Elettronica dei Sistemi Digitali
Lab 04

Flip-flops and Counters

Gruppo: A11

Pronesti, Massimiliano	S245831
Oropallo, Maria Vittoria	S245999
Nicolicchia, Riccardo	S244728
Miceli, Michelangelo	S245478

18 aprile 2020

Indice

Capitolo 1

Gated SR latch

1.1 Spiegazione teorica

L'esercizio è stato svolto creando un testbench per il codice riportato nel testo dell'esercitazione, grazie al quale è stato possibile effettuare una timing analysis. Tale codice è stato poi sintetizzato per verificare che il circuito fosse correttamente implementato.

1.2 Procedimento

La sintesi del circuito, per la cui descrizione sono stati usati gli attributi ''keep", mostra la presenza di 4 LUT, secondo **1.1**. Si è inoltre verificato che, in assenza di tali attributi, il circuito risultante è composto da un'unica LUT.

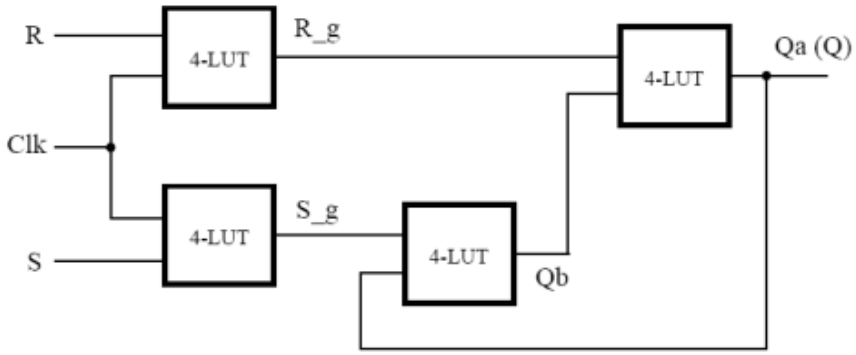


Figura 1.1: 4 LUTs implementation

1.3 Risultati

I risultati prodotti dalla simulazione e della sintesi sono mostrato in **figura 1.2** e **1.3**.

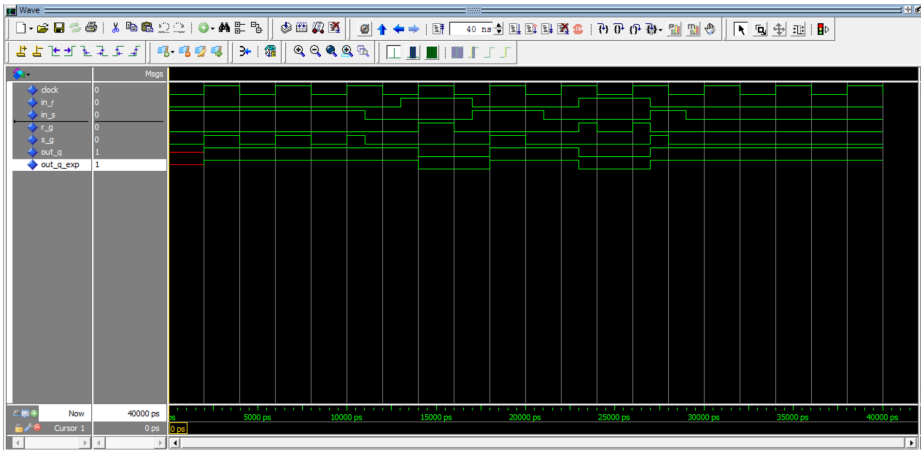


Figura 1.2: Wave exercise no. 1

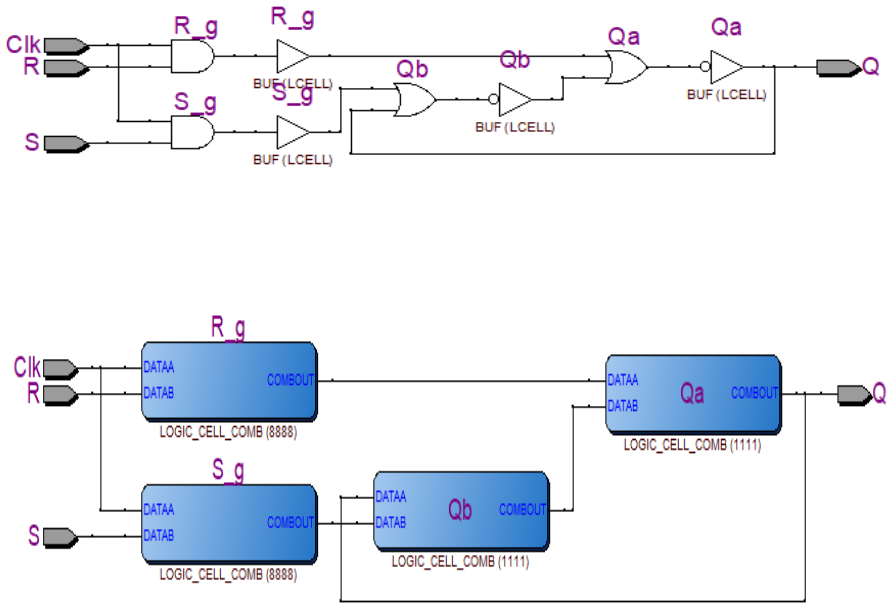


Figura 1.3: RTL viewer and technology viewer

1.4 Appendice

```
1  -- A gated RS latch described the hard way
2  library ieee;
3  use ieee.std_logic_1164.all;
4
5  entity SRlatch is
6    port (
7        Clk, R, S : in std_logic;
8        Q : out std_logic
9    );
10 end SRlatch;
11
12 architecture Structural of SRlatch is
13 signal R_g, S_g, Qa, Qb : std_logic ;
14 --ATTRIBUTE keep : boolean;
15 --ATTRIBUTE keep of R_g, S_g, Qa, Qb : signal is true;
16
17 begin
18 R_g <= R AND Clk;
19 S_g <= S AND Clk;
20 Qa <= NOT (R_g OR Qb);
21 Qb <= NOT (S_g OR Qa);
22 Q <= Qa;
23
24 end Structural;
```

Code 1.1: SRlatch.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity SRlatch_tb is
5  end SRlatch_tb;
6
7  architecture test of SRlatch_tb is
8
9    component SRlatch
10     port (
11       Clk, R, S : in std_logic;
12       Q : out std_logic
13     );
14   end component;
15
16   signal clock, in_r, in_s, out_q, out_q_exp : std_logic;
17
18 begin
19
```

