

مباحث ویژه در الکترونیک
VHDL

Dr. Mohsen Shakiba

Practice3:

12BIT Counter, Decode For 7Segment

Mohammad Niknam

951843189

تمرین 3

- هدف طراحی یک شمارنده است که مقادیر آن روی یک 7-SEGMENT نمایش داده می شود.

الف) یک شمارنده باینری 12 بیتی در VHDL توصیف کنید (شمارنده افزایشی است). عملکرد شمارنده همگام با لبه بالارونده پالس ساعت CLK است که اگر مقدار شمارنده به عدد 3EF هگزا دسیمال (معادل 999 دسیمال) برسد دوباره باید صفر شود. ورودی ریست RST سنکرون با پالس ساعت است که در صورت فعال شدن آن، خروجی شمارنده صفر می شود. همچنین در صورت فعال شدن ورودی LOAD (همگام با پالس ساعت) مقدار ورودی 12 بیتی با نام DA در شمارنده بارگذاری می گردد. برای نمایش خروجی این شمارنده به سه 7-SEGMENT نیاز داریم و خروجی مازول شمارنده نیز 24 بیت (8 بیت برای هر 7-SEGMENT) است. برای دیگری 7-SEGMENT یک مازول جداگانه بنویسید و به تعداد لازم از آن در مازول اصلی جایگذاری کنید (برای هر 4 بیت نیاز به یک دیگر داریم). برای ارزیابی صحت عملکرد مازول شمارنده یک تست بنچ نیز بنویسید. (10 نمره)

ب) این بار میخواهیم مقادیر نمایش داده شده روی 7-SEGMENT دسیمال باشند. مثلا اگر 12 بیت شمارنده 000010111000 باشد مقدار نمایش داده شده معادل دسیمال آن یعنی عدد 184 است (و نه 0B8 هگزادسیمال). برای این منظور یک مازول دیگر 12 BCD بیتی را به صورت جداگانه بنویسید و در مازول اصلی جایگذاری کنید (واضح است که مقدار شمارنده باینری ورودی این مازول است). برای ارزیابی صحت عملکرد مازول شمارنده یک تست بنچ نیز بنویسید. (10 نمره)

لطفا پاسخ نهایی را در قالب یک فایل فشرده شامل کدهای VHDL و فایل گزارش PDF ارسال بفرمایید.

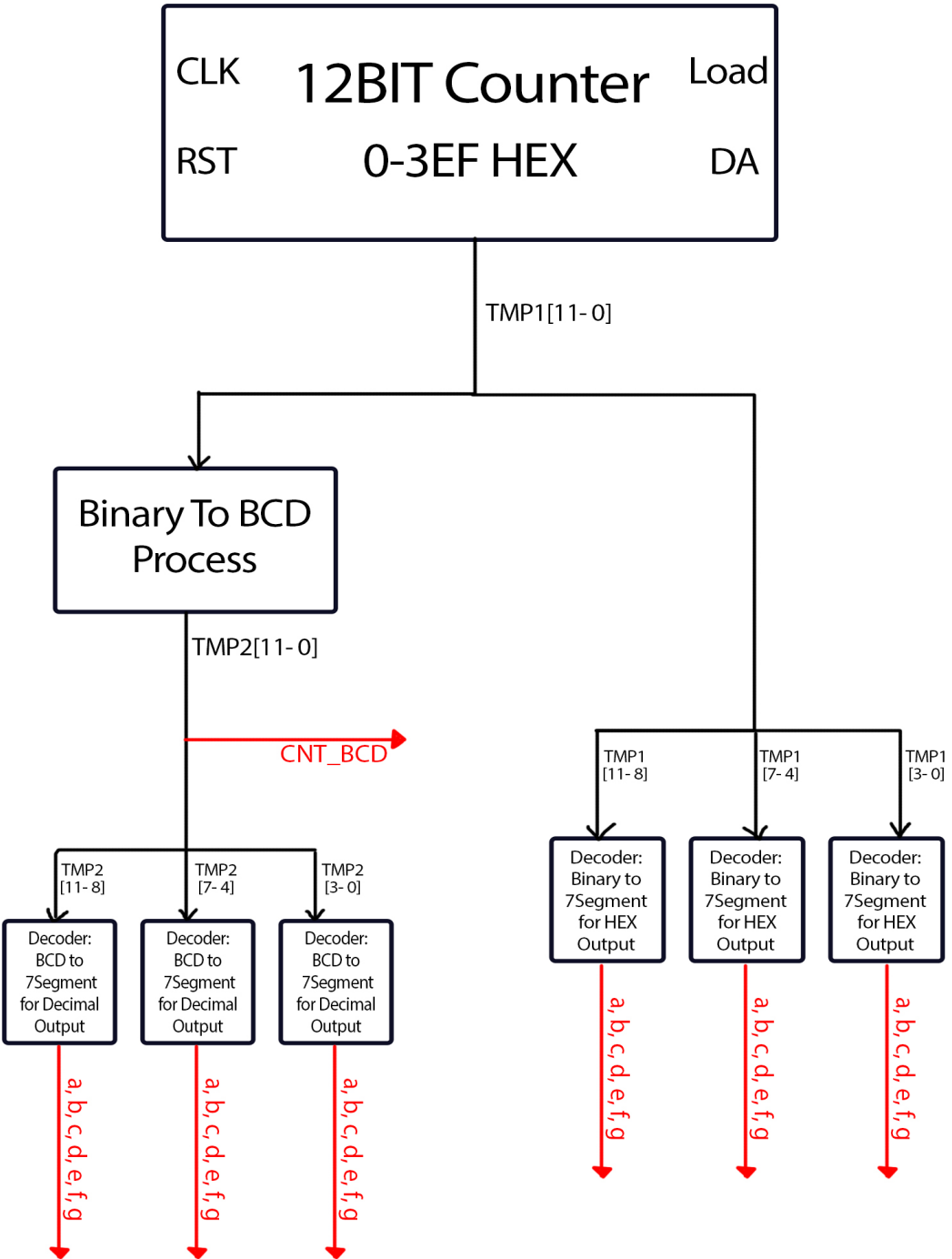
اگر نحوه عملکرد BCD DECODER را به خاطر ندارید به لینک زیر مراجعه نمایید که عملکرد یک دیگر 8 بیتی را با یک مثال نمایش میدهد (عملکرد دیگر عام است و به راحتی قابل توسعه به هر تعداد بیت دلخواه)

