Getting Started with Xilinx
for
FPGA Programing [VHDL]

For Windows

About:

Xilinx ISE® WebPACK™ design software is the industry´s only FREE, fully featured front-to-back FPGA design solution for Linux, Windows XP, and Windows 7. ISE WebPACK is the ideal downloadable solution for FPGA and CPLD design offering HDL synthesis and simulation, implementation, device fitting, and JTAG programming. ISE WebPACK delivers a complete, front-to-back design flow providing instant access to the ISE features and functionality at no cost. Xilinx has created a solution that allows convenient productivity by providing a design solution that is always up to date with error-free downloading and single file installation

How to install & use Xilinx for windows:

Installation process for Windows:

Step 1:

Create an account in the official website of Xilinx www.xilinx.com

Download the full setup file of Xilinx ISE Design Suit (latest version available)

Step 2:

Extract the files & double click on the set up file to continue the setup process

Download a license file from the web.

Now, your toolkit is ready to use.

raktimchakraborty27@gmail.com

How to use Xilinx ISE:

For the implementation of the circuits in ISE for FPGA we are using the programing language VHDL [**V**HIC **H**ardware **D**escriptive **L**anguage]

For demonstration we are here implementing a full adder circuit in Xilinx ISE

Step: 1

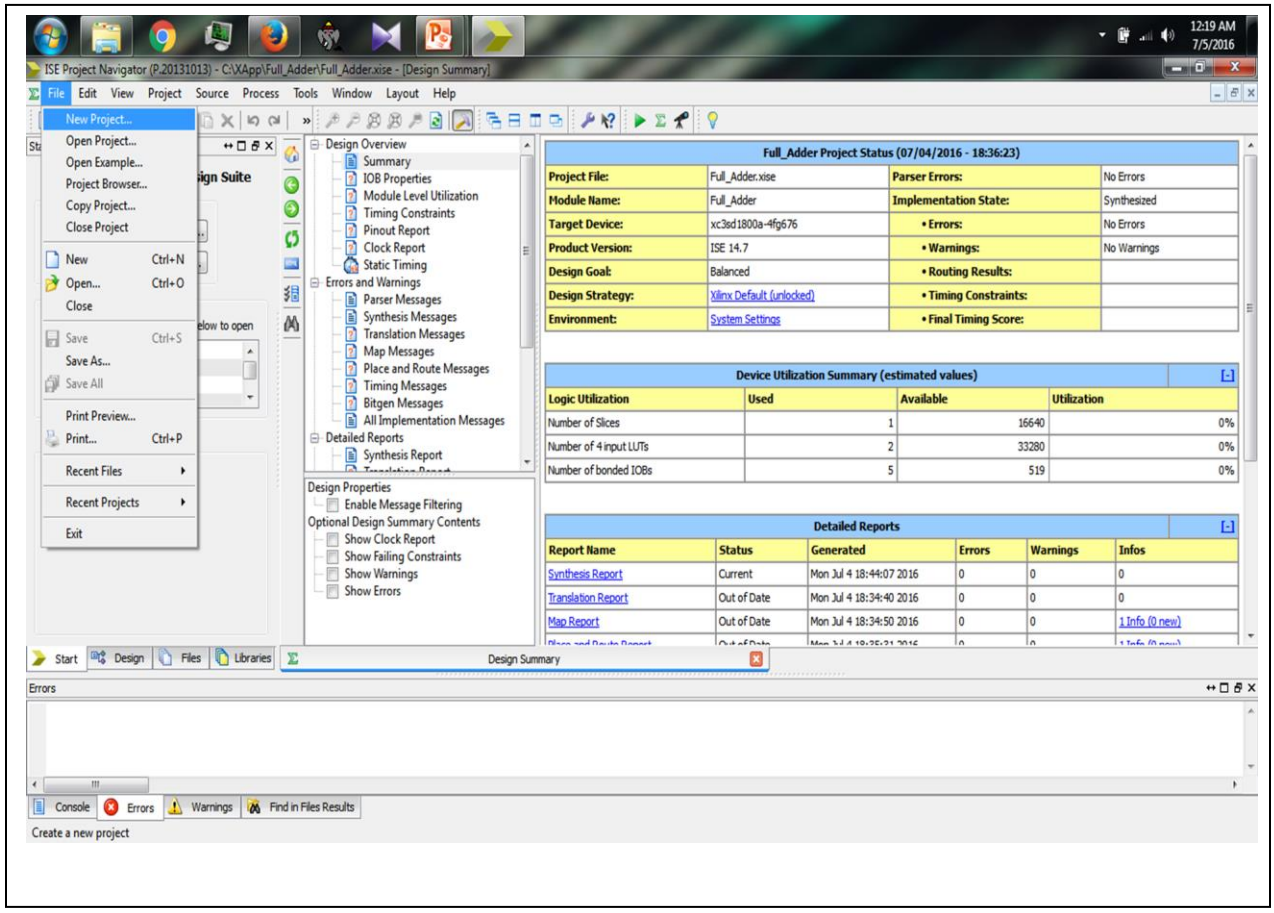


Fig-1

Click on File menu from the menu bar & select New Project option to create a new project.

Step: 2

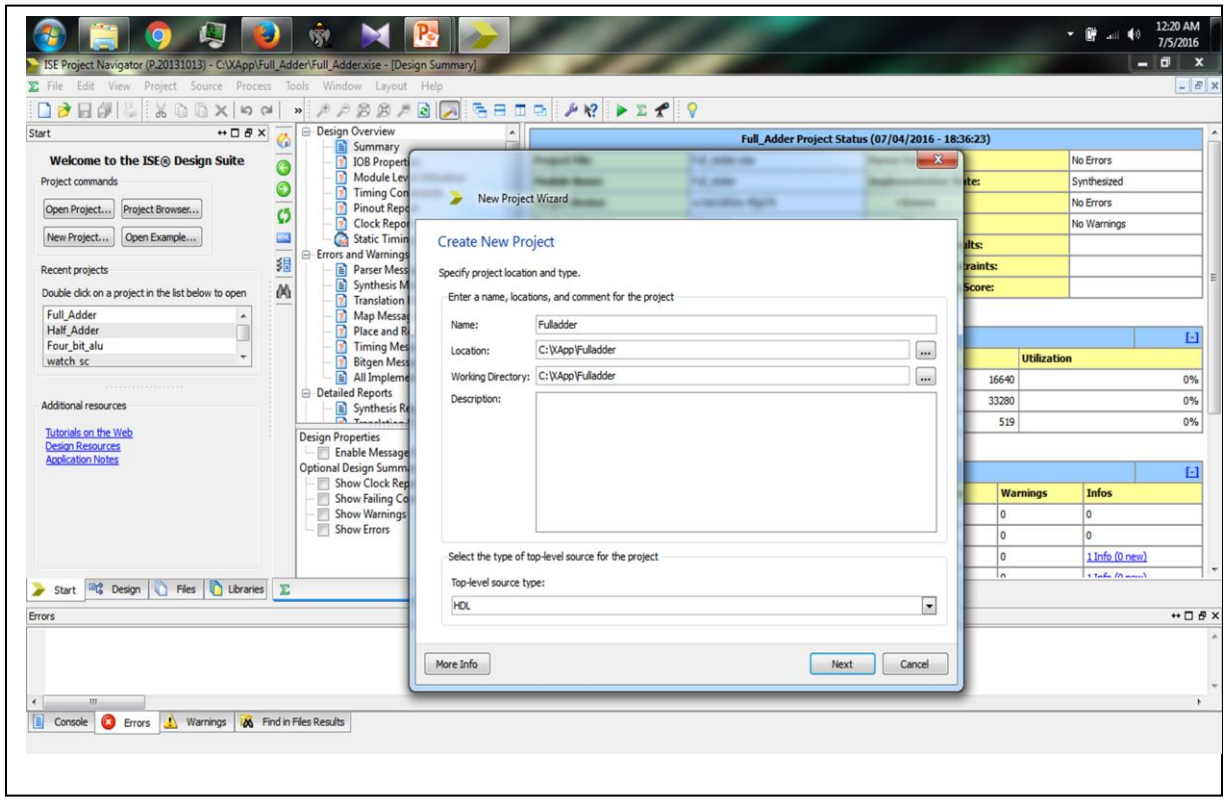


Fig: 2

After completion of Step: 1 a ‘Create New Project’ pop-up window will be appeared. Here give a suitable title to your project & select your desired destination/ directory in your hard-disk in where you want to save your project then click on the ‘Next’ button.

Step: 3

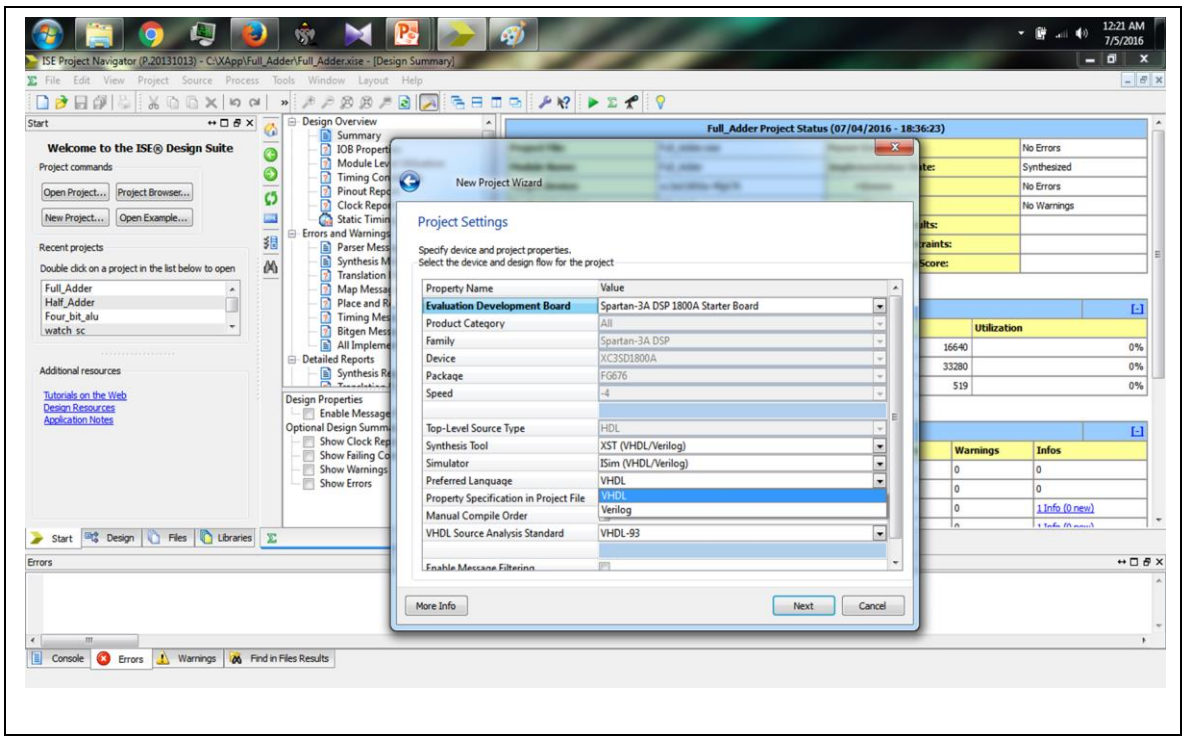


Fig: 3

Here, you select the development board on which you want to implement your designed circuit & select your preferable programming language in which you want to code.

