoj poziciji tako da će se na kraju ciklusa prijema podataka, koji traje 5 perioda takt signala, naći na poziciji sa najmanjom težinom.

Kombinaciona mreža za proveru parnosti se sastoji iz dva dela. Prvi deo je generisanje bita parne parnosti za primljenu reč (biti sa indeksima 0 do 3), dok drugi deo realizuje poređenje generisanog bita parnosti i primljenog bita parnosti koji se nalazi na poziciji primljenog bita sa indeksom 4. U slučaju da su biti jednaki generiše se signal da nema greške parnosti u primljenoj reči (`oPARITY_OK=1`). Ako se sa P označi generisani bit parnosti, a sa $D_4$ primljeni bit parnosti tada Tabela 6.1 prikazuje zavisnost izlaznog signala koji ukazuje na grešku parnosti.

| P | $D_4$ | PARITY_OK |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Tabela 6.1: Indikacija greške parnosti

Iz tabele se dobija prenosna funkcija $\text{PARITY\_OK} = \overline{\text{P} \oplus \text{D}_4}$ koja opisuje izlazni signal indikacije greške parnosti u primljenoj reči.

Sledi odgovarajući VHDL kod.

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

ENTITY ZAD04 IS PORT (
  iCLK       : IN  STD_LOGIC;
  inRESET,
  iSE, iD    : IN  STD_LOGIC;
```

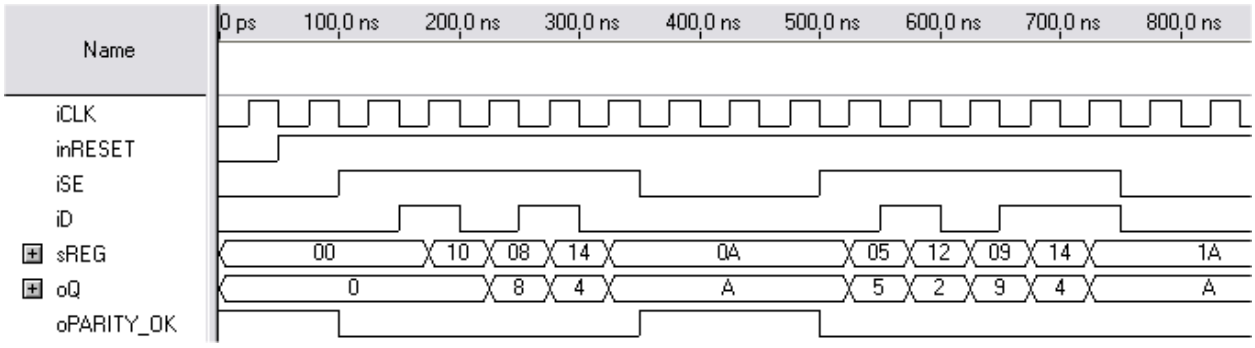
```
    oQ         : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    oPARITY_OK : OUT STD_LOGIC );
  END ZAD04;

  ARCHITECTURE ARH_ZAD04 OF ZAD04 IS
    -- stanje registra
    SIGNAL sREG: STD_LOGIC_VECTOR(4 DOWNTO 0);
    -- bit parnosti
    SIGNAL sP: STD_LOGIC;
  BEGIN
    -- registar sa sinhronim resetom
    PROCESS(iCLK) BEGIN
      IF (iCLK'EVENT AND iCLK = '1') THEN
        IF  (inRESET = '0') THEN -- sinhroni reset
          sREG <= "00000";
        ELSE
          IF (iSE = '1') THEN  -- dozvoljeno pomeranje?
             -- LSB bit ulazi prvi
            sREG <= (iD & sREG(4) & sREG(3) & sREG(2) & sREG(1));
          ELSE
            sREG <= sREG;
          END IF;
        END IF;
      END IF;
    END PROCESS;

    -- generator bita parnosti
    sP <= sREG(0) XOR sREG(1) XOR sREG(2) XOR sREG(3);

