Department of Technology

Coursework report

Coursework Title:	**Second Mandatory Assignment--VHDL Programing (Autumn 2015)**
Date of Submission:	**No. 19. 2015**
I hereby certify that the work described in this report is my own work.	

Submitted by

Name of the Student:	Qinghui Liu (Brian Liu)
Student Number:	888087
Email Address:	qinghui.liu@student.hbv.no
Date of Completion:	Nov. 17, 2015

ABSTRACT

This report is about the solutions on the secondary assignment – VHDL programming (autumn 2015), which mainly contains two parts: one is theoretical part, and another one is practical part.

Theoretical part is a problem that need to identify all errors contained in given VHDL code.

Practical part contains 6 VHDL programming tasks as below:

1. Implement an ALU capable of performing from 8 to 15 different operations.
2. Implement a 16 to 4 multiplexer that receives four vectors of four elements each and transmits one of those four-element vector at a time.
3. Implement a four digits common anode BCD to seven-segment display decoder.
4. Use 2 and 3 as components to implement the four-digit seven-segment display decoder.
5. Use a full-adder as component (implemented in one single file) to implement a four-bit ripple-carry adder.
6. Repeat 5, but implementing a four-bit ripple-carry adder/subtractor instead.

This report will give detail solutions to each question and task.

Table of Contents

VHDL INTRODUCTION

VHDL OVERVIEW

VHDL consists of an entity which can contain other entities as components of the top-level entity. Each entity is modeled by an entity declaration and an architecture body, as Fig. 1.0.

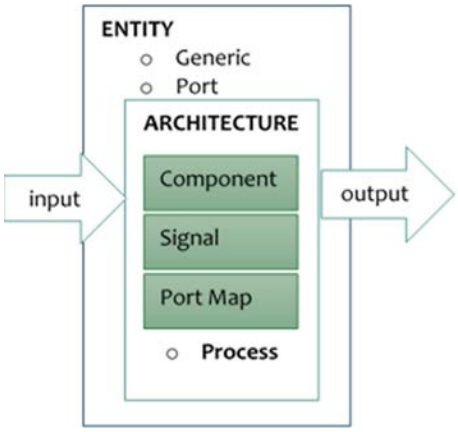


Fig. 1.0 VHDL Model

ENTITY

An ENTITY represents a template for a hardware block. It describes just the outside view of a hardware module – namely its interface with other modules in terms of input and output signals.

The inner operation of the entity is described by an ARCHITECTURE associated with it.

ARCHITECTURE

An ARCHITECTURE describes how an ENTITY operates. It can describe an entity in a structural style, behavioural style or mixed style.

TEST BENCH

To simulate a design containing a core, create a test bench file. The test bench should instantiate the top level module and should contain stimuli to drive the input ports of the design.

1 THEORETICAL PART

LIST OF ERRORS:

o	Line 3: should remove comment, need use IEEE.numeric_std.ALL;
o	Line 5: missing 'is' at the end of line
o	Line 7: missing ';' at the end of line
o	Line 9: misplacing ';' before the last ')', and missing ';' at the end of line
o	Line 10: missing ';' at the end of line
o	Line 13: missing '>' before '0'
o	Line 15: 'Q' is incompatible with 'tmp', the right statement should be "Q <= std_logic_vector(tmp);"
o	Line 16: process() missing sensitivity list, should be -- process (CLK, CLR)
o	Line 19: "CLR = 1" is wrong, should be "if(CLR = '1')" and missing "then" at the end of line
o	Line 24: should use "else" instead of "elsif"
o	Line 28: should use "end process;"

