Visvesvaraya National Institute of Technology (VNIT), Nagpur

Digital Hardware Design (ECL313)

Assignment

Submitted by :
Rajesh Nagula (BT18ECE059)
Shruti Murarka (BT18ECE099)
Dhruvi Shah (BT18ECE115)
Semester VI

Group No (9)

Submitted to :
Dr. Anamika Singh and Dr. Neha Nawandar
(Course Instructors)
Department of Electronics and Communication Engineering,
VNIT Nagpur

Contents

Problem Statement and Theory

Problem Statement: Design various models for 4-bit Excess-3 to binary converter.

Theory: The excess 3 code is a way of biased coding. In this each digit is expressed as individual binary equivalent + 3. The major advantage is that a decimal can be easily nine's complimented as simple as 1's compliment in binary. Other advantage is is doesn't use 0000 and 1111 which are the signal failure cases. Also it is difficult to write 0000 exactly in magnetic media. There are the following steps to convert the Excess-3 code to binary:

- Get the decimal equivalent of the excess 3 code using simple binary equivalence.
- Subtract 3 in each digit of the decimal number.
- From this newly generated decimal number digits get binary number.

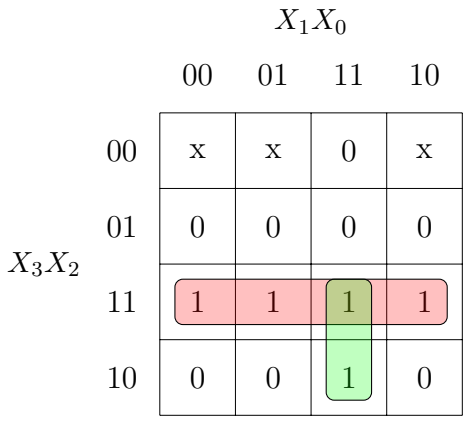
We can also subtract 0011 in each 4-bit excess-3 code to get binary equivalence. By following the duality of each step we can get excess 3 code from binary.

Truth Table: Desired Truth table is given below:

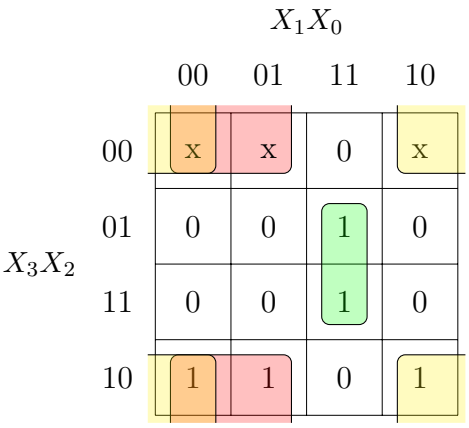
Input ($x_3x_2x_1x_0$)	Output ($b_3b_2b_1b_0$)	Valid Bit (v)
0000	xxxx	1
0001	xxxx	1
0010	xxxx	1
0011	0000	0
0100	0001	0
0101	0010	0
0110	0011	0
0111	0100	0
1000	0101	0
1001	0110	0
1010	0111	0
1011	1000	0
1100	1001	0
1101	1010	0
1110	1011	0
1111	1100	0

K-Maps: K-Maps for output are given below:

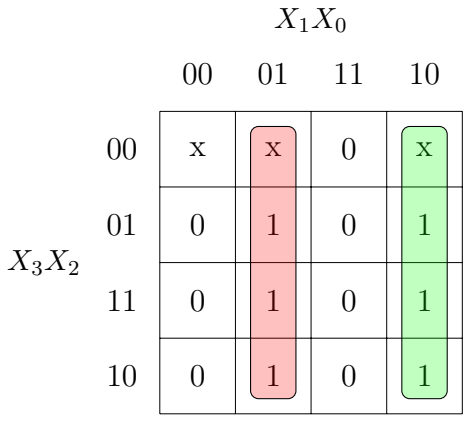
b_3



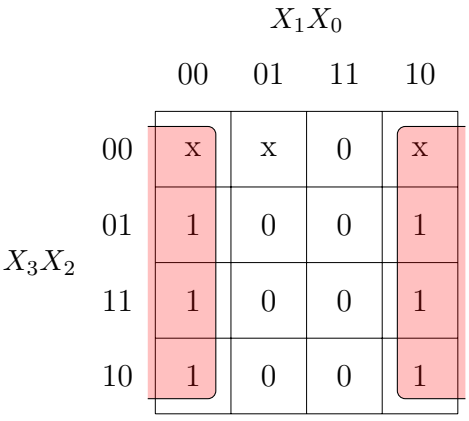
b_2



b_1



b_0



$$b_3 = x_3x_2 + x_0x_1x_3$$

$$b_2 = \overline{x_2}.\overline{x_1} + \overline{x_0}.\overline{x_2} + x_2x_1x_0$$

$$b_1 = \overline{x_1}x_0 + x_1\overline{x_0}$$

$$b_0 = \overline{x_0}$$

K-Map for valid bit is given below:

X_3X_2 \ X_1X_0	00	01	11	10
00	1	1	0	1
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

v = $\overline{x_2}.\overline{x_3}.\overline{x_1} + \overline{x_2}.\overline{x_3}.\overline{x_0}$

- Here x is used for don't care.
- The Karnaugh map is used to reduce the equation in terms of minterms, that is in sum of product form.

And final relations are:

- $b_3 = x_3x_2 + x_0x_1x_3$
- $b_2 = \overline{x_2}.\overline{x_1} + \overline{x_0}.\overline{x_2} + x_2x_1x_0$
- $b_1 = \overline{x_1}x_0 + x_1\overline{x_0}$
- $b_0 = \overline{x_0}$
- v = $\overline{x_2}.\overline{x_3}.\overline{x_1} + \overline{x_2}.\overline{x_3}.\overline{x_0}$

Code - 1

```
-- 4 bit excess-3 to binary convertor
-- Dataflow Model

library ieee;
use ieee.std_logic_1164.all;

entity e3tobin_df is
        port ( ex3 : in std_logic_vector(3 downto 0);
                    bin : out std_logic_vector( 3 downto 0));
end e3tobin_df;

architecture arch_df of e3tobin_df is
begin

bin(3) <= (ex3(3) and ex3(2)) or (ex3(3) and ex3(1) and ex3(0));
bin(2) <= ((not(ex3(2))) and (not(ex3(1))) ) or
           ( ex3(2) and ex3(1) and ex3(0)) or
            ((not(ex3(3))) and ex3(1) and (not(ex3(0))) );
bin(1) <= ( (not(ex3(1))) and ex3(0) ) or (ex3(1) and (not(ex3(0))))
bin(0) <= not(ex3(0));

end arch_df;
```

Testbench of Code - 1

```
-- Testbench Dataflow model

library ieee;
use ieee.std_logic_1164.all;

entity tb_e3tobin is
end tb_e3tobin;

architecture tb of tb_e3tobin is

-- defining the component

  component e3tobin_df
  port (ex3 : in std_logic_vector(3 DOWNTO 0);
                bin : out std_logic_vector(3 DOWNTO 0));
  end component;

-- initializing the signals for the port map

signal ex3 : std_logic_vector(3 DOWNTO 0):="0000";
signal bin : std_logic_vector(3 DOWNTO 0):="1111";

begin
-- instantiating the component defined
    dut : e3tobin_df port map(ex3 => ex3, bin => bin);
    stimuli : process
    begin
    ex3 <= "0000";
    wait for 100ps;

        ex3 <= "0001";
```

```
wait for 100ps;

ex3 <= "0010";
wait for 100ps;

ex3 <= "0011";
wait for 100ps;

ex3 <= "0100";
wait for 100ps;

ex3 <= "0101";
wait for 100ps;

ex3 <= "0110";
wait for 100ps;

ex3 <= "0111";
wait for 100ps;

ex3 <= "1000";
wait for 100ps;

    ex3 <= "1001";
wait for 100ps;

ex3 <= "1010";
wait for 100ps;

ex3 <= "1011";
wait for 100ps;

ex3 <= "1100";
```

```
    wait for 100ps;

    ex3 <= "1101";
    wait for 100ps;

    ex3 <= "1110";
    wait for 100ps;

    ex3 <= "1111";
    wait for 100ps;
    wait;
    end process;

end tb;
```

RTL View and Description of Code - 1

Description: The first code is the dataflow model , in which we made the truth table for the conversion from excess 3 to the binary system . The equations we derived using the K-Map approach . In this design we have considered a 4bit input. Hence four equations have been derived for the 4 individual bits.

RTL View: Quartus II is used for RTL View of 4 bit excess-3 to binary converter.