    -- formiranje izlaznih signala
               -- stanje registra
    oQ         <= sREG(3 DOWNTO 0);
               -- indikacija greske parnosti
    oPARITY_OK <= (sP XNOR sREG(4)) AND NOT(iSE);
  END ARH_ZAD04;
```

U prikazanom VHDL kodu je izlazni signal indikacije greške parnosti ograničen sa signalom dozvole pomeranja, tako da je signal greške parnosti validan samo ako nije aktivan signal dozvole pomeranja `iSE`. U suprotnom slučaju se generiše signal greške parnosti (`oPARITY_OK=0`). U toku prijema reči bit parnosti, kako generisani tako i primljeni, svakako nije ispravan pa je opravdano generisati signal greške parnosti.

Vremenski dijagram simulacije rada realizovanog digitalnog sistema prikazuje Slika 6.10.

Na početku vremenskog dijagrama (t=100ns÷350ns) prikazan je prijem reči $A_{HEX}$ sa ispravnim bitom parnosti koji je jednak nuli. Na kraju prijema u trenutku t=350ns je primljena reč $A_{HEX}$ i generiše se izlazni signal `oPARITY_OK` koji je jednak jedinici, čime se ukazuje da je primljena reč bez greške parnosti.

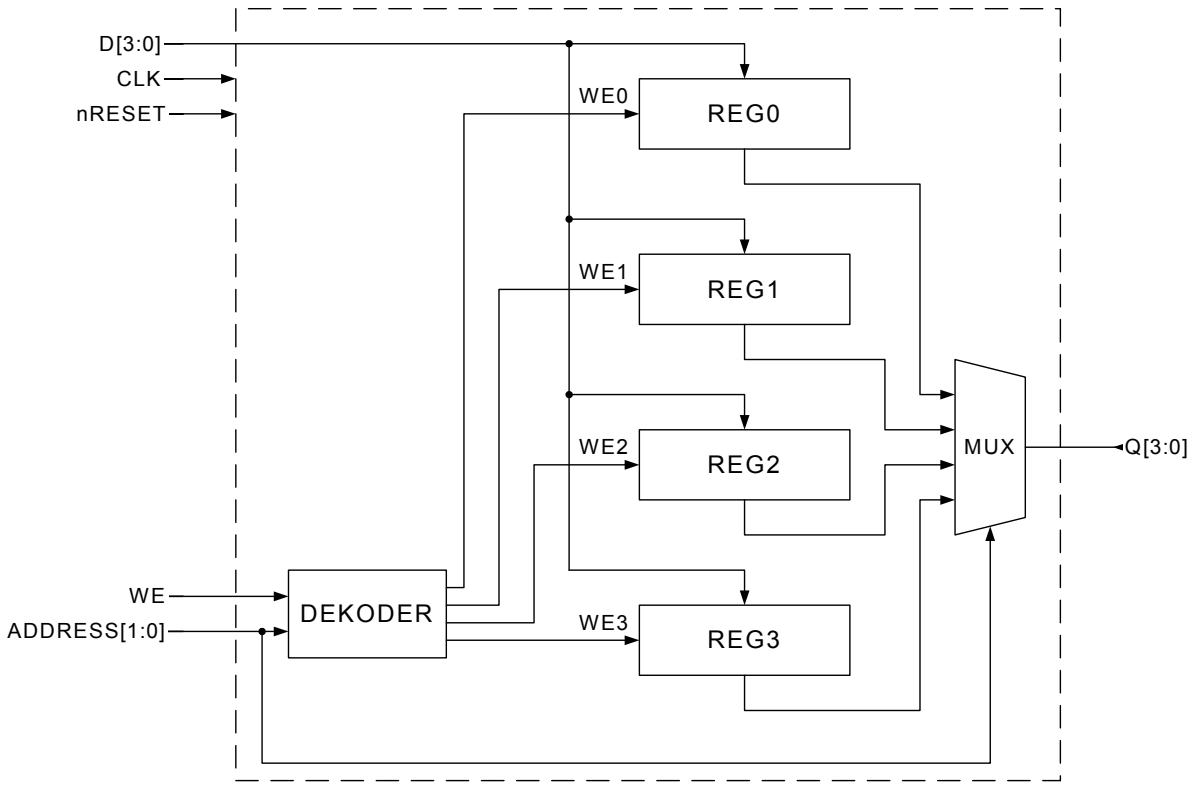
U drugom delu vremenskog dijagrama (t=500ns÷750ns) primljena je reč $A_{HEX}$ i bit parnosti koji je jednak jedinici. Time je u trenutku t=850ns primljen neparan broj jedinica što ukazuje da je primljena reč sa greškom u parnosti. To se vidi i preko stanja izlaznog signala `oPARITY_OK` koji je jednak nuli.




Slika 6.10: Simulacija rada realizovanog sistema

## 6.5 ZADATAK:

Pomoću VHDL jezika za opis fizičke arhitekture isprojektovati digitalni sistem koji prikazuje Slika 6.11.




Slika 6.11: Blok dijagram digitalnog sistema

Traženi sistem se sastoji od 4 četvorobitna registra `REG0÷REG3`. Sadržaj svih registara se briše sinhronim signalom aktivnim na niskom naponskom nivou `nRESET`. Svakom registru se pristupa na osnovu adrese `ADDRESS[1:0]` (Tabela 6.2). Podaci prisutni na ulazu `D[3:0]` se upisuju u adresirani registar ako je aktivan signal dozvole upisa `WE`. Sadržaj adresiranog registra je prisutan na izlazu `Q[3:0]` preko vektorskog multipleksera 4×1.

| Adresa | Registar |
|---|---|
| 00 | REG0 |
| 01 | REG1 |
| 10 | REG2 |
| 11 | REG3 |

Tabela 6.2: Adresiranje registara

**REŠENJE:**

Traženi sistem se može podeliti na tri dela:

- registri,
- adresni dekoder i
- izlazni multiplekser.

Iz ovog sledi da se sistem može realizovati sa tri procesa. Međutim, pošto svaki registar ima poseban signal dozvole upisa, koji zavisi od prisutne adrese, pogodnije ih je realizovati preko nezavisnih procesa. Tako na primer, navodi se VHDL kod za opis registra na adresi 0.