لینک :

<https://my.eng.utah.edu/~nmcdonal/Tutorials/BCDTutorial/BCDConversion.html>

سخت افزار به گونه ای طراحی شد که عملکرد موارد الف و ب تمرین را با هم انجام دهد

Block Diagram:



Counter 12BIT:

VHDL CODE

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  ENTITY COUNTER_12BIT IS
6      port(
7          CLK,RST,Load : IN STD_LOGIC;
8          DA : IN STD_LOGIC_VECTOR(11 downto 0);
9          CNT_BCD : OUT STD_LOGIC_VECTOR(11 downto 0);
10
11         SS_HEX_a : OUT STD_LOGIC_VECTOR(3 downto 1);           -- SS=7Segment
12         SS_HEX_b : OUT STD_LOGIC_VECTOR(3 downto 1);
13         SS_HEX_c : OUT STD_LOGIC_VECTOR(3 downto 1);
14         SS_HEX_d : OUT STD_LOGIC_VECTOR(3 downto 1);
15         SS_HEX_e : OUT STD_LOGIC_VECTOR(3 downto 1);
16         SS_HEX_f : OUT STD_LOGIC_VECTOR(3 downto 1);
17         SS_HEX_g : OUT STD_LOGIC_VECTOR(3 downto 1);
18
19         SS_Decimal_a : OUT STD_LOGIC_VECTOR(3 downto 1);
20         SS_Decimal_b : OUT STD_LOGIC_VECTOR(3 downto 1);
21         SS_Decimal_c : OUT STD_LOGIC_VECTOR(3 downto 1);
22         SS_Decimal_d : OUT STD_LOGIC_VECTOR(3 downto 1);
23         SS_Decimal_e : OUT STD_LOGIC_VECTOR(3 downto 1);
24         SS_Decimal_f : OUT STD_LOGIC_VECTOR(3 downto 1);
25         SS_Decimal_g : OUT STD_LOGIC_VECTOR(3 downto 1));
26
27  END COUNTER_12BIT;
28
29  ARCHITECTURE Behavioral of COUNTER_12BIT is
30      signal TMP1 : unsigned(11 downto 0);
31      signal TMP2 : unsigned(11 downto 0);
32
33      component Binary_To_7S_HEX is
34          port (Binary: in std_logic_vector (3 downto 0);
35               a,b,c,d,e,f,g: out std_logic);
36      end component;
37
38      component BCD_To_7S_Decimal is
39          port (bcd: in std_logic_vector (3 downto 0);
40               a,b,c,d,e,f,g: out std_logic);
41      end component;
42
43  begin
44
45      Counter_proc: process (CLK)
46      begin
47          if (CLK'event and CLK='1') then
48              if (RST='1') then
49                  TMP1 <= (others => '0');
50              elsif (Load='1') then
51                  TMP1 <= unsigned(DA);
52              elsif (TMP1="0011111100111") then
53                  TMP1 <= (others => '0');
54              else
55                  TMP1 <= TMP1+"000000000001";
56              end if;
57          end if;
58      end process;
59
60
61
62
63
64
65
66
67
68
```

Counter 12BIT:

VHDL CODE