CORRECT CODE

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.numeric_std.ALL;
4
5  entity counter_bin is
6      generic (N: integer := 4;
7               M : integer := 10);
8      port ( CLK, CLR : in STD LOGIC;
9             Q : out STD LOGIC VECTOR (N-1 downto 0));
10 end counter_bin;
11
12 architecture Behavioral of counter_bin is
13 signal tmp : unsigned (N-1 downto 0) := (others => '0');
14 begin
15     Q <= std logic vector(tmp);
16     process(CLK, CLR)
17         begin
18             if (CLK'event and CLK = '1') then
19                 if (CLR = '1') then
20                     tmp <= (others => '0');
21                 elsif (tmp = (M-1)) then
22                     tmp <= (others => '0');
23                 else
24                     tmp <= tmp +1;
25                 end if;
26             end if;
27         end process;
28 end Behavioral;
29
```

2. PRACTICAL PART

1. ALU DESIGN

OVERVIEW

An Arithmetic/Logic Unit (ALU) is a multipurpose device capable of providing several different arithmetic and logic operations. The specific operation to be performed can be chosen by the mode select inputs.

ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs. (Wikipedia, n.d.)

32-BIT ALU BLOCK DIAGRAM

The ALU will be designed has 4 inputs: **clk, A(32), B(32), opco(4),** and 2 outputs: **Y(32)** and **nzco(4).** As Fig. 1.1.

The **clk** input is clock signal which rising edge can trigger operations. The ALU operation is specified by the **opco(4)** input. **A** and **B** input store the two operands for ALU.

The **Y(32)** output stores the operation result specified by **opco(4)** input. The **nzco(4)** output contains flag values:

- **nzco(3):** "Negative" flag, which indicates the result of an arithmetic operation is negative.
- **nzco(2):** "Zero" flag, which indicates all bits of the Y bus are logic zero.
- **nzco(1):** "Carry" flag, which conveys the carry or borrow resulting from an addition/subtraction.
- **nzco(0):** "Overflow" flag, which indicates the result has exceeded the range of the Y bus.

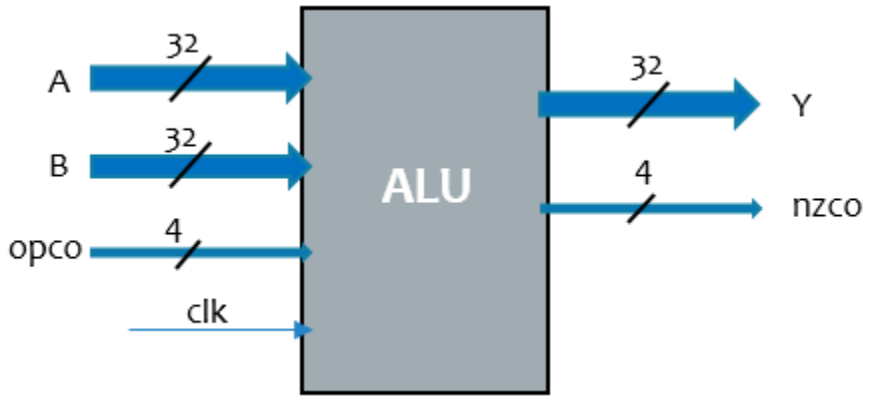


Fig. 1.1 ALU Diagram

TABLE OF ALU OPERATIONS SEPCIFICATION

NO.	Opcode	Function	Description
1	0000	Addition	Y<=A + B
2	0001	Subtraction	Y<=A - B
3	0010	Increment A	Y<=A+1
4	0011	Increment B	Y<=B+1
5	0100	Decrement A	Y<=A-1
6	0101	Decrement B	Y<=B-1
7	0110	Transfer A	Y<=A
8	0111	Transfer B	Y<=B
9	1000	AND	Y<=A and B
10	1001	OR	Y<= A or B
11	1010	NOT A	Y<= not A
12	1011	NOT B	Y<= not B
13	1100	A NAND B	Y<= A nand B
14	1101	A NOR B	Y<= A nor B
15	1110	A XOR B	Y<= A xor B
16	1111	A EX-NOR B	Y<=not (A xor B)

Fig. 1.2 ALU Operations Spec.

ENTITY OF ALU

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_unsigned.ALL;
23
24  entity ALU is
25     generic (TOTAL_BITS : natural := 32);
26     port (
27           clk   : in  std_logic;
28           opco  : in  std_logic_vector(3 downto 0);
29           A     : in  std_logic_vector(TOTAL_BITS - 1 downto 0);
30           B     : in  std_logic_vector(TOTAL_BITS - 1 downto 0);
31           Y     : out std_logic_vector(TOTAL_BITS - 1 downto 0);
32           nzco  : out std_logic_vector(3 downto 0)
33           );
34  end ALU;
```

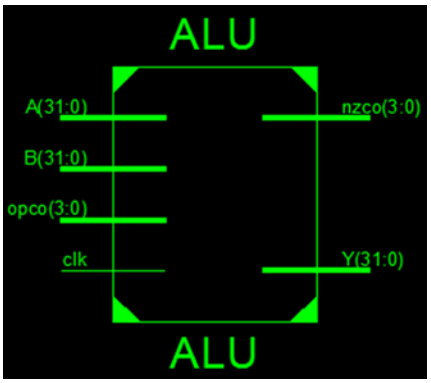


Fig. 1.4 Entity of 32-bit ALU

ARCHITECTURE OF ALU

```
36  architecture alu_arc of ALU is
37  begin
38     process (clk,A,B,opco)
39        variable temp : std_logic_vector(TOTAL_BITS downto 0):= (others =>'0');
40        variable yv: std_logic_vector(TOTAL_BITS-1 downto 0);
41        variable cfv, zfv: std_logic;
42     begin
43        zfv := '0';
44        if rising_edge(clk) then
45           case opco is
46              when "0000" =>                    -- A+B
47                 temp := ('0' & A) + ('0' & B);
48                 yv := temp(TOTAL_BITS-1 downto 0);
49                 cfv:= temp(TOTAL_BITS);
50                 nzco(0) <= yv(TOTAL_BITS-1)
51                            xor cfv
52                            xor A(TOTAL_BITS -1)
53                            xor B(TOTAL_BITS -1);
54                 nzco(1) <= cfv;
55              when "0001" =>                    -- A-B
56                 temp := ('0' & A) - ('0' & B);
57                 yv := temp(TOTAL_BITS-1 downto 0);
58                 cfv:= temp(TOTAL_BITS);
59                 nzco(0) <= yv(TOTAL_BITS-1)
60                            xor cfv
61                            xor A(TOTAL_BITS -1)
62                            xor B(TOTAL_BITS -1);
63                 nzco(1) <= cfv;
64              when "0010" =>                    -- A+1
65                 temp := ('0' & A) + 1;
66                 yv := temp(TOTAL_BITS-1 downto 0);
67                 cfv:= temp(TOTAL_BITS);
68                 nzco(0) <= yv(TOTAL_BITS-1)
69                            xor cfv
70                            xor A(TOTAL_BITS -1);
71                 nzco(1) <= cfv;
72              when "0011" =>                    -- B+1
73                 temp := ('0' & B) + 1;
74                 yv := temp(TOTAL_BITS-1 downto 0);
75                 cfv:= temp(TOTAL_BITS);
76                 nzco(0) <= yv(TOTAL_BITS-1)
77                            xor cfv
78                            xor B(TOTAL_BITS -1);
79                 nzco(1) <= cfv;
80              when "0100" =>                    -- A-1
81                 temp := ('0' & A) - 1;
82                 yv := temp(TOTAL_BITS-1 downto 0);
83                 cfv:= temp(TOTAL_BITS);
84                 nzco(0) <= yv(TOTAL_BITS-1)
85                            xor cfv
86                            xor A(TOTAL_BITS -1);
87                 nzco(1) <= cfv;
```

```
114             when "1111" =>                  -- EX-NOR
115                yv := not (A xor B);
116             when others =>
117                yv := A;
118          end case;
119
120          for i in 0 to TOTAL_BITS-1 loop
121             zfv:= zfv or yv(i);
122          end loop;
123
124          y <= yv;
125          nzco(2) <= not zfv;
126          nzco(3) <= yv(TOTAL_BITS-1);
127       end if;
128    end process;
129 end alu_arc;
```