```
20    --input values
21
22    process
23      begin
24      clock <= '0', '1' after 2 ns; --4 ns periodic clock
25      wait for 4 ns;
26    end process;
27
28      in_r <= '0', '1' after 13 ns, '0' after 17 ns, '1' after 23 ns, '0'
            after 27 ns;
29      in_s <= '1', '0' after 11 ns, '1' after 17 ns, '0' after 21 ns, '1'
            after 27 ns, '0' after 29 ns;
30
31    DUT: SRlatch port map (clock, in_r, in_s, out_q);
32
33      --expexted output
34      out_q_exp <= 'U', '1' after 2 ns, '0' after 14 ns, '1' after 18 ns,
            '0' after 23 ns, '1' after 27 ns;
35
36  end test;
```

Code 1.2: SRlatch_tb.vhd

Capitolo 2

16-bit synchronous counter

2.1 Spiegazione teorica

Per l'implementazione di un contatore sincrono a 16 bit sono stati istanziati 16 Flip-Flop di tipo toggle con reset asincrono. Settando l'enable del primo flip flop a 1, ogni flip flop incrementa di un'unità il valore del counter (come mostrato in figura wave1).

2.2 Procedimento

Si optato per un implementazione, tramite generics, di un generico contatore su N bit, in vista di un eventuale riutilizzo. Il primo T-FlipFlop riceve in ingresso il segnale di enable, mentre i successivi l' AND logico tra l'ingresso T e l'uscita Q del FlipFlop che precede.

2.3 Risultati

Di seguito sono riportati i risultati della simulazione e dall'analisi. Si osservi che, per questioni di legibilità dell'output prodotto dall'RTL viewer di Quartus, si è preferito riportare la sola sintesi di un contatore sincrono a 4 bit, ossia quello descritto da **code 2.1** per N=4. Questo non riduce la complessità per via della ricorsività del pattern.

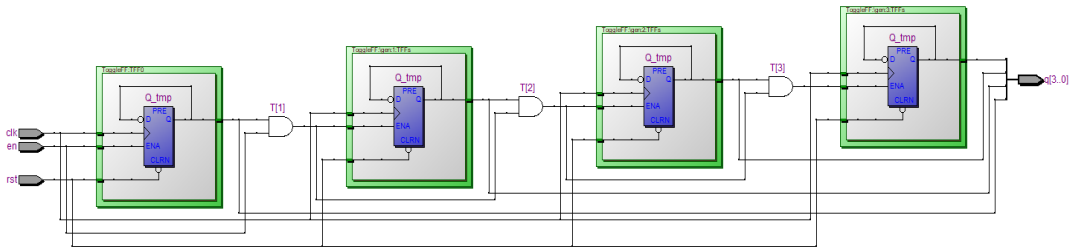


Figura 2.1: RTL viewer 4 bit counter

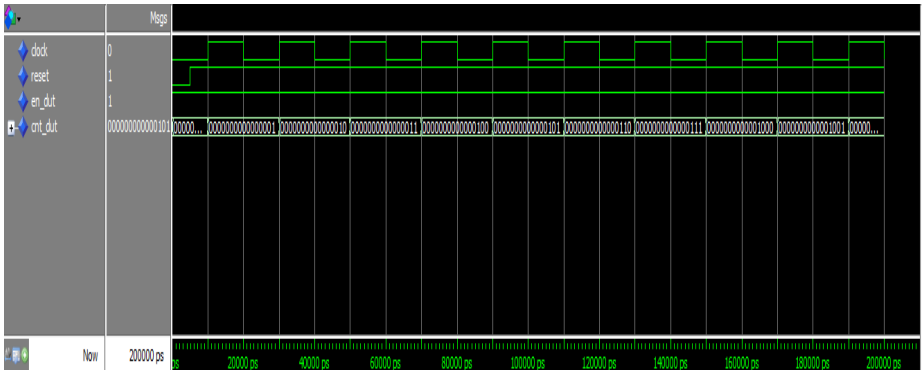


Figura 2.2: wave exercise no. 2

Si osserva dall'RTL viewer che per la versione a 16 bit si hanno altrettanti flip flop e 15 porte AND. La frequenza massima prodotta dalla time analysis risulta pari a $f_{max} = 356.9\,\text{MHz}$

2.4 Appendice

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
```

```
4
5  --KEY0=clock,SW1=Enable,SW0=Reset
6  entity counter_synth is
7    port(
8       SW : in std_logic_vector (1 downto 0);
9       KEY0 : in std_logic;
10        HEX3 : out std_logic_vector(0 to 6);
11        HEX2 : out std_logic_vector(0 to 6);
12        HEX1 : out std_logic_vector(0 to 6);
13        HEX0 : out std_logic_vector(0 to 6)
14   );
15 end counter_synth;
16
17 architecture behaviour of counter_synth is
18 constant n : integer := 16;
19
20   component counter is
21     generic (n: integer);--parallelismo
22     port (
23           en, clk, rst : in std_logic;
24           q : buffer unsigned(n-1 downto 0)
25     );
26   end component;
27
28    component hexa_display is
29      port(
30           SW : in unsigned (3 downto 0);
31           HEXA : out std_logic_vector ( 0 to 6)
32           );
33   end component;
34   signal count : unsigned(n-1 downto 0);
35
36   begin
37     CNT: counter generic map (n) port map (SW(1), KEY0, SW(0), count);
38      H0 : hexa_display port map(count(3 downto 0), HEX0);
39      H1 : hexa_display port map(count(7 downto 4), HEX1);
40      H2 : hexa_display port map(count(11 downto 8), HEX2);
41      H3 : hexa_display port map(count(15 downto 12), HEX3);
42 end architecture;
```

Code 2.1: counter_synth.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity counter is
6    generic (N: integer); --parallelism
7    port (
```

```
8           en, clk, rst : in std_logic;
9           Q : buffer unsigned( N-1 downto 0)
10    );
11  end counter;
12
13  architecture behavior of counter is
14
15    component TFF
16      port (
17         T, clk, rst : in std_logic;
18         Q           : out std_logic
19      );
20    end component;
21
22    signal T, cnt : std_logic_vector(n-1 downto 0);
23
24    begin
25      --init
26      T(0)<=en;
27      TFF0: TFF port map (t(0), clk, rst, cnt(0));
28
29      -- following FFs
30      GEN : for i in 1 to N-1 generate
31        TFFs: TFF port map (T(i), clk, rst, cnt(i));
32        T(i) <= T(i-1) and cnt(i-1);
33      end generate;
34
35     Q <= unsigned(cnt);
36  end behavior;
```

Code 2.2: counter_16b.vhd

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   -- empty entity section
6   entity counter_tb is
7   end counter_tb;
8
9   architecture test of counter_tb is
10
11  constant n : integer := 16;
12
13    component counter
14     generic (n:integer);--parallelismo
15      port (
16      en, clk, rst : in std_logic;
17           q : buffer unsigned (n-1 downto 0)
```

```
18          );
19    end component;
20
21    signal clock, reset, en_dut : std_logic;
22    signal cnt_dut : unsigned(n-1 downto 0);
23
24
25    begin
26
27      reset <= '0','1' after 5 ns ;
28      en_dut <= '1', '0' after 101 ns;
29
30      DUT: counter generic map (n) port map (en_dut, clock, reset,
            cnt_dut);
31
32
33        CLK_PR : process begin
34        clock <= '0';
35        wait for 10 ns;
36        clock <= '1';
37        wait for 10 ns;
38      end process;
39
40
41  end test;
```

