

PROGRAMIRANJE KOMUNIKACIONOG HARDVERA
– Poglavlje 4 –

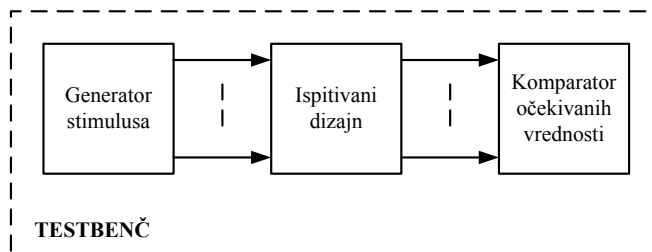
4 Simulacija dizajna

Kada se projektuje dizajn, koji treba da izvršava određenu funkciju ili grupu funkcija, veoma je bitno uveriti se da dizajn radi ispravno. U procesu verifikacije jedan od najvažnijih koraka je simulacija. Kao što smo videli u prethodnom poglavlju, razlikujemo funkcionalnu i vremensku simulaciju, pri čemu je suštinska razlika između njih u tome što vremenska simulacija uzima u obzir realna kašnjenja u dizajnu koja se dobijaju analizom postavljenog i rutiranog dizajna na konkretan izabrani FPGA čip, dok funkcionalna simulacija radi sa idealnim uslovima tj. sva kašnjenja iznose 0, sem ako se ne definišu konkretne vrednosti kašnjenja u okviru VHDL koda. Međutim, treba izbegavati upotrebu definisanja kašnjenja u VHDL kodu iz dva razloga. Prvi razlog je što tako definisana kašnjenja ne mogu da se implementiraju u hardveru, a drugi i važniji razlog je što simulacijom možemo dobiti pogrešan uvid u realan rad dizajna, jer će nakon izvršene implementacije dizajna (kada se iz dizajna uklone VHDL definisana kašnjenja), realne vrednosti kašnjenja biti drugačije.

U okviru ovog poglavlja će biti izloženi osnovni principi pisanja VHDL testbenča koji predstavlja simulaciono okruženje za simuliranje kreiranog dizajna. Takođe će biti dat i kratak pregled rada alata ModelSim koji predstavlja jedan od najpopularnijih alata za simulaciju VHDL pisanog dizajna.

4.1 Testbenč

Da bi se dizajn simulirao, neophodno je stimulisati ulaze dizajna tako da se pokrije što veći broj situacija u kojima se dizajn može naći. U idealnom slučaju treba pokriti sve moguće situacije u kojima se dizajn može naći, ali u slučaju veoma složenog dizajna to je često nemoguće postići. Veoma je važno dobro osmisliti simulaciju da bi se na što efikasniji način dobila potvrda ispravnog funkcionisanja dizajna. Osnovni princip simulacije je prikazan na slici 4.1.1.



Slika 4.1.1. – Arhitektura testbenča

Kao što se vidi simulaciono okruženje se sastoji od generatora stimulusa, dizajna koji ispitujemo i komparatora očekivanih vrednosti. Ovakvo definisano simulaciono okruženje se naziva testbenč (*testbench*). Generator stimulusa kreira vrednosti ulaznih signala. Kreirane vrednosti ulaznih signala treba da omoguće rad testiranog dizajna u svim mogućim situacijama (ili bar u većini mogućih situacija), i otuda je veoma važno dobro osmisliti generator stimulusa tako da on generiše vrednosti stimulusa koje će omogućiti efikasno ispitivanje ispravnosti rada dizajna. U slučaju testiranja kompleksnog dizajna, tipično je i generator stimulusa obiman jer zahteva generisanje velikog broja stimulusa i velikog broja vrednosti za svakog od njih, ali isplati se uložiti trud u kreiranje kvalitetnog generatora stimulusa jer ćemo na taj način pouzdano

moći utvrditi ispravnost dizajna, a takođe ćemo lakše otkriti i greške u dizajnu ukoliko ih ima. Dizajn koji ispitujemo je u stvari top level entitet kreiranog dizajna čime omogućavamo ispitivanje rada kompletnog našeg dizajna. Naravno, u početnim fazama projekta (naročito u slučaju složenog dizajna), poželjno je ispitati i ispravnost rada samo pojedinih delova kreiranog dizajna jer na taj način uglavnom se ubrzava razvoj (princip razbijanja složenog problema na više jednostavnijih) i veći broj grešaka će biti otklonjen pre ispitivanja kompletnog dizajna. Komparator očekivanih vrednosti može, ali i ne mora biti uključen u proces simulacije. Ukoliko nije uključen u proces simulacije, tada verifikujemo ispravnost dizajna posmatranjem vrednosti na izlazima testiranog (simuliranog) dizajna. Ako je komparator očekivanih vrednosti uključen u proces simulacije tada on proverava vrednosti na izlazu simuliranog dizajna sa očekivanim vrednostima i izbacuje rezultat provere. Rezultat provere može biti broj poklapanja i nepoklapanja sa očekivanim vrednostima ili prosta binarna signalizacija (jedna ili više njih u zavisnosti koliko provera se vrši) koja signalizira poklapanje i nepoklapanje u komparatoru.

4.2 Primer testbenča

Kao primer testbenča ćemo uzeti dekadni brojač koji smo koristili u primerima u prethodnom poglavlju. U svim simulacijama u okviru ovog i narednih potpoglavlja (sem potpoglavlja 4.7) ćemo koristiti ISim simulator opisan u trećem poglavlju. U trećem poglavlju je opisano kako se pokreće i koristi ovaj simulator, pa to neće biti ponovo opisivano u okviru ovog potpoglavlja. VHDL kod dekadnog brojača je:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR
USE ieee.std_logic_unsigned.all;--neophodno za operaciju sabiranja

--dekadni brojac koji kruzno broji unapred 0 do 9
ENTITY dekadni_brojac IS
PORT
(
    clk:    IN STD_LOGIC;--signal takta
    reset:  IN STD_LOGIC;--asinhroni signal reseta
    inc:    IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
    q:      OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
);
END dekadni_brojac;

ARCHITECTURE shema OF dekadni_brojac IS
    SIGNAL q_int: STD_LOGIC_VECTOR(3 DOWNTO 0);--interno stanje brojaca
BEGIN

    --proces koji definise brojanje
    PROCESS(reset,clk)
    BEGIN
        IF(reset='1')THEN--ako je asinhroni reset aktivan
            q_int<=(OTHERS=>'0');--brojac resetujemo na 0
        ELSIF(clk'EVENT and clk='1')THEN--uzlazna ivica takta
            IF(inc='1')THEN--ako je kontrolni signal za dozvolu brojanja aktivan kreni da brojis
                IF(q_int = "1001")THEN -- ako smo dosli do 9 sledeca vrednost brojaca treba da bude 0
                    q_int<=(OTHERS=>'0');
                ELSE
                    q_int<=q_int+'1';
                END IF;-- end if (q_int = "1001")
            END IF;
        END IF;
    END PROCESS;
END shema;
```

```

                END IF;-- end if inc='1'
            END IF;-- end if clk
        END PROCESS;
        -- prosledjivanje internog stanja brojaca na izlaz entiteta
        --da je q definisan u modu buffer, tada q_int ne bi bio neophodan
        q<=q_int;
    END shema;

```

Da bismo testirali ispravan rad brojača treba da pokrijemo sve situacije u kojima se brojač treba naći. To znači da treba da ga resetujemo dok je inkrement aktivan i neaktivan, ali i da testiramo aktivnu i neaktivnu vrednost inkrementa dok je reset neaktivan. Takođe treba pustiti i da brojač izbroji sve vrednosti od 0 do 9 i da se ponovo vrati na 0 i nastavi dalje da broji. VHDL kod testbenča je sledeći:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR

--testbench za dekadni brojac
ENTITY testbench IS
END testbench;

ARCHITECTURE shema OF testbench IS

--deklaracija komponente dekadnog brojaca
COMPONENT dekadni_brojac IS
PORT
(
    clk:    IN STD_LOGIC;--signal takta
    reset:  IN STD_LOGIC;--asinhroni signal reseta
    inc:    IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
    q:      OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
);
END COMPONENT;

--signali koje moramo da generisemo da bi testirali dizajn
SIGNAL clk:STD_LOGIC;
SIGNAL reset:STD_LOGIC;
SIGNAL inc:STD_LOGIC;
SIGNAL q_out:STD_LOGIC_VECTOR(3 DOWNTO 0);

BEGIN

--generisanje stimulusa takta
PROCESS
BEGIN
    --generisanje takta od 100MHz (perioda 10ns) - 5ns '1', i 5ns '0'
    loop
        clk<='0';
        wait for 5ns;
        clk<='1';
        wait for 5ns;
    end loop;
END PROCESS;

--generisanje stimulusa reseta
PROCESS
BEGIN
    reset<='1';--reset za inicijalizaciju dizajna
    wait for 15ns;--aktivna vrednost reseta traje 15ns, posle se obori na '0'
    reset<='0';
    wait for 200ns;
    reset<='1';--aktivan reset dok je inkrement aktivan

```

```

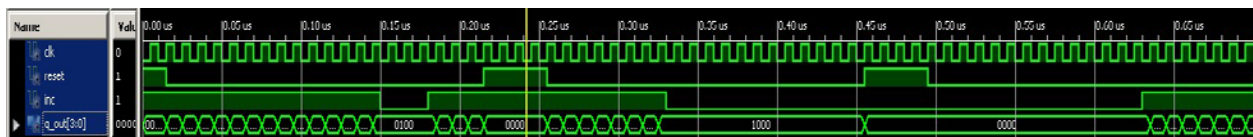
        wait for 40ns;
        reset<='0';
        wait for 200ns;
        reset<='1';--aktivan reset dok je inkrement neaktivan
        wait for 40ns;
        loop--neophodno da bi zadržali konstantno reset na '0'
            reset<='0';
            wait for 5ns;
        end loop;
    END PROCESS;
    --generisanje stimulusa inkrementa
    PROCESS
    BEGIN
        inc<='1';
        wait for 150ns;
        inc<='0';--da bi proverili da li se brojac zamrzne kad je inkrement neaktivan
        wait for 30ns;--pauza inkrementa u trajanju od 30ns
        inc<='1';
        wait for 150ns;
        inc<='0';--pauza inkrementa u trajanju od 300ns
        wait for 300ns;
        loop--neophodno da bi zadržali konstantno inkrement na '1'
            inc<='1';
            wait for 5ns;
        end loop;
    END PROCESS;
    --instanciranje komponente brojaca
    brojac_inst: dekadni_brojac
    PORT MAP
    (
        clk => clk,
        reset => reset,
        inc => inc,
        q => q_out
    );

```

END shema;

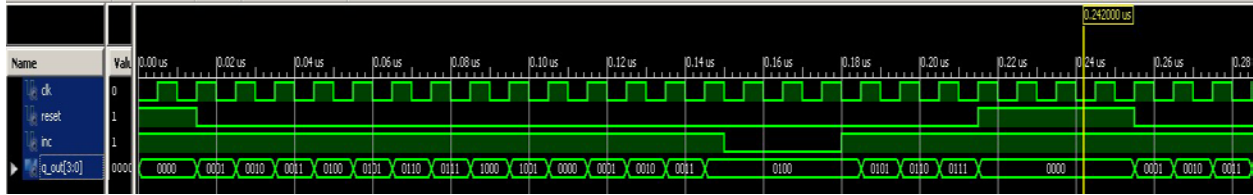
Radi preglednosti definisana su tri procesa, gde svaki proces generiše jedan stimulus (takt, reset i inkrement). Kao što se vidi kristi se konstruktija WAIT FOR radi kreiranja kašnjenja nakon koje se definiše promena signala. LOOP petlja se koristi u slučaju takta za generisanje periodičnog signala, a u slučaju reseta i inkrementa za 'zaglavljivanje' na konstantnu vrednost.

Na slici 4.2.1 je prikazan rezultat izvršene simulacije sa ovako definisanim testbenčom.