```
74  binary_to_bcd_proc: process(TMP1)
75      variable bcd: unsigned(11 downto 0); --bcd vector, with 4 bits for each display
76      variable TMP: unsigned(11 downto 0); --auxiliary vector, to copy the input and
        operate with It
77
78      begin
79          bcd := (others => '0');          -- ADDED for EVERY CONVERSION
80          TMP := TMP1;                    -- ADDED for EVERY CONVERSION
81
82          for i in 0 to 11 loop
83
84              --bit shifting. Copy the bint MSB in the bcd LSB
85              bcd(11 downto 0) := bcd(10 downto 0) & TMP(11);
86              TMP(11 downto 0) := TMP(10 downto 0) & '0';
87
88              if i < 11 and bcd(3 downto 0) > "0100" then
89                  bcd(3 downto 0) := bcd(3 downto 0) + 3;
90              end if;
91              if i < 11 and bcd(7 downto 4) > "0100" then
92                  bcd(7 downto 4) := bcd(7 downto 4) + 3;
93              end if;
94              if i < 11 and bcd(11 downto 8) > "0100" then
95                  bcd(11 downto 8) := bcd(11 downto 8) + 3;
96              end if;
97
98              TMP2 <= bcd;
99          end loop;
100  end process;
101
102  CNT_BCD <= std_logic_vector(TMP2);
103
104  Binary_To_7S_HEX_1 : Binary_To_7S_HEX port map(Binary=>std_logic_vector(TMP1(3
        downto 0)), a=>SS_HEX_a(1), b=>SS_HEX_b(1), c=>SS_HEX_c(1), d=>SS_HEX_d(1),
        e=>SS_HEX_e(1), f=>SS_HEX_f(1), g=>SS_HEX_g(1));
105  Binary_To_7S_HEX_2 : Binary_To_7S_HEX port map(Binary=>std_logic_vector(TMP1(7
        downto 4)), a=>SS_HEX_a(2), b=>SS_HEX_b(2), c=>SS_HEX_c(2), d=>SS_HEX_d(2),
        e=>SS_HEX_e(2), f=>SS_HEX_f(2), g=>SS_HEX_g(2));
106  Binary_To_7S_HEX_3 : Binary_To_7S_HEX port map(Binary=>std_logic_vector(TMP1(11
        downto 8)), a=>SS_HEX_a(3), b=>SS_HEX_b(3), c=>SS_HEX_c(3), d=>SS_HEX_d(3),
        e=>SS_HEX_e(3), f=>SS_HEX_f(3), g=>SS_HEX_g(3));
107
108  BCD_To_7S_Decimal_1 : BCD_To_7S_Decimal port map(BCD => std_logic_vector(TMP2(3
        downto 0)), a=>SS_Decimal_a(1), b=>SS_Decimal_b(1), c=>SS_Decimal_c(1),
        d=>SS_Decimal_d(1), e=>SS_Decimal_e(1), f=>SS_Decimal_f(1), g=>SS_Decimal_g(1));
109  BCD_To_7S_Decimal_2 : BCD_To_7S_Decimal port map(BCD => std_logic_vector(TMP2(7
        downto 4)), a=>SS_Decimal_a(2), b=>SS_Decimal_b(2), c=>SS_Decimal_c(2),
        d=>SS_Decimal_d(2), e=>SS_Decimal_e(2), f=>SS_Decimal_f(2), g=>SS_Decimal_g(2));
110  BCD_To_7S_Decimal_3 : BCD_To_7S_Decimal port map(BCD => std_logic_vector(TMP2(11
        downto 8)), a=>SS_Decimal_a(3), b=>SS_Decimal_b(3), c=>SS_Decimal_c(3),
        d=>SS_Decimal_d(3), e=>SS_Decimal_e(3), f=>SS_Decimal_f(3), g=>SS_Decimal_g(3));
111
112  end Behavioral;
```

Counter process:

```
Counter_proc: process (CLK)
begin
  if (CLK'event and CLK='1') then
    if (RST='1') then
      TMP1 <= (others => '0');
    elsif (Load='1') then
      TMP1 <= unsigned(DA);
    elsif (TMP1="0011111100111") then
      TMP1 <= (others => '0');
    else
      TMP1 <= TMP1+"0000000000001";
    end if;
  end if;
end process;
```

Binary To BCD process:

```
binary_to_bcd_proc: process (TMP1)
  variable bcd: unsigned(11 downto 0); --bcd vector, with 4 bits for each display
  variable TMP: unsigned(11 downto 0); --auxiliary vector, to copy the input and operate with it
begin
  bcd := (others => '0');      -- ADDED for EVERY CONVERSION
  TMP := TMP1;                -- ADDED for EVERY CONVERSION

  for i in 0 to 11 loop

    --bit shifting. Copy the bint MSB in the bcd LSB
    bcd(11 downto 0) := bcd(10 downto 0) & TMP(11);
    TMP(11 downto 0) := TMP(10 downto 0) & '0';

    if i < 11 and bcd(3 downto 0) > "0100" then
      bcd(3 downto 0) := bcd(3 downto 0) + 3;
    end if;
    if i < 11 and bcd(7 downto 4) > "0100" then
      bcd(7 downto 4) := bcd(7 downto 4) + 3;
    end if;
    if i < 11 and bcd(11 downto 8) > "0100" then
      bcd(11 downto 8) := bcd(11 downto 8) + 3;
    end if;