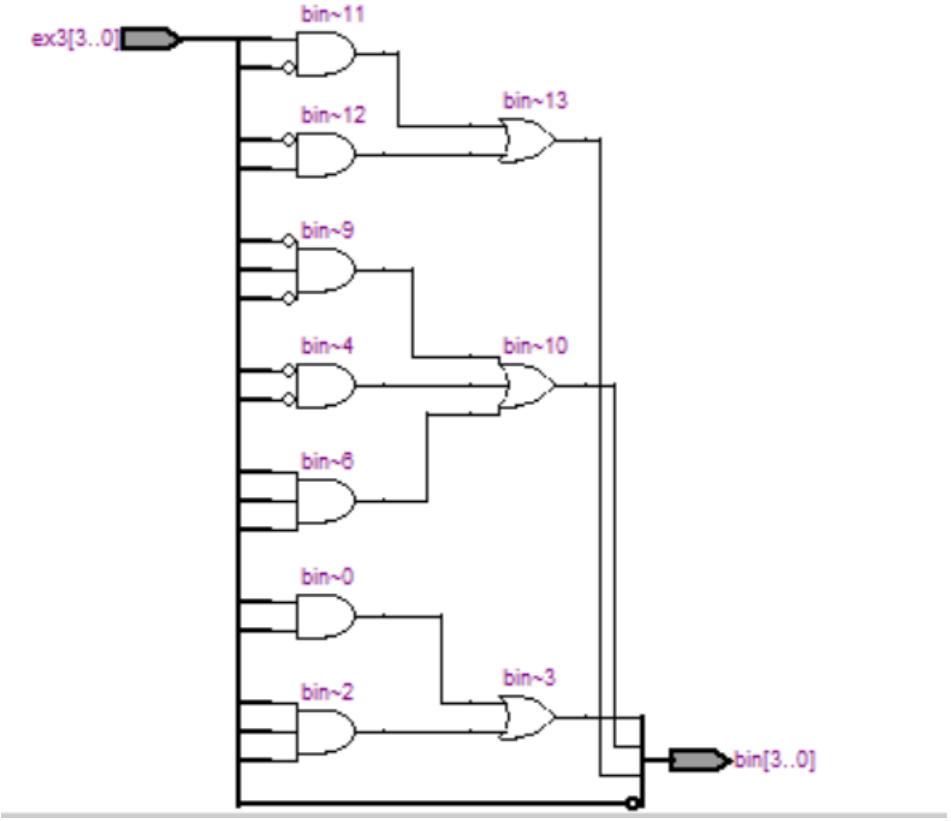


Figure 1: RTL View of behavorial model.

Simulation Results and Description of Code - 1

Description: Testbenches are used for simulation purpose. We generated the simulation waveforms using ModelSim, without changing the input signal values manually with the help of testbench. A testbench with name 'tbe3tobin' is defined. Note that, entity of testbench is always empty i.e. no ports are defined in the entity. 2 signals are defined i.e. inp and output inside the architecture body, these signals are then connected to actual excess3bin design using structural modeling using port map function. Also note that, process statement is written without the sensitivity list. Lastly, different values are assigned to input signals.

RTL View: ModelSim is used for Simulation of 4 bit excess-3 to binary converter.

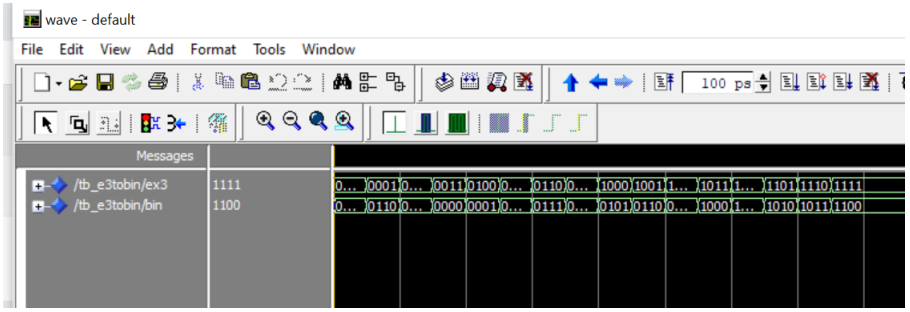


Figure 2: Simulation of behavorial model.