```
PROCESS(iCLK) BEGIN
  IF (iCLK'EVENT AND iCLK = '1') THEN
    IF  (inRESET = '0') THEN -- sinhroni reset
      sR0 <= "0000";
    ELSE
      IF (sWE(0) = '1') THEN -- dozvoljen upis?
        sR0 <= iD;
      END IF;
    END IF;
  END IF;
END PROCESS;
```

Ostali registri se realizuju na ekvivalentan način.

Signale za dozvolu upisa u adresirani registar generiše adresni dekoder. Za četiri registra je potrebno četiri signala koje je zgodno objediniti u jedinstven vektor zbog jednostavnijeg rukovanja u samom VHDL kodu. U ovom slučaju za to je namenjen vektor `sWE`, pa adresni dekoder ima sledeći izgled:

```
PROCESS(iADDRESS, iWE) BEGIN
  IF (iWE = '1') THEN
    CASE iADDRESS IS
      WHEN "00"   => sWE <= "0001"; -- upis u R0
      WHEN "01"   => sWE <= "0010"; -- upis u R1
      WHEN "10"   => sWE <= "0100"; -- upis u R2
      WHEN "11"   => sWE <= "1000"; -- upis u R3
      WHEN OTHERS => sWE <= "0000"; -- nema upisa
    END CASE;
  ELSE
    sWE <= "0000";                  -- nema upisa
  END IF;
END PROCESS;
```

Izlazi registara se dovode na ulaze vektorskog multipleksera, dok se na osnovu adrese (koja se dovodi na adresni ulaz multipleksera) samo jedan prosleđuje na izlaz.

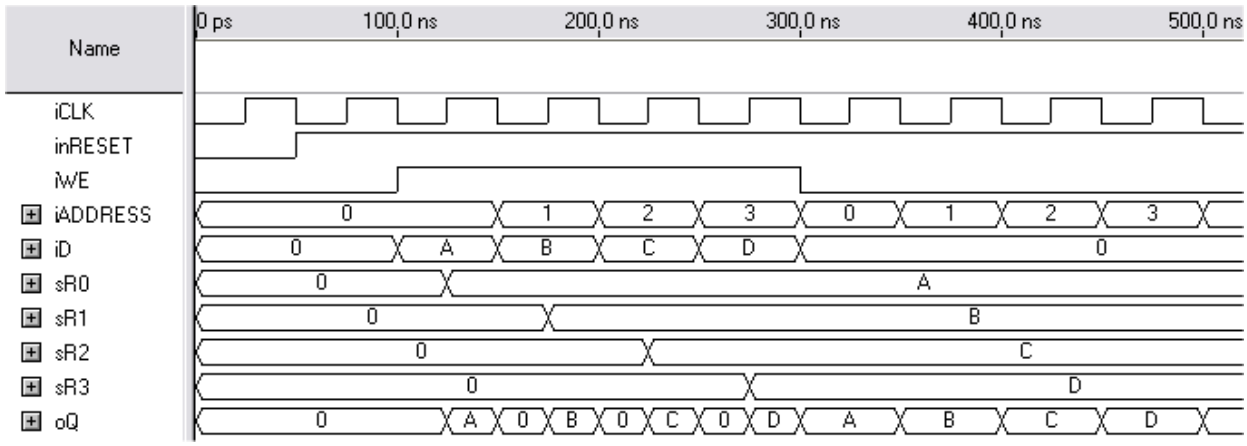
Objedinjavanjem svih potrebnih komponenti, dolazi se do sledećeg VHDL koda:

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

ENTITY ZAD05 IS PORT (
  iCLK    : IN  STD_LOGIC;
  inRESET,
  iWE     : IN  STD_LOGIC;
  iADDRESS: IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
  iD      : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
  oQ      : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
END ZAD05;

ARCHITECTURE ARH_ZAD05 OF ZAD05 IS
  -- stanje registara
  SIGNAL sR0, sR1, sR2, sR3: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- dozvola upisa u registre
  SIGNAL sWE: STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
  -- registar na adresi 0 sa sinhronim resetom
  PROCESS(iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      IF  (inRESET = '0') THEN -- sinhroni reset
        sR0 <= "0000";
      ELSE
        IF (sWE(0) = '1') THEN -- dozvoljen upis?
          sR0 <= iD;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- registar na adresi 1 sa sinhronim resetom
  PROCESS(iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      IF  (inRESET = '0') THEN -- sinhroni reset
        sR1 <= "0000";
      ELSE
        IF (sWE(1) = '1') THEN -- dozvoljen upis?
          sR1 <= iD;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- registar na adresi 2 sa sinhronim resetom
  PROCESS(iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      IF  (inRESET = '0') THEN -- sinhroni reset
        sR2 <= "0000";
      ELSE
```

```
        IF (sWE(2) = '1') THEN -- dozvoljen upis?
          sR2 <= iD;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- registar na adresi 3 sa sinhronim resetom
  PROCESS(iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      IF  (inRESET = '0') THEN -- sinhroni reset
        sR3 <= "0000";
      ELSE
        IF (sWE(3) = '1') THEN -- dozvoljen upis?
          sR3 <= iD;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- adresni dekoder
  PROCESS(iADDRESS, iWE) BEGIN
    IF (iWE = '1') THEN
      CASE iADDRESS IS
        WHEN "00"   => sWE <= "0001"; -- upis u R0
        WHEN "01"   => sWE <= "0010"; -- upis u R1
        WHEN "10"   => sWE <= "0100"; -- upis u R2
        WHEN "11"   => sWE <= "1000"; -- upis u R3
        WHEN OTHERS => sWE <= "0000"; -- nema upisa
      END CASE;
    ELSE
      sWE <= "0000";                  -- nema upisa
    END IF;
  END PROCESS;

  -- izlazni multiplekser
  PROCESS(iADDRESS, sR0, sR1, sR2, sR3) BEGIN
    CASE iADDRESS IS
      WHEN "00"   => oQ <= sR0;
      WHEN "01"   => oQ <= sR1;
      WHEN "10"   => oQ <= sR2;
      WHEN OTHERS => oQ <= sR3;
    END CASE;
  END PROCESS;

END ARH_ZAD05;
```

Slika 6.12 prikazuje vremenski dijagram simulacije upisa u sve registre digitalnog sistema kao i čitanja upisanog sadržaja iz svih registara.

