Assignment 5

Part-1 Multiplier

Due date: December 15, 2017

Group No: 5

Students:
Lalit Bhat: 01671833
Priya Parikh: 01679645

Objectives:

To design and implement a multiplier circuit which multiply two 4-bit number and displays the least significant bit first, followed by 4-most significant bits on Altera FPGA Cyclone IV using VHDL.

List Of components:

- Hardware used:

  Altera FPGA Cyclone IV board
  Power adapter
  Power supply

- Software used:
  Quartus Prime

Experimental Approach:

1. We first studied how to create a project in Quartus Prime.

2. After getting familiar with programming in VHDL we created individual projects of each components in the multiplier.

3. We tested each of the individual projects on the cyclone IV kit

4. After designing the individual codes, we described the top-level entity for the multiplier.

5. We assigned the signals and described the port maps.

6. After building the final code we ran several simulations to find any bugs.

7. We were able to multiply 2 4-bit numbers and get an 8 bit output.

VHDL Code:

1. Top Level Entity.

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY multiplier IS
5
6           PORT ( SW : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
7                  CLOCK_50 : IN  STD_LOGIC;
8                  LEDR : OUT STD_LOGIC_VECTOR(17 DOWNTO 0);
9                  HEX0 : OUT STD_LOGIC_VECTOR(0 TO 6);
10                 HEX1 : OUT STD_LOGIC_VECTOR(0 TO 6);
11           --    HEX2 : OUT STD_LOGIC_VECTOR(0 TO 6);
12                 HEX3 : OUT STD_LOGIC_VECTOR(0 TO 6);--Here we have described all the haedware ports
13                 HEX4 : OUT STD_LOGIC_VECTOR(0 TO 6);--we will be using for our code.
14                 HEX5 : OUT STD_LOGIC_VECTOR(0 TO 6);
15                 HEX6 : OUT STD_LOGIC_VECTOR(0 TO 6);
16                 HEX7 : OUT STD_LOGIC_VECTOR(0 TO 6);
17                 KEY : IN STD_LOGIC_VECTOR(3 DOWNTO 0));
18
19  END multiplier;
20
21
22
23  ARCHITECTURE Behavior OF multiplier IS
24
25      COMPONENT sevsegdecoder
26         port  (input0: in std_logic_vector(3 downto 0);--|Here we describe the components of
27                display: out std_logic_vector(0 to 6));  --|the seven segment decoder.
28      END COMPONENT;
```

```
31      COMPONENT piso
32         port ( clk,rst,load : in std_logic;         --|Here we describe the components of
33                inp : in std_logic_vector(3 downto 0);--|the parallel in - serial out shift register.
34                q : out std_logic);                  --|
35      END COMPONENT;
36
37
38      COMPONENT sipo
39         port(clk0,rst0 : in std_logic;               --|
40                a : in std_logic;                     --|Here we describe the components of
41                op : out std_logic_vector(7 downto 0));--|the serial in - parallel out shift register.
42      END COMPONENT;
43
44
45      COMPONENT alu
46         Port ( NUM1 : in  STD_LOGIC_VECTOR (4 downto 0);--|
47                NUM2 : in  STD_LOGIC_VECTOR (4 downto 0); --|Here we describe the components of the ALU
48                SUM : out  STD_LOGIC_VECTOR (4 downto 0));--|
49
50      END COMPONENT;
51
52
53      COMPONENT andgates
54         Port ( a : in std_logic_vector (3 downto 0);    --|
55                b : in std_logic;                        --|Here we describe the components of the
56                ab : out std_logic_vector (3 downto 0));  --|and gates.
57      END COMPONENT;
```

```
60        COMPONENT reg
61           port (D : in std_logic_vector(3 downto 0);   --|
62                 clk, rst : in std_logic;               --|Here we describe the components of the
63                 Q : out std_logic_vector(3 downto 0)); --|accumulator
64        END COMPONENT;
65
66        COMPONENT debounce
67
68           GENERIC (bouncetime  : INTEGER := 50000);
69           PORT (CLK, RST, sw  : IN STD_LOGIC;
70                 outp, invoutp : OUT STD_LOGIC);
71
72     END COMPONENT;
73
74   signal b0,clk0,clk_not :      std_logic;
75   --'b0' is lsb of input 'b',clk is a de-bounce clock.
76   signal aluin,regout :         std_logic_vector(3 downto 0);
77   --'aluin'-output of and logic,'regout'- output of accumulator
78   signal SUM0,aluin0,regout0 :  std_logic_vector(4 downto 0);
79   --'SUM0'output of alu,'aluin0'- alu input,'regout0'-alu input
80   Signal display :              std_logic_vector(7 downto 0);
81   --display used for seven segment decoder
```