Code 2.3: counter_16b_tb.vhd

```
1
2   library ieee;
3   use ieee.std_logic_1164.all;
4
5   entity TFF is
6     port (
7       T, clk, rst : in std_logic;
8       Q           : out std_logic
9     );
10  end TFF;
11
12  architecture behavior of TFF is
13
14    signal Q_tmp: std_logic;
15
16    begin
17
18      process(clk, rst) begin
19
20       -- async reset
21         if (rst ='0') then
```

```
22          Q_tmp<='0';
23        elsif (clk'event and clk = '1') then
24          if (T = '1') then
25             Q_tmp <= not Q_tmp;
26           end if;
27        end if;
28
29     end process;
30
31   Q <= Q_tmp;
32 end behavior;
```

Code 2.4: TFF.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4
5  -- empty entity section
6  entity ToggleFF_tb is
7  end ToggleFF_tb;
8
9  architecture test of ToggleFF_tb is
10
11 component ToggleFF is
12     port (
13       T, clk, clear : in std_logic;
14        Q           : out std_logic
15         );
16   end component;
17
18   signal clock, clear_dut, t_dut : std_logic;
19   signal q_dut : std_logic;
20
21 begin
22
23 clear_dut<= '0', '1' after 5 ns;
24
25 DUT : ToggleFF port map(t_dut, clock, clear_dut, q_dut);
26
27     CLK_PR : process begin
28          clock <= '0';
29          wait for 10 ns;
30          clock <= '1';
31          wait for 10 ns;
32          end process;
33
34 process
35   begin
```

```
36      T_dut <= '0';
37      wait for 50 ns;
38      T_dut <= '1';
39      wait for 50 ns;
40  end process;
41
42  end test;
```

Code 2.5: TFF_tb.vhd

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity hexa_display is
6     port(
7         SW : in unsigned (3 downto 0);
8         HEXA : out std_logic_vector ( 0 to 6)
9         );
10  end hexa_display;
11
12
13  architecture behaviour of hexa_display is
14
15  begin
16    with SW select
17      HEXA <=
18        "0000001" when "0000",
19        "1001111" when "0001",
20        "0010010" when "0010",
21        "0000110" when "0011",
22        "1001100" when "0100",
23        "0100100" when "0101",
24        "0100000" when "0110",
25        "0001111" when "0111",
26        "0000000" when "1000",
27        "0000100" when "1001",
28        "0001000" when "1010",
29        "0000000" when "1011",
30        "0110001" when "1100",
31        "0000001" when "1101",
32        "0110000" when "1110",
33        "0111000" when "1111",
34        "1111111" when others; --empty for undefined values
35  end architecture;
```

Code 2.6: hexa_display.vhd

Capitolo 3

16-bit synchronous counter version 2

3.1 Spiegazione teorica

In questo capitolo, ci proponiamo di implementare una nuova versione del contatore descritto al capitolo 2 basata sull'istruzione

$$Q <= Q + 1$$

e di confrontarne i risultati. Ne segue che il testbench sarà sempre **code 2.2**.

3.2 Procedimento

Si semplifica **Code 2.1** optando per una vista architetturale di tipo behavioral, i.e. fornendo solo una descrizione comportamentale.

3.3 Risultati

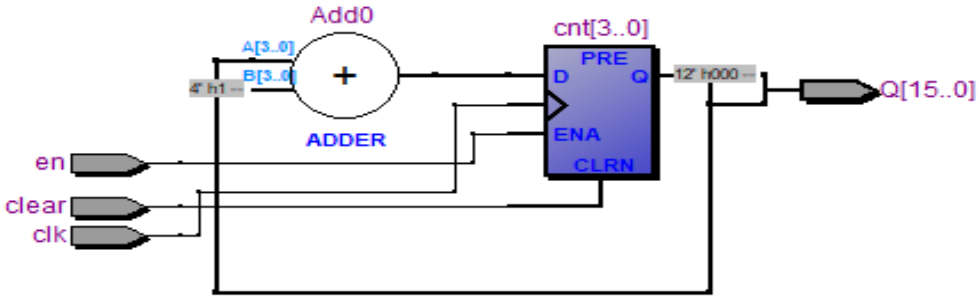
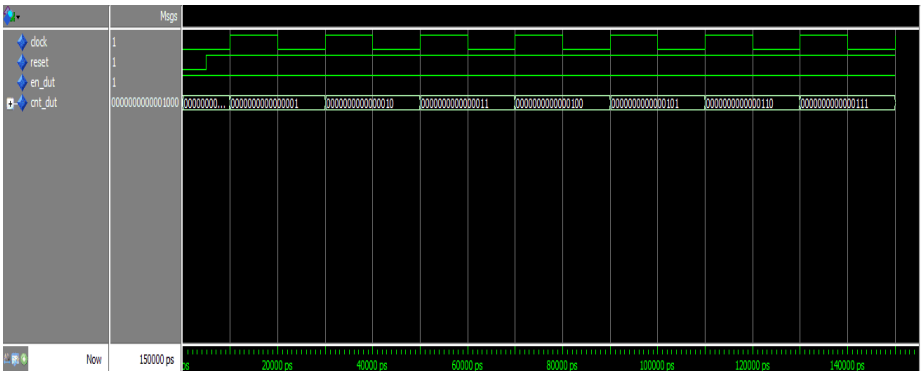


Figura 3.1: RTL viewer 4 bit synch counter v.2



La frequenza massima prodotta dalla time analysis risulta pari a $f_{max} = 383.6\,\text{MHz}$, i.e. lievemente più alta della versione descritta nel secondo esercizio. Il numero di elementi è pari a 2 : un sommatore ed un D-FF.