    TMP2 <= bcd;
  end loop;
end process;
```

Binary-to-BCD Converter

Double-Dabble Binary-to-BCD Conversion Algorithm

Basic Idea

- $Y \leftarrow X$, X is a 4-bit binary number
 - Y is a 4-bit binary number (Binary to binary)
 \Rightarrow can be done by only shifting
 ex: $1011 \leftarrow 1011$ (shift left 4 times)
 - Y is a BCD number (Binary to BCD)
 $\therefore X: 0000 \sim 1111$,
 $\therefore Y: 00 \sim 15$ (two BCD digits, at least 5 bits)
 ex: $01000 \leftarrow 1000$
 $\quad ? \leftarrow 1011$

```

if (U > 4)
    then U=U+3;
Shift left;
    
```

if (U > 4) then U will be Out of range after "shift left"

Y				X			
				1	0	1	1
			1	0	1	1	
		1	0	1	1		
	1	0	1	1			
1	0	1	1				

U				X			
				1	0	1	1
			1	0	1	1	
		1	0	1	1		
	1	0	1	1			
1	0	1	1				

Shift left

$U \leftarrow U * 2 + X[3]$

Out of range
 $U = U + 6;$

Double-Dabble Binary-to-BCD Conversion Algorithm

Shift and Add-3 Algorithm (consider 8-bit binary)

1. Shift the binary number left one bit.
2. If 8 shifts have taken place, the BCD number is in the *Hundreds*, *Tens*, and *Units* column.
3. If the binary value in any of the BCD columns is 5 or greater, add 3 to that value in that BCD column.
4. Go to 1.

Example:

Operation	Hundreds	Tens	Units	8 bits Binary			
HEX				F		F	
Start				1 1 1 1	1 1 1 1		

Steps to convert an 8-bit binary number to BCD

Operation	Hundreds	Tens	Units	Binary			
HEX				F		F	
Start				1 1 1 1	1 1 1 1		
Shift 1			1	1 1 1 1	1 1 1		
Shift 2			1 1	1 1 1 1	1 1		
Shift 3			1 1 1	1 1 1 1	1		
Add 3			1 0 1 0	1 1 1 1	1		
Shift 4		1	0 1 0 1	1 1 1 1			
Add 3		1	1 0 0 0	1 1 1 1			
Shift 5		1 1	0 0 0 1	1 1 1			
Shift 6		1 1 0	0 0 1 1	1 1			
Add 3		1 0 0 1	0 0 1 1	1 1			
Shift 7	1	0 0 1 0	0 1 1 1	1			
Add 3	1	0 0 1 0	1 0 1 0	1			
Shift 8	1 0	0 1 0 1	0 1 0 1				
BCD	2	5	5				

Example of converting hex E to BCD

Operation	Tens	Units	Binary
HEX			E
Start			1 1 1 0
Shift 1		1	1 1 0
Shift 2		1 1	1 0
Shift 3		1 1 1	0
Shift 4		1 1 1 0	
6		0 1 1 0	
Add 6	1	0 1 0 0	
BCD	1	4	

Steps to convert a 6-bit binary number to BCD

1. Clear all bits of z to zero
2. Shift B left 3 bits
 $z[8:3] = B[5:0];$
3. Do 3 times

if $Units > 4$
 then add 3 to $Units$
 (note: $Units = z[9:6]$)

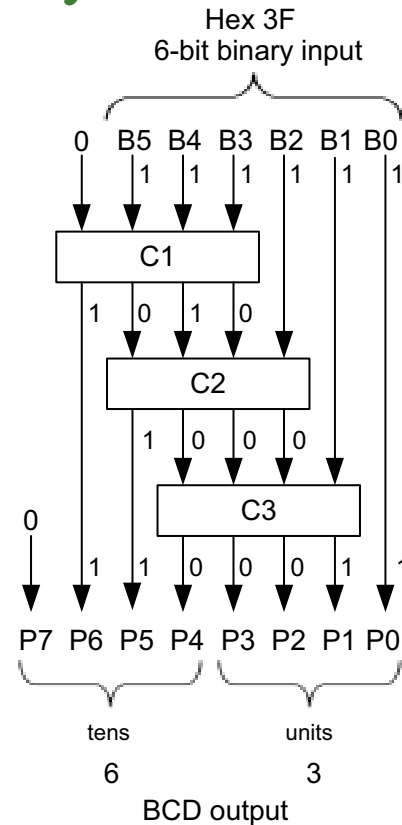
 Shift z left 1 bit
4. $Tens = P[6:4] = z[12:10]$
 $Units = P[3:0] = z[9:6]$

How to implement?

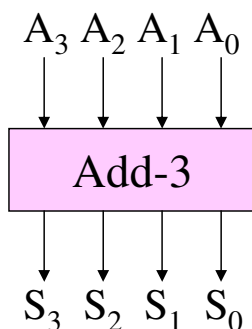
Operation	Tens	Units	Binary
B			5 4 3 2 1 0
HEX			3 F
Start			1 1 1 1 1 1
Shift 1		1	1 1 1 1 1
Shift 2		1 1	1 1 1 1
Shift 3		1 1 1	1 1 1
Add 3		1 0 1 0	1 1 1
Shift 4	1	0 1 0 1	1 1
Add 3	1	1 0 0 0	1 1
Shift 5	1 1	0 0 0 1	1
Shift 6	1 1 0	0 0 1 1	
BCD	6	3	
P	7 4	3 0	
z	13 10	9 6 5 0	

Steps to convert a 6-bit binary number to BCD (Cont'd)

Operation	Tens	Units	Binary
B			5 4 3 2 1 0
HEX			3 F
Start			1 1 1 1 1 1
Shift 1		1	1 1 1 1 1
Shift 2		1 1	1 1 1 1
Shift 3		1 1 1	1 1 1
Add 3		1 0 1 0	1 1 1
Shift 4	1	0 1 0 1	1 1
Add 3	1	1 0 0 0	1 1
Shift 5	1 1	0 0 0 1	1
Shift 6	1 1 0	0 0 1 1	
BCD	6	3	
P	7 4	3 0	
z	13 10	9 6	5 0



Truth table for Add-3 Module



A ₃	A ₂	A ₁	A ₀	S ₃	S ₂	S ₁	S ₀
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

K-Map for S_3

A_3	A_2	A_1	A_0	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

$A_3 \backslash A_1 A_0$	00	01	11	10
00				
01		1	1	1
11	x	x	x	x
10	1	1	x	x

$$S_3 = A_3 + A_2 A_0 + A_2 A_1$$

$$S_2 = A_3 A_0 + A_2 A_1' A_0'$$

$$S_1 = A_3 A_0' + A_2' A_1 + A_1 A_0$$

$$S_0 = A_3 A_0' + A_3' A_2' A_0 + A_2 A_1 A_0'$$

Component: Binary to 7Segment for HEX Output:

VHDL CODE

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  -- common cathode 7Segment
5
6  entity Binary_To_7S_HEX is
7      port (Binary: in std_logic_vector (3 downto 0);
8            a,b,c,d,e,f,g: out std_logic);
9  end Binary_To_7S_HEX;
10
11  architecture Behavioral of Binary_To_7S_HEX is
12  signal TMP: std_logic_vector(6 downto 0);
13
14  begin
15
16  process(Binary)
17  begin
18      case Binary is
19          when "0000" =>
20              TMP <= "11111110" ; -- 0 HEX
21          when "0001" =>
22              TMP <= "01100000" ; -- 1 HEX
23          when "0010" =>
24              TMP <= "1101101" ; -- 2 HEX
25          when "0011" =>
26              TMP <= "1111001" ; -- 3 HEX
27          when "0100" =>
28              TMP <= "0110011" ; -- 4 HEX
29          when "0101" =>
30              TMP <= "1011011" ; -- 5 HEX
31          when "0110" =>
32              TMP <= "1011111" ; -- 6 HEX
33          when "0111" =>
34              TMP <= "1110001" ; -- 7 HEX
35          when "1000" =>
36              TMP <= "1111111" ; -- 8 HEX
37          when "1001" =>
38              TMP <= "1110011" ; -- 9 HEX
39          when "1010" =>
40              TMP <= "1110111" ; -- A HEX
41          when "1011" =>
42              TMP <= "0011111" ; -- B HEX
43          when "1100" =>
44              TMP <= "1001110" ; -- C HEX
45          when "1101" =>
46              TMP <= "0111101" ; -- D HEX
47          when "1110" =>
48              TMP <= "1001111" ; -- E HEX
49          when "1111" =>
50              TMP <= "1000111" ; -- F HEX
51          when others =>
52              TMP <= "00000000" ;
53      end case;
54  end process;
55
56  a <= TMP(6) ;
57  b <= TMP(5) ;
58  c <= TMP(4) ;
59  d <= TMP(3) ;
60  e <= TMP(2) ;
61  f <= TMP(1) ;
62  g <= TMP(0) ;
63
64  end Behavioral;
65
66
67
```

Component:BCD to 7Segment for Decimal Output:

VHDL CODE

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  -- common cathode 7Segment
5
6  entity BCD_To_7S_Decimal is
7      port (bcd: in std_logic_vector (3 downto 0);
8            a,b,c,d,e,f,g: out std_logic);
9  end BCD_To_7S_Decimal;
10
11  architecture Behavioral of BCD_To_7S_Decimal is
12  signal TMP: std_logic_vector(6 downto 0);
13
14  begin
15
16  process(bcd)
17  begin
18      case bcd is
19          when "0000" =>
20              TMP <= "1111110" ; -- 0 decimal
21          when "0001" =>
22              TMP <= "0110000" ; -- 1 Decimal
23          when "0010" =>
24              TMP <= "1101101" ; -- 2 Decimal
25          when "0011" =>
26              TMP <= "1111001" ; -- 3 Decimal
27          when "0100" =>
28              TMP <= "0110011" ; -- 4 Decimal
29          when "0101" =>
30              TMP <= "1011011" ; -- 5 Decimal
31          when "0110" =>
32              TMP <= "1011111" ; -- 6 Decimal
33          when "0111" =>
34              TMP <= "1110001" ; -- 7 Decimal
35          when "1000" =>
36              TMP <= "1111111" ; -- 8 Decimal
37          when "1001" =>
38              TMP <= "1110011" ; -- 9 Decimal
39          when others =>
40              TMP <= "0000000" ;
41      end case;
42  end process;
43
44  a <= TMP(6);
45  b <= TMP(5);
46  c <= TMP(4);
47  d <= TMP(3);
48  e <= TMP(2);
49  f <= TMP(1);
50  g <= TMP(0);
51
52  end Behavioral;
53
54
55
```

Counter 12BIT TestBench:

VHDL CODE

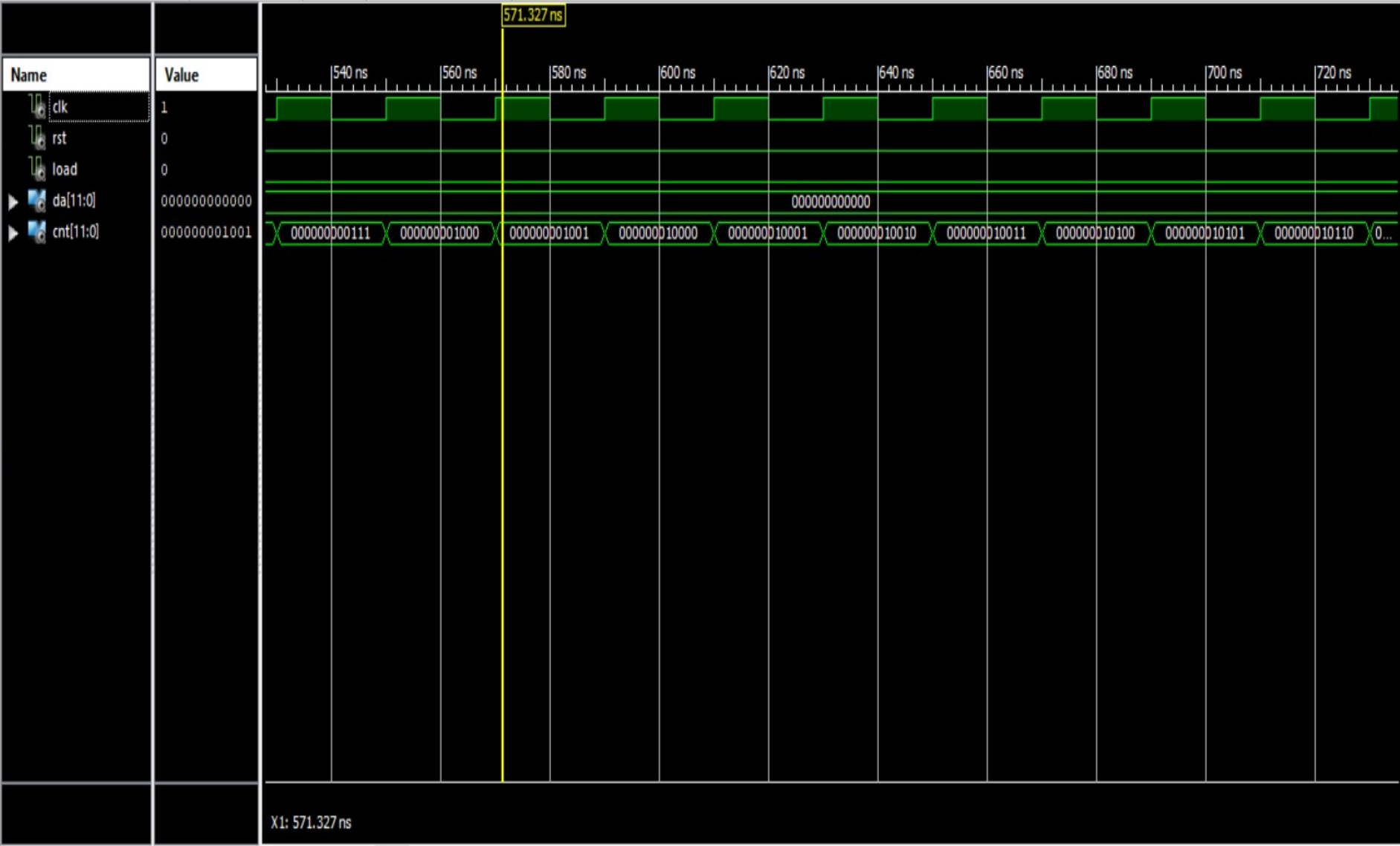
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY COUNTER_12BIT_TB IS
5  END COUNTER_12BIT_TB;
6
7  ARCHITECTURE behavior OF COUNTER_12BIT_TB IS
8
9      COMPONENT COUNTER_12BIT
10     PORT(
11         CLK,RST,Load : IN STD_LOGIC;
12         DA : IN STD_LOGIC_VECTOR(11 downto 0);
13         CNT_BCD : OUT STD_LOGIC_VECTOR(11 downto 0);
14
15         SS_HEX_a : OUT STD_LOGIC_VECTOR(3 downto 1);