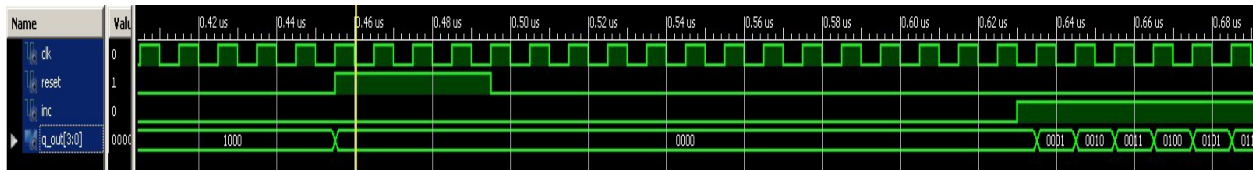


Slika 4.2.1. – Prikaz rezultata simulacije

Kao što vidimo postigli smo simulacijom pokrivanje svih željenih situacija. Reset je aktiviran kada je inkrement aktivan, a takođe i kada je inkrement neaktivan. Takođe, imamo situaciju sa aktivnom i neaktivnom vrednošću inkrementa kada je reset neaktivan. Takođe je inkrement pušten da traje dovoljno dugo da bi brojčar obišao čitav krug. Zumirani prikazi rezultata simulacije su dati na slikama 4.2.2 i 4.2.3.



Slika 4.2.2. – Zumirani prikaz rezultata simulacije 1



Slika 4.2.3. – Zumirani prikaz rezultata simulacije 2

Signali reseta i takta se tipično postavljaju unutar procesa u arhitekturi testbenč entiteta, kao što je bio slučaju i u datom primeru. Međutim, za generisanje ostalih signala se mogu koristiti i posebni entiteti kojih može biti jedan ili više, u zavisnosti od broja stimulusa i njihove logičke povezanosti. U slučaju da želimo da kreiramo poseban entitet za kreiranje stimulusa inkrementa on bi izgledao na sledeći način:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR

--generator inkrement stimulusa
ENTITY inkrement_generator IS
PORT
(
  inc:OUT STD_LOGIC
);
END inkrement_generator;

ARCHITECTURE shema OF inkrement_generator IS
BEGIN
  --generisanje stimulusa inkrementa
  PROCESS
  BEGIN
    inc<='1';
    wait for 150ns;
    inc<='0';--da bi proverili da li se brojac zamrzne kad je inkrement neaktivan
    wait for 30ns;--pauza inkrementa u trajanju od 30ns
    inc<='1';
    wait for 150ns;
    inc<='0';--pauza inkrementa u trajanju od 300ns
    wait for 300ns;
    loop--neophodno da bi zadržali konstantno inkrement na '1'
      inc<='1';
      wait for 5ns;
    end loop;
  END PROCESS;
END shema;

```

Kao što vidimo, proces koji je generisao stimulus inkrementa je pomeren sada u poseban entitet, koji se u vidu komponente poziva u okviru testbenč entiteta čiji kod sada izgleda:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR

```

```

--testbench za dekadni brojac
ENTITY testbench IS
END testbench;

ARCHITECTURE shema OF testbench IS

--deklaracija komponente generatora inkrement stimulusa
COMPONENT inkrement_generator IS
PORT
(
  inc:OUT STD_LOGIC
);
END COMPONENT;
--deklaracija komponente dekadnog brojaca
COMPONENT dekadni_brojac IS
PORT
(
  clk:    IN STD_LOGIC;--signal takta
  reset:  IN STD_LOGIC;--asinhroni signal reseta
  inc:    IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
  q:      OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
);
END COMPONENT;
--signali koje moramo da generisemo da bi testirali dizajn
SIGNAL clk:STD_LOGIC;
SIGNAL reset:STD_LOGIC;
SIGNAL inc:STD_LOGIC;
SIGNAL q_out:STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
--generisanje stimulusa takta
PROCESS
BEGIN
  --generisanje takta od 100MHz (perioda 10ns) - 5ns '1', i 5ns '0'
  loop
    clk<='0';
    wait for 5ns;
    clk<='1';
    wait for 5ns;
  end loop;
END PROCESS;
--generisanje stimulusa reseta
PROCESS
BEGIN
  reset<='1';--reset za inicijalizaciju dizajna
  wait for 15ns;--aktivna vrednost reseta traje 15ns, posle se obori na '0'
  reset<='0';
  wait for 200ns;
  reset<='1';--reset dok je inkrement aktivan
  wait for 40ns;
  reset<='0';
  wait for 200ns;
  reset<='1';--reset dok je inkrement neaktivan
  wait for 40ns;
  loop--neophodno da bi zadrzali konstantno reset na '0'
    reset<='0';
    wait for 5ns;
  end loop;
END PROCESS;
--generisanje stimulusa inkrementa

```

```

incgen_inst: inkrement_generator
PORT MAP
(
    inc => inc
);
--instanciranje komponente brojaca
brojac_inst: dekadni_brojac
PORT MAP
(
    clk => clk,
    reset => reset,
    inc => inc,
    q => q_out
);

```

END shema;

Rezultati simulacije su, naravno, identični onima prikazanim na slikama 4.2.1-4.2.3.

Često se u okviru generisanja stimulusa koji ne predstavljaju reset i takt, koriste upravo signali reseta i takta za generisanje ostalih stimulusa. Ako bi se kreirao generator stimulusa inkrementa na ovaj način tada bi VHDL kod entiteta *inkrement_generator* bio:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR

--generator inkrement stimulusa
ENTITY inkrement_generator IS
PORT
(
    clk:    IN STD_LOGIC;
    inc:    OUT STD_LOGIC
);
END inkrement_generator;

ARCHITECTURE shema OF inkrement_generator IS
    SIGNAL cnt:INTEGER:=0;--za brojanje ciklusa signala takta
    SIGNAL inc_int:STD_LOGIC:='1';
BEGIN
    --generisanje stimulusa inkrementa
    PROCESS(clk)
    BEGIN
        IF(clk'EVENT AND clk='0')THEN -- na silaznu ivicu takta
            cnt<=cnt+1;
            IF(cnt=15)THEN
                inc_int<='0';
            END IF;
            IF(cnt=18)THEN
                inc_int<='1';
            END IF;
            IF(cnt=33)THEN
                inc_int<='0';
            END IF;
            IF(cnt=63)THEN
                inc_int<='1';
            END IF;
        END IF;
    END PROCESS;
    inc<=inc_int;--prosleđjivanje interno definisanog inkrementa na izlaz
END shema;

```


U ovom slučaju se koristi samo takt kao ulazni signal, jer signal reseta više puta menja svoju vrednost tokom simulacije. Tipično se signal reseta koristi unutar entiteta za generisanje ostalih stimulusa u slučajevima kada reset vrši identičnu ulogu kao kod testiranog dizajna - resetuje interno stanje entiteta na neke inicijalne vrednosti, pri čemu se tipično aktivna vrednost reseta generiše samo na početku u procesu simulacije. Uz pomoć internog brojača, brojimo cikluse takta i na odgovarajućim vrednostima brojača menjamo vrednost stimulusa, u ovom slučaju signala inkrementa. Struktura VHDL koda testbenča je sada:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR

--testbench za dekadni brojac
ENTITY testbench IS
END testbench;

ARCHITECTURE shema OF testbench IS

--deklaracija komponente generatora inkrement stimulusa
COMPONENT inkrement_generator IS
PORT
(
  clk:   IN STD_LOGIC;
  inc:   OUT STD_LOGIC
);
END COMPONENT;
--deklaracija komponente dekadnog brojaca
COMPONENT dekadni_brojac IS
PORT
(
  clk:   IN STD_LOGIC;--signal takta
  reset: IN STD_LOGIC;--asinhroni signal reseta
  inc:   IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
  q:     OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
);
END COMPONENT;
--signali koje moramo da generisemo da bi testirali dizajn
SIGNAL clk:STD_LOGIC;
SIGNAL reset:STD_LOGIC;
SIGNAL inc:STD_LOGIC;
SIGNAL q_out:STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
--generisanje stimulusa takta
PROCESS
BEGIN
  --generisanje takta od 100MHz (perioda 10ns) - 5ns '1', i 5ns '0'
  loop
    clk<='0';
    wait for 5ns;
    clk<='1';
    wait for 5ns;
  end loop;
END PROCESS;
--generisanje stimulusa reseta
PROCESS
BEGIN
  reset<='1';--reset za inicijalizaciju dizajna
  wait for 15ns;--aktivna vrednost reseta traje 15ns, posle se obori na '0'
  reset<='0';

```

```

        wait for 200ns;
        reset<='1';--reset dok je inkrement aktivan
        wait for 40ns;
        reset<='0';
        wait for 200ns;
        reset<='1';--reset dok je inkrement neaktivan
        wait for 40ns;
        loop--neophodno da bi zadržali konstantno reset na '0'
            reset<='0';
            wait for 5ns;
        end loop;
    END PROCESS;
    --generisanje stimulusa inkrementa
    incgen_inst: inkrement_generator
    PORT MAP
    (
        clk => clk,
        inc => inc
    );
    --instanciranje komponente brojaca
    brojac_inst: dekadni_brojac
    PORT MAP
    (
        clk => clk,
        reset => reset,
        inc => inc,
        q => q_out
    );
END shema;

```

Rezultati simulacije su, naravno, identični onima prikazanim na slikama 4.2.1-4.2.3.

Primer kada je uključen i signal reseta u generisanje signala inkrementa je sledeći. Struktura VHDL koda entiteta za generisanje stimulusa za inkrement signal je:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR

--generator inkrement stimulusa
ENTITY inkrement_generator IS
PORT
(
    clk:    IN STD_LOGIC;
    reset:  IN STD_LOGIC;
    inc:    OUT STD_LOGIC
);
END inkrement_generator;

ARCHITECTURE shema OF inkrement_generator IS
    SIGNAL cnt:INTEGER:=0;--za brojanje ciklusa signala takta
    SIGNAL inc_int:STD_LOGIC;
BEGIN
    --generisanje stimulusa inkrementa
    PROCESS(reset,clk)
    BEGIN
        IF(reset='1')THEN
            inc_int<='1';
            cnt<=0;
        ELSIF(clk'EVENT AND clk='0')THEN -- na silaznu ivicu takta
            cnt<=cnt+1;
        END IF;
    END PROCESS;
END shema;

```

```

                IF(cnt=4)THEN
                    inc_int<='0';
                END IF;
                IF(cnt=7)THEN
                    inc_int<='1';
                END IF;
            END IF;

        END PROCESS;
        inc<=inc_int;--prosleđivanje interno definisanog inkrementa na izlaz
    END shema;

```

Struktura testbenča je sada:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR

```

```
--testbench za dekadni brojac
```

```

ENTITY testbench IS
END testbench;

```

```
ARCHITECTURE shema OF testbench IS
```

```
--deklaracija komponente generatora inkrement stimulusa
```

```

COMPONENT inkrement_generator IS
PORT

```

```

(
    clk:    IN STD_LOGIC;
    reset:  IN STD_LOGIC;
    inc:    OUT STD_LOGIC
);

```

```
END COMPONENT;
```

```
--deklaracija komponente dekadnog brojaca
```

```

COMPONENT dekadni_brojac IS
PORT

```

```

(
    clk:    IN STD_LOGIC;--signal takta
    reset:  IN STD_LOGIC;--asinhroni signal reseta
    inc:    IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
    q:      OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
);

```

```
END COMPONENT;
```

```
--signali koje moramo da generisemo da bi testirali dizajn
```

```

SIGNAL clk:STD_LOGIC;
SIGNAL reset:STD_LOGIC;
SIGNAL inc:STD_LOGIC;
SIGNAL q_out:STD_LOGIC_VECTOR(3 DOWNTO 0);

```

```
BEGIN
```

```
--generisanje stimulusa takta
```

```
PROCESS
```

```
BEGIN
```

```
--generisanje takta od 100MHz (perioda 10ns) - 5ns '1', i 5ns '0'
```

```
loop
```

```

    clk<='0';
    wait for 5ns;
    clk<='1';
    wait for 5ns;

```

```
end loop;
```

```
END PROCESS;
```

```
--generisanje stimulusa reseta
```