Fig. 1.5 Architecture of 32-bit ALU

SIMULATION

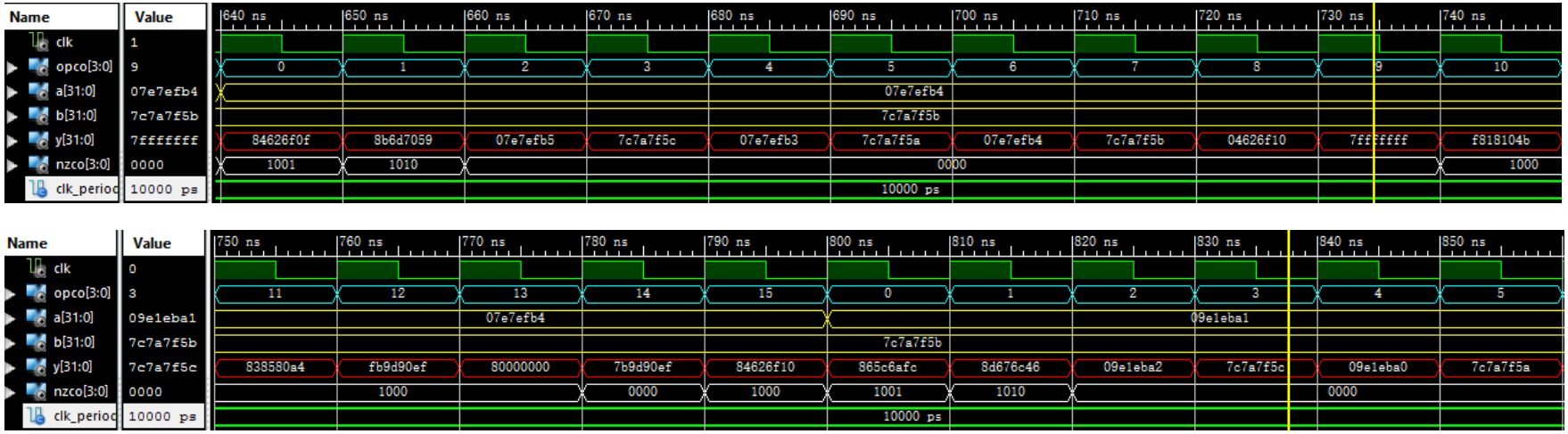


Fig. 1.6 ALU Simulation Results

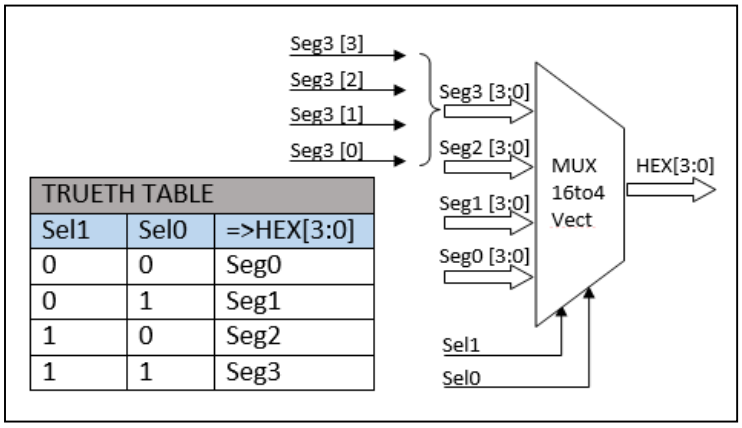
```
64      -- Clock process definitions
65      clk_process :process
66      begin
67          wait for clk_period/2;
68          clk <= not clk;
69      end process clk_process;
70
71
72      -- Stimulus process
73      opco_proc: process
74      begin
75          wait for clk_period;
76          opco <= opco+1;
77      end process opco_proc;
78
79      ab_proc: process
80      begin
81          b <= X"7C7A7F5B";
82          wait for clk_period*16;
83          a <= a+X"01F9FBED";
84      end process ab_proc;
```

Fig. 1.7 ALU Test Bench Code

2. MULTIPLEXER 16 TO 4

OVERVIEW

A multiplexer (or mux) is a device that selects one of several analog or digital input signals and forwards the selected input into a single line.