3.4 Appendice

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity counter is
6    generic (N : integer := 16);
```

```
7   port(
8       en, clk, clear : in std_logic;
9       Q : buffer unsigned (N-1 downto 0)
10  );
11 end counter;
12
13 architecture behavior of counter is
14
15  begin
16    process(clear, clk) begin
17
18      if (clear='1') then
19        Q <= (others => '0');
20      elsif clk'event and clk='1' then
21        if (en='1') then
22          Q <= Q + 1;
23        end if;
24      end if;
25
26    end process;
27
28 end architecture;
```

Code 3.1: counter_v2.vhd

Capitolo 4

Flashing digits from 0 to 9

4.1 Spiegazione teorica

Ci proponiamo di realizzare un contatore che pilota un display a 7 segmenti mostrando, incrementalmente, i numeri da 0 a 9 ad intervalli di circa 1 secondo.

4.2 Procedimento

Si implementa lo schema a blocchi mostrato in **figura 4.1** il cui funzionamento è descritto di seguito: il contatore più veloce (fast_count) detta i tempi di quello più lento (slow_count), il quale riceve il segnale di enable solo quando il primo contatore è arrivato a 49999999 ed è resettato. Questa condizione è garantita tramite una cascata tra una port XOR, che opera bitwise su ingressi da 26 bit ed una porta NOR che restituisce 1 solo se ha tutti gli ingressi nulli. Analogamente, il contatore lento è resettato quando è arrivato a 9 tramite la porta AND che preleva l'LSB e l'MSB. L'intervallo di 1 secondo è garantito facendo sì che il clock esponga un fronte di salita ogni 20 ns

$$20\,\text{ns} \cdot 50 \times 10^6 = 1\,\text{s}$$

come mostrato in **Code 4.2**.
In VHDL si è optato per una descrizione architetturale di tipo comportamentale in due process distinti per i due contatori.

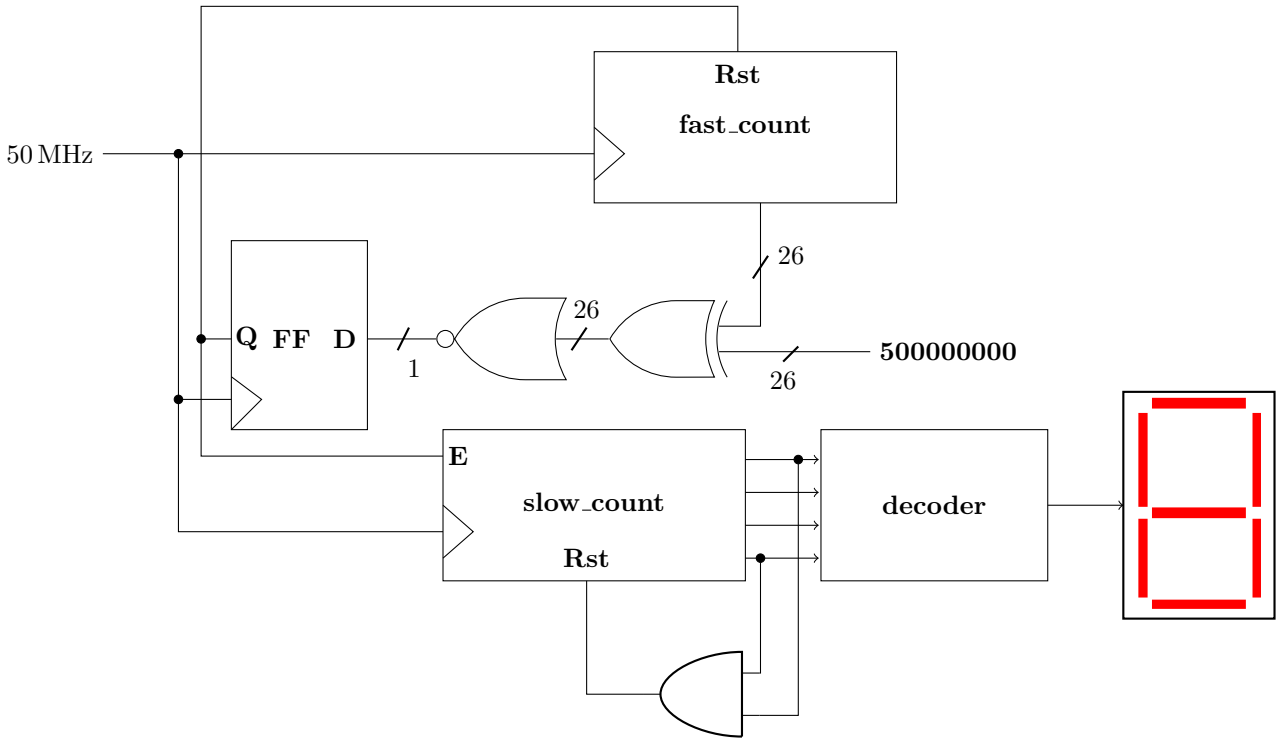


Figura 4.1: Block Scheme 0 - 9 counter

4.3 Risultati

Di seguito sono mostrati i risultati prodotti dalla simulazione e dalla sintesi. Si osservi che, per poter apprezzare le variazioni del contatore lento, si è limitato il limite superiore del contatore veloce a 500.

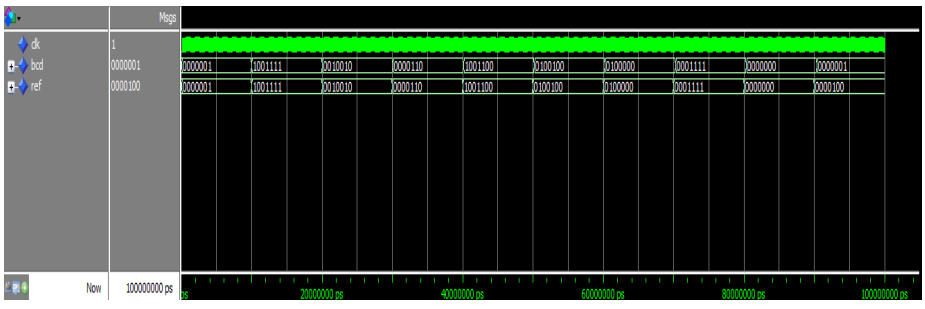


Figura 4.2: wave exercise no. 4

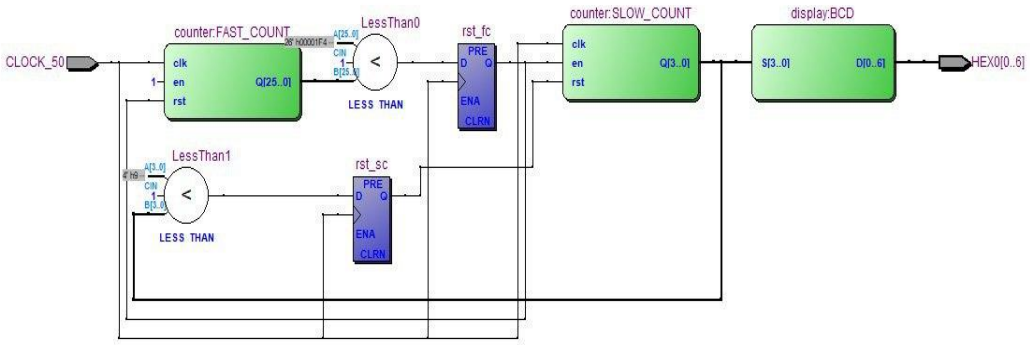


Figura 4.3: RTL viewer exercise no. 4

I blocchi di confronto corrispondono alle cascate di porte logiche nello schema a blocchi in **figura 4.1**

4.4 Appendice

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5
6  entity flasher0to9 is
7    port(
8      CLOCK_50 : in std_logic;
9      HEX0     : out std_logic_vector(0 to 6)
10   );
11 end flasher0to9;
12
13 architecture Behavioral of flasher0to9 is
14
```

```
15    component counter is
16      generic (N : integer := 16);
17      port(
18          en, clk, rst : in std_logic;
19          Q : buffer unsigned (N-1 downto 0)
20          );
21    end component;
22
23    component hexa_display is
24       port(
25          SW : in unsigned(3 downto 0);
26          HEXA : out std_logic_vector(0 to 6)
27       );
28    end component;
29
30    signal rst_fc, rst_sc, en_sc : std_logic;
31    signal Q_fc : unsigned (25 downto 0); -- output of slow counter
32    signal Q_sc : unsigned (3 downto 0); -- output of fast counter
33    signal FREQ : integer := 500;
34
35    begin
36
37    FC_PROCESS : process (CLOCK_50) begin -- fast counter process
38      if CLOCK_50'event and CLOCK_50 = '1' then
39        if Q_fc >= FREQ then
40          rst_fc <= '1';
41        else
42          rst_fc <= '0';
43        end if;
44      end if;
45    end process;
46
47    SC_PROCESS : process (CLOCK_50) begin -- slow counter process
48      if CLOCK_50'event and CLOCK_50 = '1' then
49        if Q_sc >= 9 then
50          rst_sc <= '1';
51        else
52          rst_sc <= '0';
53        end if;
54      end if;
55    end process;
56
57    FAST_COUNT : counter generic map (26) port map ('1', CLOCK_50, rst_fc
          , Q_fc );
58
59    SLOW_COUNT : counter generic map (4) port map (rst_fc, CLOCK_50,
          rst_sc, Q_sc );
60
61    BCD : hexa_display port map (Q_sc,HEX0);
62  end architecture;
```