16         SS_HEX_b : OUT STD_LOGIC_VECTOR(3 downto 1);
17         SS_HEX_c : OUT STD_LOGIC_VECTOR(3 downto 1);
18         SS_HEX_d : OUT STD_LOGIC_VECTOR(3 downto 1);
19         SS_HEX_e : OUT STD_LOGIC_VECTOR(3 downto 1);
20         SS_HEX_f : OUT STD_LOGIC_VECTOR(3 downto 1);
21         SS_HEX_g : OUT STD_LOGIC_VECTOR(3 downto 1);
22
23         SS_Decimal_a : OUT STD_LOGIC_VECTOR(3 downto 1);
24         SS_Decimal_b : OUT STD_LOGIC_VECTOR(3 downto 1);
25         SS_Decimal_c : OUT STD_LOGIC_VECTOR(3 downto 1);
26         SS_Decimal_d : OUT STD_LOGIC_VECTOR(3 downto 1);
27         SS_Decimal_e : OUT STD_LOGIC_VECTOR(3 downto 1);
28         SS_Decimal_f : OUT STD_LOGIC_VECTOR(3 downto 1);
29         SS_Decimal_g : OUT STD_LOGIC_VECTOR(3 downto 1));
30     END COMPONENT;
31
32     --Inputs
33     signal CLK : std_logic := '0';
34     signal RST : std_logic := '0';
35     signal Load : std_logic := '0';
36     signal DA : std_logic_vector(11 downto 0) := (others => '0');
37
38     --Outputs
39     signal CNT_BCD : STD_LOGIC_VECTOR(11 downto 0);
40
41     signal SS_HEX_a : STD_LOGIC_VECTOR(3 downto 1);
42     signal SS_HEX_b : STD_LOGIC_VECTOR(3 downto 1);
43     signal SS_HEX_c : STD_LOGIC_VECTOR(3 downto 1);
44     signal SS_HEX_d : STD_LOGIC_VECTOR(3 downto 1);
45     signal SS_HEX_e : STD_LOGIC_VECTOR(3 downto 1);
46     signal SS_HEX_f : STD_LOGIC_VECTOR(3 downto 1);
47     signal SS_HEX_g : STD_LOGIC_VECTOR(3 downto 1);
48
49     signal SS_Decimal_a : STD_LOGIC_VECTOR(3 downto 1);
50     signal SS_Decimal_b : STD_LOGIC_VECTOR(3 downto 1);
51     signal SS_Decimal_c : STD_LOGIC_VECTOR(3 downto 1);
52     signal SS_Decimal_d : STD_LOGIC_VECTOR(3 downto 1);
53     signal SS_Decimal_e : STD_LOGIC_VECTOR(3 downto 1);
54     signal SS_Decimal_f : STD_LOGIC_VECTOR(3 downto 1);
55     signal SS_Decimal_g : STD_LOGIC_VECTOR(3 downto 1);
56
57
58
59
60
61
62
63
64
65
```