**PROCESS
BEGIN**

```
reset<='1';--reset za inicijalizaciju dizajna  
wait for 15ns;--aktivna vrednost reseta traje 15ns, posle se obori na '0'  
reset<='0';  
wait for 200ns;  
reset<='1';--reset dok je inkrement aktivan  
wait for 40ns;  
reset<='0';  
wait for 200ns;  
reset<='1';--reset dok je inkrement neaktivan  
wait for 40ns;  
loop--neophodno da bi zadržali konstantno reset na '0'  
    reset<='0';  
    wait for 5ns;  
end loop;
```

END PROCESS;

```
--generisanje stimulusa inkrementa  
incgen_inst: inkrement_generator
```

PORT MAP

```
(  
    clk => clk,  
    reset => reset,  
    inc => inc  
);
```

```
--instanciranje komponente brojac
```

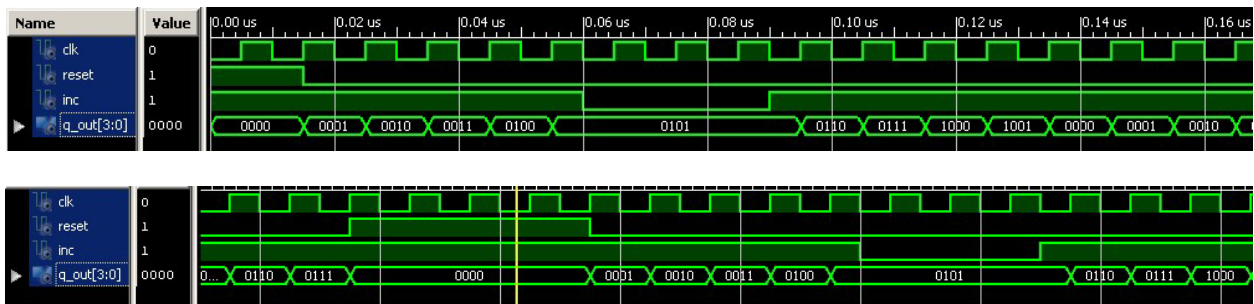
```
brojac_inst: dekadni_brojac
```

PORT MAP

```
(  
    clk => clk,  
    reset => reset,  
    inc => inc,  
    q => q_out  
);
```

END shema;

U ovom slučaju entitet stimulusa se resetuje svaki put kada se aktivira signal reseta. Entitet stimulusa generiše aktivan signal inkrementa, sa jednom pauzom koja traje tri ciklusa takta. Rezultat simulacije je naravno nešto drugačiji i prikazan je na slici 4.2.4 koja prikazuje rezultat simulacije za prve dve aktivne vrednosti signala reseta. Donja polovina slike 4.2.4 odgovara drugoj aktivaciji reseta. Kao što se vidi dizajn se ponaša identično posle deaktivacije reseta u oba slučaja.



Slika 4.2.4. – Prikaz rezultata simulacije za prve dve aktivne vrednosti signala reseta

Što se tiče testiranja dizajna koji sadrži samo kombinacionu logiku, jedina razlika je što signal takta ne postoji. Otuda se koristi princip demonstriran za signal inkrementa u prva dva

primera testbenča data u okviru ovog potpoglavlja, a to je generisanje različitih vrednosti stimulusa upotrebom WAIT FOR konstrukcije.

4.3 Komparator očekivanih vrednosti

U okviru ovog potpoglavlja ćemo prikazati kako se definiše i koristi komparator očekivanih vrednosti. I u ovom slučaju ćemo simulirati rad dekadnog brojača, čiji VHDL kod je dat u prethodnom potpoglavlju. VHDL kod testbenča u ovom slučaju je:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR
```

```
--testbench za dekadni brojac
```

```
ENTITY testbench IS
```

```
END testbench;
```

```
ARCHITECTURE shema OF testbench IS
```

```
--deklaracija komponente komparatora ocekivanih vrednosti
```

```
COMPONENT komparator_ocekivanih IS
```

```
PORT
```

```
(
```

```
    clk:    IN STD_LOGIC;
    reset:  IN STD_LOGIC;
    q_in:   IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    ok:     OUT STD_LOGIC;
    error:  OUT STD_LOGIC
```

```
);
```

```
END COMPONENT;
```

```
--deklaracija komponente dekadnog brojaca
```

```
COMPONENT dekadni_brojac IS
```

```
PORT
```

```
(
```

```
    clk:    IN STD_LOGIC;--signal takta
    reset:  IN STD_LOGIC;--asinhroni signal reseta
    inc:    IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
    q:      OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
```

```
);
```

```
END COMPONENT;
```

```
--signali koje moramo da generisemo da bi testirali dizajn
```

```
SIGNAL clk:STD_LOGIC;
```

```
SIGNAL reset:STD_LOGIC;
```

```
SIGNAL inc:STD_LOGIC;
```

```
SIGNAL q_out:STD_LOGIC_VECTOR(3 DOWNTO 0);
```

```
SIGNAL ok:STD_LOGIC;
```

```
SIGNAL error:STD_LOGIC;
```

```
BEGIN
```

```
--generisanje stimulusa takta
```

```
PROCESS
```

```
BEGIN
```

```
--generisanje takta od 100MHz (perioda 10ns) - 5ns '1', i 5ns '0'
```

```
loop
```

```
    clk<='0';
```

```
    wait for 5ns;
```

```
    clk<='1';
```

```
    wait for 5ns;
```

```
end loop;
```

```

END PROCESS;
--generisanje stimulusa reseta
PROCESS
BEGIN
    reset<='1';--reset za inicijalizaciju dizajna
    wait for 15ns;--aktivna vrednost reseta traje 15ns, posle se obori na '0'
    reset<='0';
    wait for 200ns;
    reset<='1';--reset dok je inkrement aktivan
    wait for 40ns;
    reset<='0';
    wait for 200ns;
    reset<='1';--reset dok je inkrement neaktivan
    wait for 40ns;
    loop--neophodno da bi zadržali konstantno reset na '0'
        reset<='0';
        wait for 5ns;
    end loop;
END PROCESS;
--generisanje stimulusa inkrementa
PROCESS
BEGIN
    loop--neophodno da bi zadržali konstantno inkrement na '1'
        inc<='1';
        wait for 5ns;
    end loop;
END PROCESS;

--instanciranje komponente komparator
komparator_inst: komparator_ocekivanih
PORT MAP
(
    clk => clk,
    reset => reset,
    q_in => q_out,
    ok => ok,
    error => error
);

--instanciranje komponente brojac
brojac_inst: dekadni_brojac
PORT MAP
(
    clk => clk,
    reset => reset,
    inc => inc,
    q => q_out
);

```

END shema;

U ovom slučaju generišemo neprekidno aktivnu vrednost signala inkrementa. VHDL kod komparatora očekivanih vrednosti je:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR

--komparator ocekivanih vrednosti
ENTITY komparator_ocekivanih IS
PORT

```

```

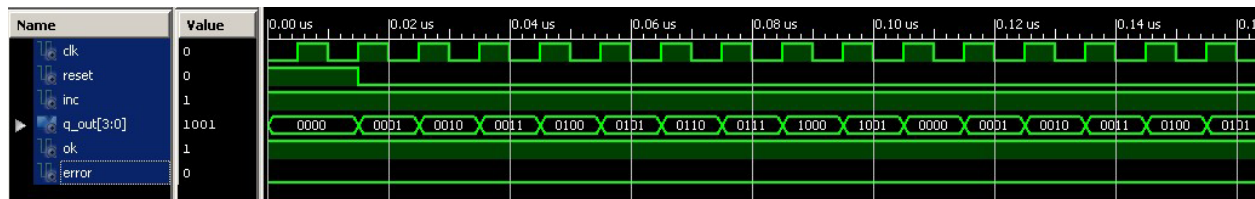
(
    clk:    IN STD_LOGIC;
    reset:  IN STD_LOGIC;
    q_in:   IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    ok:     OUT STD_LOGIC;
    error:  OUT STD_LOGIC
);
END komparator_cekivanih;

ARCHITECTURE shema OF komparator_cekivanih IS
    SIGNAL q_cekivano:    STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
    --definisiranje cekivanih vrednosti
    PROCESS(reset,clk)
    BEGIN
        IF(reset='1')THEN
            q_cekivano<="0000";
        ELSIF(clk'EVENT AND clk='1')THEN
            CASE q_cekivano IS
                WHEN "0000" =>
                    q_cekivano<="0001";
                WHEN "0001" =>
                    q_cekivano<="0010";
                WHEN "0010" =>
                    q_cekivano<="0011";
                WHEN "0011" =>
                    q_cekivano<="0100";
                WHEN "0100" =>
                    q_cekivano<="0101";
                WHEN "0101" =>
                    q_cekivano<="0110";
                WHEN "0110" =>
                    q_cekivano<="0111";
                WHEN "0111" =>
                    q_cekivano<="1000";
                WHEN "1000" =>
                    q_cekivano<="1001";
                WHEN "1001" =>
                    q_cekivano<="0000";
                WHEN OTHERS =>
                    q_cekivano<="XXXX";
            END CASE;
        END IF;
    END PROCESS;
    --komparacija cekivanih vrednosti sa vrednostima
    --dobijenim iz testiranog dekadnog brojaca
    ok<= '1' WHEN q_cekivano=q_in ELSE
        '0';
    error<= '0' WHEN q_cekivano=q_in ELSE
        '1';
END shema;

```

Komparator proverava očekivane vrednosti sa onim primljenim od simuliranog dekadnog brojača i generiše signal greške i signal ispravnog rada (ova dva signala su u suštini međusobno komplementarna što se može videti i iz njihove definicije u priloženom VHDL kodu komparatora). Na slici 4.3.1 je prikazan rezultat simulacije, i može se videti da komparator

generiše neprekidno aktivnu vrednost signala ispravnog rada, odnosno neaktivnu vrednost signala greške.



Slika 4.3.1. – Prikaz rezultata simulacije kada se koristi komparator očekivanih vrednosti

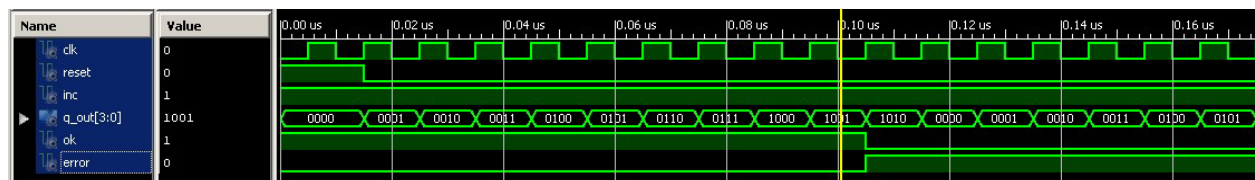
Sada namerno unesimo grešku u dekadnom brojaču tako što ćemo izmeniti sledeću liniju koda:

```
IF(q_int = "1001")THEN
```

u

```
IF(q_int = "1010")THEN
```

Sada će se detektovati greška, odnosno signal greške će se aktivirati, što se vidi sa slike 4.3.2. Kao što vidimo ovaj metod sa upotrebom komparatora očekivanih vrednosti omogućava veoma jednostavno vizuelno utvrđivanje postojanja grešaka u dizajnu jer se prati samo binarni signal greške (ili komplementarnog signala ispravne vrednosti). Naravno, da bi došli do ovako jednostavnog vizuelnog načina utvrđivanja ispravnosti rada dizajna, neophodno je uložiti trud u pisanje koda komparatora očekivanih vrednosti. U opštem slučaju broj komparatora očekivanih vrednosti može biti i veći od jedan.