We are selecting the evaluation development board as ‘Spartan’ as we are doing a FPGA stuff here & preferred language as VHDL.

After doing these things click on ‘Next’ & then ‘Finish’.

Step: 4

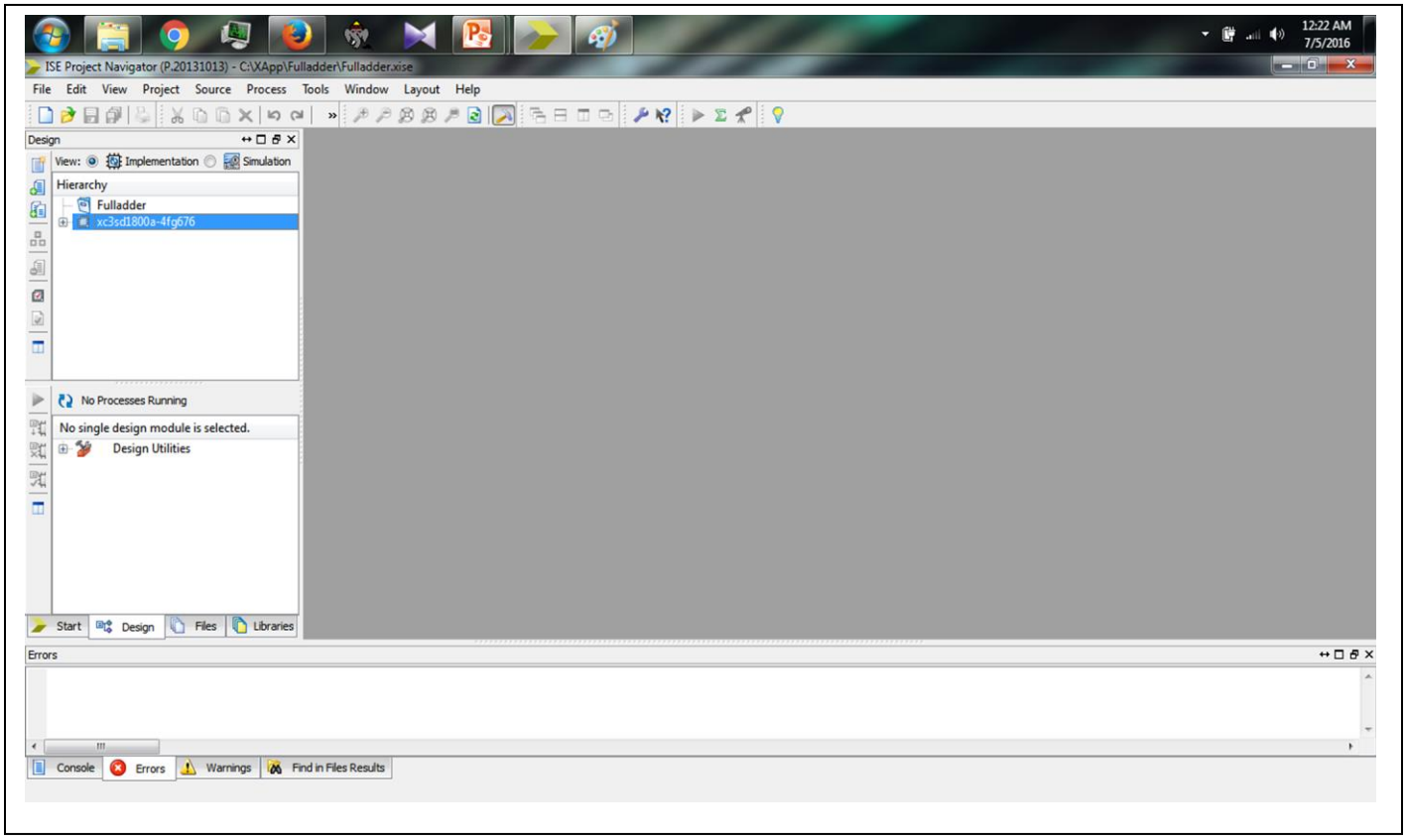


Fig: 4

This screen will appear after finish the 3rd step

raktimchakraborty27@gmail.com

Step: 5

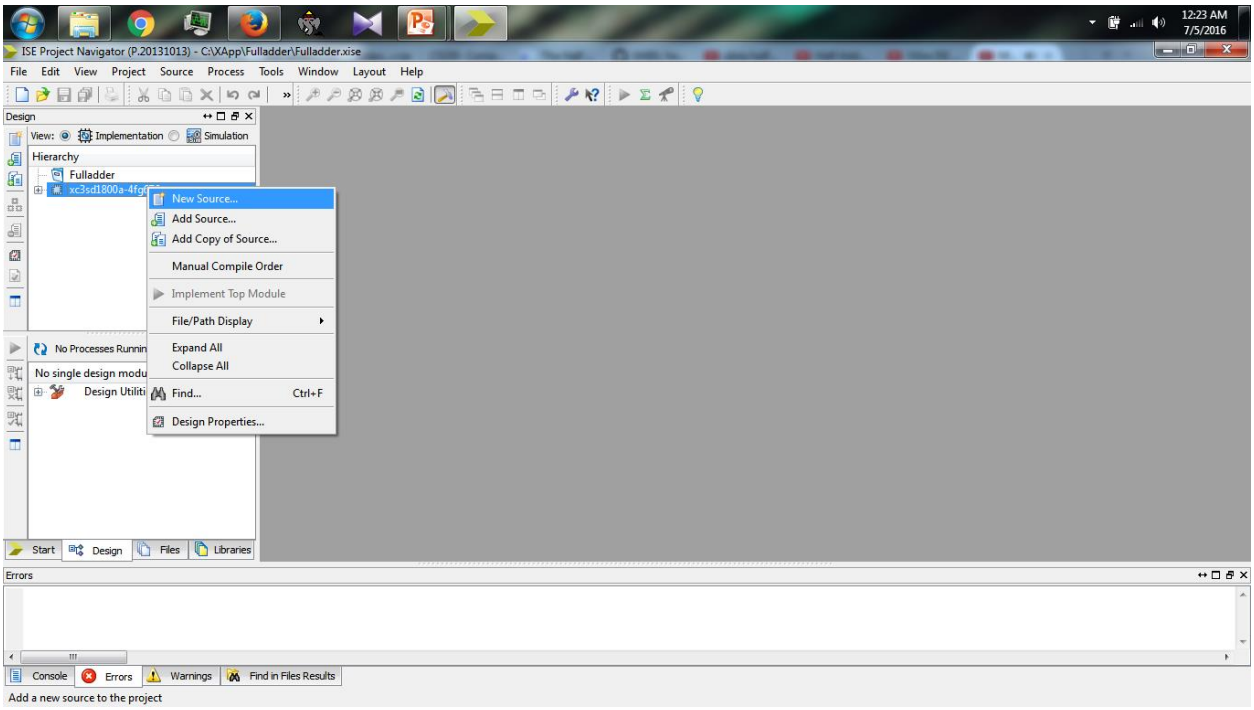


Fig: 5

Right click on the hierarchy window & select ‘New source’ option to add the Behavioral file of your circuit.

raktimchakraborty27@gmail.com

Step: 6

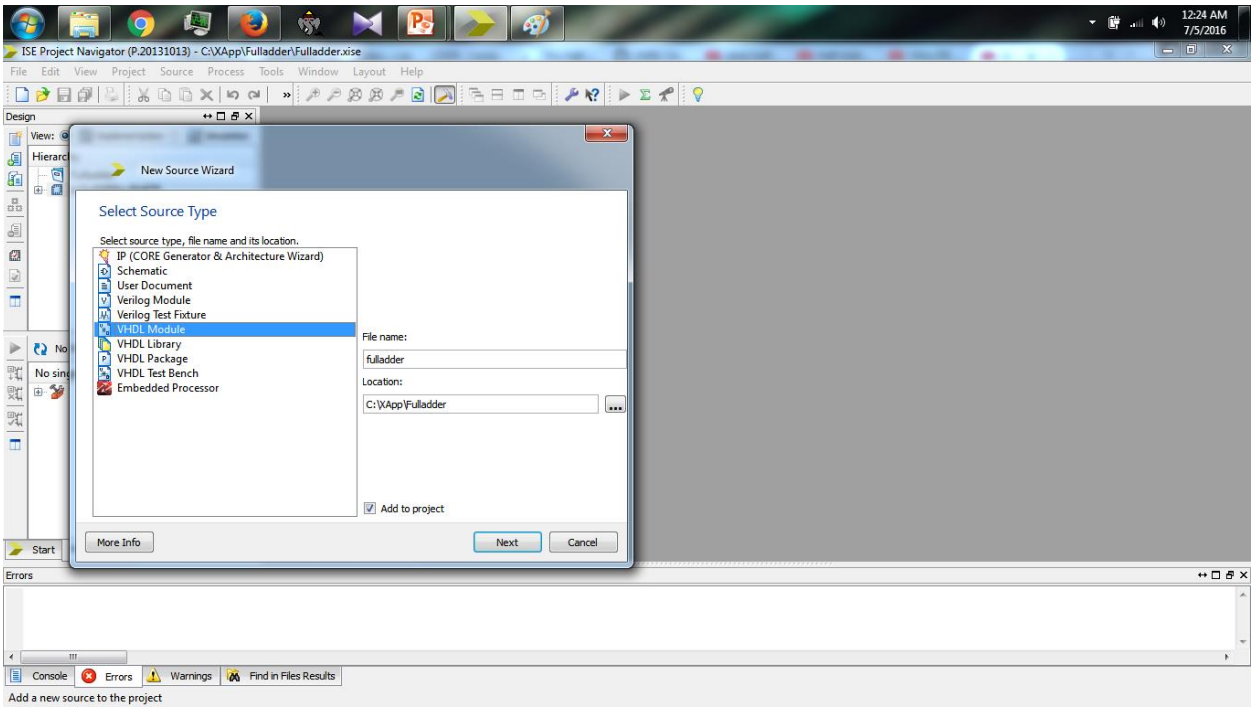


Fig: 6

After completion of step: 5 a ‘New Source Wizard’ window will be appeared. Select ‘VHDL Module’ from the source type & give a suitable title of the new module. Save the module in the same directory in which you are saving your project.