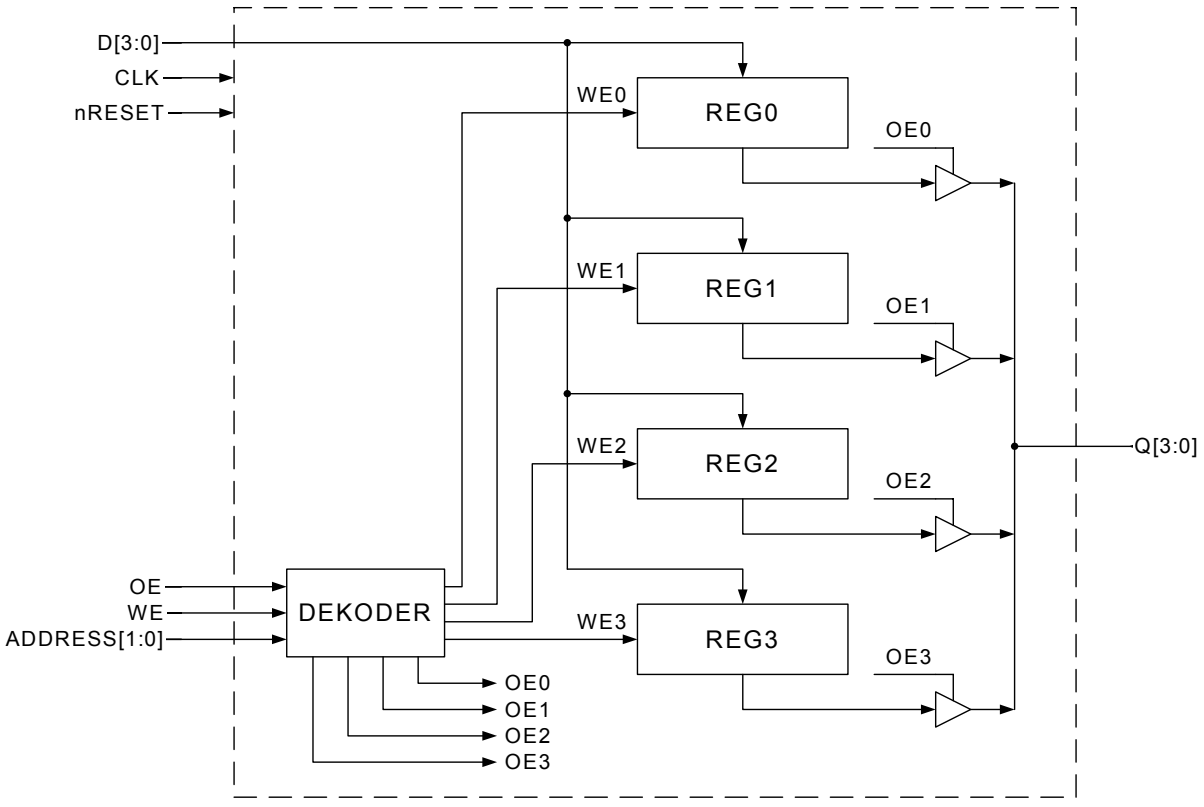
Na početku vremenskog dijagrama, u vremenskom intervalu t=100ns do t=300ns, se u registre na adresama nula, jedan dva i tri upisuju vrednosti $A_{HEX}$, $B_{HEX}$, $C_{HEX}$ i $D_{HEX}$ respektivno. Upis se vrši na rastuću ivicu takt signala `iCLK`. U nastavku vremenskog dijagrama (t=300ns do t=500ns) se na izlaznoj magistrali `oQ` redom čitaju upisane vrednosti u sve registre.




Slika 6.12: Simulacija rada realizovanog sistema

## 6.6 ZADATAK:

Pomoću VHDL jezika za opis fizičke arhitekture isprojektovati digitalni sistem koji prikazuje Slika 6.13.




Slika 6.13: Blok dijagram digitalnog sistema

Traženi sistem se sastoji od 4 četvorobitna registra `REG0÷REG3`. Sadržaj svih registara se briše sinhronim signalom aktivnim na niskom naponskom nivou `nRESET`. Svakom registru se pristupa na osnovu adrese `ADDRESS[1:0]` (Tabela 6.3). Podaci prisutni na ulazu `D[3:0]` se upisuju u adresirani registar ako je aktivan signal dozvole upisa `WE`. Svaki registar ima na svom izlazu bafer sa tri stanja. Upravljački signal za otvaranje bafera je aktivan na viskom nivou. Svi izlazi

bafera sa tri stanja su povezani u jedinstvenu izlaznu magistralu `Q[3:0]`. Sadržaj adresiranog registra je prisutan na izlazu, preko bafera sa tri stanja, ako je aktivan ulazni signal `OE`. U suprotnom, na izlazu `Q[3:0]`je prisutno stanje visoke impedanse (HIGH-Z)

| Adresa | Registar |
|---|---|
| 00 | REG0 |
| 01 | REG1 |
| 10 | REG2 |
| 11 | REG3 |

Tabela 6.3: Adresiranje registara

**REŠENJE:**

Traženi digitalni sistem je veoma sličan sistemu iz prethodnog zadatka. Razlika je u tome što u ovom slučaju svi registri na izlazima imaju bafere sa tri stanja. Ovakvom realizacijom se eleminiše upotreba izlaznog vektorskog multipleksera, što dovodi do višestrukog smanjenja potrebnih logičkih kola za realizaciju digitalnog sistema.