Slika 4.3.2. – Prikaz rezultata simulacije kada se koristi komparator očekivanih vrednosti kada postoji greška

Pored generisanja binarnih vrednosti, mogu se generisati i vrednosti broja poklapanja i nepoklapanja između očekivanih vrednosti i onih dobijenih simulacijom testiranog dizajna. VHDL kod komparatora očekivanih vrednosti u ovom slučaju je:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR

--komparator ocekivanih vrednosti
ENTITY komparator_ocekivanih IS
PORT
(
    clk:           IN STD_LOGIC;
    reset:        IN STD_LOGIC;
    q_in:         IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    cnt_ok:       OUT INTEGER;
    cnt_error:    OUT INTEGER
);
END komparator_ocekivanih;

ARCHITECTURE shema OF komparator_ocekivanih IS
    SIGNAL q_ocekivano: STD_LOGIC_VECTOR(3 DOWNTO 0);
```



```

SIGNAL cnt_ok1, cnt_error1:INTEGER;--interni brojaci
BEGIN
  --definisanje ocekivanih vrednosti
  PROCESS(reset,clk)
  BEGIN
    IF(reset='1')THEN
      q_cekivano<="0000";
      cnt_ok1<=0;
      cnt_error1<=0;
    ELSIF(clk'EVENT AND clk='1')THEN
      CASE q_cekivano IS
        WHEN "0000" =>
          q_cekivano<="0001";
        WHEN "0001" =>
          q_cekivano<="0010";
        WHEN "0010" =>
          q_cekivano<="0011";
        WHEN "0011" =>
          q_cekivano<="0100";
        WHEN "0100" =>
          q_cekivano<="0101";
        WHEN "0101" =>
          q_cekivano<="0110";
        WHEN "0110" =>
          q_cekivano<="0111";
        WHEN "0111" =>
          q_cekivano<="1000";
        WHEN "1000" =>
          q_cekivano<="1001";
        WHEN "1001" =>
          q_cekivano<="0000";
        WHEN OTHERS =>
          q_cekivano<="XXXXX";
      END CASE;
      --komparacija ocekivanih vrednosti sa vrednostima
      --dobijenim iz testiranog dekadnog brojaca
      IF(q_cekivano=q_in)THEN
        cnt_ok1<=cnt_ok1+1;
      ELSE
        cnt_error1<=cnt_error1+1;
      END IF;
    END IF;
  END PROCESS;
  --prosledjivanje vrednosti internih brojaca na izlaze
  cnt_error<=cnt_error1;
  cnt_ok<=cnt_ok1;
END shema;

```

Kao što vidimo sada se umesto binarnih signalizacija greške i ispravnog rada, koriste brojači poklapanja i nepoklapanja očekivanih vrednosti sa onima koje su ostvarene simulacijom. Osim te izmene, ostatak koda je veoma sličan prvoj varijanti komparatora očekivanih vrednosti. VHDL kod testbenč entiteta je sada:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR
--testbench za dekadni brojac
ENTITY testbench IS
END testbench;

```

ARCHITECTURE shema OF testbench IS

--deklaracija komponente komparatora ocekivanih vrednosti

COMPONENT komparator_okekivanih **IS**

PORT

(

 clk: **IN** STD_LOGIC;
 reset: **IN** STD_LOGIC;
 q_in: **IN** STD_LOGIC_VECTOR(3 **DOWNTO** 0);
 cnt_ok: **OUT** INTEGER;
 cnt_error: **OUT** INTEGER

);

END COMPONENT;

--deklaracija komponente dekadnog brojaca

COMPONENT dekadni_brojac **IS**

PORT

(

 clk: **IN** STD_LOGIC;--signal takta
 reset: **IN** STD_LOGIC;--asinhroni signal reseta
 inc: **IN** STD_LOGIC;--kontrolni signal za dozvolu brojanja
 q: **OUT** STD_LOGIC_VECTOR(3 **DOWNTO** 0)--vrednost brojaca

);

END COMPONENT;

--signali koje moramo da generisemo da bi testirali dizajn

SIGNAL clk:STD_LOGIC;

SIGNAL reset:STD_LOGIC;

SIGNAL inc:STD_LOGIC;

SIGNAL q_out:STD_LOGIC_VECTOR(3 **DOWNTO** 0);

SIGNAL cnt_ok:INTEGER;

SIGNAL cnt_error:INTEGER;

BEGIN

--generisanje stimulusa takta

PROCESS

BEGIN

 --generisanje takta od 100MHz (perioda 10ns) - 5ns '1', i 5ns '0'

loop

 clk<='0';

wait for 5ns;

 clk<='1';

wait for 5ns;

end loop;

END PROCESS;

--generisanje stimulusa reseta

PROCESS

BEGIN

 reset<='1';--reset za inicijalizaciju dizajna

wait for 15ns;--aktivna vrednost reseta traje 15ns, posle se obori na '0'

 reset<='0';

wait for 200ns;

 reset<='1';--reset dok je inkrement aktivan

wait for 40ns;

 reset<='0';

wait for 200ns;

 reset<='1';--reset dok je inkrement neaktivan

wait for 40ns;

loop--neophodno da bi zadržali konstantno reset na '0'

 reset<='0';

wait for 5ns;

end loop;

```

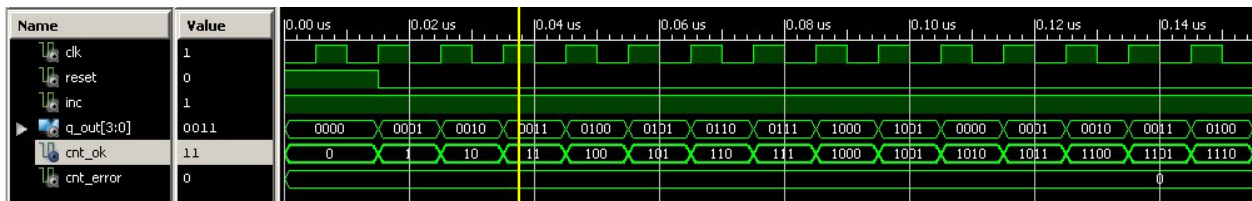
END PROCESS;
--generisanje stimulusa inkrementa
PROCESS
BEGIN
    loop--neophodno da bi zadržali konstantno inkrement na '1'
        inc<='1';
        wait for 5ns;
    end loop;
END PROCESS;

--instanciranje komponente komparator
komparator_inst: komparator_ocekivanih
PORT MAP
(
    clk => clk,
    reset => reset,
    q_in => q_out,
    cnt_ok => cnt_ok,
    cnt_error => cnt_error
);

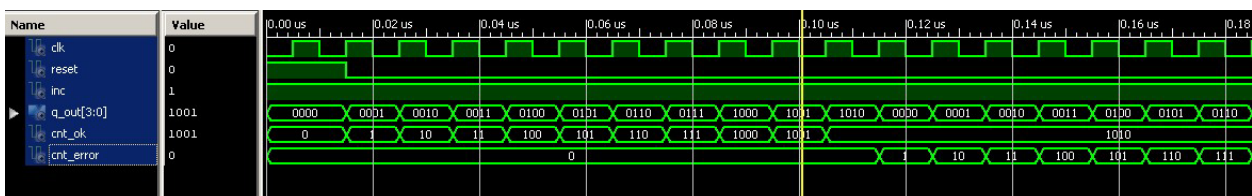
--instanciranje komponente brojača
brojac_inst: dekadni_brojac
PORT MAP
(
    clk => clk,
    reset => reset,
    inc => inc,
    q => q_out
);
END shema;

```

Na slikama 4.3.3 i 4.3.4 su prikazani rezultati simulacije za ispravan kod dekadnog brojača i neispravan kod dekadnog brojača, respektivno. Netačan kod brojača je dobijen na identičan način kao u prethodnom slučaju opisanom u ovom potpoglavlju.



Slika 4.3.3. – Prikaz rezultata simulacije kada se posmatraju brojači poklapanja i nepoklapanja - ispravan dekadni brojač



Slika 4.3.4. – Prikaz rezultata simulacije kada se posmatraju brojači poklapanja i nepoklapanja - neispravan dekadni brojač

Kao što vidimo u prvom slučaju samo brojač poklapanja se inkrementira, dok se u drugom slučaju inkrementira i brojač grešaka tj. nepoklapanja. I ovaj način omogućava

jednostavnu vizuelnu identifikaciju ispravnosti rada testiranog dizajna, pri čemu i dalje važi napomena da je potrebno uložiti trud u pisanje koda komparatora očekivanih vrednosti.

4.4 ASSERT komanda

Tokom simulacije je veoma praktično ispisivati obaveštenja koja ukazuju na ponašanje testiranog dizajna. Ispisana obaveštenja mogu, na primer, ukazati da se prošlo kroz izvestan deo koda (preciznije rečeno korišćena je struktura koju opisuje dotični deo koda) ili da je došlo do greške. Pametnim pozicioniranjem ispisa obaveštenja može se olakšati proces debugovanja dizajna i brže otkloniti postojeće greške u dizajnu. U ovu svrhu se koristi ASSERT komanda čija opšta struktura izgleda na sledeći način:

```
ASSERT (uslov)
REPORT poruka
SEVERITY nivo_poruke;
```

U okviru uslova ASSERT komande se definiše kada će biti ispisano obaveštenje. Važno je napomenuti da se obaveštenje ispisuje kada uslov ima vrednost FALSE. Sadržaj obaveštenja u vidu stringa se ispisuje u REPORT delu (*poruka*), dok SEVERITY deo ukazuje na nivo obaveštenja (*note*, *warning*, *error* ili *failure*). ASSERT struktura se može ubaciti i u konkurentni kod i u sekvencijalni kod. ASSERT struktura ne može da se implementira u hardveru. Većina kompajlera će ignorisati ASSERT strukturu u procesu kompajliranja dizajna ili generisati upozorenje. ISE i Quartu s razvojna okruženja će u procesu kompajliranja dizajna ignorisati ASSERT strukture. U slučaju da se dizajn, koji sadrži ASSERT strukture, ne može kompajlirati, tada prosto treba izbrisati ili staviti pod komentare sve ASSERT strukture u dizajnu.

Kreirajmo dve ASSERT strukture u primeru simulacije dekadnog brojača koje ćemo ubaciti unutar koda dekadnog brojača. Može se koristiti bilo koji testbenč primer iz prethodnih potpoglavlja. VHDL kod dekadnog brojača sa ubačenom ASSERT strukturom je sada:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR
USE ieee.std_logic_unsigned.all;--neophodno za operaciju sabiranja

--dekadni brojac koji kružno broji unapred 0 do 9
ENTITY dekadni_brojac IS
PORT
(
    clk:      IN STD_LOGIC;--signal takta
    reset:    IN STD_LOGIC;--asinhroni signal reseta
    inc:      IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
    q:        OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
);
END dekadni_brojac;

ARCHITECTURE shema OF dekadni_brojac IS
    SIGNAL q_int: STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";--interno stanje brojaca
BEGIN

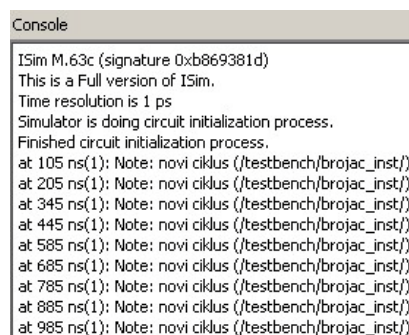
    --proces koji definise brojanje
    PROCESS(reset,clk)
    BEGIN
        IF(reset='1')THEN--ako je asinhroni reset aktivan
            q_int<=(OTHERS=>'0');--brojac resetujemo na 0
```

```

ELSEIF(clk'EVENT and clk='1')THEN--uzlazna ivica takta
    IF(inc='1')THEN--ako je kontrolni signal za dozvolu brojanja aktivan kreni da brojis
        IF(q_int = "1001")THEN -- ako smo dosli do 9 sledeca vrednost brojacu treba da bude 0
            q_int<=(OTHERS=>'0');
            --obavestjenje da je brojac zapoceo novi ciklus brojanja
            ASSERT (FALSE)
                REPORT "novi ciklus"
                    SEVERITY note;
        ELSE
            -- u suprotnom inkrementiramo brojac za 1
            q_int<=q_int+'1';
        END IF;-- end if (q_int = "1001")
    END IF;-- end if inc='1'
END IF;-- end if clk
END PROCESS;
-- prosledjivanje internog stanja brojacu na izlaz entiteta
--da je q definisan u modu buffer, tada q_int ne bi bio neophodan
q<=q_int;
--obavestjenje o gresci ako brojac izađe van opsega
ASSERT (q_int<="1001")
    REPORT "Brojac je van opsega"
        SEVERITY error;
END shema;

```

Kao što vidimo dodate su dve ASSERT strukture. Jedna služi da generiše obaveštenje svaki put kada se otpočne novi ciklus brojanja od 0 do 9, i ona se ispisuje bezuslovno jer je uslov postavljen na vrednost FALSE. Druga služi da obavesti da se desila greška ako brojač izađe iz dekadnog opsega (0 do 9). Obaveštenje o grešci se ispisuje samo ako brojač ima vrednost veću od "1001", pri čemu treba primetiti da je uslov formiran tako da ako brojač ima vrednost veću od "1001" tada će uslov dobiti vrednost FALSE. Takođe, u ovom primeru je ASSERT ubačen u sekvencijalni deo koda (prvi ASSERT) i u konkurentni deo koda (drugi ASSERT). Kada se pokrene simulacija, u konzoli će biti ispisana obaveštenja o početku novog ciklusa brojanja kao što je prikazano na slici 4.4.1. Napomenimo da je u deklaraciji signala *q_int* podešena njegova inicijalna vrednost. Ako se ona ne postavi, onda bi bila generisana poruka o grešci da je vrednost tog signala van opsega u trenutku Ops (sam početni trenutak simulacije) jer u tom trenutku simulator uzima inicijalne vrednosti signala ako su definisane, a u suprotnom ih postavlja na nepoznate vrednosti tj. 'X'.