TRUETH TABLE		
Sel1	Sel0	=>HEX[3:0]
0	0	Seg0
0	1	Seg1
1	0	Seg2
1	1	Seg3

Fig. 2.1 True Table of Mux 16 to 4

VHDL CODE

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity Mux16to4v is
5       Port ( Seg0, Seg1, Seg2, Seg3 : in  STD_LOGIC_VECTOR (3 downto 0);
6              Sel : in  STD_LOGIC_VECTOR (1 downto 0);
7              HEX : out  STD_LOGIC_VECTOR (3 downto 0));
8   end Mux16to4v;
9
10  architecture Behavioral of Mux16to4v is
11
12  begin
13     HEX <= Seg0 when Sel="00" else
14            Seg1 when Sel="01" else
15            Seg2 when Sel="10" else
16            Seg3 ;
17
18  end Behavioral;
```

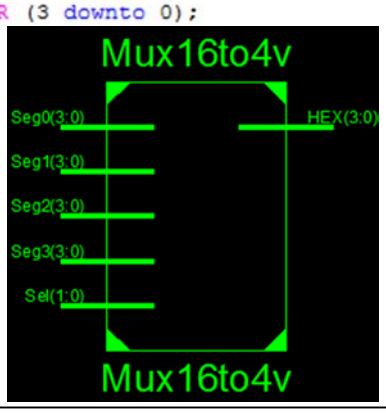


Fig. 2.2 VHDL Code of Mux 16 to 4

SIMULATION

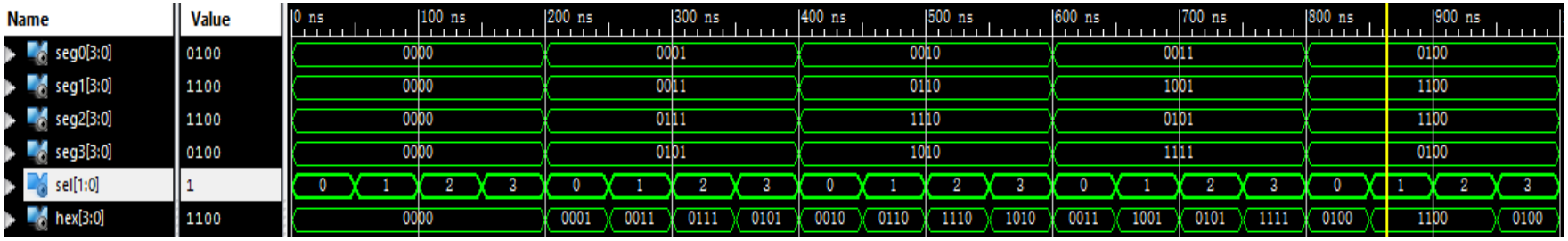


Fig. 2.3 Simulation Results of Mux16to4

```
54      -- Stimulus process
55      stim_proc_sel: process
56      begin
57         wait for 50 ns;
58         sel <= sel + 1;
59      end process;
60
61      stim_proc_seg: process
62      begin
63         wait for 200 ns;
64         seg0 <= seg0+1;
65         seg1 <= seg1+3;
66         seg2 <= seg2+7;
67         seg3 <= seg3+5;
68      end process;
```

Fig. 2.4 Mux16to4 Test Bench

3. BCD TO 7-SEGMENT DISPLAY

OVERVIEW

Binary Coded Decimal (BCD) to 7-Segment Display Decoder provides a very convenient way of displaying information or digital data with 7-segment LED (Light Emitting Diode), as Fig. 3.1

A standard 7-segment LED display generally has 8 input connections, one for each LED segment and one that acts as a common terminal or connection for all the internal display segments. Some single displays have also have an additional input pin to display a decimal point in their lower right or left hand corner.

There are two important types of 7-segment LED digital display.

- The Common Cathode Display (CCD)
- The Common Anode Display (CAD)

They can be made to display a variety of numbers or characters, as Fig. 3.2

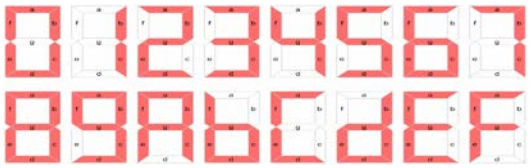


Fig. 3.2 7-Segment Display Elements

A truth table can be giving the segments that need to be illuminated in order to produce the required character as shown below, as Fig 3.3

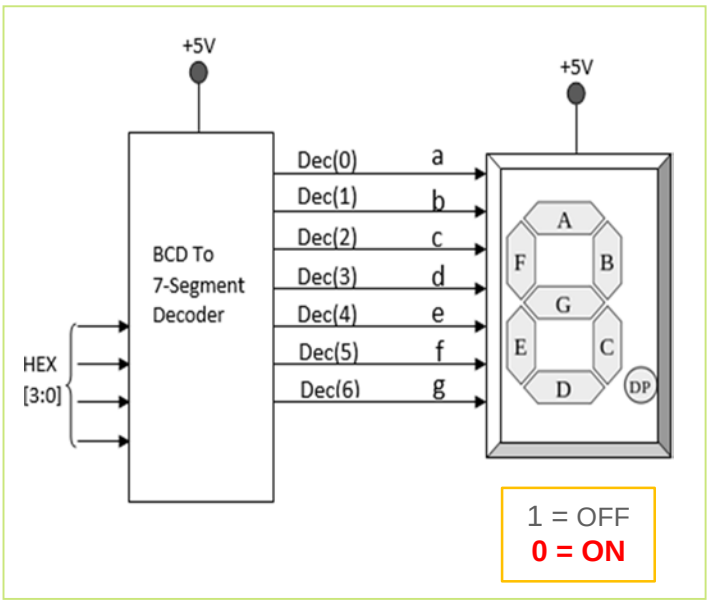


Fig. 3.1 BCD to 7-Segment Decoder

Character	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
b	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
d	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

Fig. 3.3 Truth Table for a 7-segment display

VHDL CODE

