1η Εργαστηριακή Άσκηση

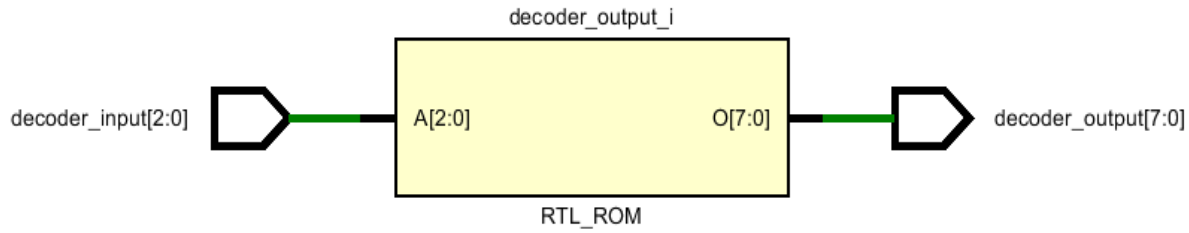
VLSI

Παντελαίος Δημήτριος Α.Μ.: 03118049

Μοίρας Αλέξανδρος Α.Μ.: 03118081

Ζήτημα Α.2.:

Το RTL σχηματικό του δυαδικού αποκωδικοποιητή 3 σε 8 είναι:



Πρόκειται για ένα απλό συνδυαστικό κύκλωμα που ενεργοποιεί διαφορετική έξοδο για κάθε διαφορετικό συνδυασμό των 3 εισόδων.

Ο κώδικας VHDL για την dataflow αρχιτεκτονική:

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

entity decoder_dataflow is
    port(
           decoder_input : in  std_logic_vector(2 downto 0);
           decoder_output : out std_logic_vector(7 downto 0)
          );
end entity;

architecture dataflow_arch of decoder_dataflow is
begin
  with decoder_input select decoder_output <=
    "00000001" when "000",
    "00000010" when "001",
    "00000100" when "010",
    "00001000" when "011",
    "00010000" when "100",
    "00100000" when "101",
    "01000000" when "110",
    "10000000" when "111",
    "--------" when others;
end architecture;
```

Ο κώδικας VHDL για την behavioral αρχιτεκτονική:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity decoder_behavioral is
    port(
           decoder_input : in  std_logic_vector(2 downto 0);
           decoder_output : out std_logic_vector(7 downto 0)
          );
end entity;

architecture behavioral_arch of decoder_behavioral is

begin
  MUX_LOGIC : process(decoder_input)
  begin
    case decoder_input is
      when "000" =>
        decoder_output <= "00000001";
      when "001" =>
        decoder_output <= "00000010";
      when "010" =>
        decoder_output <= "00000100";
      when "011" =>
        decoder_output <= "00001000";
      when "100" =>
        decoder_output <= "00010000";
      when "101" =>
        decoder_output <= "00100000";
      when "110" =>
        decoder_output <= "01000000";
      when "111" =>
        decoder_output <= "10000000";
      when others =>
        decoder_output <= (others => '-');
    end case;
  end process;
end architecture ; -- arch
```

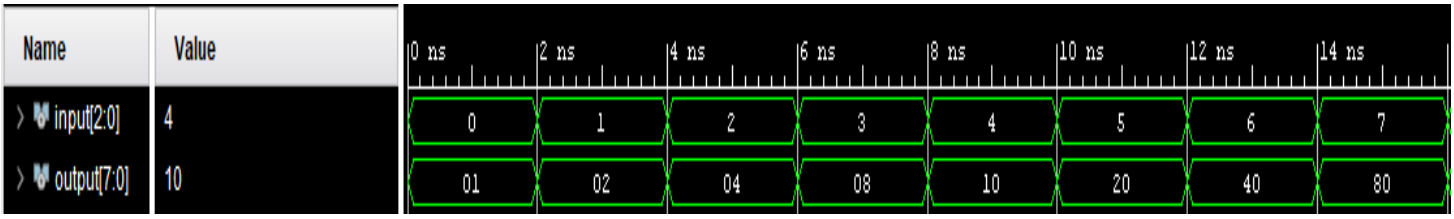
Για την επιλογή της κατάλληλης εξόδου στην dataflow αρχιτεκτονική χρησιμοποιήθηκε η εντολή with select when, ενώ στην behavioral περιγραφή χρησιμοποιήθηκε ένα process, το οποίο με κάθε αλλαγή της εισόδου έδινε την αντίστοιχη έξοδο με την χρήση της εντολής case.