Step: 7

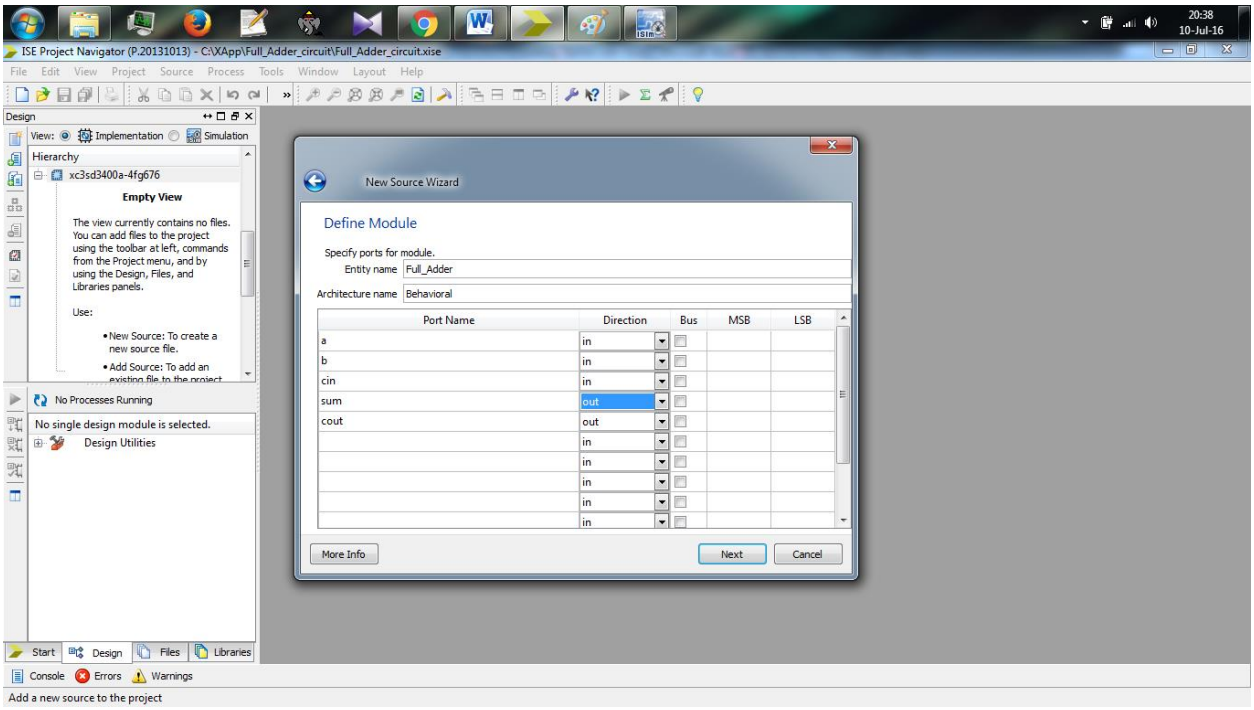


Fig: 7

Here, you define your module by specifying the pin configuration of the circuit which you are designing.

Here, we are taken a, b & c_{in} as input port, so we are selecting here 'in' from the dropdown box under the 'Direction' option available in the wizard; sum & c_{out} are taken as an output port & select 'out' option, as we are designing a full adder circuit here, we are defining the input & the output ports as per the configuration of a full adder circuit.

After that Click on 'Next' button then click on 'Finish'.

Step: 8

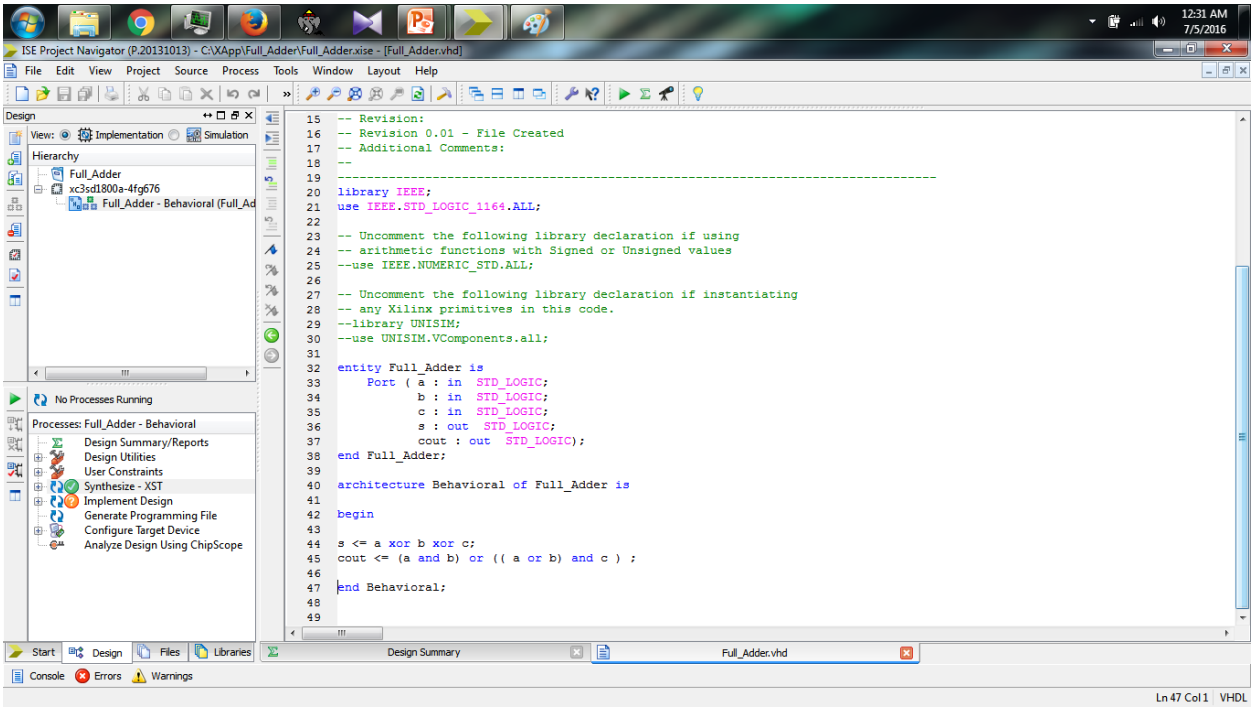


Fig: 8

As shown in fig: 8, this screen will appear after completion the previous step.

Here, we will define the entities of the circuit i.e. the pin configuration & the behavioral i.e. how will the circuit going to compute the output which will be generated from our designed circuit.

Here, for demonstration the full code of the implementation of a full adder circuit is shown below using VHDL:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
```

```
entity Full_Adder is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : in  STD_LOGIC;
           s : out  STD_LOGIC;
           cout : out  STD_LOGIC);
end Full_Adder;