Code 4.1: flasher0to9.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity counter0to9_tb is
5  end entity;
6
7  architecture test of counter0to9_tb is
8    component flasher0to9 is
9      port(
10       CLOCK_50 : in std_logic;
11       HEX0     : out std_logic_vector(0 to 6)
12     );
13   end component;
14
15   signal clk : std_logic;
16   signal bcd, ref : std_logic_vector(6 downto 0);
17   constant Ts : time := 20 ns ;
18
19   begin
20
21   -- one raising edge for each 20 ns block
22   -- after 5e7 edges ,i.e. after 1 sec
23   -- the 7 seg display increases its value
24    CLK_PR: process begin
25       clk <= '1';
26       wait for Ts/2;
27       clk <= '0';
28       wait for Ts/2;
29   end process;
30
31   ref <= "0000001" ,   -- 0
32   "1001111" after 10 us, -- 1
33   "0010010" after 20 us, -- 2
34   "0000110" after 30 us, -- 3
35   "1001100" after 40 us, -- 4
36   "0100100" after 50 us, -- 5
37   "0100000" after 60 us, -- 6
38   "0001111" after 70 us, -- 7
39   "0000000" after 80 us, -- 8
40   "0000100" after 90 us;
41
42    UUT : flasher0to9 port map (clk, bcd);
43
44 end architecture;
```

Code 4.2: flasher0to9_tb.vhd

Capitolo 5

Reaction timer

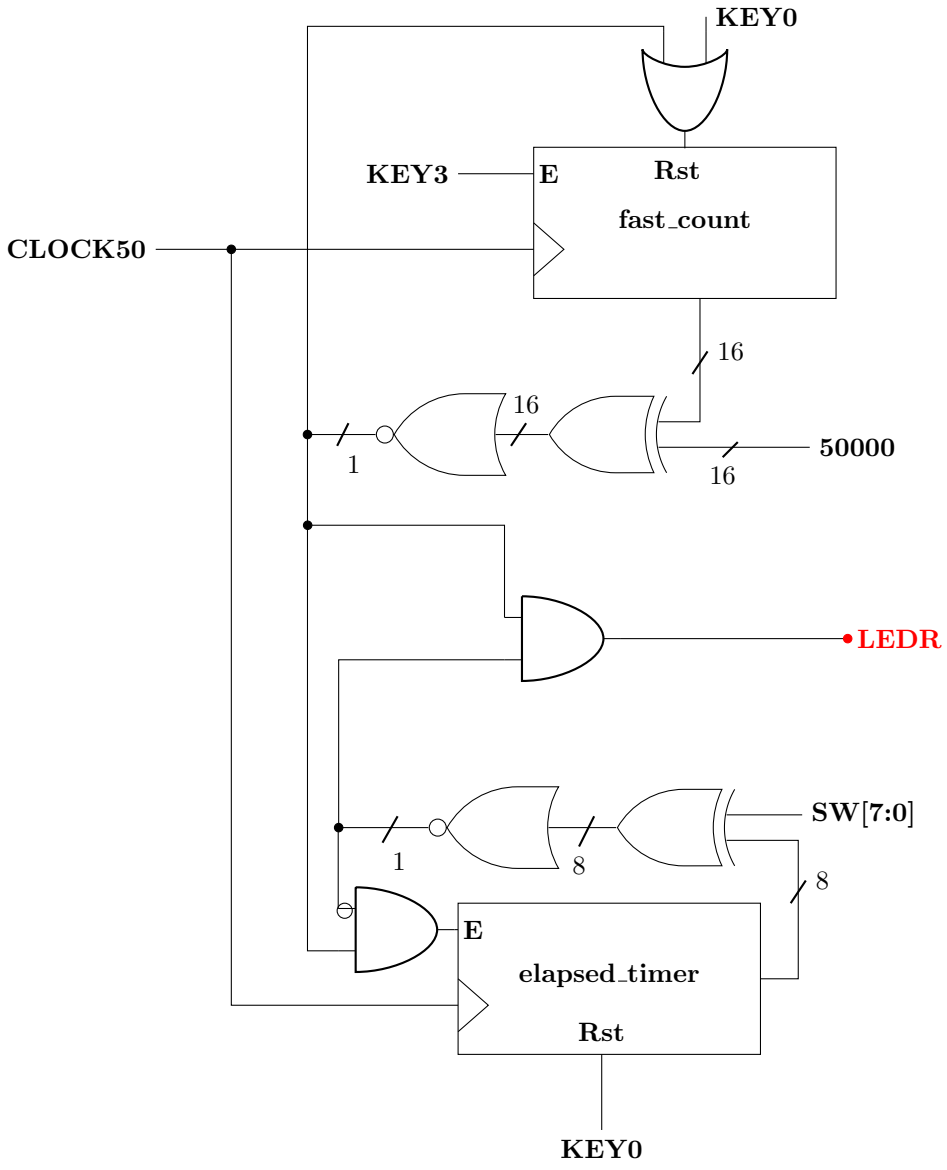
5.1 Spiegazione teorica

In quest'ultimo capitolo, ci focalizzeremo sull'implementazione di un reaction timer tramite dei contatori asincroni in grado di contare gli l'intervallo di tempo tra il reset del circuito e l'attivazione del segnale KEY3. In particolare, il componente counter incrementa il suo valore ad ogni colpo di clock ; 4_digit_counter pilota i quattro decoder.

5.2 Procedimento

Il componente count_fast scandisce il passare dei millisecondi. Esso, infatti, riceve il CLOCK_50 ed incrementa ad ogni colpo di clock il suo valore che viene, poi, comparato come nell'esercizio 4, con il valore 49999, la cui uscita è collegata ai successivi componenti per abilitarne le operazioni col giusto timing (1 ms). Il componente elaps_timer utilizza un contatore, incrementato ogni ms grazie al segnale di enable proveniente dal fast_counter. Il suo contenuto viene comparato con quello di SW; se i valori sono uguali, cioè se sono stati aspettati SW ms, allora i quattro decoder vengono attivati tramite 4_digit_counter. In particolare, ogni volta che il decoder che conta le unità raggiunge quota 10, viene incrementato il decoder che conta le decine. Analogamente per i restanti due decoder: ogni volta che il precedente counter conta '10', incrementano il numero mostrato sul display. Ognuno dei quattro counter, inoltre, viene resettato quando KEY0 vale zero (in modo da mostrare una sequenza di zeri durante

l'elapsed time) oppure quando il contatore è a quota 10 (in modo da mostrare solo l'unità, '0'). Quando KEY3 è attivato, l'enable del count_fast vale zero. Viene dunque disabilitato il primo contatore, e di conseguenza tutti gli altri, permettendo uno stato di freeze. Lo schema a blocchi è mostrato in **figura 5.1** (divisa in 2 pagine per questioni grafiche).



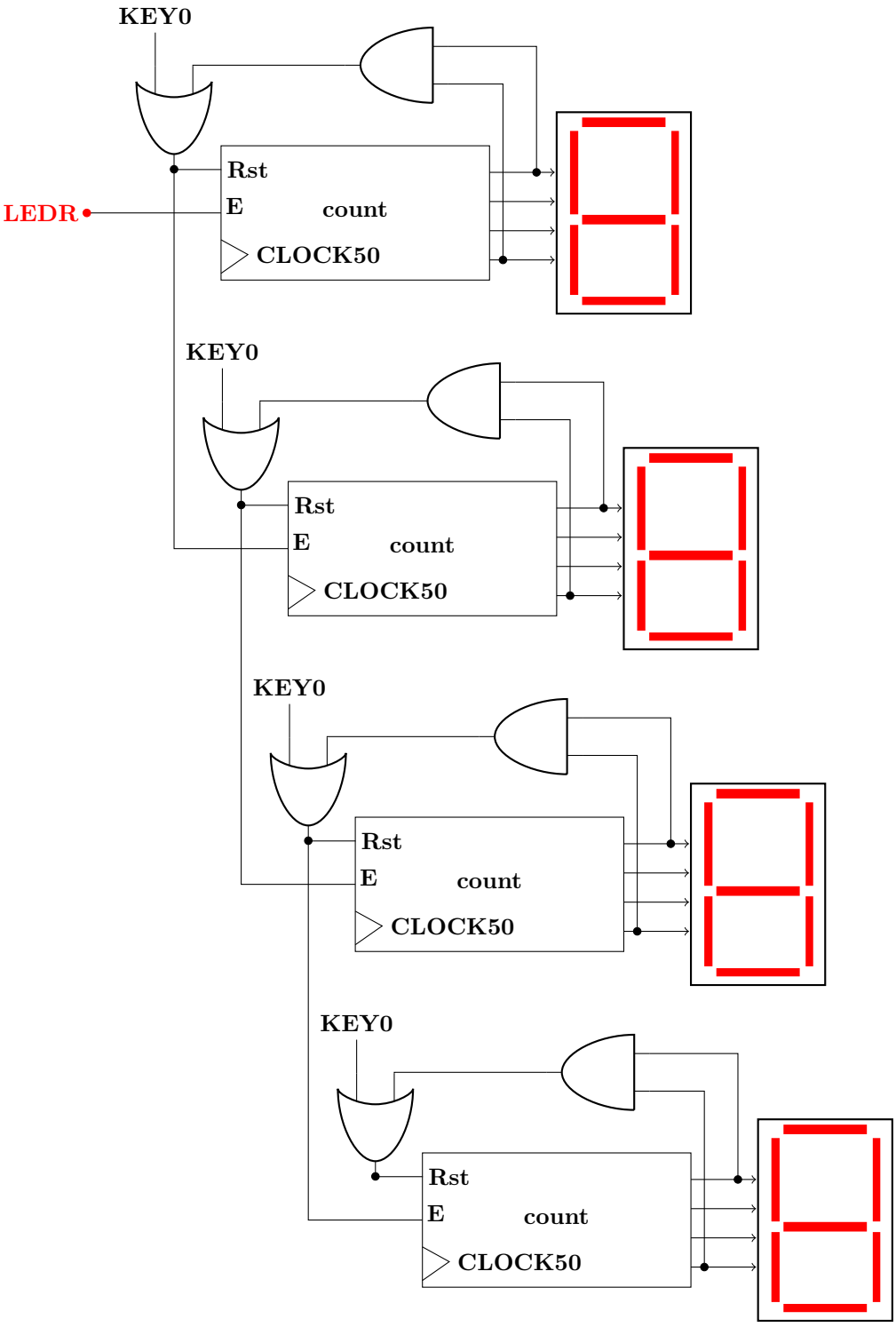


Figura 5.1: Block scheme exercise no. 5

Come nell'esercizio 4, sulle uscite delle porte NOR dovrebbero figurare dei FF di tipo D che sono stati omessi per ragioni grafiche.

5.3 Risultati

Si osservi che la wave di seguito riportata presenta un numero di segnali intermedi maggiore rispetto a quelli presenti nel testbench, inseriti soltanto per una più chiara lettura della simulazione. Per le analoghe ragioni descritte al capitolo 4, si è abbassato il limite superiore del contatore veloce.