```
4   entity BCDto7seg is
5       Port ( HEX : in  STD_LOGIC_VECTOR(3 downto 0);
6              Dec : out  STD_LOGIC_VECTOR(6 downto 0)
7              );
8   end BCDto7seg;
10  architecture BCD_arc of BCDto7seg is
11     signal digit : std_logic_vector(6 downto 0);
12  begin
13     process (HEX)
14        variable sel : std_logic_vector(3 downto 0);
15     begin
16        sel := HEX;
17        case sel is
18           when x"0" => digit <= "0000001";--display_0;
19           when x"1" => digit <= "1001111";--display_1;
20           when x"2" => digit <= "0010010";--display_2;
21           when x"3" => digit <= "0000110";--display_3;
22           when x"4" => digit <= "1001100";--display_4;
23           when x"5" => digit <= "0100100";--display_5;
24           when x"6" => digit <= "0100000";--display_6;
25           when x"7" => digit <= "0001111";--display_7;
26           when x"8" => digit <= "0000000";--display_8;
27           when x"9" => digit <= "0000100";--display_9;
28           when x"A" => digit <= "0001000";--display_A;
29           when x"b" => digit <= "1100000";--display_b;
30           when x"C" => digit <= "0110001";--display_C;
31           when x"d" => digit <= "1000010";--display_d;
32           when x"E" => digit <= "0110000";--display_E;
33           when others => digit <= "0111000";--display_F;
34        end case;
35     end process;
36     Dec<= digit;
37  end BCD_arc;
```

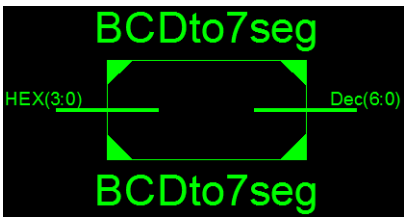


Fig. 3.4 BCDto7Seg Code

SIMULATION

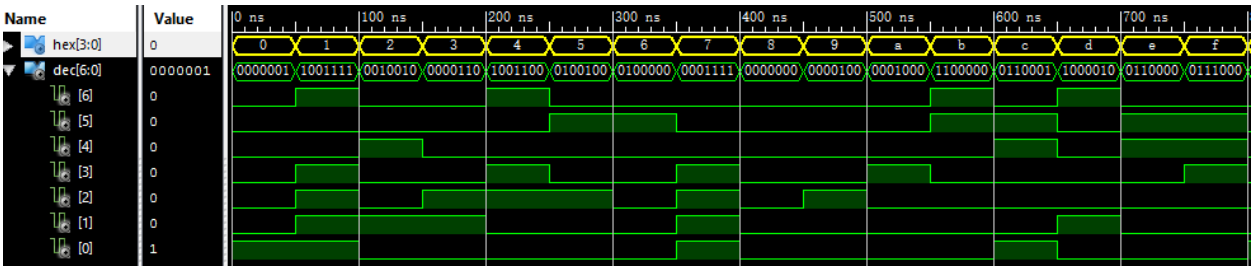


Fig. 3.5 Simulation Results

```
41     -- Stimulus process
42     stim_proc: process
43     begin
44        wait for 50 ns;
45        HEX <= HEX + 1;
46     end process;
```

Fig. 3.6 BCDto7Seg Test Bench

4. FOUR-DIGIT SEVEN-SEGMENT DISPLAY DECODER

OVERVIEW

This decoder contains several different parts. A counter is provided to provide the timing for the anode select lines (AN3-AN0). The two bits of the counter are also used to select the digit (and digit point) that is driven on the display. Using the top-two bits of a binary counter ensures that each digit will be driven for the same amount of time, as Fig. 4.1

This decoder also contains two multiplexers. One multiplexer is used to select the appropriate 4-bit segment input to display. This is a 4-bit, 4 to 1 multiplexer. The second multiplexer is used to select the appropriate decimal point to display. The controller also contains two decoders: one decoder is used to decode the four-bit value into the seven segment signals and the second decoder is used to decode the two-bit anode_select signal and generate the four independent anode control signals.

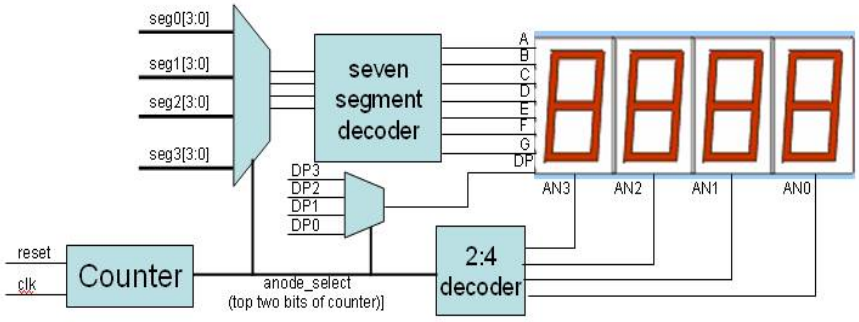


Fig. 4.1 Four_Bit-7Seg-Display Diagram

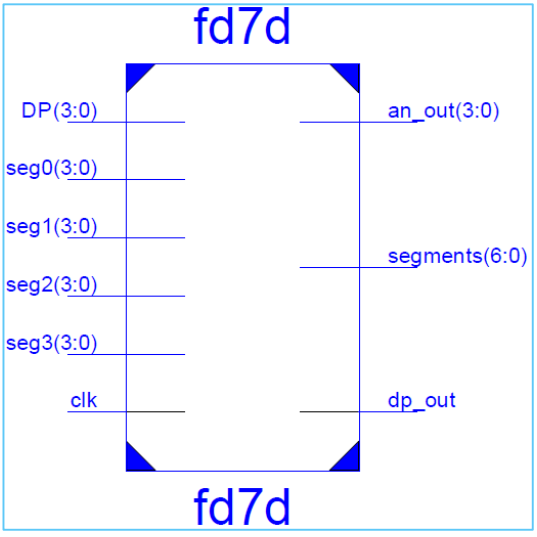


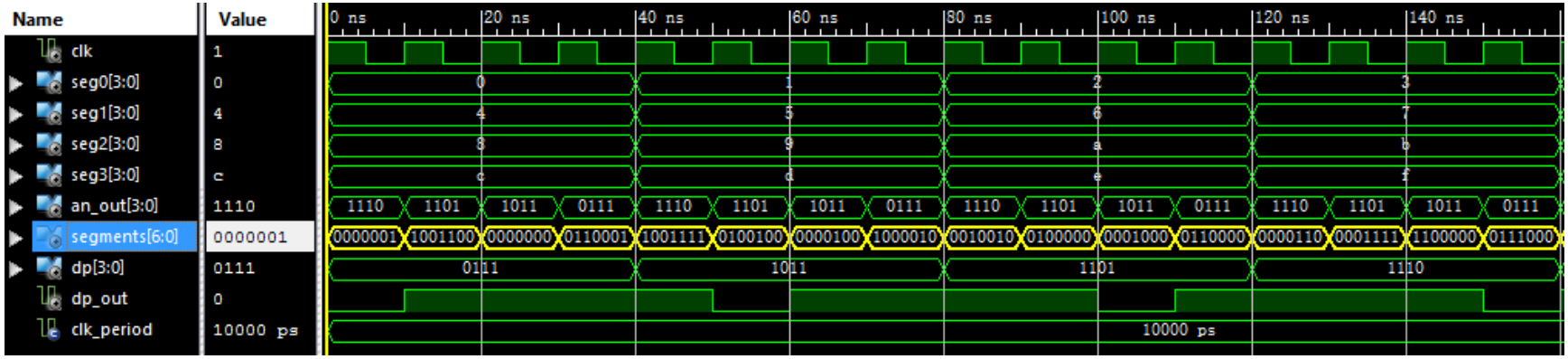
Fig. 4.2 Four_Bit-7Seg-Decocer