architecture Behavioral of Full_Adder is
begin
s <= a xor b xor c;
cout <= (a and b) or (( a or b) and c ) ;
end Behavioral;
```

raktimchakraborty27@gmail.com

Step: 9

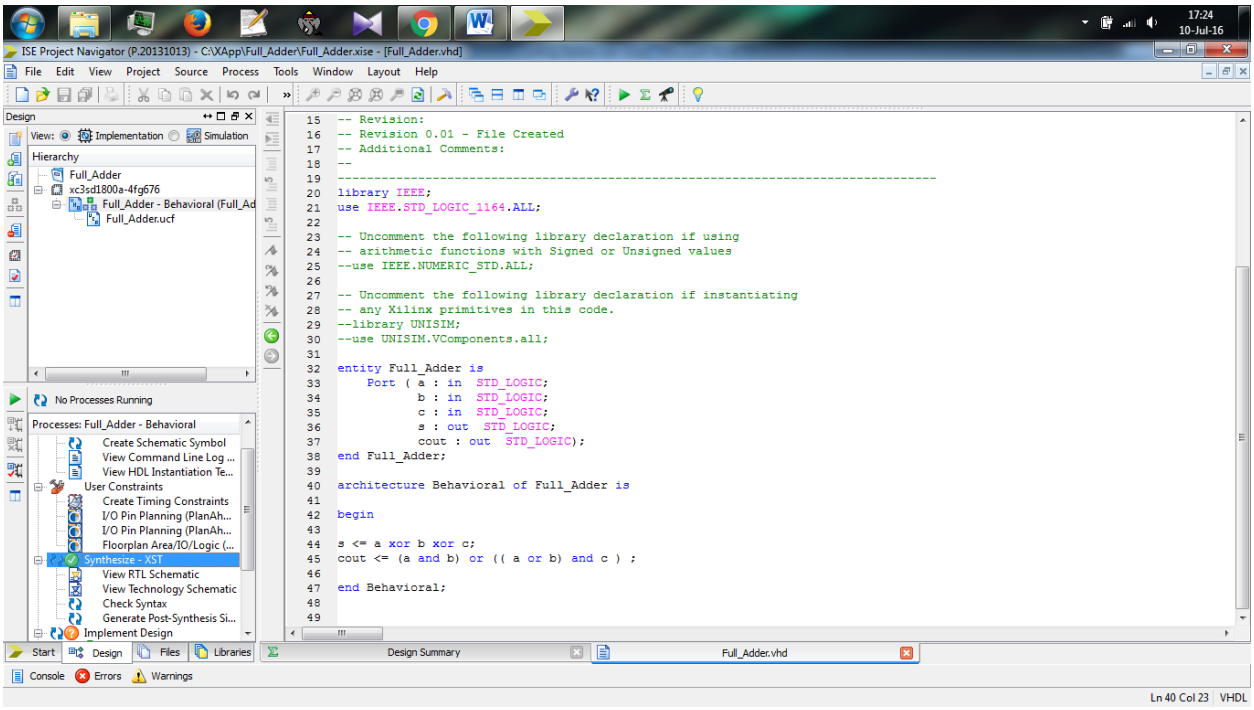


Fig: 9

After defining the behavioral of the circuit; double click on the ‘Synthesize-XST’ option which is available under the ‘Processes’ window.

It will check for errors in the code & generate the RTL-schematic diagram.

Step: 10

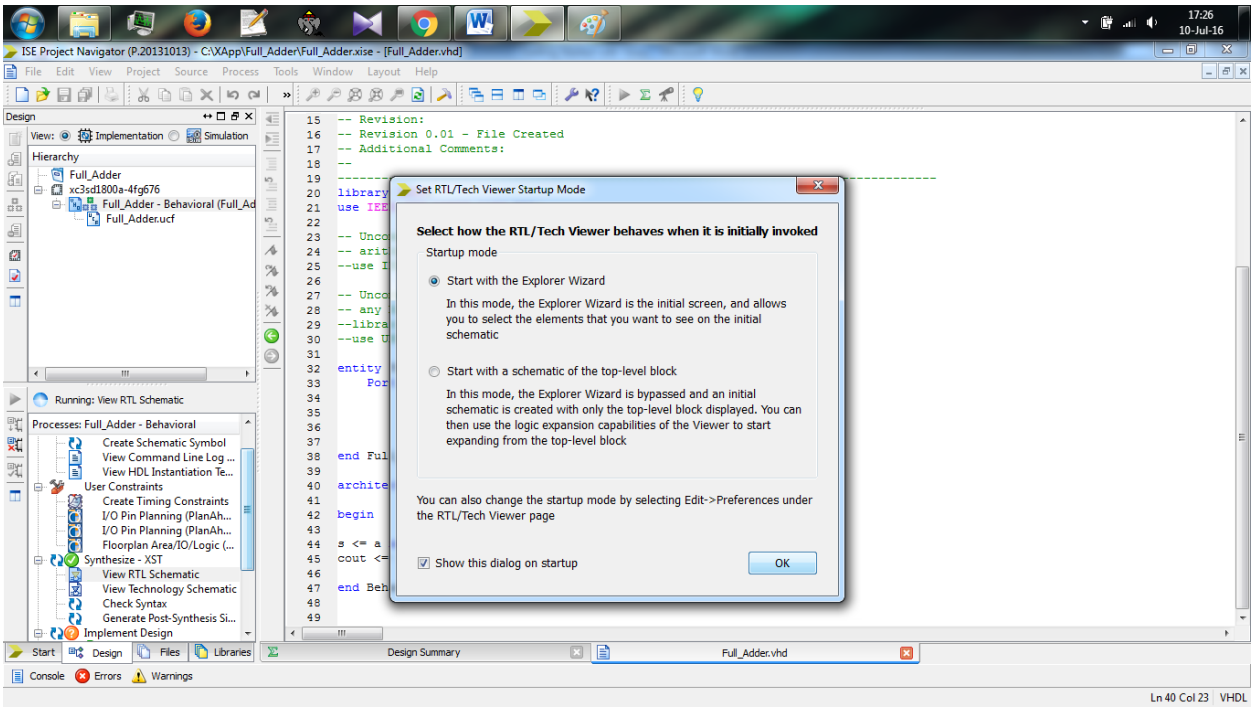


Fig: 10

After completion of the process of synthesize we need to view the RTL-schematic diagram of our circuit. To do so, double click on the 'View RTL-Schematic' option.

A set-up window will appear as shown in fig: 10.

By selecting the very first option you can able to view the every module by simply adding that module & draw the schematics. After choosing the first option from the set up window the screen will be shown as Fig.11.

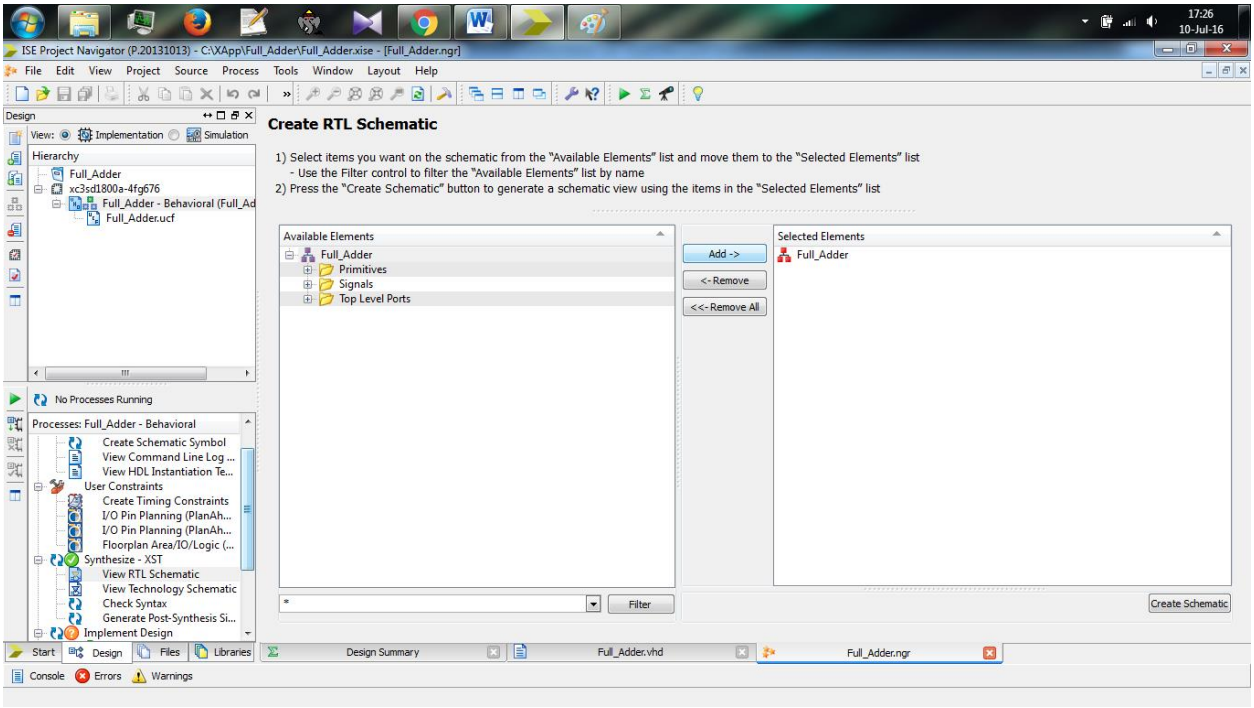


Fig: 11

Here, except the top module there are more 3 modules in the list. You can select any of the modules & create their schematics.

For example here, we are going to generate the RTL-Schematic for the lower most module i.e. the 'Top Level Ports' as depicted in Fig.11.

For the generation of RTL-Schematic we expand that module, here the last module from the list & select the components of that module & click on 'Add' button as shown in Fig.11. Then click on 'Create Schematic' button, as shown in Fig.12.

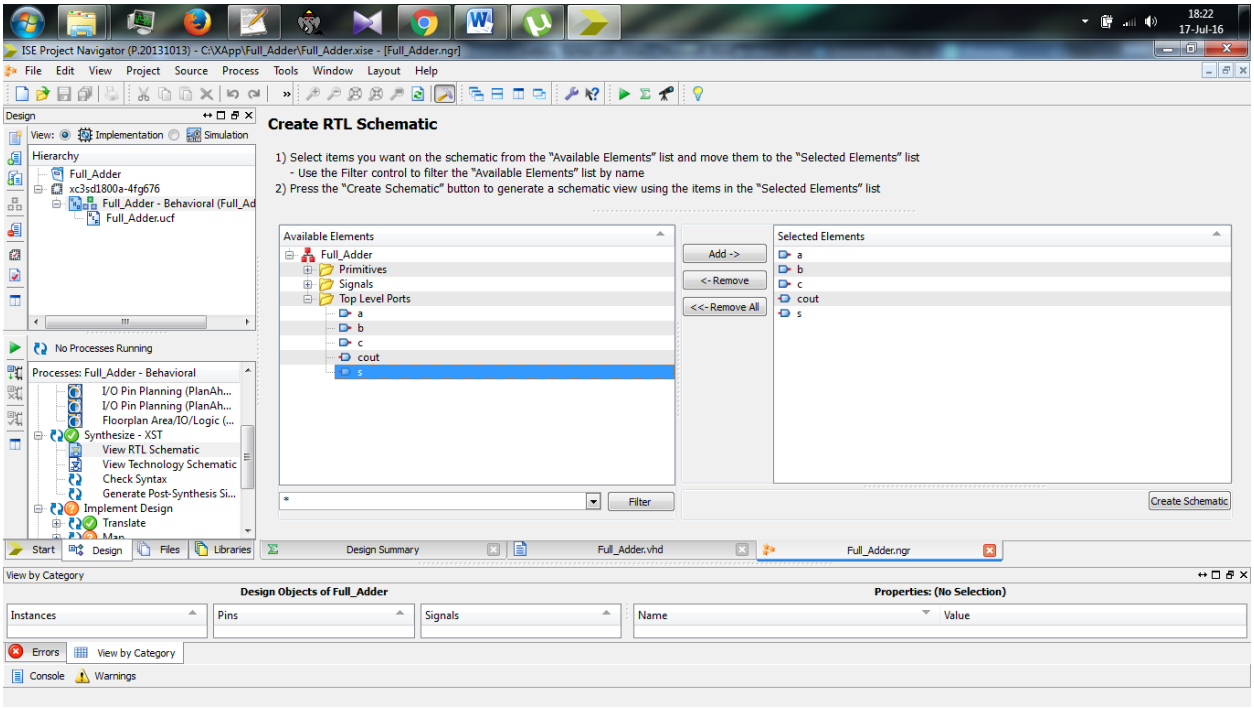


Fig.12

After clicking the 'Create Schematic' button the RTL will be generated as shown in Fig.13.

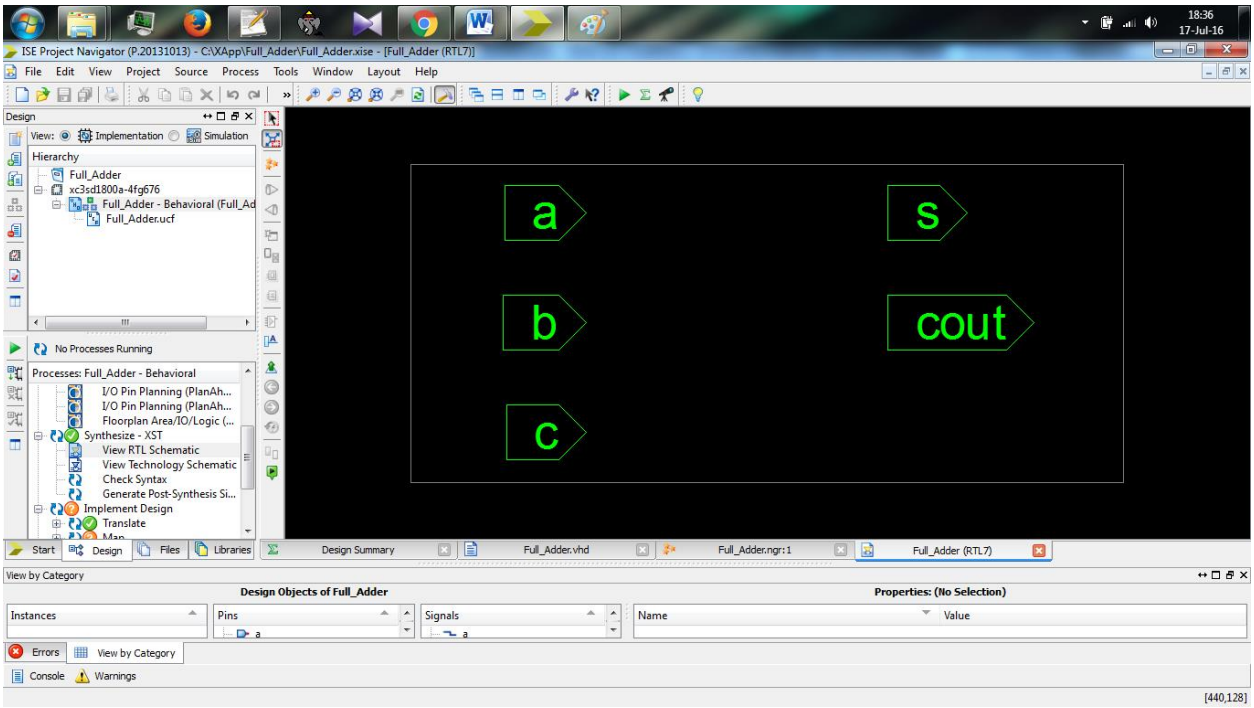


Fig.13

raktimchakraborty27@gmail.com

Double click on the black portion/ background of the RTL-Schematic to view the full RTL of that module will be shown by Xilinx as shown in Fig.14.

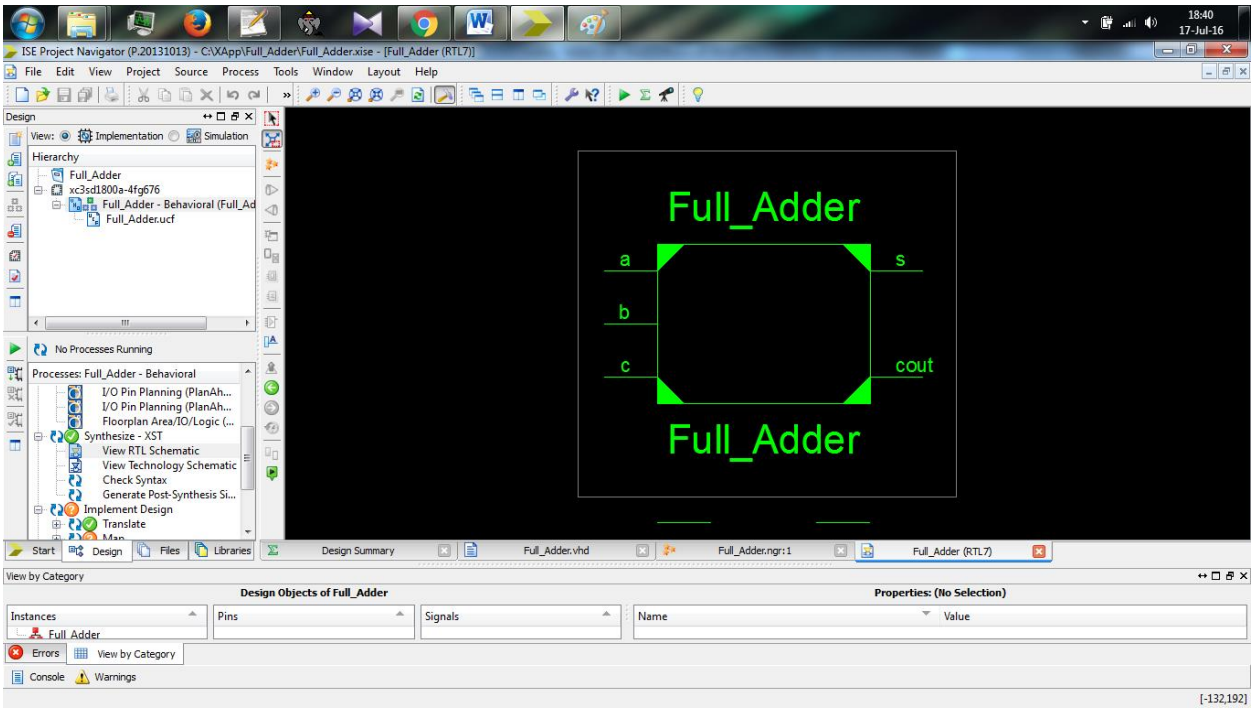


Fig.14

By repeat this whole procedure of this step user can able to create the RTL-Schematic of every module & by adding the top module user can get the complete RTL-Schematic of the circuit as shown in Fig.15.

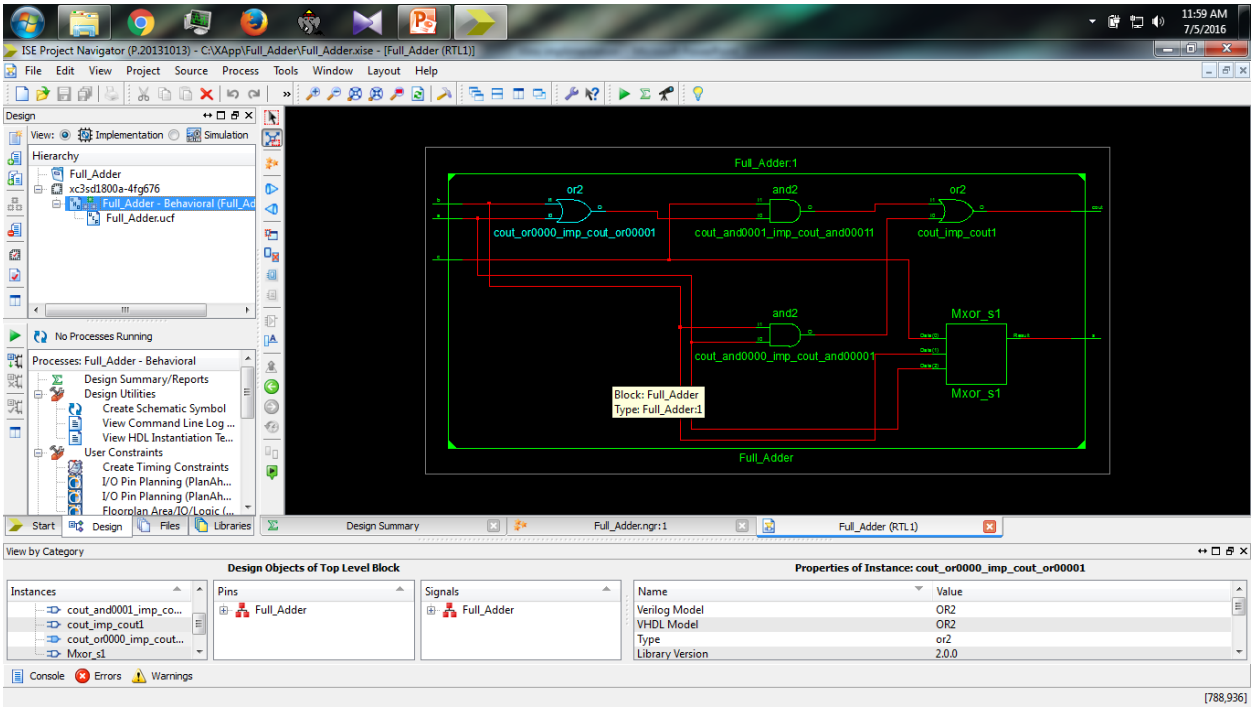


Fig.15

And if you select the second option from the window than you need not to do anything just select the second option & click ‘Ok’. The RTL-Schematic will be generated after the click as shown in Fig.15.

Step: 11

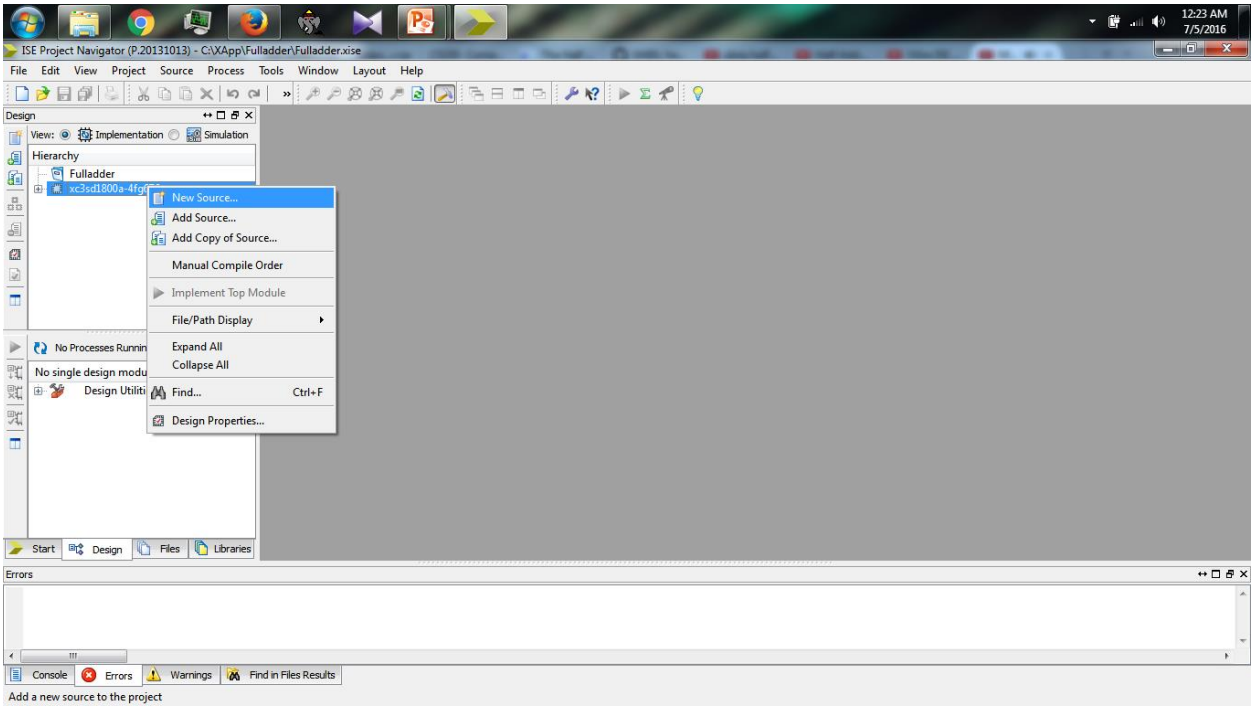


Fig: 16

Till step: 10, we are doing our implementation portion of our circuit. Now, we are going forward towards the simulation option.

To do so, we are now adding another source to our project as we have done in step: 5.

In this step, repeat step 5.

Step: 12

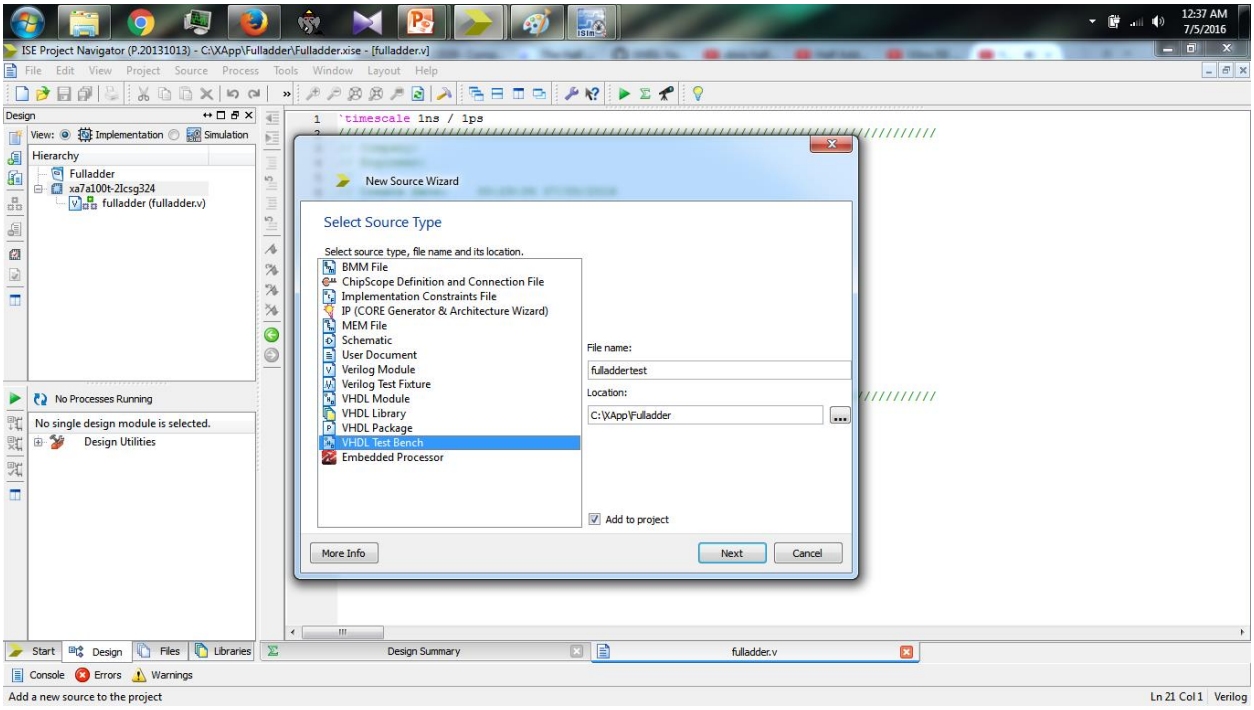


Fig: 17

Here we will select the ‘VHDL Test Bench’ option for simulation & give a suitable file name to this file & save to the same directory in which we are saving our whole project. Xilinx will automatically set that directory in ‘Location’ if it is not set to that directory you we have to set that path manually.

After that click on ‘Next’ and again ‘Next’ and then click on ‘Finish’ button.

Step: 13

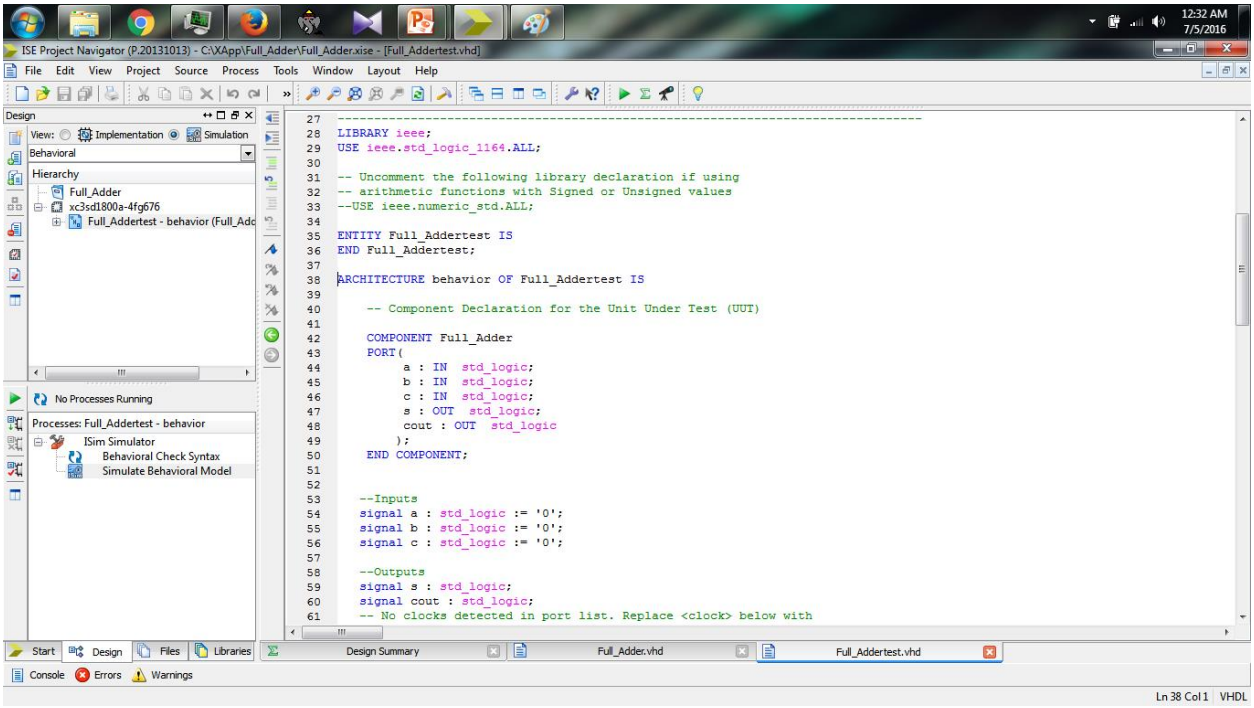


Fig: 18

The screen as shown in fig: 18 will appear after completion the previous step.

Here we mainly define the truth table i.e. the input stimulus of our circuit.

To make the thing easy to understand we are attaching the sample code of a full adder circuit for simulation is shown below:

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

ENTITY Full_Addertest IS

END Full_Addertest;


ARCHITECTURE behavior OF Full_Addertest IS
```

```
-- Component Declaration for the Unit Under Test (UUT)

 COMPONENT Full_Adder
 PORT(
      a : IN  std_logic;
      b : IN  std_logic;
      c : IN  std_logic;
      s : OUT  std_logic;
      cout : OUT  std_logic
     );
 END COMPONENT;

--Inputs
signal a : std_logic := '0';
signal b : std_logic := '0';
signal c : std_logic := '0';

                                        --Outputs
signal s : std_logic;
signal cout : std_logic;

constant clock_period : time := 10 ns;
                                        signal clock : std_logic;
```