Dalla sintesi si evince la presenza di un elemento di memoria sull'enable del contatore veloce dovuto ai segnali asincroni KEY0 e KEY1.

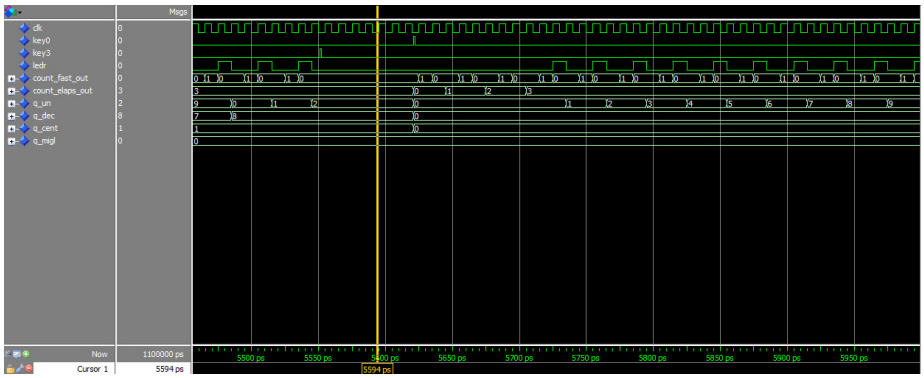


Figura 5.2: wave exercise no. 5

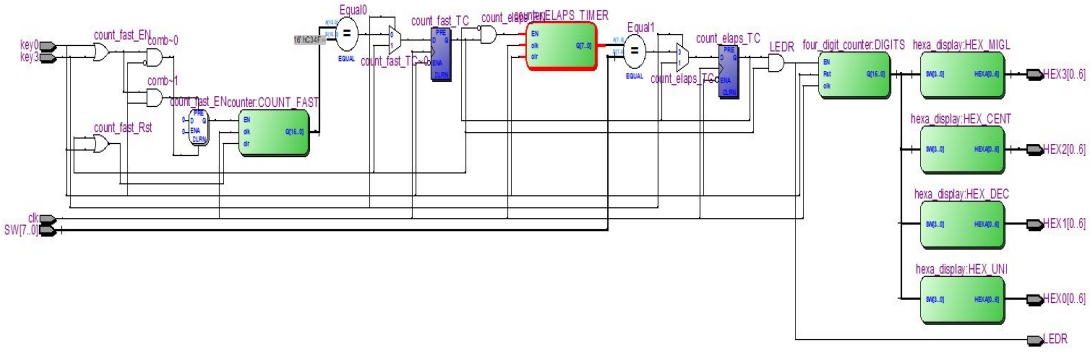


Figura 5.3: RTL viewer timer

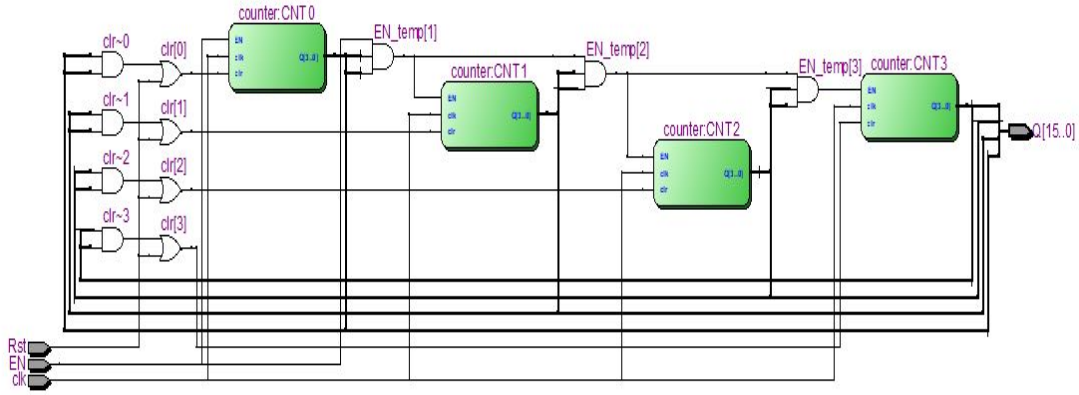


Figura 5.4: RTL viewer decoders

5.4 Appendice

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
```

```
4
5  entity cronometer is
6    port ( clk,key0,key3 : in std_logic;
7           SW : in std_logic_vector (7 downto 0);
8           LEDR : buffer std_logic;
9           HEX0,HEX1,HEX2,HEX3 : out std_logic_vector ( 0 to 6)
10        );
11 end cronometer;
12
13
14 architecture gated of cronometer is
15
16
17   component four_digit_counter
18      port ( EN, clk, Rst : in std_logic;
19          Q : buffer unsigned (15 downto 0)
20        );
21   end component;
22
23
24
25   component counter
26     generic (n : integer := 4);
27     port ( EN, clk, clr : in std_logic;
28            Q : buffer unsigned (n-1 downto 0)
29          );
30   end component;
31
32
33   component hexa_display
34     port(
35       SW : in unsigned (3 downto 0);
36       HEXA : out std_logic_vector ( 0 to 6)
37       );
38   end component;
39
40
41    signal count_fast_TC,count_fast_EN,count_fast_Rst : std_logic;
42    signal count_fast_out : unsigned (15 downto 0);
43
44    signal count_elaps_TC,count_elaps_EN : std_logic;
45    signal count_elaps_out : unsigned (7 downto 0);
46
47    signal Q : unsigned (15 downto 0);
48
49
50 begin
51
52
53   count_elaps_EN <= count_fast_TC and not (count_elaps_TC);
```

```
54  --counts time elapsed before start cronometer
55  ELAPS_TIMER: counter generic map(8)
56                       port map
                             (count_elaps_EN,clk,KEY0,count_elaps_out);
57
58
59   count_fast_Rst <= KEY0 or count_fast_TC;
60   --counts from 0 to 49999 at 50Mhz (1 ms per cycle)
61  COUNT_FAST: counter generic map(16)
62                       port map
                             (count_fast_EN,clk,count_fast_Rst,count_fast_out);
63
64  --lits when elapsed timer has finished and 1 ms is elapsed
65  LEDR <= count_fast_TC and count_elaps_TC;
66
67
68
69  DIGITS: four_digit_counter port map (LEDR,clk,KEY0,Q);
70
71
72
73  HEX_UNI: hexa_display port map( Q(3 downto 0), HEX0 );
74  HEX_DEC: hexa_display port map( Q(7 downto 4), HEX1 );
75  HEX_CENT: hexa_display port map( Q(11 downto 8), HEX2 );
76  HEX_MIGL: hexa_display port map( Q(15 downto 12), HEX3 );
77
78
79  Async_CONTR:process (clk,key0,key3)
80    begin
81
82      --async reset
83      if key0 = '1' then
84        count_fast_EN <= '1';
85
86      --async freeze
87      elsif key3 = '1' then
88        count_fast_EN <= '0';
89
90      --sync evaluation of TCs
91      elsif clk'event and clk = '1' then
92
93        if count_fast_out = to_unsigned(49999,16) then
94          count_fast_TC <= '1';
95        else
96          count_fast_TC <= '0';
97        end if;
98
99
100       if count_elaps_out = unsigned(SW)then
101         count_elaps_TC <= '1';
```

```
102         else
103           count_elaps_TC <= '0';
104         end if;
105
106       end if;
107
108   end process;
109
110
111
112 end architecture;
```