```
83   begin
84
85    piso0:                     piso port map (clk0,SW(0),SW(1),SW(13 downto 10),b0);
86    andgates0:                 andgates port map (SW(17 downto 14),b0,aluin);
87                               aluin0 <= "00000" OR aluin;
88    alu0 :                     alu port map (aluin0,regout0,SUM0);
89    reg0 :                     reg port map (SUM0(4 downto 1),clk0,SW(0),regout);
90                               regout0 <= "00000" OR regout;
91    sipo0 :                    sipo port map (clk0,SW(0),SUM0(0),display);
92
93   -- sipo0:                  sipo port map (KEY(0),SW(0),x,y);
94   debounce0:                 debounce GENERIC MAP (bouncetime => 100000) PORT MAP (CLOCK_50,KEY(1),KEY(0),clk0,clk_not);
95                                    --default is 50000 (1ms@50MHz)
96
97
98   disp0: sevsegdecoder port map (SW(17 downto 14),HEX7);--Input 'A'.
99   disp1: sevsegdecoder port map (SW(13 downto 10),HEX6);--Input 'B'.
100  disp2: sevsegdecoder port map (aluin,HEX5);
101  disp3: sevsegdecoder port map (SUM0(4 downto 1),HEX4);--AND logic output.
102  disp4: sevsegdecoder port map (regout,HEX3);--displays accumulator output.
103  --disp5: singlebit port map (SUM0(0),HEX2);
104  disp6: sevsegdecoder port map (display(7 downto 4),HEX1);--displays the lower nibble.
105  disp7: sevsegdecoder port map (display(3 downto 0),HEX0);--Displays the upper nibble.
106
107
108  END Behavior;
```

2. Seven segment decoders.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   --This code describes the sevensegment decoder
4   entity sevsegdecoder is
5      port  (input0: in std_logic_vector(3 downto 0);
6         display: out std_logic_vector(0 to 6));
7   end sevsegdecoder;
8
9   architecture behavior of sevsegdecoder is
10  begin
11     with input0 select
12        display <=  "0000001" when "0000",--0
13                    "1001111" when "0001",--1
14                    "0010010" when "0010",--2
15                    "0000110" when "0011",--3
16                    "1001100" when "0100",--4
17                    "0100100" when "0101",--5
18                    "0100000" when "0110",--6
19                    "0001111" when "0111",--7
20                    "0000000" when "1000",--8
21                    "0000100" when "1001",--9
22                    "0001000" when "1010",--A
23                    "1100000" when "1011",--b
24                    "0110001" when "1100",--C
25                    "1000010" when "1101",--d
26                    "0110000" when "1110",--E
27                    "0111000" when others;--F
28  end behavior;
```

3. Parallel in serial out shift register.

```
1  library ieee;
2  use ieee.std_logic_1164.all,ieee.numeric_std.all;
3
4  --This code describes a parallel in and serial out shift register.
5
6  entity piso is
7
8     port ( clk,rst,load : in std_logic;
9           inp : in unsigned(3 downto 0);
10          --the above port is where the variable 'b' to be multiplied
11          --is loaded parallely
12          q : out std_logic);--this output is sent to the AND logic
13
14 end entity piso;
15
16 architecture behavior of piso is
17
18 begin
19
20       p0: process(clk,rst,load) is
21
22       variable reg : unsigned ( 3 downto 0) :="0000";
```

```
24       begin
25
26          if (rising_edge (clk)) then
27
28             if (rst = '1') then
29             reg := (others => '0');
30
31             elsif (load = '1') then
32             reg := inp; --4 bit parallel input assigned to a temp variable.
33
34             else
35             reg := shift_right (reg,1); --logic for shift.
36
37             end if;
38          end if;
39
40          q <= reg(0); --serial output.
41
42       end process p0;
43
44 end behavior;
```