Za upravljanje sa baferima je potrebno uvesti četiri signala koji regulišu otvaranje/zatvaranje bafera sa tri stanja. Slično kao kod generisanja upravljačkih signala za upis u registre, i signali za otvaranje bafera sa tri stanja su objedinjeni u jedan četvorobitni vektor (`sOE`). Potrebno je obezbediti da u jednom trenutku samo jedan bafer propušta signale na izlaz, jer bi u suprotnom slučaju došlo do sukoba vrednosti signala iz više izvora i vrednost izlaznog vektora ne bi bila ispravno definisana.

Bafer sa tri stanja je moguće realizovati uslovnim `IF` iskazom koji prema VHDL standardu mora biti u okviru VHDL procesa. Radi pojednostavljenja analize VHDL koda u ovom slučaju su baferi sa tri stanja realizovani `WHEN-ELSE` iskazom na sledeći način:

```
oQ <= (OTHERS => 'Z') WHEN (sOE(0) = '0') ELSE sR0;
oQ <= (OTHERS => 'Z') WHEN (sOE(1) = '0') ELSE sR1;
oQ <= (OTHERS => 'Z') WHEN (sOE(2) = '0') ELSE sR2;
oQ <= (OTHERS => 'Z') WHEN (sOE(3) = '0') ELSE sR3;
```

Na ovaj način je očiglednije da su izlazi svih bafera sa tri stanja spojeni na jedinstveni izlazni vektor `oQ`.

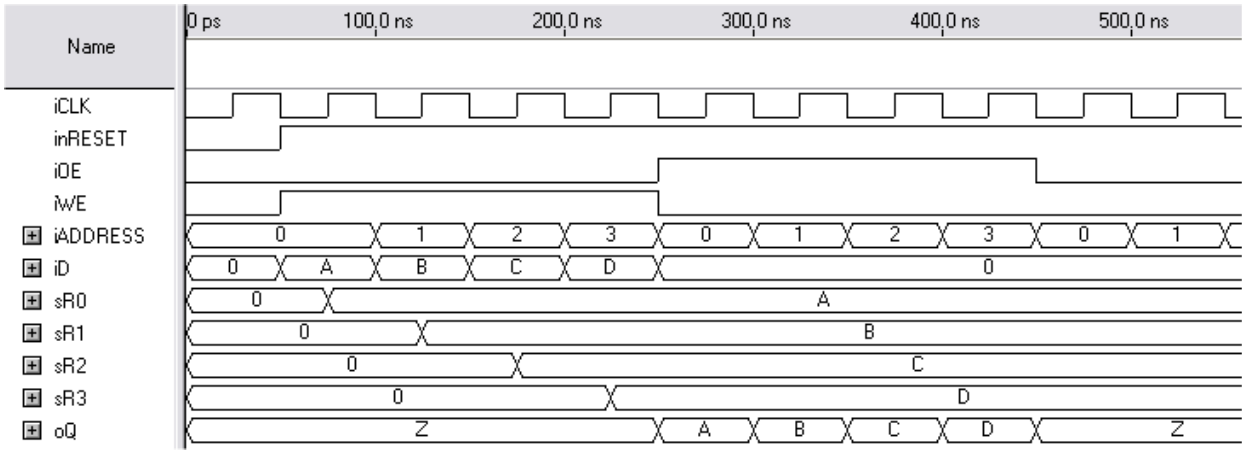
Potrebno je naglasiti da je samo izlaze bafera sa tri stanja moguće spojiti jedinstvenim signalom. U suprotnom neminovno dolazi do sukoba signala iz više izvora, pa svaki VHDL prevodilac tokom prevođenja detektuje takve situacije i javlja odgovarajuću grešku.

Sledi kompletan VHDL kod traženog digitalnog sistema:

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

ENTITY ZAD06 IS PORT (
  iCLK    : IN  STD_LOGIC;
  inRESET,
  iWE, iOE: IN  STD_LOGIC;
  iADDRESS: IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
  iD      : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
  oQ      : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
END ZAD06;

ARCHITECTURE ARH_ZAD06 OF ZAD06 IS
  -- stanje registara
  SIGNAL sR0, sR1, sR2, sR3: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- dozvola upisa u registre
  SIGNAL sWE: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- dozvola citanja iz registra
  SIGNAL sOE: STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN

  -- registar na adresi 0 sa sinhronim resetom
  PROCESS(iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      IF  (inRESET = '0') THEN -- sinhroni reset
        sR0 <= "0000";
      ELSE
        IF (sWE(0) = '1') THEN -- dozvoljen upis?
          sR0 <= iD;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- registar na adresi 1 sa sinhronim resetom
  PROCESS(iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      IF  (inRESET = '0') THEN -- sinhroni reset
        sR1 <= "0000";
      ELSE
        IF (sWE(1) = '1') THEN -- dozvoljen upis?
          sR1 <= iD;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- registar na adresi 2 sa sinhronim resetom
  PROCESS(iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      IF  (inRESET = '0') THEN -- sinhroni reset
        sR2 <= "0000";
      ELSE
        IF (sWE(2) = '1') THEN -- dozvoljen upis?
          sR2 <= iD;
        END IF;
      END IF;
```