Code 5.1: cronometer.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity cronometer_tb is
6
7  end cronometer_tb;
8
9
10 architecture gate of cronometer_tb is
11
12
13
14   component cronometer
15       port ( clk,key0,key3 : in std_logic;
16         SW : in std_logic_vector (7 downto 0);
17         LEDR : buffer std_logic;
18         HEX0,HEX1,HEX2,HEX3 : out std_logic_vector ( 0 to 6)
19       );
20   end component;
21
22
23
24   signal clk, key0, key3,LEDR : std_logic;
25   signal HEX0_dut,HEX1_dut,HEX2_dut,HEX3_dut : std_logic_vector (0 to 6);
26   signal HEX0_dut_dec,HEX1_dut_dec,HEX2_dut_dec,HEX3_dut_dec : integer;
27
28
29
30 begin
31
32   CLK_GEN: process
33   begin
34
35     clk <= '0','1' after 5 ps;
```

```
36
37      wait for 10 ps;
38
39    end process CLK_GEN;
40
41
42
43    key0 <= '1', '0' after 1 ps, '1' after 5621 ps,'0' after 5622 ps;
44    key3 <= '0','1' after 5551 ps,'0' after 5552 ps;
45
46
47
48
49    COUN: cronometer port map (clk,key0,key3,"00000011",ledr,
50                               HEX0_dut,HEX1_dut,HEX2_dut,HEX3_dut);
51
52
53    with HEX0_dut select
54      HEX0_dut_dec <=
55      0 when "0000001",
56      1 when "1001111",
57      2 when "0010010",
58      3 when "0000110",
59      4 when "1001100",
60      5 when "0100100",
61      6 when "0100000",
62      7 when "0001111",
63      8 when "0000000",
64      9 when "0000100",
65      10 when others;
66
67    with HEX1_dut select
68      HEX1_dut_dec <=
69      0 when "0000001",
70      1 when "1001111",
71      2 when "0010010",
72      3 when "0000110",
73      4 when "1001100",
74      5 when "0100100",
75      6 when "0100000",
76      7 when "0001111",
77      8 when "0000000",
78      9 when "0000100",
79      10 when others;
80
81    with HEX2_dut select
82      HEX2_dut_dec <=
83      0 when "0000001",
84      1 when "1001111",
85      2 when "0010010",
```

```
86      3 when "0000110",
87      4 when "1001100",
88      5 when "0100100",
89      6 when "0100000",
90      7 when "0001111",
91      8 when "0000000",
92      9 when "0000100",
93      10 when others;
94
95    with HEX3_dut select
96      HEX3_dut_dec <=
97      0 when "0000001",
98      1 when "1001111",
99      2 when "0010010",
100     3 when "0000110",
101     4 when "1001100",
102     5 when "0100100",
103     6 when "0100000",
104     7 when "0001111",
105     8 when "0000000",
106     9 when "0000100",
107     10 when others;
108
109
110
111 end architecture;
```

Code 5.2: cronometer_tb.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  --it use 4 counter to count from 0 to 9999 on 16 bit
6
7  entity four_digit_counter is
8    port ( EN, clk, Rst : in std_logic;
9           Q : buffer unsigned (15 downto 0)
10        );
11 end four_digit_counter;
12
13 architecture gated of four_digit_counter is
14
15   component counter
16     generic (n : integer := 4);
17     port ( EN, clk, clr : in std_logic;
18            Q : buffer unsigned (n-1 downto 0)
19          );
20   end component;
```

```
21
22
23    signal clr ,EN_temp: std_logic_vector (3 downto 0);
24
25  begin
26
27    EN_temp(0) <= EN;
28    clr(0) <= Rst or (Q(3) and Q(1));
29    CNT0: counter port map ( EN_temp(0),clk,clr(0),Q(3 downto 0));
30
31    EN_temp(1) <= Q(3) and Q(0) and EN_temp(0);
32    clr(1) <= Rst or (Q(7) and Q(5));
33    CNT1: counter port map ( EN_temp(1),clk,clr(1),Q(7 downto 4));
34
35    EN_temp(2) <= Q(7) and Q(4)and EN_temp(1);
36    clr(2) <= Rst or (Q(11) and Q(9));
37    CNT2: counter port map ( EN_temp(2),clk,clr(2),Q(11 downto 8));
38
39    EN_temp(3) <= Q(11) and Q(8) and EN_temp(2);
40    clr(3) <= Rst or (Q(15) and Q(13));
41    CNT3: counter port map ( EN_temp(3),clk,clr(3),Q(15 downto 12));
42
43  end  architecture;
```

Code 5.3: 4_digit_counter.vhd

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity four_digit_counter_tb is
6
7   end four_digit_counter_tb;
8
9
10  architecture gate of four_digit_counter_tb is
11
12    component four_digit_counter
13       port ( EN, clk, Rst : in std_logic;
14           Q : buffer unsigned (15 downto 0)
15         );
16    end component;
17
18    signal EN, clk, Rst : std_logic;
19    signal Q_un,Q_dec,Q_cent,Q_migl : unsigned (3 downto 0);
20    signal Q_dut : unsigned (15 downto 0);
21
22
23
```

```
24  begin
25
26    CLK_GEN: process
27    begin
28
29      clk <= '0','1' after 5 ps;
30
31      wait for 10 ps;
32
33    end process CLK_GEN;
34
35
36
37    EN <= '1', '0' after 100 ps, '1' after 111 s;
38    Rst <= '1', '0' after 1 ps;
39   --  Q_dut <= Q_migl & Q_cent & Q_dec & Q_un;
40
41
42    COUNT: four_digit_counter port map (EN,clk,Rst,Q_dut);
43    Q_un <= Q_dut (3 downto 0);
44    Q_dec <= Q_dut (7 downto 4);
45    Q_cent <= Q_dut (11 downto 8);
46    Q_migl <= Q_dut (15 downto 12);
47  end architecture;
```

Code 5.4: 4_digit_counter_tb.vhd