Code - 2

```
-- 4 bit excess-3 to binary convertor
-- Behavioral Model

LIBRARY ieee;
use ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY excs3tbin IS
port(
    ex3 : in std_logic_vector(3 DOWNTO 0);
        bin : out std_logic_vector(3 DOWNTO 0);
        valid: out std_logic
        );
end excs3tbin;

ARCHITECTURE arch OF excs3tbin IS
BEGIN
process(ex3)        -- process sensitive to input
begin
if (ex3 = "0000" or ex3 = "0001" or ex3 = "0010") then
        valid <= '1';
        bin <= "1111";
else
        valid <= '0';
        bin <= ex3 - "0011";
end if;
end process;
end arch;
```

Testbench of Code - 2

```
-- Testbench behavioral model

library ieee;
use ieee.std_logic_1164.all;

entity tb_ex3tbin is
end tb_ex3tbin;

architecture tb of tb_ex3tbin is

-- defining the component

  component excs3tbin
  port (ex3 : in std_logic_vector(3 DOWNTO 0);
                bin : out std_logic_vector(3 DOWNTO 0);
                valid: out std_logic);
  end component;

-- initializing the signals for the port map

signal inp : std_logic_vector(3 DOWNTO 0);
signal op : std_logic_vector(3 DOWNTO 0);
signal z : std_logic;

begin
-- instantiating the component defined
    dut : excs3tbin port map(ex3 => inp, bin => op, valid => z);
    stimuli : process
    begin
    inp <= "0000";
    wait for 100ps;
```

```
inp <= "0001";
wait for 100ps;

inp <= "0010";
wait for 100ps;

inp <= "0011";
wait for 100ps;

inp <= "0100";
wait for 100ps;

inp <= "0101";
wait for 100ps;

inp <= "0110";
wait for 100ps;

inp <= "0111";
wait for 100ps;

inp <= "1000";
wait for 100ps;

inp <= "1001";
wait for 100ps;

inp <= "1010";
wait for 100ps;

inp <= "1011";
wait for 100ps;
```

```
    inp <= "1100";
    wait for 100ps;

    inp <= "1101";
    wait for 100ps;

    inp <= "1110";
    wait for 100ps;

    inp <= "1111";
    wait for 100ps;
    wait;
    end process;

end tb;
```

RTL View and Description of Code - 2

Description: For behavioral model, we defined process sensitive to input data. Sequential statements if then else are used for conditioning. If input < 3 we assigned output $= 1111$ along with high valid bit indicating invalid input else binary of given input data is difference of input data and 0011 (3 in binary) arithmetic operation is done in code. Valid bit is low during this time indicating valid input data.

RTL View: Quartus II is used for RTL View of 4 bit excess-3 to binary converter.

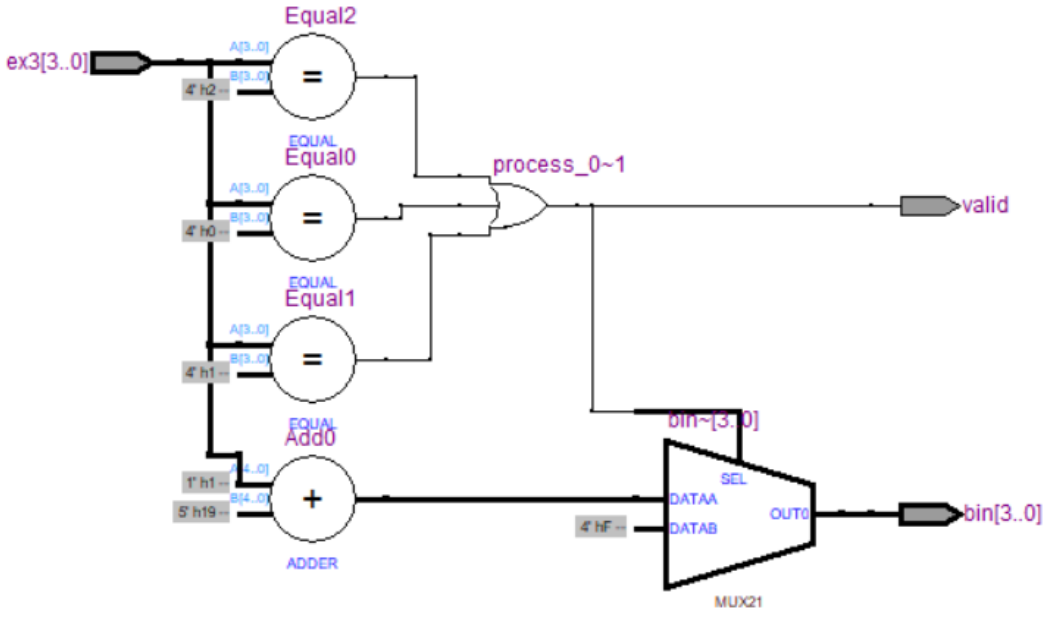


Figure 3: RTL View of behavorial model.