```vhdl
--- define component 2to4 decoder for anode_select
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Decoder2to4 is
   port(
         A : in std_logic_vector(1 downto 0);
         AN : out std_logic_vector(3 downto 0)
        );
end Decoder2to4;

architecture dec_arc of Decoder2to4 is
begin
   AN <= "1110" when A="00" else
         "1101" when A="01" else
         "1011" when A="10" else
         "0111";
end dec_arc;

--- define component Mux4to1 for  dp_select
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux4to1 is
   port(
         inp : in std_logic_vector(3 downto 0);
         sel : in std_logic_vector(1 downto 0);
         oup : out std_logic
        );
end Mux4to1;

architecture mux4_arc of Mux4to1 is
begin
   with sel select
    oup <= inp(3) when "00",
           inp(2) when "01",
           inp(1) when "10",
           inp(0) when others;

end mux4_arc;

--- define component Mux16to4vector for digit_select
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux16to4 is
   port(
         seg0, seg1, seg2,seg3 : in std_logic_vector(3 downto
         sel : in std_logic_vector(1 downto 0);
         o_hex: out std_logic_vector(3 downto 0)
        );

end Mux16to4;

architecture mux16_arc of Mux16to4 is

begin
   with sel select
    o_hex <= seg0 when "00",
             seg1 when "01",
             seg2 when "10",
             seg3 when others;

end mux16_arc;
```

```vhdl
--- build 4 digit 7-segment display
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.ALL;

entity fd7d is
   port (
         clk: in std_logic;
         seg0, seg1, seg2, seg3 : in std_logic_vector(3 downto 0);
         DP : in std_logic_vector(3 downto 0);
         segments : out std_logic_vector(6 downto 0);
         dp_out : out std_logic;
         an_out : out std_logic_vector(3 downto 0)
         );

end fd7d;
architecture Behavioral of fd7d is
   COMPONENT Mux4to1
   PORT(
      inp : IN std_logic_vector(3 downto 0);
      sel : IN std_logic_vector(1 downto 0);
      oup : OUT std_logic
      );
   END COMPONENT;
   COMPONENT Mux16to4
   PORT(
      seg0 : IN std_logic_vector(3 downto 0);
      seg1 : IN std_logic_vector(3 downto 0);
      seg2 : IN std_logic_vector(3 downto 0);
      seg3 : IN std_logic_vector(3 downto 0);
      sel : IN std_logic_vector(1 downto 0);
      o_hex : OUT std_logic_vector(3 downto 0)
      );
   END COMPONENT;
   COMPONENT Decoder2to4
   PORT(
      A : IN std_logic_vector(1 downto 0);
      AN : OUT std_logic_vector(3 downto 0)
      );
   END COMPONENT;
   COMPONENT BCDto7seg
   PORT(
      HEX : IN std_logic_vector(3 downto 0);
      Dec : OUT std_logic_vector(6 downto 0)
      );
   END COMPONENT;

   signal hex_in : std_logic_vector(3 downto 0);
begin
   sel_pro: process(clk)
   begin
      if ( clk'EVENT and clk = '1' ) then
         sel <= sel + '1';
      end if;
   end process ;

   comp0: Decoder2to4 port map(sel, an_out);
   comp1: Mux16to4 port map(seg0, seg1, seg2, seg3,
                            sel, hex_in);
   comp2: BCDto7seg port map(hex_in, segments);
   comp3: Mux4to1 port map(DP, sel, dp_out);

end Behavioral;
```

Fig. 4.3 Part of VHDL code for fd7d

SIMULATION




```
68    -- Clock process definitions
69    clk_process :process
70    begin
71       wait for clk_period/2;
72          clk <= not (clk);
73    end process clk_process;
74
75    -- segment process definitions
76    seg_proc: process
77    begin
78          wait for 4*clk_period;
79          seg0 <= seg0 + 1;
80          seg1 <= seg1 + 1;
81          seg2 <= seg2 + 1;
82          seg3 <= seg3 + 1;
83
84    end process seg_proc;
```