```
    END IF;
  END PROCESS;

  -- registar na adresi 3 sa sinhronim resetom
  PROCESS(iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      IF  (inRESET = '0') THEN -- sinhroni reset
        sR3 <= "0000";
      ELSE
        IF (sWE(3) = '1') THEN -- dozvoljen upis?
          sR3 <= iD;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- adresni dekoder za upis u registar
  PROCESS(iADDRESS, iWE) BEGIN
    IF (iWE = '1') THEN
      CASE iADDRESS IS
        WHEN "00"   => sWE <= "0001"; -- upis u R0
        WHEN "01"   => sWE <= "0010"; -- upis u R1
        WHEN "10"   => sWE <= "0100"; -- upis u R2
        WHEN "11"   => sWE <= "1000"; -- upis u R3
        WHEN OTHERS => sWE <= "0000"; -- nema upisa
      END CASE;
    ELSE
      sWE <= "0000";                  -- nema upisa
    END IF;
  END PROCESS;

  -- adresni dekoder za citanje iz registra
  PROCESS(iADDRESS, iOE) BEGIN
    IF (iOE = '1') THEN
      CASE iADDRESS IS
        WHEN "00"   => sOE <= "0001"; -- citanje iz R0
        WHEN "01"   => sOE <= "0010"; -- citanje iz R1
        WHEN "10"   => sOE <= "0100"; -- citanje iz R2
        WHEN "11"   => sOE <= "1000"; -- citanje iz R3
        WHEN OTHERS => sOE <= "0000"; -- citanje nije dozvoljeno
      END CASE;
    ELSE
      sOE <= "0000";                  -- citanje nije dozvoljeno
    END IF;
  END PROCESS;

  -- formiranje izlaznog vektora
  oQ <= (OTHERS => 'Z') WHEN (sOE(0) = '0') ELSE sR0;
  oQ <= (OTHERS => 'Z') WHEN (sOE(1) = '0') ELSE sR1;
  oQ <= (OTHERS => 'Z') WHEN (sOE(2) = '0') ELSE sR2;
  oQ <= (OTHERS => 'Z') WHEN (sOE(3) = '0') ELSE sR3;