Simulation Results and Description of Code - 2

Description: Testbenches are used for simulation purpose. We generated the simulation waveforms using ModelSim, without changing the input signal values manually with the help of testbench. A testbench with name 'tbex3tbin' is defined. Note that, entity of testbench is always empty i.e. no ports are defined in the entity. 3 signals are defined i.e. inp, op and z inside the architecture body, these signals are then connected to actual excess3bin design using structural modeling using port map function. Also note that, process statement is written without the sensitivity list. Lastly, different values are assigned to input signals.

RTL View: ModelSim is used for Simulation of 4 bit excess-3 to binary converter.

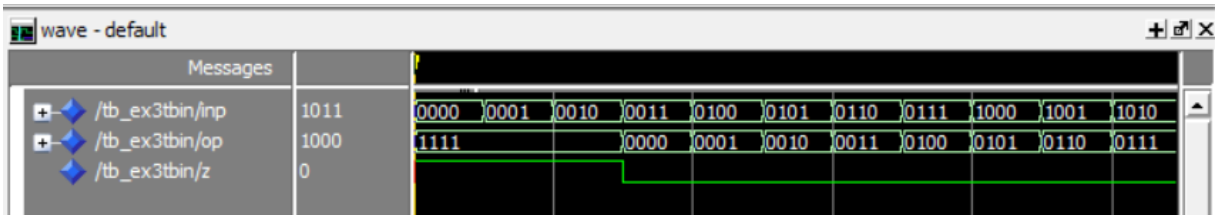


Figure 4: Simulation of behavorial model.

Code - 3

```
-- Structural Model XS3 to Binary
-- fulladder component
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

-- Defining entity
ENTITY fulladd IS
PORT ( Cin, x, y : IN STD_LOGIC ;
Sum, Cout : OUT STD_LOGIC ) ;
END fulladd ;

--data flow model
ARCHITECTURE df_fulladd OF fulladd IS
BEGIN
Sum <= x XOR y XOR Cin ;
Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y) ;
END df_fulladd ;


-- 4 bit excess-3 to binary convertor

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY xs3_2_bin IS
PORT ( ex3 : IN std_logic_vector(3 DOWNTO 0);
                bin : OUT std_logic_vector( 3 DOWNTO 0);
                valid: out std_logic);
END xs3_2_bin;

ARCHITECTURE structural OF xs3_2_bin IS
```

```
CONSTANT TMP : std_logic_vector(3 downto 0) := "1100";
CONSTANT C0: std_logic := '1';
-- defining the component
COMPONENT fulladd IS
  PORT( Cin, x, y : IN std_logic;
        Sum, Cout : OUT std_logic);
END COMPONENT;
-- initializing the signals for port map
SIGNAL C1, C2, C3, C4: std_logic;

BEGIN
        valid<='1' WHEN (ex3 = "0000" or ex3 = "0001" or ex3 = "0010"
        '0';
        -- instantiating the component defined
        FA0:fulladd PORT MAP(C0, ex3(0), TMP(0), bin(0), C1);
        FA1:fulladd PORT MAP(C1, ex3(1), TMP(1), bin(1), C2);
        FA2:fulladd PORT MAP(C2, ex3(2), TMP(2), bin(2), C3);
        FA3:fulladd PORT MAP(C3, ex3(3), TMP(3), bin(3), C4);

END structural ;
```