raktimchakraborty27@gmail.com

```
BEGIN

                                        -- Instantiate the Unit Under Test (UUT)
  uut: Full_Adder PORT MAP (
        a => a,
        b => b,
        c => c,
        s => s,
        cout => cout
      );

  -- Clock process definitions
  clock_process :process
  begin
                                        clock <= '0';
                                        wait for clock_period/2;
                                        clock <= '1';
                                        wait for clock_period/2;
  end process;


  -- Stimulus process
```

```
  stim_proc: process
  begin
     -- hold reset state for 100 ns.
     wait for 100 ns;
     wait for clock_period*10;
     -- insert stimulus here
a <= '1' ;
b <= '0' ;
c <= '0' ;
     wait for clock_period*10;
a <= '1' ;
b <= '0' ;
c <= '1' ;
     wait for clock_period*10;
a <= '1' ;
b <= '1' ;
c <= '0' ;
wait for clock_period*10;


a <= '1' ;
b <= '0' ;
c <= '1' ;
```

```
    wait for clock_period*10;
 a <= '0' ;
 b <= '0' ;
 c <= '0' ;
    wait for clock_period*10;
 a <= '0' ;
 b <= '0' ;
 c <= '1' ;
    wait for clock_period*10;
 a <= '0' ;
 b <= '1' ;
 c <= '0' ;
    wait for clock_period*10;
 a <= '0' ;
 b <= '1' ;
 c <= '1' ;
    wait for clock_period*10;
  end process;
END;
```