4. Serial in parallel out shift register.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all,ieee.numeric_std.all;
3
4  --This code describes a serial in and parallel out shift register.
5
6  entity sipo is
7
8        port(clk0,rst0 : in std_logic;
9           a : in std_logic;                --single bit input
10          op : out unsigned(7 downto 0));  --stores the 8 bit final result
11
12
13 end entity sipo;
14
15 architecture Behavior of sipo is
16
17 begin
18    p1: process (clk0,rst0) is
19    variable temp: unsigned(7 downto 0) :="00000000";
```

```
20    begin
21
22       if rising_edge(clk0) then
23
24          if (rst0 = '1') then
25             temp := "00000000";
26             op<= temp; --Assign the single bit input to
27                        --a temp variable
28          else
29             temp := temp(6 downto 0) & a;
30             op(0) <= temp(7);--|
31             op(1) <= temp(6);--|
32             op(2) <= temp(5);--|
33             op(3) <= temp(4);--|-->logic for shift operation
34             op(4) <= temp(3);--|
35             op(5) <= temp(2);--|
36             op(6) <= temp(1);--|
37             op(7) <= temp(0);--|
38          end if;
39       end if;
40
41
42
43    end process p1;
44
45 end architecture Behavior;
```

5. ALU

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

--This code describes the ALU adder logic.

entity alu is
    Port ( NUM1 : in  STD_LOGIC_VECTOR (4 downto 0);-- 5-bit number
           NUM2 : in  STD_LOGIC_VECTOR (4 downto 0);-- 5-bit number
           SUM : out  STD_LOGIC_VECTOR (4 downto 0));-- 5 bit result

end alu;

architecture Behavioral of alu is
begin

      SUM <= NUM1+NUM2;


end Behavioral;
```

6. AND logic.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--This is the code for 4 bit AND operation
-- this operation is performed with 4 bit input 'a'
--is each anded with lsb of input 'b'

entity andgates is
    Port ( a : in std_logic_vector (3 downto 0);
           b : in std_logic;
           ab : out std_logic_vector (3 downto 0));
end andgates;

architecture Behavior of andgates is

   begin

      ab(0) <=a (0) and b;--|
      ab(1) <=a (1) and b;--|-->and logic
      ab(2) <=a (2) and b;--|
      ab(3) <=a (3) and b;--|

   end Behavior;

```

7. Accumulator.

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

--This is the code for the accumulator

entity reg is


      port (D : in std_logic_vector(3 downto 0);   -- the output of the ALU is connected to this port.
            clk, rst : in std_logic;               -- clock and reset.
            Q : out std_logic_vector(3 downto 0)); -- this output is sent back to the input for add
                                                   --operation used in multiplier logic.
end entity reg;


architecture Behaviour of reg is

begin
   p0: process (clk,rst) is
          begin
             if rising_edge(clk) then
                if (rst = '1') then
                   Q <= (others => '0'); --reset conditions
                else
                   Q <= D;  --set the input to the output Q
                end if;
             end if;

    end process p0;

end Behaviour;
```