Counter 12BIT TestBench:

VHDL CODE

```
74 BEGIN
75
76     -- Instantiate the Unit Under Test (UUT)
77     uut: COUNTER_12BIT PORT MAP (
78         CLK => CLK,
79         RST => RST,
80         Load => Load,
81         DA => DA,
82         CNT_BCD => CNT_BCD,
83         SS_HEX_a => SS_HEX_a,
84         SS_HEX_b => SS_HEX_b,
85         SS_HEX_c => SS_HEX_c,
86         SS_HEX_d => SS_HEX_d,
87         SS_HEX_e => SS_HEX_e,
88         SS_HEX_f => SS_HEX_f,
89         SS_HEX_g => SS_HEX_g,
90         SS_Decimal_a => SS_Decimal_a,
91         SS_Decimal_b => SS_Decimal_b,
92         SS_Decimal_c => SS_Decimal_c,
93         SS_Decimal_d => SS_Decimal_d,
94         SS_Decimal_e => SS_Decimal_e,
95         SS_Decimal_f => SS_Decimal_f,
96         SS_Decimal_g => SS_Decimal_g
97     );
98
99     CLK <= not(CLK) after 10 ns;
100     RST <= '1', '0' after 400 ns;
101     DA <= x"000", x"02D" after 900 ns;
102     Load <= '0', '1' after 920 ns, '0' after 1040 ns;
103
104
105 END;
```