raktimchakraborty27@gmail.com

Step: 14

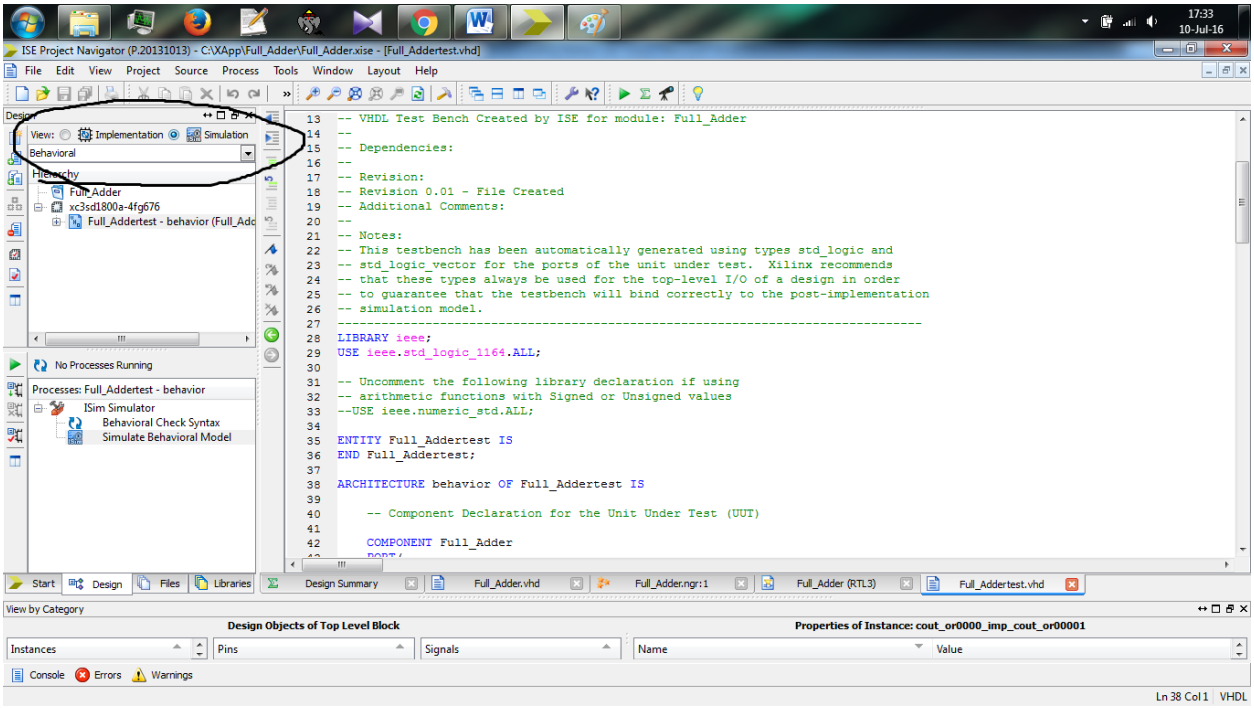


Fig: 19

After defining the input stimulus in the simulation code we select the option simulation as shown in fig: 19 [if it is not selected].

