CS 288: 4 - Bit ALU Design

Dheerendra Singh Rathor
120050033
dheeru.rathor14@gmail.com

March 4, 2014

Aim:

Use Xilinx ISE to design and simulate a module which can perform following operations on pair of 4-bit input (A and B).

Truth Table –

S3	S2	S1	S0	Operations
0	0	0	0	Complement A
0	0	0	1	Bit-wise And
0	0	1	0	Bitwise OR
0	0	1	1	Addition
0	1	0	0	Subtraction
0	1	0	1	Right logic shift for A
0	1	1	0	Left logic shift for A
0	1	1	1	Right logic shift for B
1	0	0	0	Left logic shift for B
1	0	0	1	Complement B

Procedure:

For designing the 4 bit Arithmetic Logic Unit, I first designed a 4 bit Adder which will be used for Addition and subtraction. For complement, Bit shift and bitwise operations I used sequential behavioral model.

Complement:
1' = 0
0' = 1

Bitwise AND:
0 AND x = 0
1 AND 1 = 1

Bitwise OR:
1 OR x = 1
0 OR 0 = 0

XOR:
1 XOR 1 = 0
0 XOR 0 = 0
1 XOR 0 = 1

0 XOR 1 = 1

Timing Diagrams

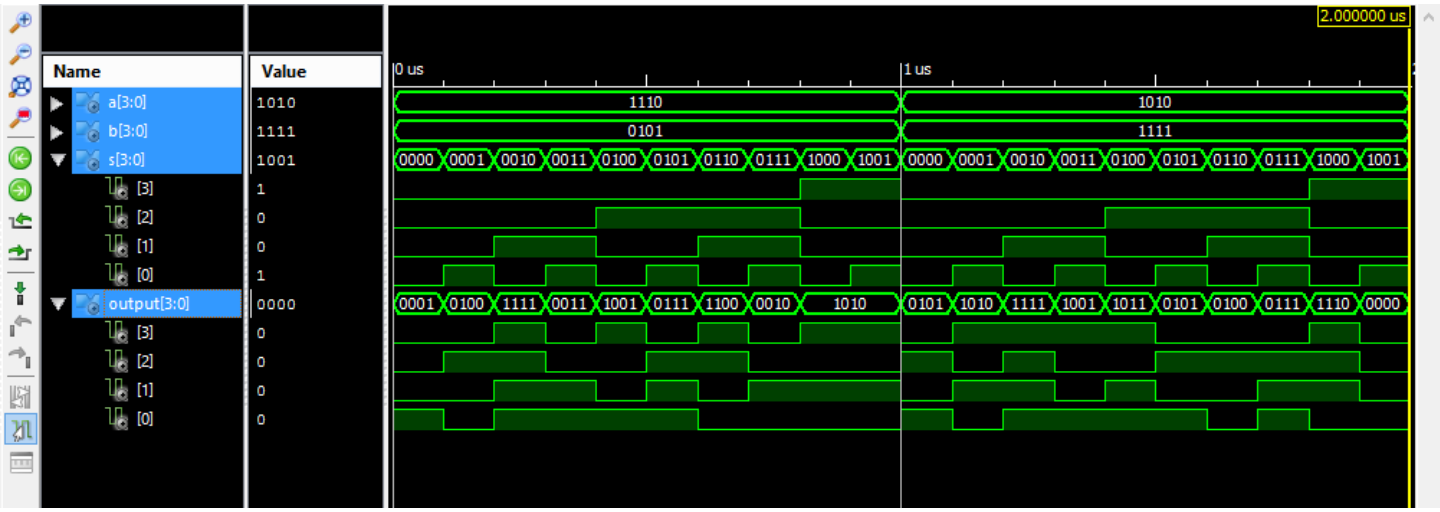

Figure 1: Timing Diagram for 4 bit Arithmetic Logic Unit

Codes

Here I have include working code for Arithmetic Logic Unit and 4-bit Adder.