Testbench of Code - 3

```
-- Testbench Structural Model

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY tb_xs3_2_bin IS
END tb_xs3_2_bin;

ARCHITECTURE tb OF tb_xs3_2_bin IS
-- defining the component
  COMPONENT xs3_2_bin
  PORT (ex3 : IN std_logic_vector(3 DOWNTO 0);
                bin : OUT std_logic_vector(3 DOWNTO 0);
                valid: OUT std_logic);
  END COMPONENT;
-- initializing the signals for port map
SIGNAL inp : std_logic_vector(3 DOWNTO 0);
SIGNAL op : std_logic_vector(3 DOWNTO 0);
SIGNAL z : std_logic;

BEGIN
-- instantiating the component defined
    dut : xs3_2_bin PORT MAP(ex3 => inp, bin => op, valid => z);
    stimuli : PROCESS
    BEGIN
    inp <= "0000";
    wait for 100ps;

        inp <= "0001";
    wait for 100ps;
```

```
inp <= "0010";
wait for 100ps;

inp <= "0011";
wait for 100ps;

inp <= "0100";
wait for 100ps;

inp <= "0101";
wait for 100ps;

inp <= "0110";
wait for 100ps;

inp <= "0111";
wait for 100ps;

inp <= "1000";
wait for 100ps;

    inp <= "1001";
wait for 100ps;

inp <= "1010";
wait for 100ps;

inp <= "1011";
wait for 100ps;

inp <= "1100";
wait for 100ps;
```

```
    inp <= "1101";
    wait for 100ps;

    inp <= "1110";
    wait for 100ps;

    inp <= "1111";
    wait for 100ps;
    wait;
    END PROCESS;

END tb;
```

RTL View and Description of Code - 3

Description: For the structural modelling of 4 bit excess 3 bit to binary converter we need a full adder component 5because we need to subtract 3 that is b0011 from excess 3 to get binary. This can be achieved using a fulladder for subtractions. We need to subtract 3 hence we need to get 2's complement of three. This can be done by inverting b0011 to get b1100 and setting the Cin that is input carry as high. Now with the use of full adder component and the logic discussed above we can get binary output of corresponding excess3 code.

RTL View: Quartus prime lite 20.1 is used for RTL View of 4 bit excess-3 to binary converter using sequential modelling.

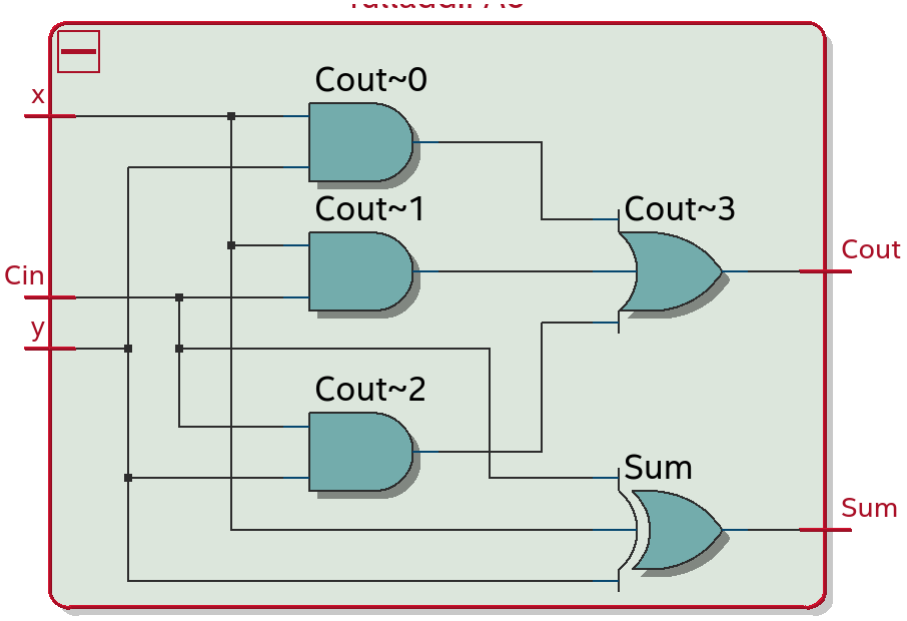


Figure 5: Full adder component.

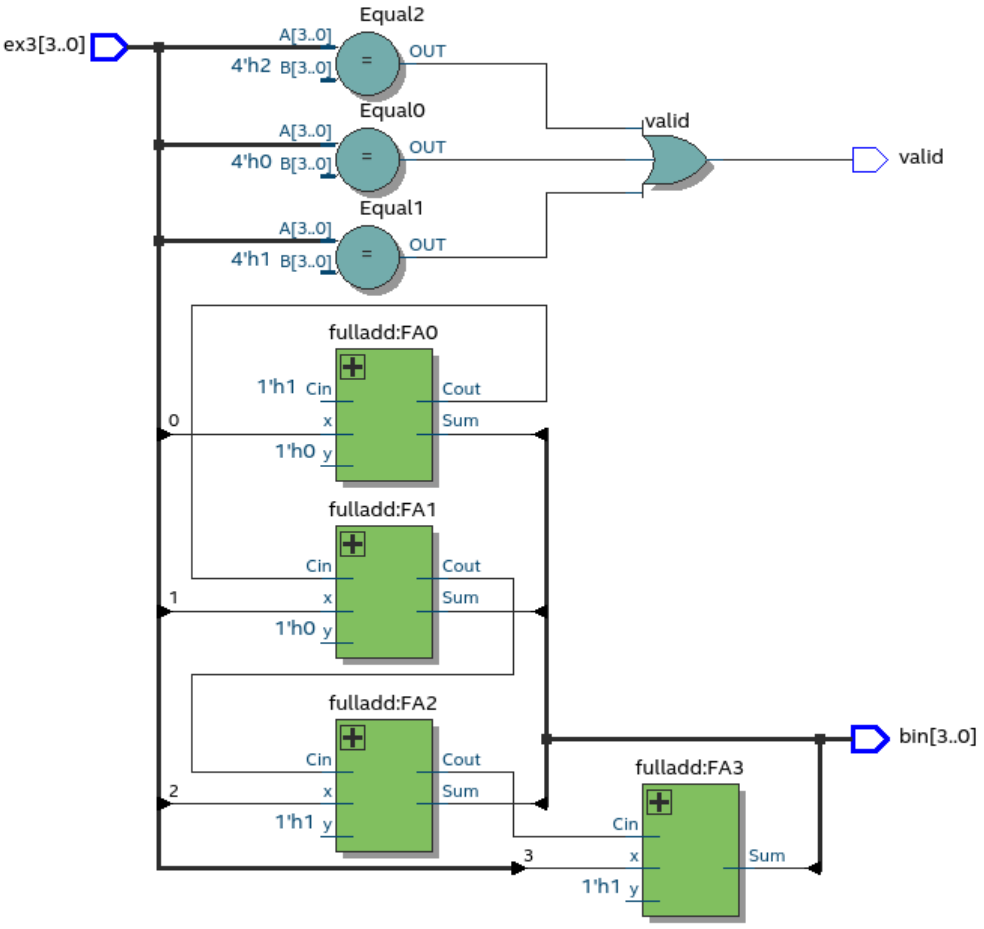


Figure 6: RTL View of Structural model.

Simulation Results and Description of Code - 3

Description: Testbenches are used for simulation purpose. We generated the simulation waveforms using ModelSim, without changing the input signal values manually with the help of testbench. A testbench with name 'tbxs32bin' is defined. Note that, entity of testbench is always empty i.e. no ports are defined in the entity. 3 signals are defined i.e. inp, op and z inside the architecture body, these signals are then connected to actual excess3bin design using structural modeling using port map function. Also note that, process statement is written without the sensitivity list. Lastly, different values are assigned to input signals.

RTL View: ModelSim is used for Simulation of 4 bit excess-3 to binary converter.

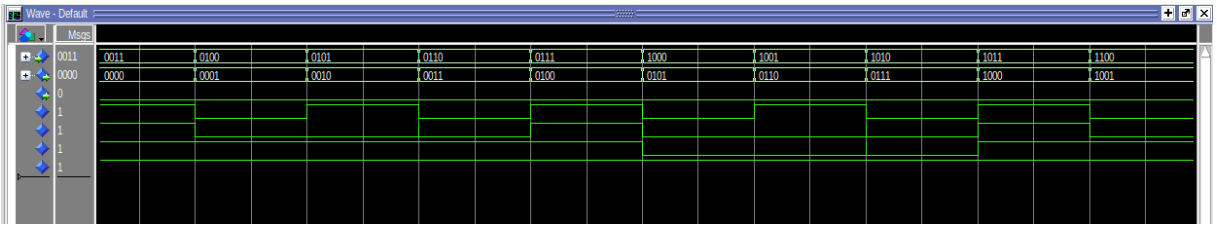


Figure 7: Simulation of Structural model.

Conclusion

We have designed three models for the conversion of excess-3 to binary. The first approach used is the data flow model which is the simplest approach in which the equations are derived using the truth table. The data flow model is like a direct gate-level implementation. The next approach is the Behavioral model, in which within the architecture we define the process which has the sequential statements in it. The process contains a sensitivity list, which ensures that the process is run every time there is a change in any of the elements in it. The last approach used is the Structural model, in which a component is declared and defined and then instantiated in the main architecture.