```

Console
ISim M.63c (signature 0xb869381d)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 105 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
at 205 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
at 345 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
at 445 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
at 585 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
at 685 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
at 785 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
at 885 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
at 985 ns(1): Note: novi ciklus (/testbench/brojac_inst/).

```

Slika 4.4.1. – Ispis obaveštenja u konzoli

Ako namerno napravimo grešku u kodu brojača tako što uslov

```
IF(q_int = "1001")THEN
```

promenimo u

```
IF(q_int = "1010")THEN
```

tada će se ispisati i poruke o detektovanoj grešci kao što je prikazano na slici 4.4.2.

```
Console
ISim M.63c (signature 0xb869381d)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 105 ns(2): Error: Brojac je van opsega
at 115 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
at 345 ns(2): Error: Brojac je van opsega
at 355 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
at 585 ns(2): Error: Brojac je van opsega
at 595 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
at 695 ns(2): Error: Brojac je van opsega
at 705 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
at 805 ns(2): Error: Brojac je van opsega
at 815 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
at 915 ns(2): Error: Brojac je van opsega
at 925 ns(1): Note: novi ciklus (/testbench/brojac_inst/).
```

Slika 4.4.2. – Ispis obaveštenja o grešci u konzoli

Treba naglasiti da u situaciji kada se ispisuje obaveštenje nivoa *failure*, tada se simulacija nasilno zaustavlja, a za sve ostale nivoe obaveštenja se simulacija nastavlja. Ako promenimo nivo obaveštenja o grešci brojača (drugi ASSERT) iz nivoa *error* u nivo *failure* doći će do nasilnog zaustavljanja simulacije kao što je prikazano na slici 4 4 3 , za slu čaj već opisane namerno unete greške u kod dekadnog brojača.

```
Console
ISim M.63c (signature 0xb869381d)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.

** Failure:Brojac je van opsega
User(VHDL) Code Called Simulation Stop
In process dekadni_brojac.vhd:43

INFO: Simulator is stopped.
```

Slika 4.4.3. – Nasilno zaustavljanje simulatora usled aktivacije ASSERT komande nivoa *failure*

4.5 Čitanje vrednosti stimulusa iz fajla

Osim direktnog definisanja vrednosti stimulusa u vidu odgovarajućeg VHDL koda, moguće je i učitati vrednosti stimulusa iz fajlova. Ovo je naročito zgodno u slučaju kada možemo da generišemo vrednosti stimulusa u okviru aplikacija napisanih u programskim jezicima poput C, C++, Matlab i dr. Naravno, kada već generišemo vrednosti stimulusa iz aplikacije napisane u nekom od programskih jezika i zapisujemo ih u fajl ili fajlove, tada možemo i da u te fajlove ispišemo direktno dodelu vrednosti stimulusima po sintaksi VHDL koda i potom iskopiramo tako generisani VHDL kod na odgovarajuća mesta u testbenču.

Za čitanje vrednosti iz fajlova se koristi paket *textio* biblioteke *std*. U okviru ovog paketa je definisano učitavanje tipova podataka definisanih u okviru *std* biblioteke (INTEGER, BIT, BIT_VECTOR, REAL, TIME, BOOLEAN, CHARACTER i STRING). Ovaj paket se mora pozvati upotrebom USE instrukcije. Ako se želi izvršiti čitanje vrednosti tipova STD_LOGIC, STD_ULOGIC, STD_LOGIC_VECTOR i STD_ULOGIC_VECTOR tada se mora koristiti dodatno i paket *std_logic_textio* biblioteke *ieee*.

Fajl koji se želi otvoriti za čitanje se deklarise sa:

```
FILE promenjiva : TEXT OPEN READ_MODE IS "naziv_fajla.txt";
```

gde je *promenjiva* naziv koji je dodeljen fajlu za pozive unutar VHDL koda, a *naziv_fajla.txt* označava naziv fajla koji se otvara. Ako se fajl ne nalazi na istoj lokaciji gde i projekat, onda treba navesti i relativnu ili apsolutnu lokaciju fajla pored njegovog naziva. Fajl se tumači kao niz stringova, gde svaka linija u fajlu označava jedan string. Da bi se učitala jedna linija iz fajla koristi se procedura READLINE:

```
READLINE(f, l);
```

gde je *f* naziv dodeljen fajlu unutar VHDL koda, a *l* naziv promenljive tipa LINE.

Procedura READ omogućava potom učitavanje pročitane vrednosti u željeni signal stimulusa koji može da bude bilo koji od tipova podataka navedenih na početku ove sekcije:

```
READ (l,s,g);
```

gde je *l* naziv promenljive tipa LINE u koju smo učitali liniju iz fajla, *s* naziv signala stimulusa, a *g* je tipa BOOLEAN i predstavlja signalizaciju da li je učitavanje vrednosti u signal stimulusa bilo uspešno ili ne (TRUE ili FALSE). Argument *g* se može izostaviti. U suštini argument *g* je pogodan za proveru da li je došlo do greške u učitavanju, ali i za slučaj kada u tekstualni fajl pored vrednosti ubacujemo i linije komentara, čime proverom vrednosti argumenta *g* možemo utvrditi da li je učitana linija komentar ili ne. Za učitavanje vrednosti STD_LOGIC_VECTOR i STD_ULONGIC_VECTOR u heksadecimalnom zapisu na raspolaganju je procedura HREAD, a u oktalnom zapisu procedura OREAD. Argumenti ovih procedura su identični argumentima procedure READ.

Kreirajmo sada testbenč koji će učitati vrednosti stimulusa inkrementa identične onima iz prvog primera testbenča datog u sekciji 4.2. Struktura dekadnog brojača je ista kao u tom primeru. VHDL kod testbenča je sada:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR
USE ieee.std_logic_textio.all;--za rad sa fajlovima
USE std.textio.all;
--testbench za dekadni brojac
ENTITY testbench IS
END testbench;

ARCHITECTURE shema OF testbench IS

--deklaracija komponente dekadnog brojaca
COMPONENT dekadni_brojac IS
PORT
(
    clk:    IN STD_LOGIC;--signal takta
    reset:  IN STD_LOGIC;--asinhroni signal reseta
    inc:    IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
    q:      OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
);
END COMPONENT;

--signali koje moramo da generisemo da bi testirali dizajn
SIGNAL clk:STD_LOGIC;
SIGNAL reset:STD_LOGIC;
SIGNAL inc:STD_LOGIC;
SIGNAL q_out:STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
```

```

--generisanje stimulusa takta
PROCESS
BEGIN
    --generisanje takta od 100MHz (perioda 10ns) - 5ns '1', i 5ns '0'
    loop
        clk<='0';
        wait for 5ns;
        clk<='1';
        wait for 5ns;
    end loop;
END PROCESS;
--generisanje stimulusa reseta
PROCESS
BEGIN
    reset<='1';--reset za inicijalizaciju dizajna
    wait for 15ns;--aktivna vrednost reseta traje 15ns, posle se obori na '0'
    reset<='0';
    wait for 200ns;
    reset<='1';--reset dok je inkrement aktivan
    wait for 40ns;
    reset<='0';
    wait for 200ns;
    reset<='1';--reset dok je inkrement neaktivan
    wait for 40ns;
    loop--neophodno da bi zadržali konstantno reset na '0'
        reset<='0';
        wait for 5ns;
    end loop;
END PROCESS;
--generisanje stimulusa inkrementa
PROCESS
    FILE fajn_ulazni : TEXT OPEN READ_MODE IS "inkrement.txt";
    VARIABLE line_id : LINE;
    VARIABLE t : TIME;
    VARIABLE tmp : STD_LOGIC;
BEGIN
    if (not endfile(fajn_ulazni)) then--ako nismo stigli do kraja fajla
        READLINE(fajn_ulazni,line_id);
        READ(line_id,tmp);
        READLINE(fajn_ulazni,line_id);
        READ(line_id,t);
        inc<=tmp;
        wait for t;
    else
        --difolt vrednost stimulusa
        inc<='1';
        wait for 5ns;
    end if;
END PROCESS;

--instanciranje komponente brojaca
brojac_inst: dekadni_brojac
PORT MAP
(
    clk => clk,
    reset => reset,
    inc => inc,
    q => q_out
);
END shema;

```


Kao što vidimo u deklarativnom delu procesa zaduženog za generisanje stimulusa inkrementa se nalazi deklaracija fajla, kao i varijabli koje služe za učitavanje vrednosti inkrementa (*tmp*) i učitavanje vrednosti (*t*) neophodne za WAIT FOR konstrukciju. Varijabla *line_id* je neophodna kao pomoćna promenljiva u koju se smešta pročitana linija iz fajla. Čitanje iz fajla se vrši dok se ne dođe do kraja fajla, a potom se u ELSE delu postavlja difolt vrednost stimulusa kada su pročitane sve linije iz fajla. Funkcija *endfile* proverava da li se došlo do kraja fajla ili ne. Sadržaj tekstualnog fajla *inkrement.txt* je:

```
1
150ns
0
30ns
1
150ns
0
300ns
1
5ns
```

Očigledno, neparne linije odgovaraju vrednosti inkrementa, a parne linije vrednosti koja se koristi u WAIT FOR konstrukciji. Sadržaj je kreiran tako da odgovara vrednostima za stimulus inkrementa definisanim u okviru prvog testbenč primera iz sekcije 4.2. Takođe treba primetiti da su dodata dva paketa (*std.textio.all* i *ieee.std_logic_textio.all*) koja su neophodna za rad sa fajlovima, pri čemu paket *ieee.std_logic_textio.all* moramo da koristimo jer je signal inkrementa STD_LOGIC tipa.