Simulation, just for Binary to BCD:

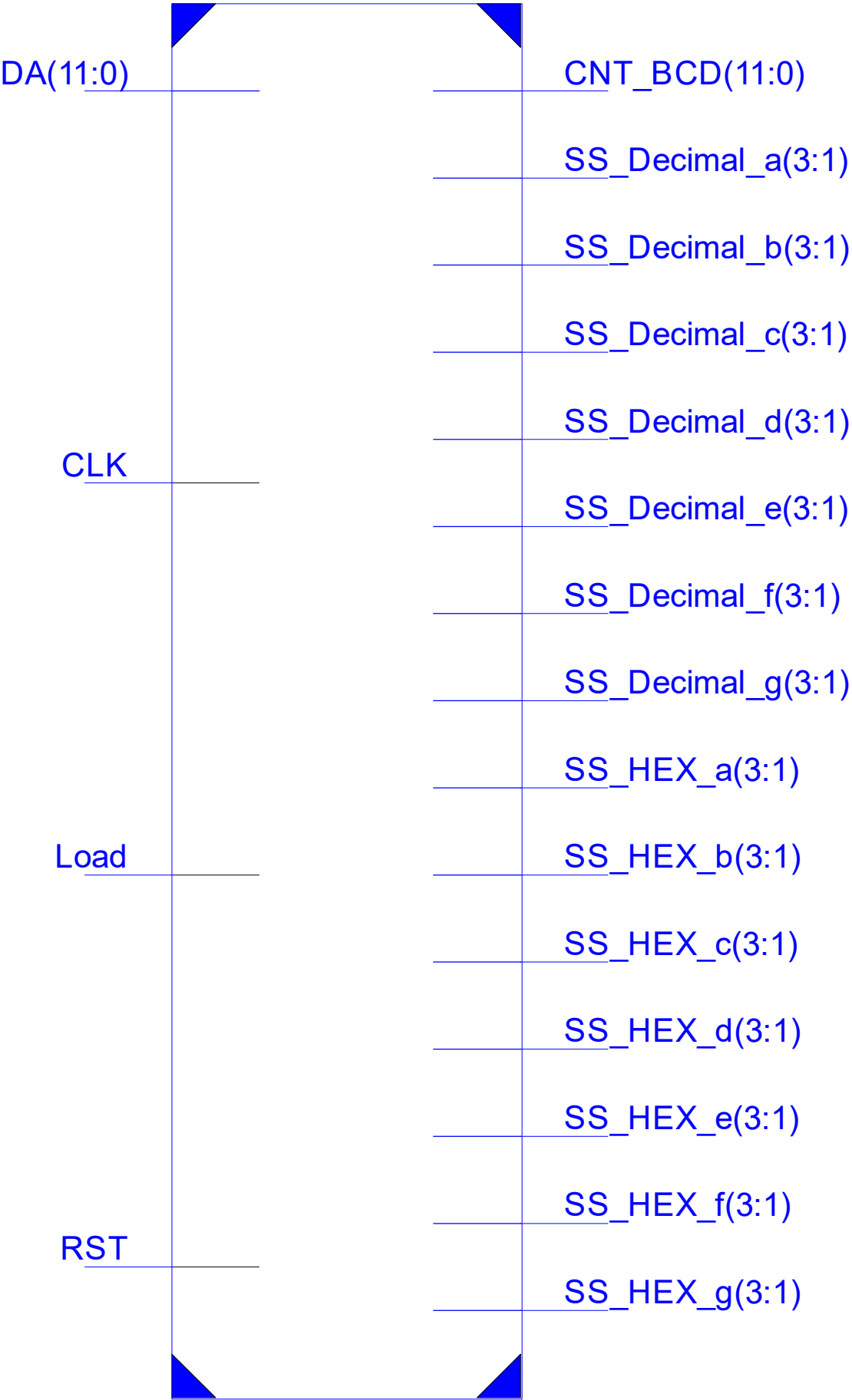


Simulation:

Name	Value	480 ns	500 ns	520 ns	540 ns	560 ns	580 ns	600 ns	620 ns	640 ns	660 ns	
clk	0											
rst	0											
load	0											
da[11:0]	000000000000											
cnt_bcd[11:0]	000000000111											
ss_hex_a[3:1]	111											
ss_hex_b[3:1]	111											
ss_hex_c[3:1]	111											
ss_hex_d[3:1]	110											
ss_hex_e[3:1]	110											
ss_hex_f[3:1]	110											
ss_hex_g[3:1]	001											
ss_decimal_a[3:1]	111											
ss_decimal_b[3:1]	111											
ss_decimal_c[3:1]	111											
ss_decimal_d[3:1]	110											
ss_decimal_e[3:1]	110											
ss_decimal_f[3:1]	110											
ss_decimal_g[3:1]	001											

X1: 541.000 ns

COUNTER_12BIT



COUNTER_12BIT

Synthesize & RTL Schematic:

