Labb3 johannes joujo

This code was given by the teacher

```
Library IEEE;
use work.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.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 leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity add4 is
Port ( in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
s_out : out STD_LOGIC_VECTOR (4 downto 0));
end add4;
architecture Behavioral of add4 is
begin
 s_out<= ('0' & in_1) + ('0' & in_2);
end Behavioral;
Library IEEE;
use work.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.all;
entity mul4 is -- m_out <= in_1 * in_2
Port ( in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
m_out : out STD_LOGIC_VECTOR (7 downto 0));
end mul4;
```

```
architecture Behavioral of mul4 is
begin
m_out <= in_1 * in_2;
end Behavioral;
Library IEEE;
use work.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.all;
entity invrt4 is -- i_out <= NOT ( in_1 )
Port ( in_1 : in STD_LOGIC_VECTOR (3 downto 0);
i_out : out STD_LOGIC_VECTOR (3 downto 0));
end invrt4;
architecture Behavioral of invrt4 is
begin
 i_out <= NOT ( in_1 );
end Behavioral;
Library IEEE;
use work.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.all;
entity bit_and4 is -- a_out <= ( in_1 AND in_2 );
Port ( in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
a_out : out STD_LOGIC_VECTOR (3 downto 0));
end bit_and4;
architecture Behavioral of bit_and4 is
begin
a_out <= ( in_1 AND in_2 );
end Behavioral;
Library IEEE;
use work.all;
use IEEE.STD_LOGIC_1164.all;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

use IEEE.STD_LOGIC_ARITH.all;

entity bit_or4 is -- o_out <= ( in_1 OR in in_2 )

Port ( in_1 : in STD_LOGIC_VECTOR (3 downto 0);

in_2 : in STD_LOGIC_VECTOR (3 downto 0);

o_out : out STD_LOGIC_VECTOR (3 downto 0));

end bit_or4;

architecture Behavioral of bit_or4 is

begin

o_out <= ( in_1 OR in_2 );

end Behavioral;
```

I used what was given to me for add4.

Explaining the testbench

Inputs for all the components

The testbench has five components which are add4, mul4, invert4, bit_and4 and bit_or4. add4, mul4, bit_and4 and bit_or4 have the input port in_1 and in_2 both of them are vectors from 0-3 (four bits). The invert4 only has one input port called in_1 and is a vector from 0-3 (four bits). Note that all entities have the same input in_1 and most of them has in_2, this makes all the different entity get the same input but it gives different result because of what operation it does.

Outputs for the components

The output for add4 is s_out and that is a vector 4 down to 0 (5 bits), the reason for the output to be a vector that can contain 5 bits is because when you add two four bits you can get a carry number, ex 1111_2 + 1111_2 = 11110_2 (in binary)

The output for mul4 is m_out is a vector 7 down to 0 (8 bits). The reason for the output being 8 bits is because that is the maximum amount of bits you need when multiplying two 4 bit vectors, ex 1111_2 x 1111_2 = 11100001_2 (in binary).

The reason for the invrt4 only having one input and one output (I_out) being the same size vector (4 bit) is because it takes the input vector ex 1010_2 and inverts each bit to 0101_2 (in binary).

The output (a_out) for bit_and4 is the same size as the two inputs because it takes the two input vectors (4 bits) and compare the numbers in a boolean way, ex 1101_2 and 1001_2 = 1001_2 (in binary)

The output for bit_or4 has the output o_out and workes almost the same as the bit_and4 but instead of the boolean and it does boolean or, ex 1101_2 or 1001_2 = 1101_2 (in binary)

The testbench

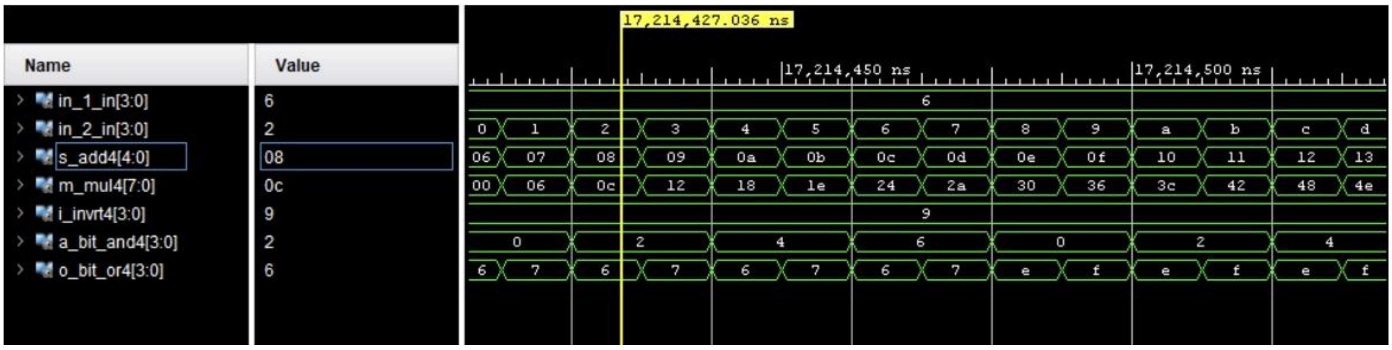
```
Library IEEE;
use work.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.all;
entity fyra_tb is
end fyra_tb;
architecture behave of fyra_tb is
component add4
    port (
      in_1, in_2 : in  std_logic_vector(3 downto 0);
      s_out  : out std_logic_vector(4 downto 0));
  end component;
component mul4
      port (
        in_1, in_2 : in  std_logic_vector(3 downto 0);
        m_out  : out std_logic_vector(7 downto 0));
    end component;
component invrt4
        port (
          in_1 : in  std_logic_vector(3 downto 0);
          i_out  : out std_logic_vector(3 downto 0));
      end component;
component bit_and4
         port (
           in_1, in_2 : in  std_logic_vector(3 downto 0);
           a_out  : out std_logic_vector(3 downto 0));
       end component;
component bit_or4
    port (
      in_1, in_2 : in  std_logic_vector(3 downto 0);
      o_out  : out std_logic_vector(3 downto 0));
  end component;

    signal in_1_in, in_2_in : std_logic_vector(3 downto 0);
    signal s_out_behave :std_logic_vector(4 downto 0);
    signal m_out_behave :std_logic_vector(7 downto 0);
    signal i_out_behave :std_logic_vector(3 downto 0);
    signal a_out_behave :std_logic_vector(3 downto 0);
    signal o_out_behave :std_logic_vector(3 downto 0);
begin  -- behave
add : add4 port map (
    in_1        => in_1_in,
    in_2        => in_2_in,
    s_out     => s_out_behave);
mul : mul4 port map (
    in_1        => in_1_in,
    in_2        => in_2_in,
    m_out     => m_out_behave);
invrt : invrt4 port map (
    in_1        => in_1_in,
    i_out     => i_out_behave);
bit_and : bit_and4 port map (
    in_1        => in_1_in,
    in_2        => in_2_in,
    a_out     => a_out_behave);
bit_or : bit_or4 port map (
    in_1        => in_1_in,
    in_2        => in_2_in,
    o_out     => o_out_behave);
process
   variable a_vector, b_vector : std_logic_vector(3 downto 0):=(others=>'0');
begin  -- process
  for a_vector in 0 to 15 loop
```

```
    for b_vector in 0 to 15 loop
      in_1_in <= conv_std_logic_vector(a_vector,4);
      in_2_in <= conv_std_logic_vector(b_vector,4);
      wait for 10 ns;
--        if ((conv_integer(s_behave) /= conv_integer(s_struct)) or (cout_struct /= cout_behave)) then
--          assert false report "Simulation failed!" severity failure;
--        end if;
    end loop;  -- b_vector
  end loop;  -- a_vector
end process;
end behave;
```

I looked at testbench for labb2 and made my own for labb3.

I only had two files one was the testbench and the other was add4 where I used the text from labb 3 instructions. Most of the functions worked the same they had two inputs and one output, but the result was different because of the operations are different.



Name	Value
in_1_in[3:0]	6
in_2_in[3:0]	2
s_add4[4:0]	08
m_mul4[7:0]	0c
i_invrt4[3:0]	9
a_bit_and4[3:0]	2
o_bit_or4[3:0]	6

17,214,427.036 ns
17,214,450 ns
17,214,500 ns
6
0 1 2 3 4 5 6 7 8 9 a b c d
06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13
00 06 0c 12 18 1e 24 2a 30 36 3c 42 48 4e
9
0 2 4 6 0 2 4
6 7 6 7 6 7 6 7 e f e f e f

Figure 1 shows the result from the different operations (adding, multiplying, and, or, inverting).