8. De-bounce.

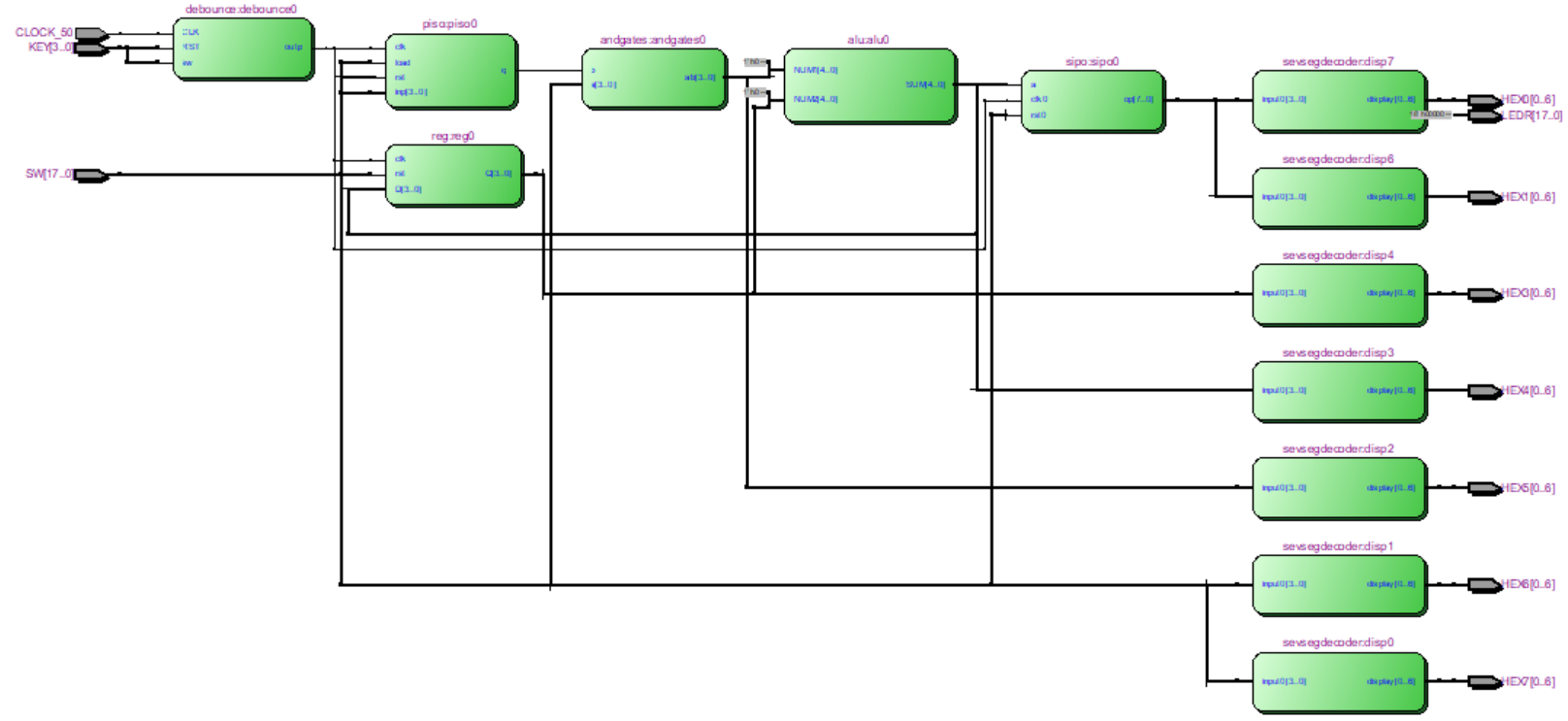
```
module debounce(
    CLK,
    RST,
    sw,
    outp,
    invoutp);

parameter bouncetime = 50000;
parameter clkwidth = $clog2(bouncetime);
input CLK;
input RST;
input sw;
output reg outp;
output reg invoutp;
reg [(clkwidth-1):0] count;
reg lsw; //last switch state

```

```
18    always@(posedge CLK or negedge RST)
19  begin
20        if(RST==0) begin
21           count <= bouncetime;
22        end
23        else begin
24           lsw<=sw;
25
26           if(lsw != sw) begin
27              count <= bouncetime;
28           end
29           else
30
31           if (count == 0) begin
32              outp <= sw;
33              invoutp <= ~sw;
34           end
35           else begin
36              count <= count - 1;
37           end;
38
39        end
40  end
41
42  endmodule
```

RTL –



Results:

During the demonstration the TA asked us to multiply different combinations of 2 four-bit numbers to check if our code executes accurately and it displays the answer.

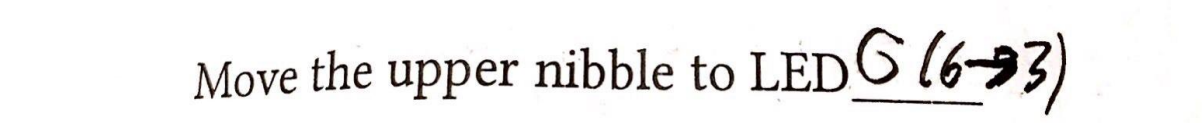
In each of these combinations we got our desired outputs.

Conclusion:

We successfully designed and simulated VHDL code to multiply 2 4-bit numbers and get an 8 bit output.

FTQs:

1. Lalit Bhat



To move the upper nibble to the led we need to add the following code:

a. Inside entity

```
ENTITY multiplier IS
    Port (LEDG: OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
End entity multiplier
```

b. Inside architecture

```
LEDG (6) <=display (3);
LEDG (5) <=display (2);
LEDG (4) <=display (1);
LEDG (3) <=display (0);
```

2. Priya Parekh

Move the upper nibble to display HEX 2.

In our code, HEX1 displays upper nibble. I need to change the code by adding following instruction to move the upper nibble to HEX2.

disp1: sevsegdecoder port map (SW (7 downto 4), HEX2);