```
86    -- dp input process definitions
87    dp_proc: process (seg0, seg1, seg2,seg3)
88       variable no_dp : integer := 0;
89    begin
90          no_dp := no_dp + 1;
91
92          if (no_dp = 1) then
93             dp <= "1110";
94          elsif (no_dp = 2) then
95             dp <= "1101";
96          elsif (no_dp = 3) then
97             dp <= "1011";
98          elsif (no_dp = 4) then
99             dp <= "0111";
100         else
101            dp <= "1111";
102            no_dp := 0;
103         end if;
104   end process dp_proc;
```

Fig. 4.4 Simulation Result and Test Bench

5. FOUR-BIT RIPPLE-CARRY ADDER

OVERVIEW

A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. Full adder is a logic circuit that adds two input operand bits plus a Carry in bit and outputs a Carry out bit and a sum bit.

VHDL CODE

```
1  -- full_adder define
2  library IEEE;
3  use ieee.std_logic_1164.ALL;
4
5  entity full_adder is
6     port (
7           a, b, cin : in std_logic;
8           sum, cout: out std_logic
9           );
10 end full_adder;
11
12 architecture behavior of full_adder is
13 begin
14    sum <= (a XOR b) XOR cin;
15    cout <= (a AND b) or (a AND cin) or (b AND cin);
16 end behavior;
```

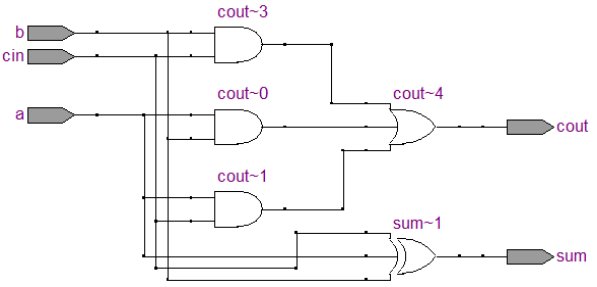


Fig. 5.1 Full Adder block and Code

```
22 entity nbit_adder is
23    generic ( N : natural := 4);  -- initiallize 4-bit adder
24    port(
25          A, B : in std_logic_vector(N-1 downto 0);
26          Cin  : in std_logic;
27          Sum  : out std_logic_vector(N-1 downto 0);
28          Cout : out std_logic
29         );
30 end nbit_adder;
31
32 architecture behavior of nbit_adder is
33    component full_adder
34       port(
35             a, b, cin : in std_logic;
36             sum, cout: out std_logic
37            );
38    end component;
39    signal co : std_logic_vector(N downto 0);
40 begin
41    co(0) <= Cin;
42    Cout <= co(N-1);
43
44    GEN: for i in 0 to N-1 generate
45       nb_adder: full_adder port map(A(i), B(i), co(i),
46                                     Sum(i),co(i+1) );
47    end generate GEN;
48
49 end behavior;
```

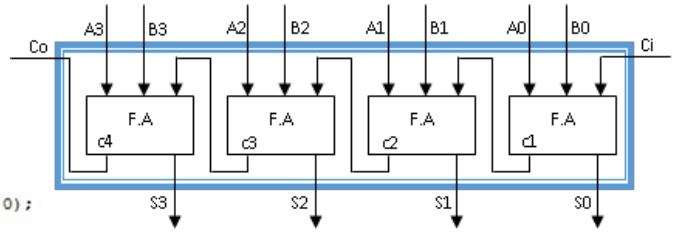
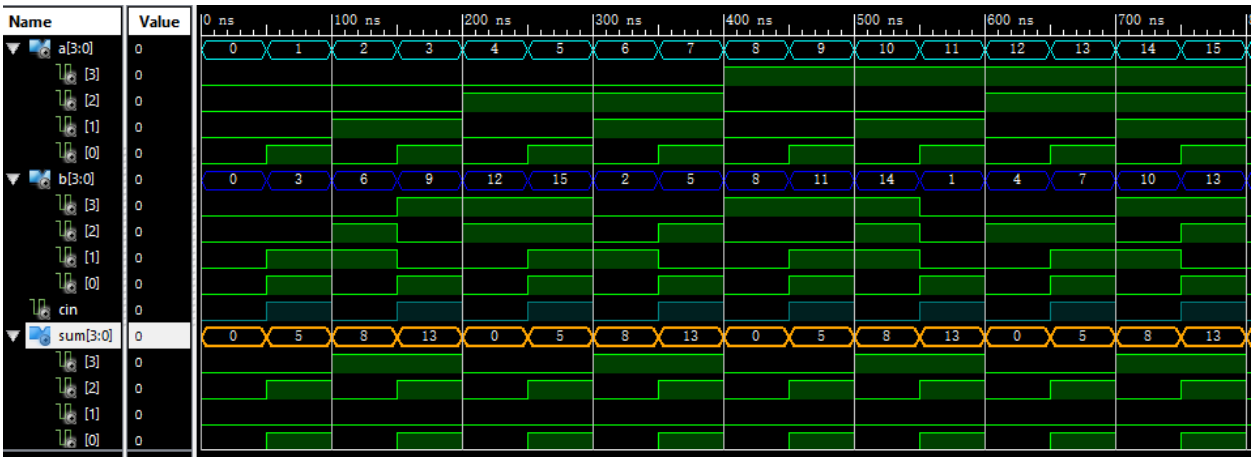


Fig. 5.2 Four-Bit Full Adder Block and Code