Ο κώδικας VHDL για το testbench της dataflow αρχιτεκτονικής:

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

entity decoder_dataflow_testbench is
end decoder_dataflow_testbench;

architecture test of decoder_dataflow_testbench is

    signal input : std_logic_vector(2 downto 0) := (others => '0');
    signal output : std_logic_vector(7 downto 0) := (others => '0');

    component decoder_dataflow is
        port(
               decoder_input : in  std_logic_vector(2 downto 0);
               decoder_output : out std_logic_vector(7 downto 0)
              );
    end component;
begin
    uut: decoder_dataflow
        port map (
        decoder_input => input,
        decoder_output => output
        );

    stimulus: process begin
         input <= "000";
         wait for 2ns;
         input <= "001";
         wait for 2ns;
         input <= "010";
         wait for 2ns;
         input <= "011";
         wait for 2ns;
         input <= "100";
         wait for 2ns;
         input <= "101";
         wait for 2ns;
         input <= "110";
         wait for 2ns;
         input <= "111";
         wait;
    end process;
end architecture;
```

Ο αντίστοιχος κώδικας για την behavioral περιγραφή είναι ο ίδιος με την διαφορά ότι χρησιμοποιείται το component decoder_behavioral αντί για το decoder_dataflow.

Η προσομοίωση που επιβεβαιώνει την ορθή λειτουργία του κυκλώματός μας:

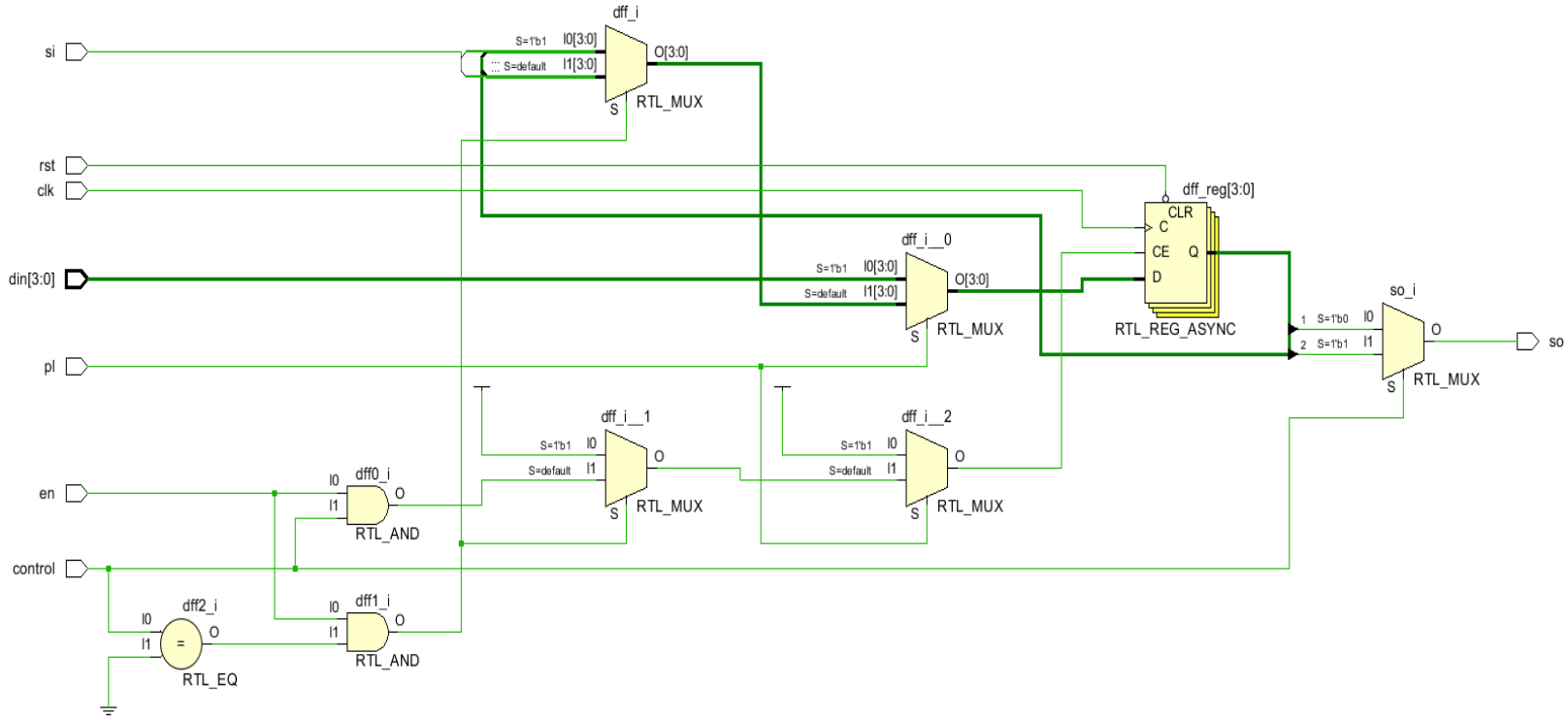


Ζήτημα Β.2.:

Στόχος είναι να περιγραφεί καταχωρητής ολίσθησης των 4 bits με παράλληλη φόρτωση. Το σήμα με την μεγαλύτερη προτεραιότητα είναι το pl, το οποίο όταν τίθεται ίσο με ‘1’, φορτώνεται η παράλληλη είσοδος των 4 bit din. Η ενεργοποίηση της ολίσθησης γίνεται μέσω του σήματος en. Ο καταχωρητής έχει και μια σειριακή είσοδο si, καθώς και μια σειριακή έξοδο so. Προσθέτουμε ένα σήμα που το ονομάζουμε control, το οποίο όταν παίρνει την τιμή ‘0’, έχουμε δεξιά ολίσθηση, ενώ όταν παίρνει την τιμή ‘1’ έχουμε αριστερή ολίσθηση.

Κατά την δεξιά ολίσθηση το περιεχόμενο του καταχωρητή ολισθαίνει κατά μια θέση προς το LSB , το LSB βγαίνει στην έξοδο so και η είσοδος si περνά στο MSB του καταχωρητή. Αντίθετα, κατά την αριστερή ολίσθηση το περιεχόμενο του καταχωρητή ολισθαίνει κατά μια θέση προς το MSB , το MSB βγαίνει στην έξοδο so και η είσοδος si περνά στο LSB του καταχωρητή. Η ανανέωση αυτή γίνεται σε κάθε θετικό παλμό του ρολογιού clk. Τέλος υπάρχει και το σήμα rst, το οποίο όταν παίρνει την τιμή ‘0’, μηδενίζεται το περιεχόμενο του καταχωρητή.

Το RTL σχηματικό του καταχωρητή που περιεγράφη παραπάνω:



Η διαφορά του παραπάνω RTL από το αντίστοιχο του καταχωρητή που υποστηρίζει μόνο δεξιά ολίσθηση είναι ότι υπάρχει ένα επιπλέον σήμα control και δύο πολυπλέκτες οι οποίοι δέχονται ως είσοδο αυτό το σήμα και αποφασίζουν αν θα γίνει δεξιά η αριστερή ολίσθηση και αντίστοιχα αν θα δοθεί στην έξοδο το LSB ή το MSB.

Το RTL σχηματικό του σχήματος 2.8 είναι ορθό και δεν μπορεί να απλοποιηθεί άλλο με το χέρι, καθώς το vivado εφαρμόζει αλγορίθμους ώστε να βρει το απλούστερο δυνατό σχηματικό διάγραμμα στην behavioral περιγραφή.

Ο κώδικας VHDL για την behavioral αρχιτεκτονική:

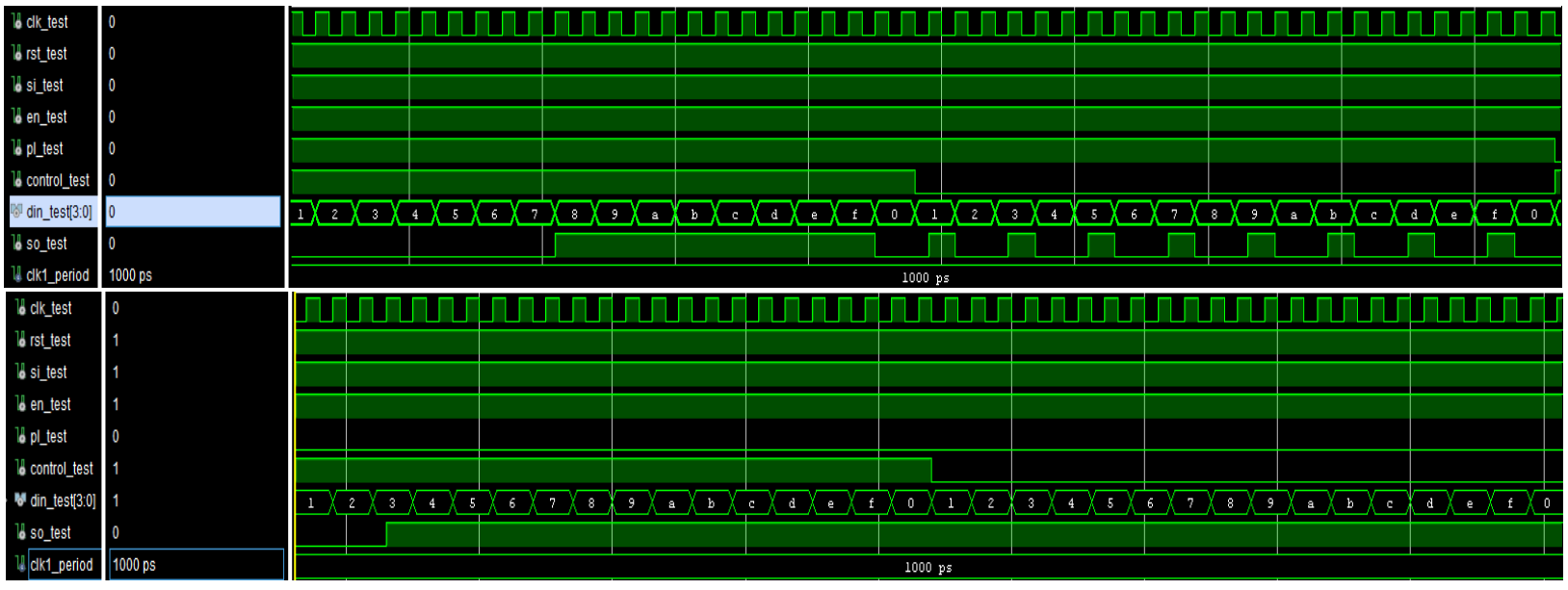
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.ALL;

entity shift_right_left_behavioral is
    port (
    clk,rst,si,en,pl, control: in std_logic;
    din: in std_logic_vector(3 downto 0);
    so: out std_logic);
end shift_right_left_behavioral;

architecture rtl of shift_right_left_behavioral is
    signal dff: std_logic_vector(3 downto 0);
begin
    edge: process (clk,rst)
    begin
    if rst='0' then
        dff<=(others=>'0');
    elsif clk'event and clk='1' then
        if pl='1' then
            dff<=din;
        elsif en='1' and control='0' then
            dff<=si&dff(3 downto 1);
        elsif en='1' and control='1' then
            dff<=dff(2 downto 0)&si;
        end if;
    end if;
    end process;

    with control select so <=
        dff(0) when '0',
        dff(3) when '1',
        '-' when others;
end rtl;
```

Ο κώδικας VHDL για το testbench της behavioral αρχιτεκτονικής:

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

entity shift_right_left_behavioral_testbench is
end shift_right_left_behavioral_testbench;

architecture B2_test of shift_right_left_behavioral_testbench is

    constant clk1_period : time := 1 ns;
    signal clk_test,rst_test,si_test,en_test,pl_test,control_test : std_logic := '0';
    signal din_test : std_logic_vector(3 downto 0) := (others => '0');
    signal so_test : std_logic := '0';

    component shift_right_left_behavioral is
        port (
            clk,rst,si,en,pl, control: in std_logic;
            din: in std_logic_vector(3 downto 0);
            so: out std_logic);
    end component;
begin
    uut: shift_right_left_behavioral
        port map (
            clk => clk_test,
            rst => rst_test,
            si => si_test,
            en => en_test,
            pl => pl_test,
            control => control_test,
            din => din_test,
            so => so_test
        );
```

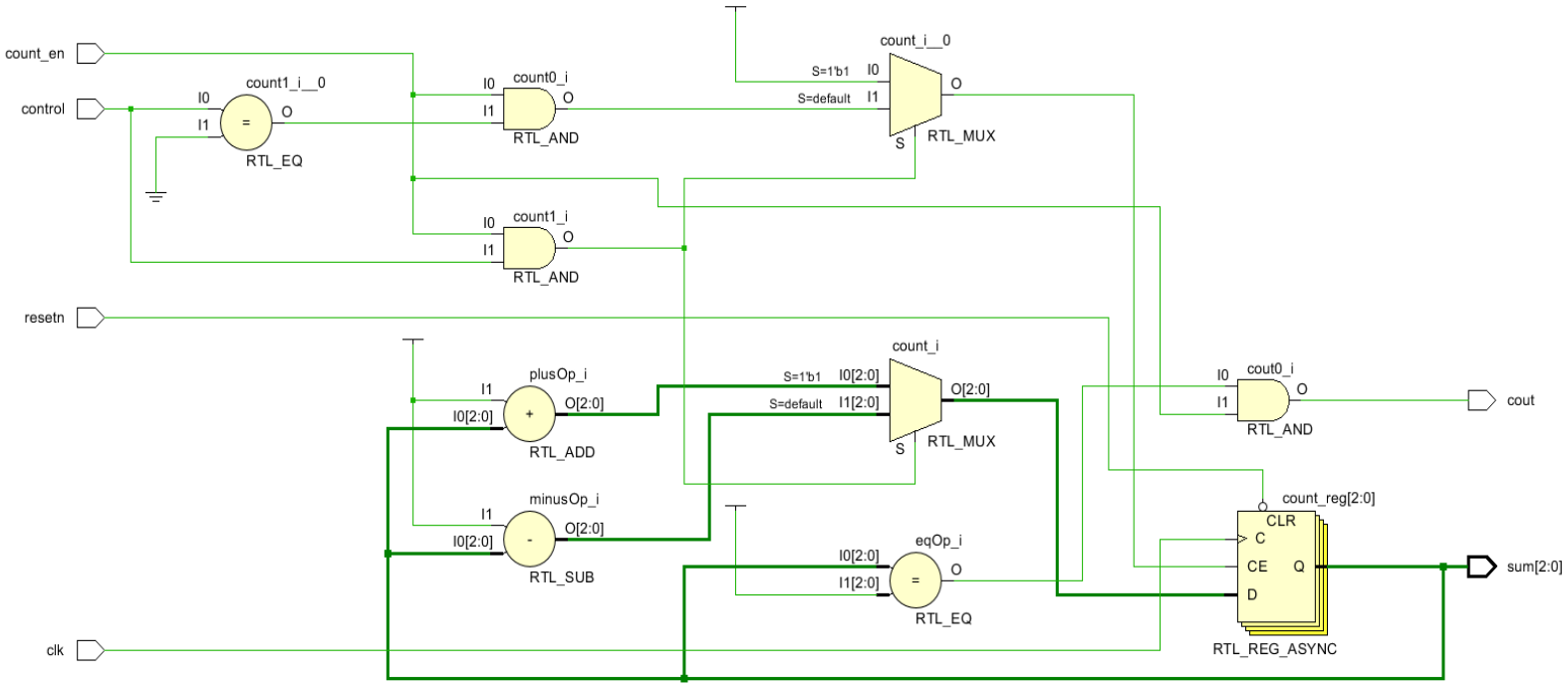
```
    stimulus: process begin
        for i in 0 to 1 loop
            rst_test <= not rst_test;
            for j in 0 to 1 loop
                si_test <= not si_test;
                for k in 0 to 1 loop
                    en_test <= not en_test;
                    for l in 0 to 1 loop
                        pl_test <= not pl_test;
                        for m in 0 to 1 loop
                            control_test <= not control_test;
                            for n in 0 to 15 loop
                                din_test <= std_logic_vector( unsigned(din_test) + 1 );
                                wait for 1500ps;
                            end loop;
                        end loop;
                    end loop;
                end loop;
            end loop;
        end loop;
        wait;
    end process;

    clk1_generator: process begin
            clk_test <= '0';
            wait for clk1_period/2;
            clk_test <= '1';
            wait for clk1_period/2;
    end process;

end architecture;
```

Ένα μέρος της προσομοίωσης που επιβεβαιώνει την ορθή λειτουργία του κυκλώματός μας



Ζήτημα Β.3.:

1. Προσθέτουμε στον δοσμένο μετρητή ένα σήμα control, το οποίο όταν παίρνει την τιμή ‘0’, ο μετρητής μετράει προς τα κάτω, ενώ όταν παίρνει την τιμή ‘1’, μετράει προς τα πάνω.
   Το RTL σχηματικό του μετρητή :



Ο κώδικας VHDL της behavioral αρχιτεκτονικής:

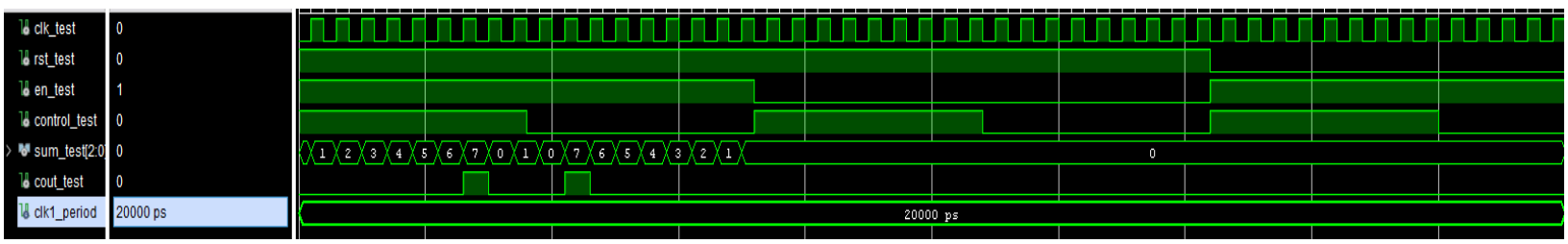
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity up_down_counter_behavioral is
    port( clk,
        resetn,
        count_en : in std_logic;
        sum : out std_logic_vector(2 downto 0);
        control: in std_logic;
        cout : out std_logic);
end;

architecture rtl_nolimit of up_down_counter_behavioral is
signal count: std_logic_vector(2 downto 0);
begin
    process(clk, resetn)
    begin
        if resetn='0' then
            count <= (others=>'0');
        elsif clk'event and clk='1' then
            if count_en='1' and control='1' then
                count<=count+1;
             elsif count_en='1' and control='0' then
                count<=count-1;
            end if;
        end if;
    end process;
    sum <= count;
    cout <= '1' when count=7 and count_en='1' else '0';
end rtl_nolimit;
```

Ο κώδικας VHDL για το testbench της behavioral αρχιτεκτονικής:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity up_down_counter_behavioral_testbench is
end up_down_counter_behavioral_testbench;

architecture B3_test of up_down_counter_behavioral_testbench is

    constant clkl_period : time := 20 ns;
    signal clk_test,rst_test,en_test,control_test : std_logic := '0';
    signal sum_test : std_logic_vector(2 downto 0) := (others => '0');
    signal cout_test : std_logic;
```

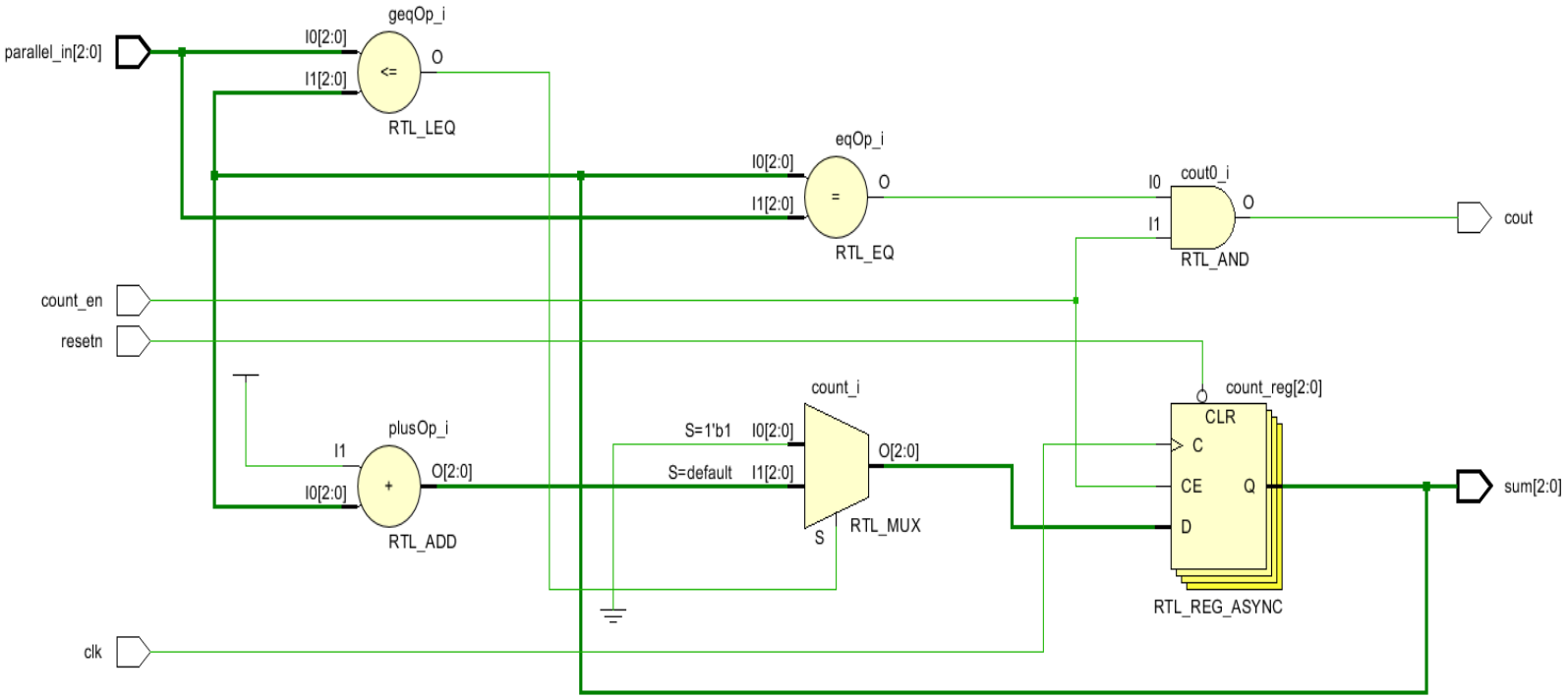
```
    component up_down_counter_behavioral is
    port( clk,
        resetn,
        count_en : in std_logic;
        sum : out std_logic_vector(2 downto 0);
        control: in std_logic;
        cout : out std_logic);
    end component;
begin
    uut: up_down_counter_behavioral
        port map (
            clk => clk_test,
            resetn => rst_test,
            count_en => en_test,
            control => control_test,
            sum => sum_test,
            cout => cout_test
        );

    stimulus: process begin
        for i in 0 to 1 loop
            rst_test <= not rst_test;
            for k in 0 to 1 loop
                en_test <= not en_test;
                for m in 0 to 1 loop
                    control_test <= not control_test;
                    wait for 180ns;
                end loop;
            end loop;
        end loop;
        wait;
    end process;

    clk1_generator: process begin
            clk_test <= '0';
            wait for clk1_period/2;
            clk_test <= '1';
            wait for clk1_period/2;
    end process;

end architecture;
```

Ενα μέρος της προσομοίωσης που επιβεβαιώνει την ορθή λειτουργία του κυκλώματός:



2. Προσθέτουμε στον μετρητή ένα σήμα εισόδου 3 bit parallel_in και στην θετική ακμή του ρολογιού ελέγχουμε αν η τρέχουσα τιμή του μετρητή είναι μεγαλύτερη η ίση από την τιμή αυτής της εισόδου. Σε αυτή την περίπτωση μηδενίζουμε τον μετρητή.

   Το RTL σχηματικό του παραπάνω μετρητή:



Ο κώδικας VHDL της behavioral αρχιτεκτονικής:

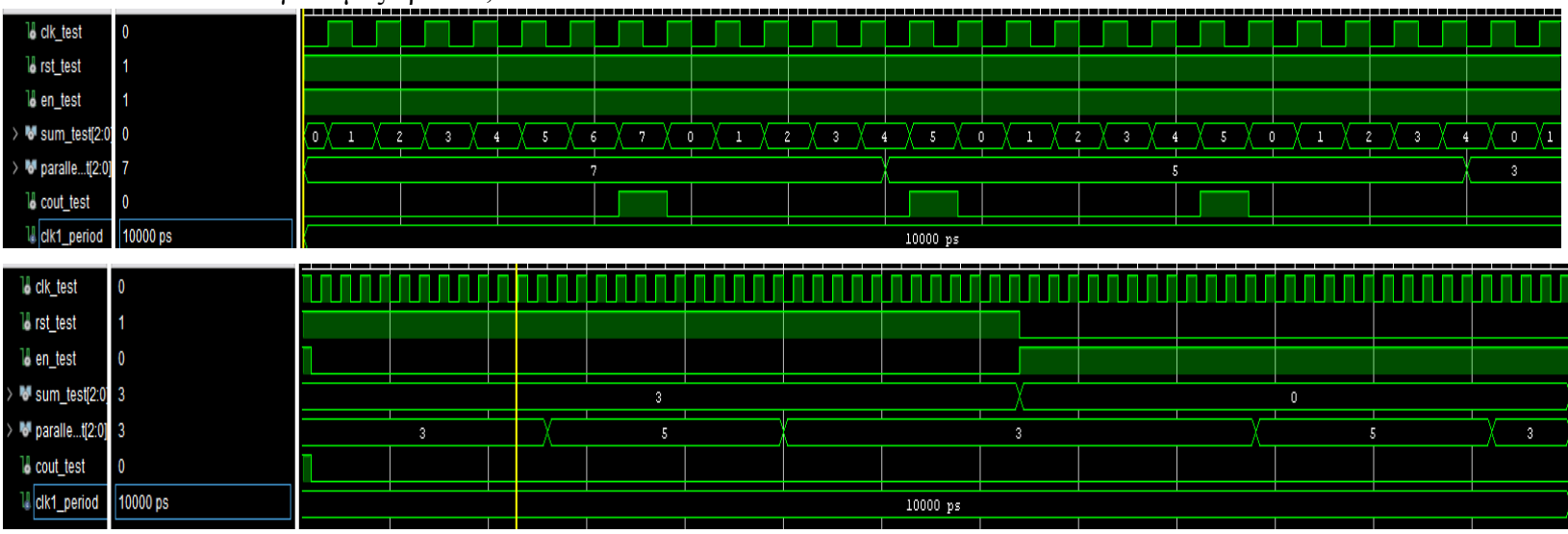
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity counter_with_limit_behavioral is
    port( clk,
        resetn,
        count_en : in std_logic;
        parallel_in : in std_logic_vector(2 downto 0);
        sum : out std_logic_vector(2 downto 0);
        cout : out std_logic);
end;

architecture rtl_limit of counter_with_limit_behavioral is
signal count: std_logic_vector(2 downto 0);
begin
    process(clk, resetn)
    begin
        if resetn='0' then
            count <= (others=>'0');
        elsif clk'event and clk='1' then
            if count_en='1' then
                if count>=parallel_in then
                    count<=(others=>'0');
                else
                    count <= count + 1;
                end if;
            end if;
        end if;
    end process;
    sum <= count;
    cout <= '1' when count=parallel_in and count_en='1' else '0';
end rtl_limit;
```

Ο κώδικας VHDL για το testbench της behavioral αρχιτεκτονικής:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity counter_with_limit_behavioral_testbench is
end counter_with_limit_behavioral_testbench;

architecture B3_test of counter_with_limit_behavioral_testbench is

    constant clk1_period : time := 10 ns;
    signal clk_test,rst_test,en_test : std_logic := '0';
    signal sum_test : std_logic_vector(2 downto 0) := (others => '0');
    signal parallel_in_test : std_logic_vector(2 downto 0) := "111";
    signal cout_test : std_logic;

    component counter_with_limit_behavioral is
    port( clk,
        resetn,
        count_en : in std_logic;
        parallel_in : in std_logic_vector(2 downto 0);
        sum : out std_logic_vector(2 downto 0);
        cout : out std_logic);
    end component;
begin
    uut: counter_with_limit_behavioral
        port map (
            clk => clk_test,
            resetn => rst_test,
            count_en => en_test,
            parallel_in => parallel_in_test,
            sum => sum_test,
            cout => cout_test
        );
```

```
    stimulus: process begin
        for i in 0 to 1 loop
            rst_test <= not rst_test;
            for k in 0 to 1 loop
                en_test <= not en_test;
                    wait for 120ns;
                    parallel_in_test <= "101";
                    wait for 120ns;
                    parallel_in_test <= "011";
                    wait for 120ns;
            end loop;
        end loop;
        wait;
    end process;

    clk1_generator: process begin
            clk_test <= '0';
            wait for clk1_period/2;
            clk_test <= '1';
            wait for clk1_period/2;
    end process;

end architecture;
```

Ενα μέρος της προσομοίωσης που επιβεβαιώνει την ορθή λειτουργία του κυκλώματός, για τιμές ορίου 7, 5 και 3:



3. Τα κυκλώματα που δίνει ο synthesizer για τα ζητούμενα 1 και 2 είναι ορθά και δεν μπορούν να απλοποιηθούν περισσότερο, καθώς έχουμε χρησιμοποιήσει behavioral περιγραφή και συνεπώς το vivado εφαρμόζει αλγορίθμους ώστε να βρει το απλούστερο δυνατό σχηματικό διάγραμμα.