Napomenimo još da je moguće otvoriti istovremeno više fajlova. Na primer, vrednosti za stimulus inkrementa i vrednosti za WAIT FOR konstrukciju su mogle biti razdvojene. Tada bi struktura procesa za generisanje stimulusa inkrementa bila (ostatak testbenča ostaje isti):

```
--generisanje stimulusa inkrementa
PROCESS
    FILE fajl_ulazni_inc : TEXT OPEN READ_MODE IS "inkrement.txt";
    FILE fajl_ulazni_time : TEXT OPEN READ_MODE IS "vreme.txt";
    VARIABLE line_id: LINE;
    VARIABLE t : TIME;
    VARIABLE tmp : STD_LOGIC;
BEGIN
    if ((not endfile(fajl_ulazni_inc))AND(not endfile(fajl_ulazni_time))) then--ako nismo stigli do kraja fajla
        READLINE(fajl_ulazni_inc,line_id);
        READ(line_id,tmp);
        READLINE(fajl_ulazni_time,line_id);
        READ(line_id,t);
        inc<=tmp;
        wait for t;
    else
        --difolt vrednost stimulusa
        inc<='1';
        wait for 5ns;
    end if;
END PROCESS;
```

Struktura *inkrement.txt* je sada:

```
1
0
1
```

0
1

dok je struktura fajla *vreme.txt*:

150ns
30ns
150ns
300ns
5ns

Za kraj prikazimo još primer kada se koristi HREAD procedura. U ovom primeru ćemo testirati displej čiji je VHDL kod dat u prethodnom poglavlju. VHDL kod testbenča u ovom slučaju bi bio:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR
USE ieee.std_logic_textio.all;--za rad sa fajlovima
USE std.textio.all;
--testbench za dekadni brojac
ENTITY testbench IS
END testbench;

ARCHITECTURE shema OF testbench IS

--deklaracija komponente dekadnog brojaca
COMPONENT displej IS
PORT
(
    broj:    IN STD_LOGIC_VECTOR(3 DOWNTO 0);--vrednost broja na ulazu
    displej: OUT STD_LOGIC_VECTOR(6 DOWNTO 0)--vektor ciji svaki indeks utice na jednu diodu displeja
);
END COMPONENT;

--signali koje moramo da generisemo da bi testirali dizajn
SIGNAL broj:STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL displej_out:STD_LOGIC_VECTOR(6 DOWNTO 0);

BEGIN

--generisanje stimulusa ulaza u displej
PROCESS
    FILE fajl_ulazni_hex : TEXT OPEN READ_MODE IS "hexulaz.txt";
    FILE fajl_ulazni_time : TEXT OPEN READ_MODE IS "vreme.txt";
    VARIABLE line_id: LINE;
    VARIABLE t : TIME;
    VARIABLE tmp : STD_LOGIC_VECTOR(3 DOWNTO 0);

    BEGIN
        if ((not endfile(fajl_ulazni_hex))AND(not endfile(fajl_ulazni_time))) then--ako nismo stigli do kraja
            READLINE(fajl_ulazni_hex,line_id);
            HREAD(line_id,tmp);
            READLINE(fajl_ulazni_time,line_id);
            READ(line_id,t);
            broj<=tmp;
            wait for t;
        else
            --difolt vrednost stimulusa
            broj<=X"0";
            wait for 5ns;
        end if;
    END PROCESS;
```

```

--instanciranje komponente displeja
displej_inst: displej
PORT MAP
(
    broj => broj,
    displej => displej_out
);
END shema;

```

Sadržaj fajla *hexulaz.txt* je:

```

0
5
A
F
7

```

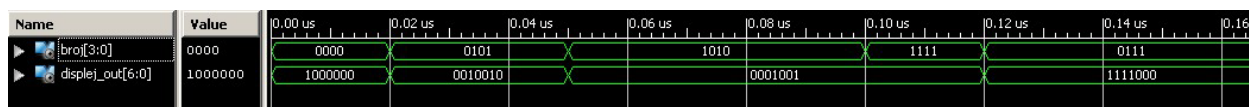
dok je struktura fajla *vreme.txt*:

```

20ns
30ns
50ns
20ns
50ns

```

Na slici 4.5.1 je prikazan deo rezultata simulacije za ovaj primer. Napomenimo još da se mogao koristiti i samo jedan fajl u koji bi bili učešljani sadržaji fajlova *hexulaz.txt* (sadržaj ovog fajla u neparne linije) i *vreme.txt* (sadržaj ovog fajla u parne linije).



Slika 4.5.1. – Prikaz rezultata simulacije displeja

4.6 Upis generisanih vrednosti u fajl

Osim posmatranja vremenskog prikaza rezultata simulacije, poželjno je često i sačuvati ostvarene vrednosti u simulaciji za kasniju analizu u drugim programskim jezicima poput C, C++, Matlaba i dr. Takođe, umesto da se radi komparacija sa očekivanim vrednostima u okviru simulacije, komparacija se može uraditi i van simulacije tako što se rezultati simulacije sačuvaju u tekstualnom fajlu ili fajlovima, pa se potom oni porede sa očekivanim vrednostima koristeći neki drugi programski jezik poput na primer C jezika.

Za upis vrednosti u fajlove se koristi paket *textio* biblioteke *std*. U okviru ovog paketa je definisan upis u fajlove za tipove podataka definisanih u okviru *std* biblioteke (INTEGER, BIT, BIT_VECTOR, REAL, TIME, BOOLEAN, CHARACTER i STRING). Ovaj paket se mora pozvati upotrebom USE instrukcije. Ako se želi izvršiti upis vrednosti tipova STD_LOGIC, STD_ULONGIC, STD_LOGIC_VECTOR i STD_ULONGIC_VECTOR tada se dodatno mora koristiti i paket *std_logic_textio* biblioteke *ieee*.

Fajl koji se želi otvoriti za upis se deklarise sa:

```
FILE promenjiva : TEXT OPEN WRITE_MODE IS "naziv_fajla.txt";
```

ili

```
FILE promenjiva : TEXT OPEN APPEND_MODE IS "naziv_fajla.txt";
```

gde je *promenjiva* naziv koji je dodeljen fajlu za pozive unutar VHDL koda, a *naziv_fajla.txt* označava naziv fajla koji se otvara. WRITE_MODE označava da se stari sadržaj fajla briše, dok APPEND_MODE označava da se upisani (novi) sadržaj dodaje na već postojeći sadržaj u fajlu. Ako fajl sa navedenim nazivom ne postoji, biće kreiran. Ako se fajl ne nalazi na istoj lokaciji gde i projekat, onda treba navesti i relativnu ili apsolutnu lokaciju fajla pored njegovog naziva. Fajl se tumači kao niz stringova, gde svaka linija u fajlu označava jedan string. Da bi se upisala jedna linija u fajl koristi se procedura WRITELINE:

```
WRITELINE(f, l);
```

gde je *f* naziv dodeljen fajlu unutar VHDL koda, a *l* naziv promenljive tipa LINE.

Procedura WRITE omogućava upis vrednosti željenog signala (koji može da bude bilo koji od tipova podataka navedenih na početku ove sekcije) u promenljivu tipa LINE:

```
WRITE (l, s, j, d);
```

gde je *l* naziv promenljive tipa LINE u koju smo učitali liniju iz fajla, *s* naziv signala čiju vrednost upisujemo, *j* definiše poravnanje upisane vrednosti (LEFT ili RIGHT, pri čemu je difolt vrednost RIGHT), a *d* predstavlja minimalan broj karaktera za upisanu vrednost signala (difolt vrednost je 0). Ako je broj karaktera koji se ispisuje manji od minimalnog broja definisanog argumentom *d*, tada se ispis dopunjuje brojem praznih karaktera (*space*) tako da ukupan broj ispisanih karaktera odgovara vrednosti argumenta *d*. Argumenti *j* i *d* se mogu izostaviti. Za upis vrednosti STD_LOGIC_VECTOR i STD_ULONGIC_VECTOR u heksadecimalnom zapisu na raspolaganju je procedura HWRITE, a u oktalnom zapisu procedura OWRITE. Argumenti ovih procedura su identični argumentima procedure WRITE.

Kreirajmo testbenč koji će vršiti simulaciju dekadnog brojača i izvršimo ispis generisanih vrednosti brojača:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR
USE ieee.std_logic_textio.all;--za rad sa fajlovima
USE std.textio.all;
--testbench za dekadni brojac
ENTITY testbench IS
END testbench;

ARCHITECTURE shema OF testbench IS

--deklaracija komponente dekadnog brojaca
COMPONENT dekadni_brojac IS
PORT
(
    clk:    IN STD_LOGIC;--signal takta
    reset:  IN STD_LOGIC;--asinhroni signal reseta
    inc:    IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
    q:      OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojaca
);
END COMPONENT;
--signali koje moramo da generisemo da bi testirali dizajn
SIGNAL clk:STD_LOGIC;
SIGNAL reset:STD_LOGIC;
```

```

SIGNAL inc:STD_LOGIC;
SIGNAL q_out:STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
--generisanje stimulusa takta
PROCESS
BEGIN
--generisanje takta od 100MHz (perioda 10ns) - 5ns '1', i 5ns '0'
loop
    clk<='0';
    wait for 5ns;
    clk<='1';
    wait for 5ns;
end loop;
END PROCESS;
--generisanje stimulusa reseta
PROCESS
BEGIN
    reset<='1';--reset za inicijalizaciju dizajna
    wait for 15ns;--aktivna vrednost reseta traje 15ns, posle se obori na '0'
    reset<='0';
    wait for 200ns;
    reset<='1';--reset dok je inkrement aktivan
    wait for 40ns;
    reset<='0';
    wait for 200ns;
    reset<='1';--reset dok je inkrement neaktivan
    wait for 40ns;
    loop--neophodno da bi zadržali konstantno reset na '0'
        reset<='0';
        wait for 5ns;
    end loop;
END PROCESS;
--generisanje stimulusa inkrementa
PROCESS
BEGIN
    inc<='1';
    wait for 150ns;
    inc<='0';--da bi proverili da li se brojca zamrzne kad je inkrement neaktivan
    wait for 30ns;--pauza inkrementa u trajanju od 30ns
    inc<='1';
    wait for 150ns;
    inc<='0';--pauza inkrementa u trajanju od 300ns
    wait for 300ns;
    loop--neophodno da bi zadržali konstantno inc na '1'
        inc<='1';
        wait for 5ns;
    end loop;
END PROCESS;
--proces za ispis rezultata simulacije
PROCESS(clk)
    FILE fajl_izlaz : TEXT OPEN WRITE_MODE IS "rezultat.txt";
    VARIABLE line_id : LINE;
    VARIABLE tmp: STD_LOGIC_VECTOR(3 DOWNTO 0);
    VARIABLE t:TIME;
BEGIN
    IF(clk'EVENT and clk='1')THEN
        tmp:=q_out;
        t:=NOW;
        WRITE(line_id,t);
        WRITELINE(fajl_izlaz,line_id);

```

```

        WRITE(line_id,tmp);
        WRITELINE(fajl_izlaz,line_id);
    END IF;
END PROCESS;
--instanciranje komponente brojaca
brojac_inst: dekadni_brojac
PORT MAP
(
    clk => clk,
    reset => reset,
    inc => inc,
    q => q_out
);
END shema;

```

Ključna reč NOW uzima trenutno vreme iz simulacije. U izlazni fajl *rezultat.txt* se ispisuju u neparnim linijama trenuci simulacije, a u parnim linijama vrednosti izlaza brojača u tim trenucima. Deo sadržaja ovog fajla je:

```

5 ns
0000
15 ns
0000
25 ns
0001
35 ns
0010
45 ns
0011
55 ns
0100

```

Ako bi liniju

```
WRITE(line_id,tmp);
```

zamenili sa

```
HWRITE(line_id,tmp);
```

tada bi vrednosti brojača bile ispisane u heksadecimalnom formatu i sadržaj fajla *rezultat.txt* izgledao (prikazan je deo sadržaja):

```

5 ns
0
15 ns
0
25 ns
1
35 ns
2
45 ns
3
55 ns
4

```

Ako želimo da razdvojimo ispis vremenskih trenutaka i vrednosti brojača, tada bi VHDL kod procesa za ispis navedenih vrednosti u tekstualne fajlove bio (ostatak testbenča je isti kao u početnom primeru datom u ovoj sekciji):

```

--proces za ispis rezultata simulacije
PROCESS(clk)
    FILE fajl_izlaz_q : TEXT OPEN WRITE_MODE IS "rezultat_q.txt";
    FILE fajl_izlaz_t : TEXT OPEN WRITE_MODE IS "rezultat_t.txt";
    VARIABLE line_id : LINE;
    VARIABLE tmp: STD_LOGIC_VECTOR(3 DOWNTO 0);
    VARIABLE t: TIME;

BEGIN
    IF (clk'EVENT and clk='1') THEN
        tmp:=q_out;
        t:=NOW;
        WRITE(line_id,t);
        WRITELINE(fajl_izlaz_t,line_id);
        WRITE(line_id,tmp);
        WRITELINE(fajl_izlaz_q,line_id);
    END IF;
END PROCESS;

```

Fajl *rezultat_t.txt* bi sadržavao vremenske trenutke, a *rezultat_q* bi sadržavao vrednosti brojača. Često je zgodno da se umesto vremenskih trenutaka (ili uz njih) generišu redni brojevi ciklusa takta. Tada bi proces za generisanje vrednosti koje se ispisuju u fajl bio (ostatak testbenča je isti kao u početnom primeru datom u ovoj sekciji):

```

--proces za ispis rezultata simulacije
PROCESS(clk)
    FILE fajl_izlaz : TEXT OPEN WRITE_MODE IS "rezultat.txt";
    VARIABLE line_id : LINE;
    VARIABLE tmp: STD_LOGIC_VECTOR(3 DOWNTO 0);
    VARIABLE t: TIME;
    VARIABLE cnt: INTEGER:=0;

BEGIN
    IF (clk'EVENT and clk='1') THEN
        tmp:=q_out;
        WRITE(line_id,cnt);
        WRITELINE(fajl_izlaz,line_id);
        WRITE(line_id,tmp);
        WRITELINE(fajl_izlaz,line_id);
        cnt:=cnt+1;
    END IF;
END PROCESS;