1 Arithmetic Logic Unit

```
1  ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    14:20:01 03/04/2014
6  -- Design Name:
7  -- Module Name:    alu - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
```

```
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 --use IEEE.NUMERIC_STD.ALL;
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity alu is
33     Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
34            b : in  STD_LOGIC_VECTOR (3 downto 0);
35            s : in  STD_LOGIC_VECTOR (3 downto 0);
36            output : out  STD_LOGIC_VECTOR (3 downto 0));
37 end alu;
38
39 architecture Behavioral of alu is
40 signal c :std_logic_vector(3 downto 0);
41 signal c1 : std_logic_vector(3 downto 0);
42 signal c2 : std_logic_vector(3 downto 0);
43 signal temp : std_logic_vector(3 downto 0);
44
45 component add is
46     Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
47            b : in  STD_LOGIC_VECTOR (3 downto 0);
48            output : out  STD_LOGIC_VECTOR (3 downto 0));
49 end component;
50
51 begin
52 c1(0) <= not b(0);
53 c1(1) <= not b(1);
54 c1(2) <= not b(2);
55 c1(3) <= not b(3);
56
57 add1: add
58         port map(a,b,c);
59 add2: add
60         port map(c1,"0001",c2);
```

```
add3: add
        port map(a,c2,temp);
process(a,b,s)
begin
case s is
        when "0000" =>
                output(0) <= not a(0);
                output(1) <= not a(1);
                output(2) <= not a(2);
                output(3) <= not a(3);
        when "0001" =>
                output(0) <= a(0) and b(0);
                output(1) <= a(1) and b(1);
                output(2) <= a(2) and b(2);
                output(3) <= a(3) and b(3);
        when "0010" =>
                output(0) <= a(0) or b(0);
                output(1) <= a(1) or b(1);
                output(2) <= a(2) or b(2);
                output(3) <= a(3) or b(3);
        when "0011" =>
                output <= c;
        when "0100" =>
                output <= temp;
        when "0101" =>
                output(0) <= a(1);
                output(1) <= a(2);
                output(2) <= a(3);
                output(3) <= '0';
        when "0110" =>
                output(3) <= a(2);
                output(2) <= a(1);
                output(1) <= a(0);
                output(0) <= '0';
        when "0111" =>
                output(0) <= b(1);
                output(1) <= b(2);
                output(2) <= b(3);
                output(3) <= '0';
        when "1000" =>
                output(3) <= b(2);
                output(2) <= b(1);
                output(1) <= b(0);
                output(0) <= '0';
        when "1001" =>
```

```
                output(0) <= not b(0);
                output(1) <= not b(1);
                output(2) <= not b(2);
                output(3) <= not b(3);
        when others =>
                null;
end case;
end process;

end Behavioral;
```

2 4 bit Adder

```
----------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    14:34:15 03/04/2014
-- Design Name:
-- Module Name:    add - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

```

```
32 entity add is
33     Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
34            b : in  STD_LOGIC_VECTOR (3 downto 0);
35            output : out  STD_LOGIC_VECTOR (3 downto 0));
36 end add;
37
38
39 architecture Behavioral of add is
40 signal c1 :std_logic;
41 signal c2 : std_logic;
42 signal c3 : std_logic;
43 begin
44
45 output(0) <= a(0) xor b(0);
46 c1 <= a(0) and b(0);
47 output(1) <= a(1) xor b(1) xor c1;
48 c2 <= (a(1) and b(1)) or (a(1) and c1) or (b(1) and c1);
49 output(2) <= a(2) xor b(2) xor c2;
50 c3 <= (a(2) and b(2)) or (a(2) and c2) or (b(2) and c2);
51 output(3) <= a(3) xor b(3) xor c3;
52
53
54 end Behavioral;
```

Inference

In this assignment I inferred the following:

1. Designing a basic Arithmetic Logic Unit
2. Learned the Bit-wise operations
3. Learned the Bit Shifting
4. Learned clock implementation