```
51      -- Stimulus process
52      stim_proc: process
53      begin
54         wait for 50 ns;
55         A <= A+1;
56         B <= B+3;
57         Cin <= not Cin;
58      end process;
```

Fig. 5.3 Simulation result and test bench

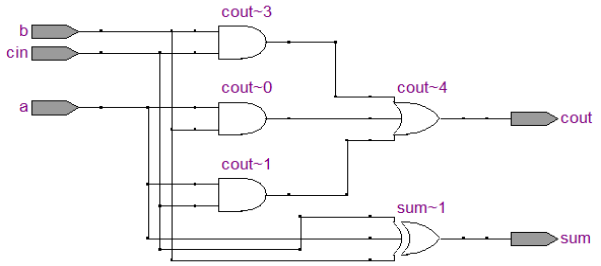
6. FOUR-BIT RIPPLE-CARRY ADDER/SUBTRACTOR

OVERVIEW

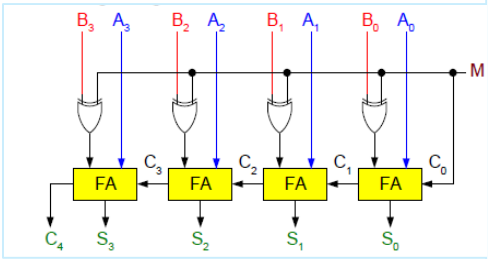
An adder–subtractor is a circuit that is capable of adding or subtracting numbers. A 4-bit ripple-carry adder–subtractor based on a 4-bit adder that performs two's complement on **A** when **MOD = 1** to yield **Sum = B- A.**

VHDL CODE

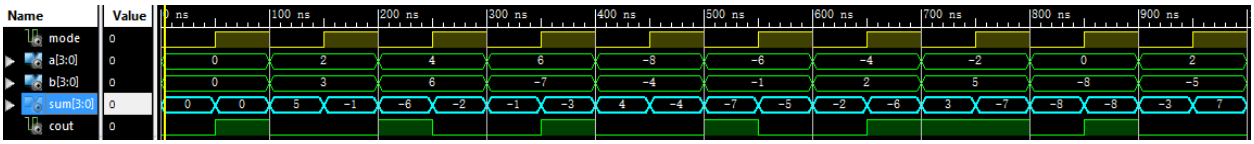
```vhdl
-- full_adder define
library IEEE;
use ieee.std_logic_1164.ALL;

entity full_adder is
   port (
         a, b, cin : in std_logic;
         sum, cout: out std_logic
         );
end full_adder;

architecture behavior of full_adder is
begin
   sum <= (a XOR b) XOR cin;
   cout <= (a AND b) or (a AND cin) or (b AND cin);
end behavior;
```




```vhdl
entity add_sub is
   generic ( N : natural := 4);  -- initiallize 4-bit adder/substractor
   port(
         A, B : in std_logic_vector(N-1 downto 0);
         Mode : in std_logic;
         Sum  : out std_logic_vector(N-1 downto 0);
         Cout : out std_logic
        );
end add_sub;

architecture fbas_arc of add_sub is
   component fulladder
      port(
            a, b, cin : in std_logic;
            sum, cout: out std_logic
           );
   end component;
   signal co   : std_logic_vector(N downto 0);
   signal xorB : std_logic_vector(N-1 downto 0);
begin
   co(0) <= Mode;
   Cout <= co(N-1);

   SUB: for i in 0 to N-1 generate
      mod_b: xorB(i) <= B(i) XOR Mode;
   end generate SUB;

   GEN: for i in 0 to N-1 generate
      nb_adder: fulladder port map(A(i), xorB(i), co(i),
                                   Sum(i),co(i+1) );
   end generate GEN;
end fbas_arc;
```



SIMULATION




```
50    -- Stimulus process
51    ab_proc: process
52    begin
53       wait for 100 ns;
54       A <= A+2;
55       B <= B+3;
56    end process;
57
58    mode_proc: process
59    begin
60       wait for 50 ns;
61       Mode <= not Mode;
62    end process;
```

Fig. 6.1 Simulation result and test bench

CONCLUSION

In this coursework, I have designed, implemented **a 32 bit ALU**, a **Mux16to4**, a **BCDto7Segment** Display, a **Four-digit 7-Segment Display Decoder**, a **4-Bit Ripple-Carry Adder**, and **4-bit Ripple-Carry Adder/Substractor**.

All the above design are then verified to see whether they match theoretically or not. All above given simulation waveforms show that they match completely thereby verifying our desired results.

Through this coursework, I have mastered the basic VHDL programming skills and simulation test bench method.

REFERENCES

[1]. https://www.youtube.com/watch?v=r8xVQ3ThQK8
[2]. Geetanjali, Nishant Tripathi，VHDL Implementation of 32-Bit Arithmetic Logic Unit (Alu)
[3]. http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-simple-alu.html
[4]. https://en.wikipedia.org/wiki/Arithmetic_logic_unit
[5]. https://en.wikipedia.org/wiki/Adder%E2%80%93subtractor
[6]. https://en.wikipedia.org/wiki/Adder_(electronics)
[7]. http://ece320web.groups.et.byu.net/labs/Lab3-SevenSegmentDisplay/SevenSegmentDecoder.html
[8]. Spartan-3 Starter Kit Board User Guide http://ece320web.groups.et.byu.net/resources/S3BOARD-rm.pdf
[9]. Nexys3 Reference Manual Revision: April 10, 2013
[10]. http://www.electronics-tutorials.ws/combination/comb_6.html
[11]. https://en.wikipedia.org/wiki/Multiplexer
[12]. http://www.circuitstoday.com/ripple-carry-adder
[13]. http://1.bp.blogspot.com/-Jq5PlCZIe18/TsLQcUfUe4I/AAAAAAAAANk/71sLNGItC3s/s1600/Capture6.PNG

APPENDICES

[1]. VHDL Source Code :  VHDL_ASG.rar