```

Za brojanje ciklusa takta koristi se brojač *cnt*, pri čemu je uzeto da se prvi ciklus takta indeksira sa 0. Ako bi se htelo da se prvi ciklus takta indeksira sa 1, tada bi bilo dovoljno pomeriti liniju u kojoj se inkrementira varijabla *cnt* tako da ona bude prva linija koja se izvršava na uzlaznu ivicu takta u okviru ovog procesa. Deo sadržaja kreiranog fajla *rezultat.txt* je:

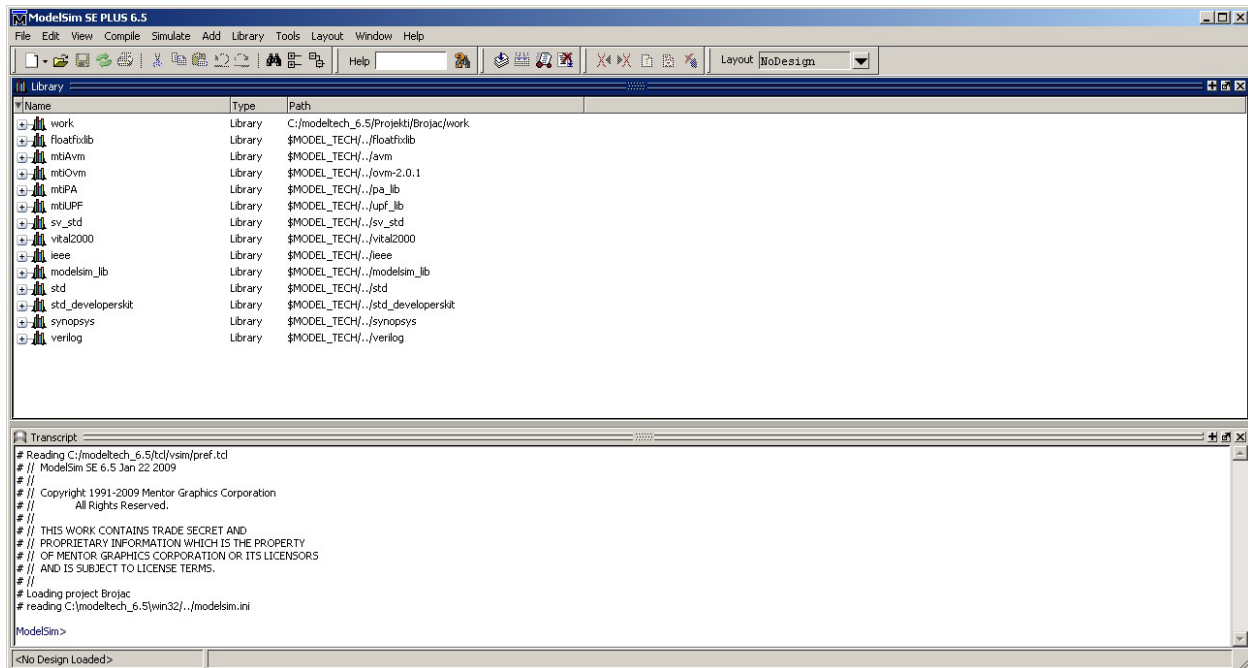
```

0
0000
1
0000
2
0001
3
0010
4
0011
5
0100

```

4.7 ModelSim

ModelSim alat predstavlja jedan od najpopularnijih softverskih paketa za simulaciju shema opisanih HDL jezikom (VHDL i Verilog). Xilinx-ov ISim simulator je zasnovan na ModelSim simulatoru i stoga je veoma sličan njemu. U ovom potpoglavlju će biti ukratko izložene najosnovnije opcije ModelSim simulatora koje omogućavaju simulaciju dizajna pisanog VHDL jezikom. Za sve naprednije opcije može se konsultovati pomoćna literatura dostupna u okviru ModelSim softverskog paketa.



Slika 4.7.1. – ModelSim aplikacija kada nije otvoren projekat

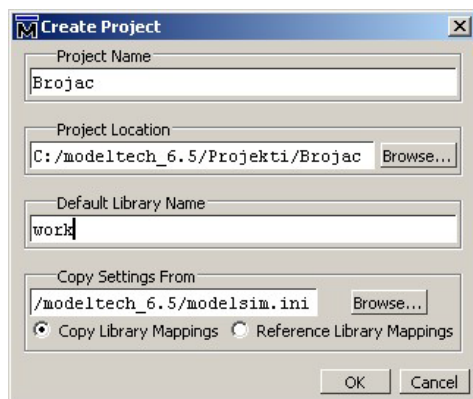
Izgled prozora ModelSim aplikacije za slučaj kada nema otvorenih projekata je prikazan na slici 4.7.1. Kao što vidimo u okviru ove aplikacije postoji glavni meni, red izvučenih ikonica, centralni deo prozora, i donji deo prozora.

Glavni meni sadrži grupisane opcije ModelSim simulatora. **File** grupa sadrži opcije za kreiranje novog izvorišnog fajla, za kreiranje, otvaranje i zatvaranje projekata i dr. **Edit** grupa sadrži opcije za kopiranje, brisanje, pretragu u okviru otvorenih izvorišnih fajlova, kao i druge srodne opcije. **View** grupa daje mogućnost aktiviranja i deaktiviranja različitih prikaza poput vremenskog prikaza rezultata simulacije, simuliranih objekata i dr. **Compile** grupa sadrži komande za aktivaciju kompajliranja projekta ili pojedinih fajlova u projektu, kao i postavljanje parametara kompajliranja. Ovde se pod kompajliranjem podrazumeva samo analiza izvorišnih fajlova da bi se dobila netlista koja se koristi za potrebe simulacije. **Simulate** grupa opcija se koristi za pokretanje procesa simulacije. **Add** grupa opcija se koristi za razna dodavanja unutar simulatora, na primer, dodavanje signala u vremenski prikaz pokrenute simulacije radi posmatranja njihovih vrednosti i sl. **Tools** grupa sadrži pomoćne alate poput komparatora vremenskih dijagrama signala. **Layout** grupa sadrži opcije za podešavanje izgleda prozora aplikacije, kao i snimanje i učitavanje željenog izgleda prozora aplikacije. Pod izgledom se

podrazumeva raspored potprozora unutar aplikacije. **Window** grupa sadrži opcije za kontrolu prikaza prozora. **Help** grupa omogućava pristup pomoćnoj dokumentaciji ModelSim aplikacije.

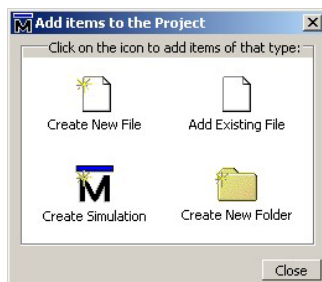
Red izvučenih ikonica sadrži ikonice preko kojih se pristupa najčešće korišćenim komandama ModelSim simulatora. U centralnom delu prozora je prikazana samo lista dostupnih biblioteka kada projekat nije otvoren. U okviru centralnog dela prozora se po defaultu otvaraju svi ostali potprozori (kada se aktivira njihov prikaz), poput sadržaja projekta, vremenskog prikaza rezultata simulacije i sl. Takođe, kada se aktivira simulacija ili editor za prikaz sadržaja nekog izvorišnog fajla centralni deo prozora se deli na dve polovine. Leva polovina sadrži prikaz biblioteka ili projekta u zavisnosti od izabranog taba (u glavnom meniju se dobija **Library** ili **Project** grupa opcija u zavisnosti od izabranog taba), a desna polovina ostale potprozore, poput editora izvorišnog fajla ili prikaza vremenske simulacije, pri čemu i tu postoje tabovi za izbor prikaza ako je istovremeno otvoreno više potprozora. Takođe, u zavisnosti od otvorenih potprozora se delimično menja i red sa izvučenim ikonicama tako da se uvek dobiju one ikonice koje odgovaraju otvorenim potprozorima.

U donjem delu se nalazi konzola koja daje ispis obaveštenja, ali pruža i mogućnost za tekstualni upis komandi za aktivaciju željenih akcija poput, na primer, pokretanja procesa simulacije.



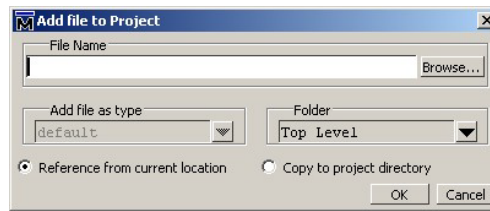
Slika 4.7.2. – Unos naziva i lokacije kreiranog projekta

Da bi kreirali novi projekat, bismo opciju **File->New->Project** iz glavnog menija čime se otvara prvi prozor dijaloga za kreiranje projekta koji je prikazan na slici 4.7.2. U okviru ovog prozora se unosi naziv kreiranog projekta, kao i njegova lokacija. Kada se unesu traženi podaci, otvara se sledeći prozor u dijalogu prikazan na slici 4.7.3, koji omogućava kreiranje izvorišnih fajlova, kao i dodavanje izvorišnih fajlova. Naravno, uvek je moguće i kasnije dodati izvorišne fajlove ili ih kreirati.



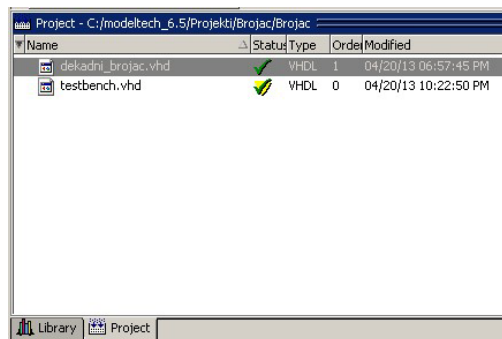
Slika 4.7.3. – Prozor za dodavanje ili kreiranje izvorišnih fajlova

Nakon toga, se kreira projekat i pojavljuje se tab za izbor prikaza projekta (**Project**), koji se nalazi uz tab za prikaz biblioteka (**Library**). Naknadno dodavanje fajlova je moguće kada se izabere tab **Project**, a potom iz glavnog menija opcija **Project->Add to Project->Existing File**, čime se otvara prozor za brauzovanje do željenog fajla ili fajlova (slika 4.7.4).



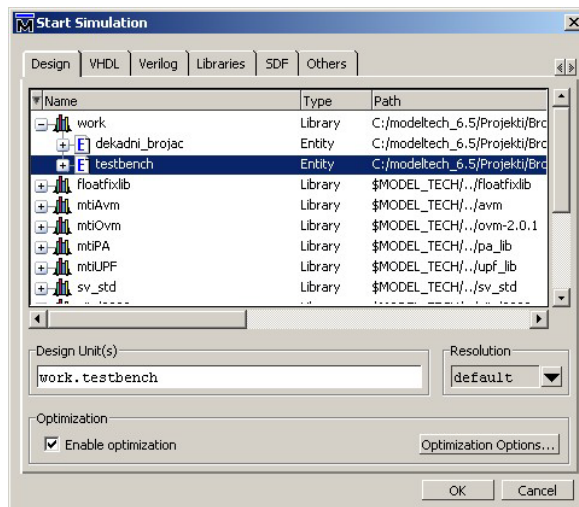
Slika 4.7.4. – Naknadno dodavanje izvorišnih fajlova

Nakon što su dodati fajlovi u projekat, neophodno je kompajlirati projekat. Kompajliranje svih fajlova u projektu se pokreće komandom **Compile->Compile All** iz glavnog menija. Takođe, mogu se selektovati samo neki fajlovi i za njih pokrenuti kompajliranje opcijom **Compile->Compile Selected** iz glavnog menija. Opcija **Compile->Compile Order** iz glavnog menija daje mogućnost manuelnog podešavanja redosleda kojim će fajlovi iz projekta biti kompajlirani, ako nismo zadovoljni redosledom koji je preporučio ModelSim.