END ARH_ZAD06;
```

Slika 6.14 prikazuje vremenski dijagram simulacije upisa u sve registre digitalnog sistema kao i čitanja upisanog sadržaja iz svih registara.

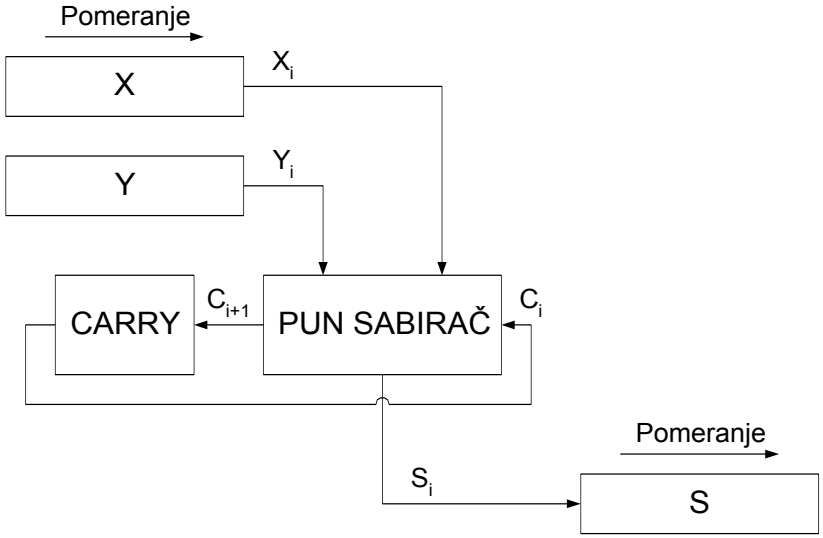



Slika 6.14: Simulacija rada realizovanog sistema

Na početku vremenskog dijagrama, u vremenskom intervalu t=50ns do t=250ns, se u registre na adresama nula, jedan, dva i tri upisuju vrednosti $A_{HEX}$, $B_{HEX}$, $C_{HEX}$ i $D_{HEX}$ respektivno. U tom intervalu je aktivan signal dozvole upisa `iWE`, pa se upis vrši na rastuću ivicu takt signala `iCLK`. Tokom upisa se na izlaznoj magistrali nalazi stanje visoke impedanse pošto nije aktivan signal dozvole čitanja sadržaja registara, tj. `iOE=0`. U nastavku vremenskog dijagrama (t=250ns do t=450ns) se na izlaznoj magistrali `oQ` redom čitaju upisane vrednosti u sve registre. Čitanje sadržaja adresiranog registra je omogućeno aktiviranjem signala dozvole čitanja `iOE=1`. Nakon vremenskog trenutka t=450ns, kada nisu aktivni signali dozvole upisa i čitanja, na izlaznoj magistrali je prisutno stanje visoke impedanse.

## 6.7 ZADATAK:

Isprojektovati serijski sabirač četvorobitnih brojeva X i Y. Na izlazu sabirača obezbediti zbir S i izlazni prenos C. Blok šemu serijskog sabirača prikazuje Slika 6.15.




Slika 6.15: Blok šema serijskog sabirača

Isprojektovani digitalni sistem pored ulaznog takt signala CLK treba da ima i signal za inicijalizaciju sadržaja pomeračkih registara (INIT) koji je aktivan na visokom logičkom nivou. Takođe, treba da postoji i ulazni signal dozvole sabiranja ADD_EN koji je aktivan tokom celog ciklusa sabiranja.

Serijski sabirač isprojektovati u VHDL jeziku za opis fizičke arhitekture.

**REŠENJE:**

Ulazne parametre X i Y treba prilikom inicijalizacije sistema upisati u odgovarajuće pomeračke registre. Pomerački registri pomeraju svoj sadržaj u desno prema bitu najmanje važnosti, ako je aktivan signal dozvole sabiranja. Pun sabirač sabira sadržaj bita najmanje važnosti registara X i Y. Ulazni prenos je sadržaj ćelije za kašnjenje izlaznog prenosa punog sabirača. Zbog toga, prilikom inicijalizacije treba obrisati sadržaj ćelije za kašnjenje izlaznog prenosa. Rezultat punog sabirača se smešta na poziciju bita najveće važnost pomeračkog registra za smeštanje sume ulaznih parametara. Ovaj registar je takođe pomerački sa pomeranjem u desno. Na taj način će nakon četiri periode takt signala, rezultat biti pravilno upisan u registru.

U realizovanom VHDL kodu se prilikom inicijalizacije sistema ulazni parametri `iX` i `iY` upisuju u pomeračke registre `srX` i `srY` aktiviranjem ulaznog signala `iINIT`. Ciklus sabiranja treba da traje 4 periode takt signala `iCLK`. Ovaj ciklus se ograničava ulaznim signalom dozvole sabiranja `iADD_EN`.

Pun sabirač se realizuje direktno na osnovu poznatih prenosnih funkcija punog sabirača:

$$S_o = X_i \oplus Y_i \oplus C_i$$

$$C_o = X_i \cdot Y_i + X_i \cdot C_i + Y_i \cdot C_i$$

U VHDL kodu su ulazi punog sabirača realizovani signalima `sX`, `sY` i `sC_IN`, gde su `sX` i `sY` biti najmanje važnosti odgovarajućih registara, a ulazni prenos `sC_IN` predstavlja sadržaj ćelije za kašnjenje izlaznog prenosa punog sabirača. Izlazi punog sabirača su označeni signalima `sS` (suma) i `sC_OUT` (izlazni prenos). Izlazni prenos se smešta u ćeliju za kašnjenje izlaznog prenosa čime se obezbeđuje ispravan ulazni prenos punog sabirača u narednoj iteraciji. Rezultat sabiranja se smešta u pomerački registar `srS`. Prilikom inicijalizacije sistema briše se sadržaj ćelije za kašnjenje izlaznog prenosa kao i sadržaj registra za smeštanje rezultata.

Pošto se prilikom ciklusa sabiranja menja sadržaj registra za smeštanje rezultata i izlaznog prenosa, izlaz sistema se ograničava sa signalom `iADD_EN`. Ako je signal aktivan, tj. ako je u toku ciklus sabiranja, izlazi `oS` (suma) i `oC` (izlazni prenos) se postavljaju na nulu, dok u suprotnom slučaju (`iADD_EN=0`) oni primaju ispravan sadržaj odgovarajućih registara.

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

ENTITY SER_SABIRAC IS PORT (
  iCLK:    IN  STD_LOGIC;
  iINIT:   IN  STD_LOGIC;
  iADD_EN: IN  STD_LOGIC;
  iX, iY:  IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
  oS:      OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
  oC:      OUT STD_LOGIC );
END SER_SABIRAC;

ARCHITECTURE ARH_SER_SABIRAC OF SER_SABIRAC IS
  -- registri za smestanje parametara X i Y i sume S
  SIGNAL srX, srY, srS: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- biti koji se sabiraju
  SIGNAL sX, sY: STD_LOGIC;
  -- ulazni i izlzni prenos punog sabiraca
  SIGNAL sC_IN, sC_OUT: STD_LOGIC;
  -- rezultat sabiraca
  SIGNAL sS: STD_LOGIC;
BEGIN

  -- registar za smestanje parametra X
  PROCESS (iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK='1') THEN
      IF (iINIT='1') THEN
        srX <= iX; -- ucitaj parametar X
      ELSE
        IF (iADD_EN='1') THEN
          srX <= ('0' & srX(3 DOWNTO 1));
        END IF;
      END IF;
    END IF;
  END PROCESS;
  sX <= srX(0);  -- sabira se LSB bit

  -- registar za smestanje parametra Y
  PROCESS (iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK='1') THEN
      IF (iINIT='1') THEN
        srY <= iY; -- ucitaj parametar Y
      ELSE
        IF (iADD_EN='1') THEN
          srY <= ('0' & srY(3 DOWNTO 1));
        END IF;
      END IF;
    END IF;
  END PROCESS;
  sY <= srY(0);  -- sabira se LSB bit

  -- pun sabirac
  sS     <= (sX XOR sY XOR sC_IN); -- suma
  sC_OUT <= (sX AND sY)    OR      -- izlazni prenos
            (sX AND sC_IN) OR
            (sY AND sC_IN);
```

```
    -- celija za kasnjenje izlaznog prenosa
    PROCESS (iCLK) BEGIN
      IF (iCLK'EVENT AND iCLK='1') THEN
        IF (iINIT='1') THEN
          sC_IN <= '0'; -- inicijalno 0
        ELSE
          IF (iADD_EN='1') THEN
            sC_IN <= sC_OUT;
          END IF;
        END IF;
      END IF;
    END PROCESS;

    -- registar za smestanje sume S
    PROCESS (iCLK) BEGIN
      IF (iCLK'EVENT AND iCLK='1') THEN
        IF (iINIT='1') THEN
          srS <= "0000"; -- obrisi sadrzaj
        ELSE
          IF (iADD_EN='1') THEN
            srS <= (sS & srS(3 DOWNTO 1));
          END IF;
        END IF;
      END IF;
    END PROCESS;

    -- prosledjivanje rezultata na izlazne signale
    PROCESS (srS, sC_IN, iADD_EN) BEGIN
      IF (iADD_EN='1') THEN
        -- u toku je ciklus sabiranja
        -- postavi nulu na izlaz
        oS <= "0000";
        oC <= '0';
      ELSE
        -- sabiranje zavrseno
        -- postavi rezultat na izlaz
        oS <= srS;   -- suma X+Y
        oC <= sC_IN; -- izlazni prenos
      END IF;
    END PROCESS;

  END ARH_SER_SABIRAC;
```