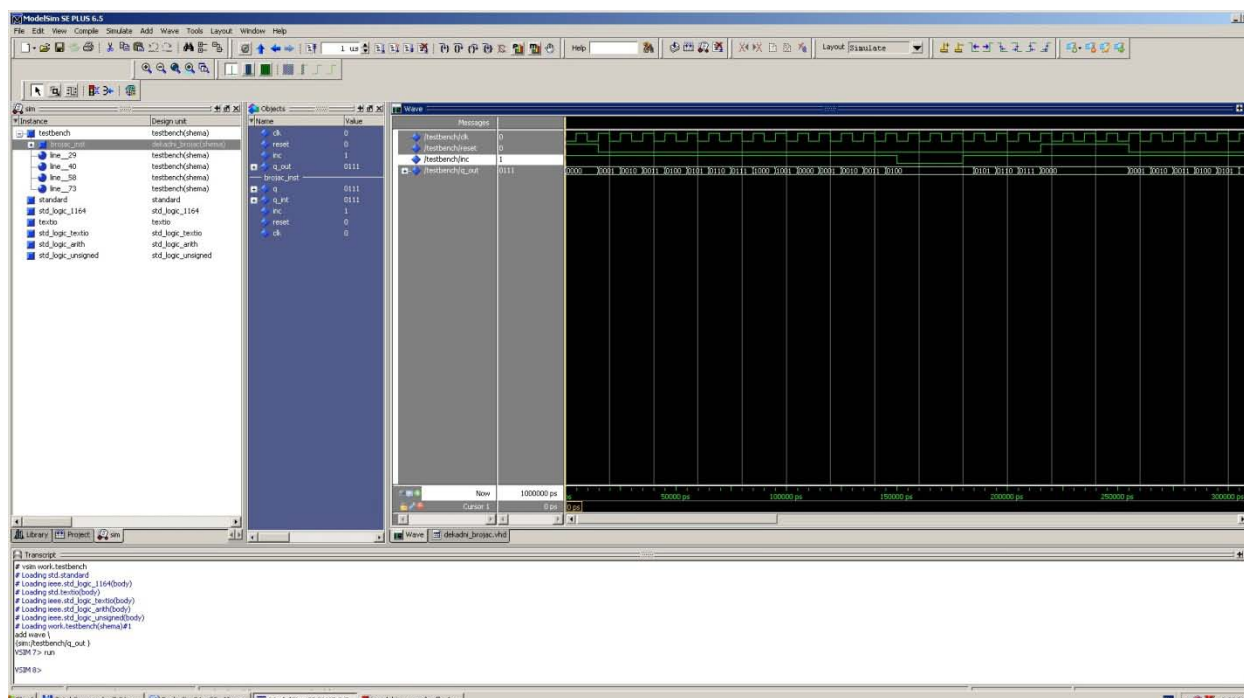
Slika 4.7.5. – Lista fajlova projekta kada je izabran Project tab

Dodajmo VHDL fajlove za dekadni brojač i testbenč definisanih u prvom primeru iz sekcije 4.2. I potom uradimo kompajliranje kompletnog projekta sa komandom **Compile->Compile All** iz glavnog menija. Uspešno kompajliranje nekog fajla je označeno kvačicom u koloni status (pre kompajliranja se nalazi simbol upitnika), kao što se vidi sa slike 4.7.5. Pokrenimo simulaciju sa **Simulate->Start Simulation** iz glavnog menija. Tada se otvara prozor prikazan na slici 4.7.6, u kome treba izabrati entitet koji želimo da simuliramo. Izabraćemo entitet *testbench* jer je on top level entitet za simulaciju. Drugi način za pokretanje simulacije je kucanje komande **vsim work.testbench** u konzoli čime se pokreće simulacija testbenč entiteta. Komanda **vsim** pokreće simulator i potrebno je iza ove komande definisati naziv entiteta koji se simulira, a to je entitet *testbench* koji se nalazi u okviru *work* biblioteke. Komanda **vsim** se može pokrenuti sa različitim parametrima, a njihov opis se nalazi u pomoćnoj dokumentaciji dostupnoj preko **Help** grupe opcija iz glavnog menija. Napomenimo da jedan od parametara (**-t**) definiše rezoluciju simulatora, pri čemu se vrednost rezolucije navodi kao 1, 10 ili 100 vremenskih jedinica (fs, ps, ns, us, ms, sec). Na primer, komanda **vsim -t 10ns work.testbench**, generiše simulaciju testbenč entiteta sa rezolucijom od 10ns. Difolt rezolucija je 1ns.



Slika 4.7.6. – Izbor entiteta koji će se simulirati

Na slici 4.7.7 je prikazan izgled prozora kad je aktivirana i izvršena simulacija testbenča za dekadni brojč.



Slika 4.7.7. – Prikaz rezultata simulacije u ModelSim aplikaciji

Na levoj strani centralnog dela prozora se pojavio tab **sim** koji predstavlja aktivnu simulaciju. Izborom ovog taba dobijamo hijerarhijsku strukturu prikaza projekta, odnosno entiteta u projektu. Tada se u glavnom meniju pojavljuje **Wave** grupa opcija. U srednjoj koloni (**Objects** potprozor) se nalazi prikaz signala za izabrane entitete. *Drag & Drop* principom mogu se dodavati signali iz srednje kolone u potprozor za vremenski prikaz simulacije koji se nalazi sa desne strane. Drugi način je da se selektuju željeni signali, a potom aktivira opcija **Add ->To Wave->Selected Signals** iz glavnog menija. Sa desne strane se nalazi vremenski prikaz simulacije koji je veoma sličan onome u Isim simulatoru. Desnim klikom na neki signal u ovom

potprozoru se dobija meni u okviru koga se može izabrati i format prikaza vrednosti signala (**Radix** opcija) poput binarnog, heksadecimalnog i dr. U redu izvučenih ikonica se nalaze ikonice za pokretanje ili restartovanje simulacije, za definisanje trajanja simulacije kada se pokrene, za dodavanje i brisanje markera, za nalaženje prethodne ili sledeće tranzicije selektovanog signala u vremenskom prikazu, za nalaženje prethodne/sledeće uzlazne/silazne ivice (opcije za nalaženje su dostupne i preko opcije **Edit->Signal Search** iz glavnog menija). Napomenimo još da u slučaju da je samo **Wave** potprozor otvoren sa desne strane tada neće biti tabova sa desne strane, u suprotnom će tabovi postojati i **Wave** tab označava izbor prikaza **Wave** potprozora koji sadrži vremenski prikaz rezultata simulacije.

```

Ln#
16 ARCHITECTURE shema OF dekadni_brojac IS
17   SIGNAL q_int: STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";--interno stanje brojac
18 BEGIN
19
20   --proces koji definise brojanje
21   PROCESS(reset,clk)
22   BEGIN
23     IF(reset='1') THEN--ako je asinhroni reset aktivan
24       q_int<=(OTHERS=>'0');--brojac resetujemo na 0
25     ELSIF (clk'EVENT and clk='1') THEN--uzlazna ivica takta
26       IF(inc='1') THEN--ako je kontrolni signal za dozvolu brojanja aktivan kreni da brojis
27         IF(q_int = "1001") THEN -- ako smo dosli do 9 sledeca vrednost brojac
28           q_int<=(OTHERS=>'0');
29         ELSE -- u suprotnom inkrementiramo brojac za 1
30           q_int<=q_int+'1';
31         END IF;-- end if (q_int = "1001")
32       END IF;-- end if inc='1'
33     END IF;-- end if clk
34   END PROCESS;
35   -- prosledjivanje internog stanja brojac na izlaz entiteta
36   --da je q definisan u modu buffer, tada q_int ne bi bio neophodan
37   q<=q_int;
38 END shema;
39
40
41 --LIBRARY ieee;
42 --USE ieee.std_logic_1164.all;--neophodno jer radimo sa STD_LOGIC i STD_LOGIC_VECTOR
43 --
44 ---dekadni brojac koji kružno broji unapred 0 do 9
45 --ENTITY dekadni_brojac IS
46 --PORT
47 --{
48 -- clk:  IN STD_LOGIC;--signal takta
49 -- reset: IN STD_LOGIC;--asinhroni signal reseta
50 -- inc:  IN STD_LOGIC;--kontrolni signal za dozvolu brojanja
51 -- q:   OUT STD_LOGIC_VECTOR(3 DOWNTO 0)--vrednost brojac
52 --};

```

Slika 4.7.8. – Aktivna *breakpoint* tačka

U okviru VHDL koda mogu se dodati tzv. *breakpoint* tačke, u kojima se zaustavlja simulacija kada simulator naiđe na njih. Uloga *breakpoint* tačaka je ista kao i u drugim programskim jezicima (C, C++), odnosno, one se koriste za debugovanje dizajna. Na ovaj način možemo utvrditi da li se prolazi kroz pojedine delove koda ili možemo da zaustavimo proces simulacije kada se krene sa izvršavanjem instrukcije na koju je stavljena *breakpoint* tačka i da ispitamo vrednosti svih signala radi utvrđivanja izvora greške i sl. *Breakpoint* tačke se dodaju u editoru VHDL fajlova, koji se aktivira prostim dvostrukim klikom na entitet čiji VHDL kod želimo otvoriti u okviru prikaza fajlova kada je izabran **Project** tab sa leve strane. Dodavanje se vrši tako što se klikne na kolonu sa brojevima linija, pri čemu se klikne na liniju na koju želimo da dodamo *breakpoint* tačku. Linije za koje je moguće postaviti *breakpoint* tačku su označene crvenom bojom. Crvena tačka označava aktivnu *breakpoint* tačku (slika 4.7.8), a klikom na tu tačku ona postaje plava čime je označeno da je *breakpoint* tačka neaktivna. Ponovnim klikom na tačku se vrši njena ponovna aktivacija (i postaje ponovo crvena). Ako želimo da obrišemo *breakpoint* tačku onda desnim klikom otvaramo meni i biramo opciju **Remove Breakpoint**. Kada se aktivira simulacija sa aktivnim *breakpoint* tačkama, simulator će se zaustaviti svaki put kada na njih naiđe (slika 4.7.9), a da bi se simulacija nastavila potrebno je izabrati opciju

Simulate->Run->Continue iz glavnog menija ili odgovarajuću ikonicu iz reda ikonica ispod glavnog menija (druga ikonica sa desne strane od polja za unos trajanja simulacije).

```
23 IF(reset='1') THEN--ako je asinhroni reset aktivan
24   q_int<=(OTHERS=>'0');--brojac resetujemo na 0
25 ELSIF(clk'EVENT and clk='1') THEN--uzlazna ivica takta
26   IF(inc='1') THEN--ako je kontrolni signal za dozvolu brojanja akt:
27     IF(q_int = "1001") THEN -- ako smo dosli do 9 sledeca vrednost }
28     q_int<=(OTHERS=>'0');
29     ELSE -- u suprotnom inkrementiramo brojac za 1
30     q_int<=q_int+'1';
```

Slika 4.7.9. – Simulacija zaustavljena u *breakpoint* tački

Simulacija se zatvara komandom **Simulate->End Simulation** iz glavnog menija. ModelSim je veoma sličan ISim simulatoru pa uglavnom upotrebu ModelSim alata nije teško savladati ako je pre toga korišćen ISim simulator. ModelSim možemo posmatrati i kao znatno moćniju verziju ISim simulatora jer nudi veći broj opcija, ali pre svega nudi i bolje performanse sa stanovišta brzine simulacije i potrošnje memorije računara što je naročito izraženo u slučaju simulacije veoma kompleksnog dizajna. Takođe, važna napomena je da u slučaju dizajna koji sadrži megafunkcije iz ISE ili Quartus okruženja, neophodno je u ModelSim dodati i kompajlirati odgovarajuće biblioteke neophodne za te megafunkcije da bi simuliranje takvog dizajna bilo moguće unutar ModelSim alata. Otuda se nude i verzije ModelSim alata sa već prekompajliranim bibliotekama koje su neophodne za megafunkcije iz ISE ili Quartus okruženja (ModelSim Xilinx Edition ili ModelSim Altera Edition).