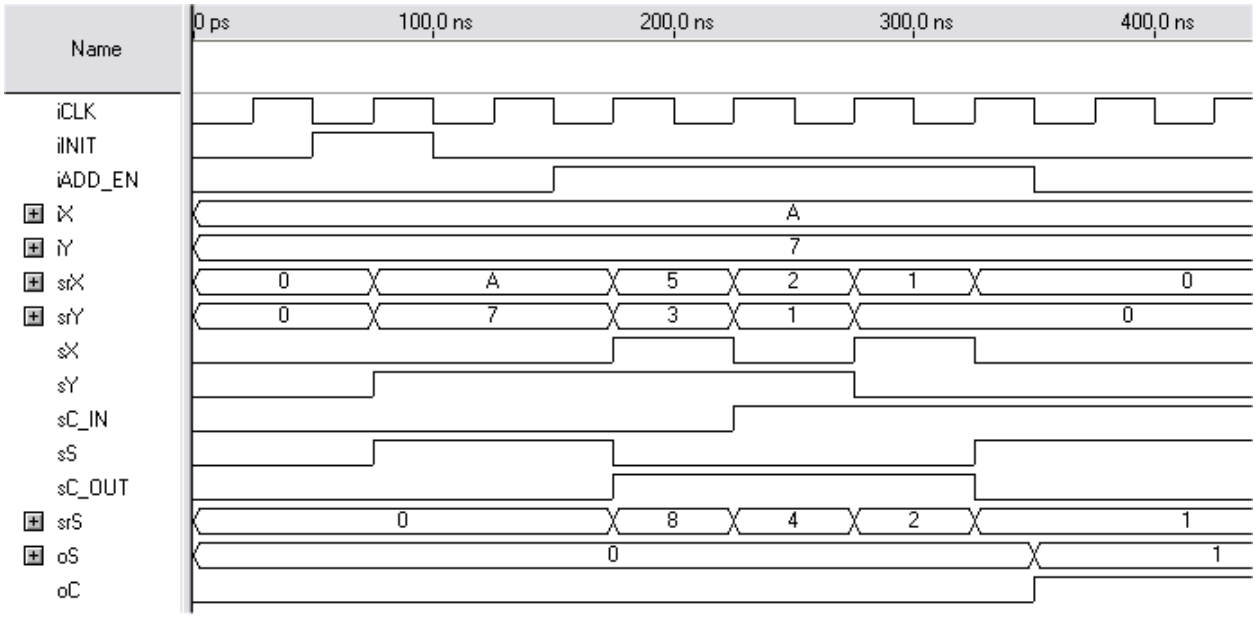
Rad serijskog sabirača se ilustruje primerom sabiranja brojeva $A_{HEX}$ i $7_{HEX}$.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 0 | 1 | 0 | | | A |
| + | 0 | 1 | 1 | 1 | | + | 7 |
| 1 | 0 | 0 | 0 | 1 | | | 11 |

U ovom slučaju na kraju ciklusa sabiranja digitalni sistem za rezultat sabiranja treba da izlazni signal `oS` postavi na vrednost $1_{HEX}$. Vrednost izlaznog prenosa `oC` je 1.

Slika 6.16 ilustruje rad serijskog sabirača na ovom primeru. Sistem se inicijalizuje u vremenskom trenutku t=75ns. Registri `srX` i `srY` primaju vrednosti

ulaznih parametara, dok se sadržaj registra za smeštanje rezultata `srS` i ćelije za kašnjenje izlaznog prenosa `sC_IN` postavlja na vrednost nula. Ciklus sabiranja započinje aktiviranjem signala dozvole sabiranja `iADD_EN`, i traje četiri periode takt signala `iCLK` (t=150ns÷350ns). Tokom ovog ciklusa se sa svakom rastućom ivicom takt signala pomera sadržaj registara `srX`, `srY` i `srS`. Izlaz punog sabirača `sS` se upisuje na poziciju bita najveće važnosti registra `srS`, a izlazni prenos `sC_OUT` u ćeliju za kašnjenje `sC_IN`. Na vremenskom dijagramu se vidi i da su izlazni signali `oS` i `oC` ograničeni sa signalom dozvole sabiranja `iADD_EN`, tj. rezultat sabiranja se na izlaznim signalima pojavljuje tek kad je ciklus sabiranja završen (`iADD_EN=0`).




Slika 6.16: Ilustracija rada serijskog sabirača