After that in the ‘Processes’ window double click on ‘Behavioral Check Syntax’ option to check the errors & to make the code ready for simulation as shown in fig:20.

After that, double click on ‘Simulate Behavioral Model’ to generate the timing diagram of our circuit as shown in fig: 21.

Fig: 21 depict the final simulated result of a full adder circuit which we have implemented here for demonstration.

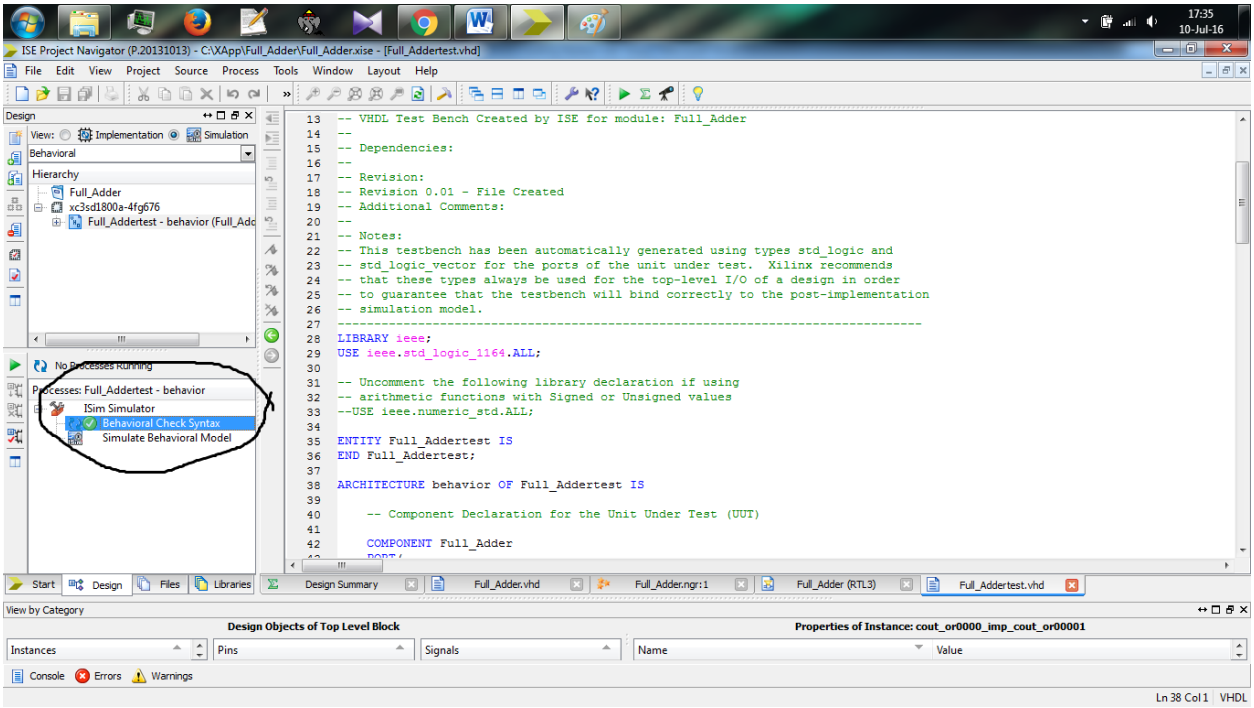

ISE Project Navigator (P.20131013) - C:\XApp\Full_Adder\Full_Adder.xise - [Full_Addertest.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Behavioral
Hierarchy
Full_Adder
xc3sd1800a-4fg676
Full_Addertest - behavior (Full_Adc
No Processes Running
Processes: Full_Addertest - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
-- VHDL Test Bench Created by ISE for module: Full_Adder
-- Dependencies:
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test. Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
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;
ENTITY Full_Addertest IS
END Full_Addertest;
ARCHITECTURE behavior OF Full_Addertest IS
-- Component Declaration for the Unit Under Test (UUT)
COMPONENT Full_Adder
Start Design Files Libraries
Design Summary Full_Adder.vhd Full_Adder.ngr:1 Full_Adder (RTL3) Full_Addertest.vhd
View by Category
Design Objects of Top Level Block
Properties of Instance: cout_or0000_imp_cout_or00001
Instances Pins Signals Name Value
Console Errors Warnings
Ln 38 Col 1 VHDL

Fig: 20

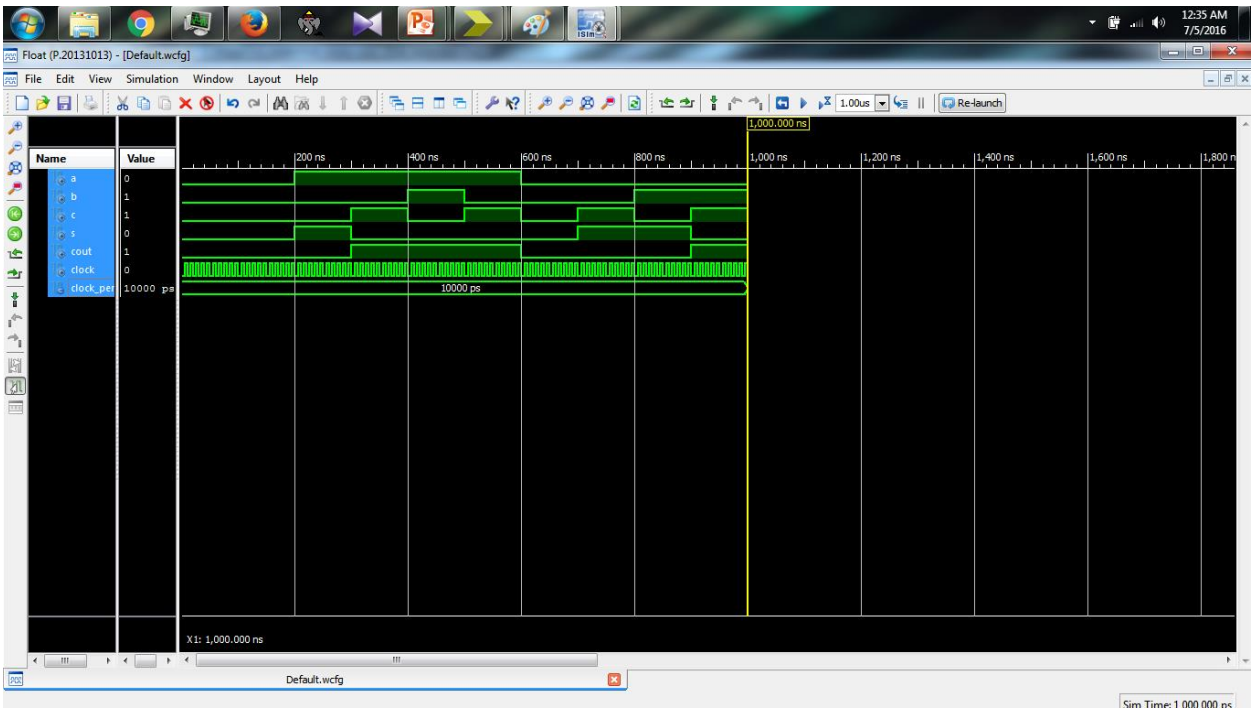

Float (P.20131013) - [Default.wcfg]
File Edit View Simulation Window Layout Help
Re-launch
Name
Value
a
b
c
s
cout
clock
clock_per
10000 ps
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns
1,400 ns
1,600 ns
1,800 n
1,000.000 ns
10000 ps
X1: 1,000.000 ns
Default.wcfg

Fig: 21

Few more combinational circuits using Xilinx ISE

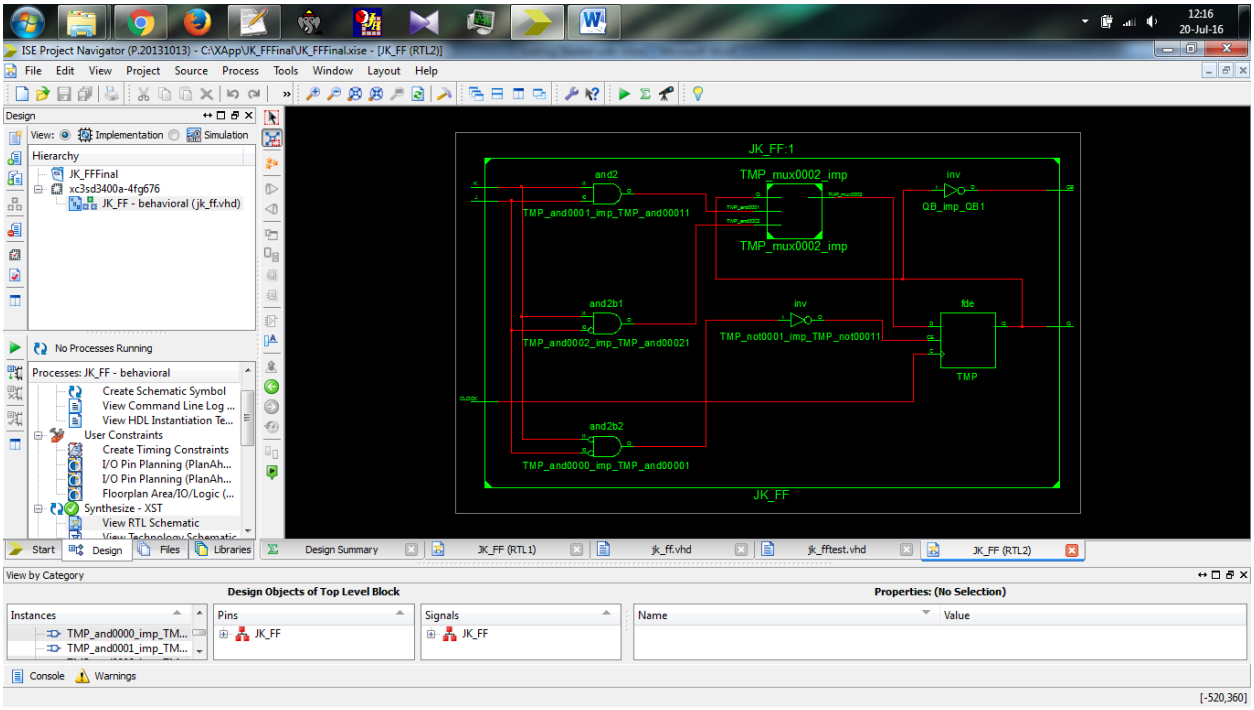
JK Flip-Flop:

JK Flip-flop behavioral:

```
library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;

entity JK_FF is
PORT( J,K,CLOCK: in std_logic;
Q, QB: out std_logic);
end JK_FF;

Architecture behavioral of JK_FF is
begin
PROCESS(CLOCK)
variable TMP: std_logic;
begin
if(CLOCK='1' and CLOCK'EVENT) then
if(J='0' and K='0')then
TMP:=TMP;
elsif(J='1' and K='1')then
```

```
TMP:= not TMP;
elsif(J='0' and K='1')then
TMP:='0';
else
TMP:='1';
end if;
end if;
Q<=TMP;
QB <=not TMP;
end PROCESS;
end behavioral;
```

The RTL-Schematic Diagram of JK-Flip Flop:



raktimchakraborty27@gmail.com

JK Flip-Flop Simulation:

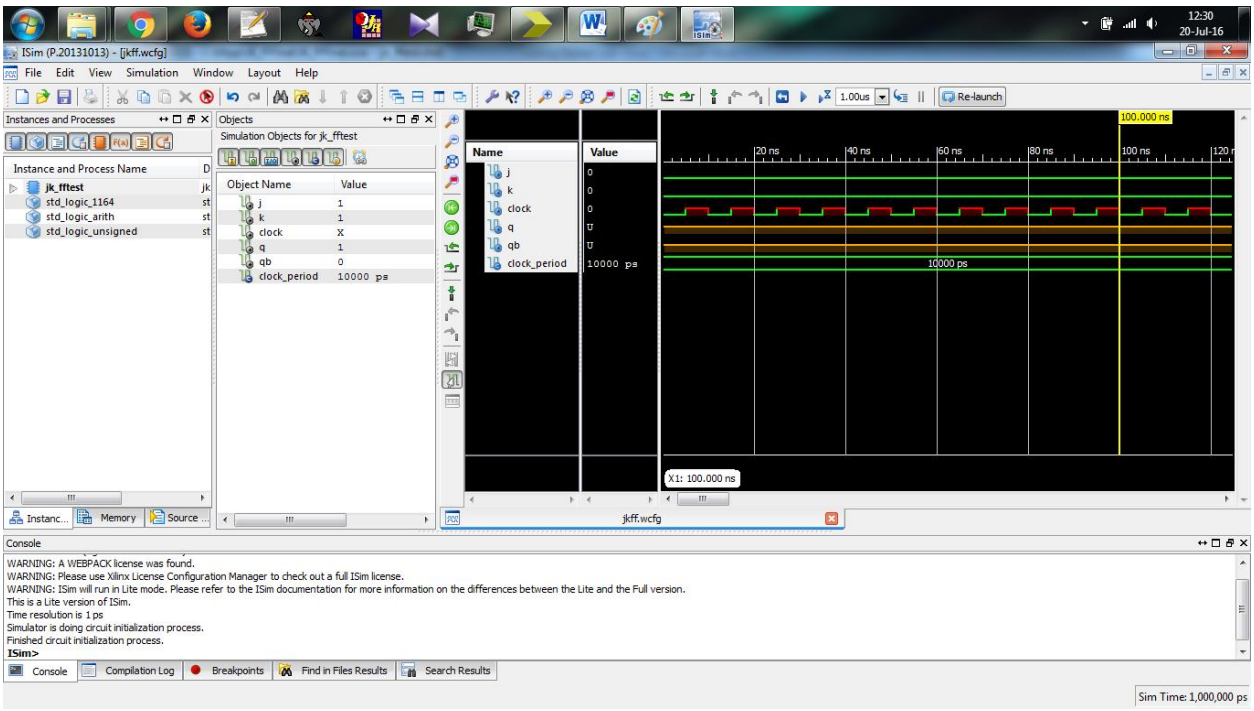
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY jk_fftest IS
END jk_fftest;
ARCHITECTURE behavior OF jk_fftest IS
    COMPONENT JK_FF
    PORT(
         J : IN  std_logic;
         K : IN  std_logic;
         CLOCK : IN  std_logic;
         Q : OUT  std_logic;
         QB : OUT  std_logic
        );
    END COMPONENT;
   --Inputs
   signal J : std_logic := '0';
   signal K : std_logic := '0';
   signal CLOCK : std_logic := '0';
                                        --Outputs
   signal Q : std_logic;
   signal QB : std_logic;
   -- Clock period definitions
```

```
  constant CLOCK_period : time := 10 ns;
BEGIN
  uut: JK_FF PORT MAP (
        J => J,
        K => K,
        CLOCK => CLOCK,
        Q => Q,
        QB => QB
      );

  -- Clock process definitions
  CLOCK_process :process
  begin
                                    CLOCK <= '0';
                                    wait for CLOCK_period/2;
                                    CLOCK <= '1';
                                    wait for CLOCK_period/2;
  end process;
  -- Stimulus process
  stim_proc: process
  begin
    -- hold reset state for 100 ns.
    wait for 100 ns;
```

```
    wait for CLOCK_period*10;

    -- insert stimulus here
J<='0';
K<='0';
CLOCK<='0';
wait for CLOCK_period*10;

J<='0';
K<='1';
CLOCK<='0';
wait for CLOCK_period*10;

J<='1';
K<='0';
CLOCK<='0';
wait for CLOCK_period*10;

J<='1';
K<='1';
CLOCK<='0';
wait for CLOCK_period*10;
```

```
J<='0';
K<='0';
CLOCK<='1';
wait for CLOCK_period*10;

J<='0';
K<='1';
CLOCK<='1';
wait for CLOCK_period*10;
J<='1';
K<='0';
CLOCK<='1';
wait for CLOCK_period*10;
J<='1';
K<='1';
CLOCK<='1';
wait for CLOCK_period*10;
      wait;
   end process;
END;
```

The Clock Pulse Simulation Diagram [0-100 ns]:



- 8:1 Multiplexer:

8:1 Multiplexer behavioral:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity eight_input_mux is
    Port ( d0 : in  STD_LOGIC;
           d1 : in  STD_LOGIC;
           d2 : in  STD_LOGIC;
           d3 : in  STD_LOGIC;
           d4 : in  STD_LOGIC;
           d5 : in  STD_LOGIC;
           d6 : in  STD_LOGIC;
```

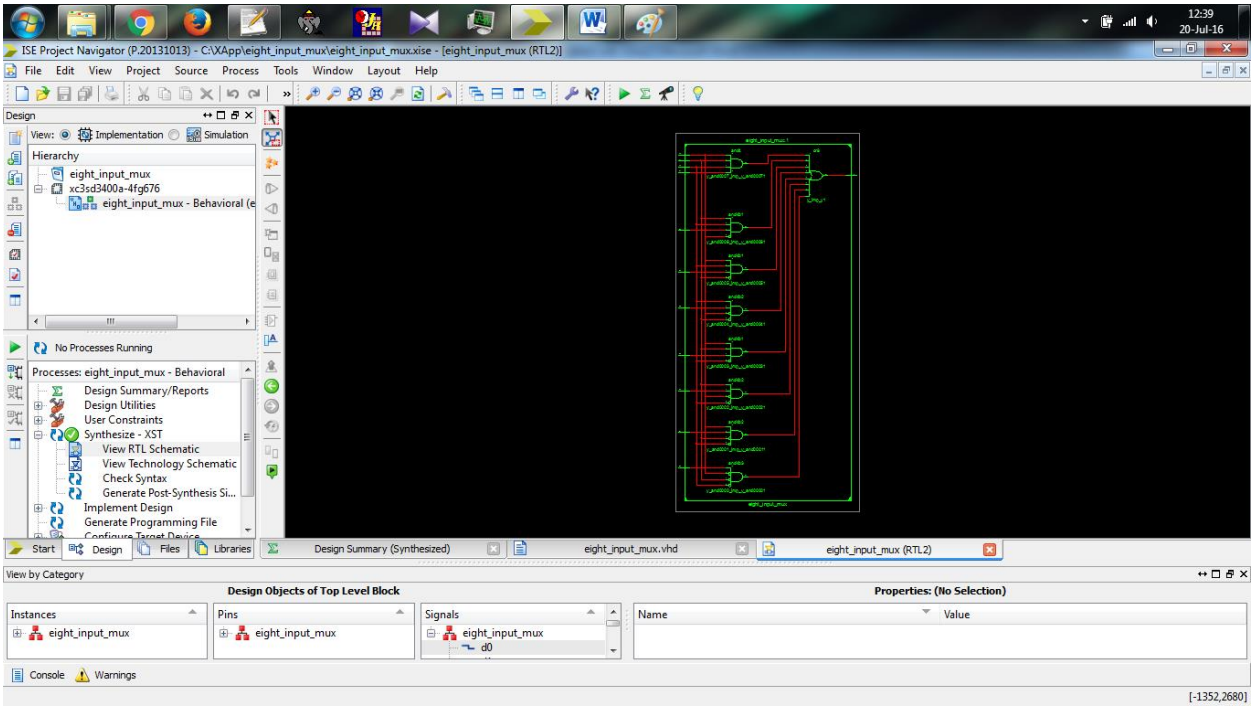
```
        d7 : in  STD_LOGIC;
        s0 : in  STD_LOGIC;
        s1 : in  STD_LOGIC;
        s2 : in  STD_LOGIC;
        y : out  STD_LOGIC);
end eight_input_mux;

architecture Behavioral of eight_input_mux is

begin
y <= ( (  d0) and (not s2) and ( not s1) and (not s0) ) or

( (  d1) and (not s2) and ( not s1) and s0 ) or

( (  d2) and (not s2) and (  s1) and  (not s0) ) or


( (  d3) and (not s2) and (  s1) and s0 ) or

( (  d4) and ( s2) and ( not s1) and (not s0) ) or

( (  d5) and ( s2) and ( not s1) and s0 ) or
```

((d6) and (s2) and (s1) and (not s0)) or

((d7) and (s2) and (s1) and s0);

end Behavioral;

The RTL-Schematic Diagram of 8:1 Multiplexer:



8:1 Multiplexer Simulation:

LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

ENTITY eight_in_mux IS

END eight_in_mux;

```
ARCHITECTURE behavior OF eight_in_mux IS

   -- Component Declaration for the Unit Under Test (UUT)

   COMPONENT eight_input_mux
   PORT(
        d0 : IN  std_logic;
        d1 : IN  std_logic;
        d2 : IN  std_logic;
        d3 : IN  std_logic;
        d4 : IN  std_logic;
        d5 : IN  std_logic;
        d6 : IN  std_logic;
        d7 : IN  std_logic;
        s0 : IN  std_logic;
        s1 : IN  std_logic;
        s2 : IN  std_logic;
        y : OUT  std_logic
       );
   END COMPONENT;


  --Inputs
```

```
   signal d0 : std_logic := '0';
   signal d1 : std_logic := '0';
   signal d2 : std_logic := '0';
   signal d3 : std_logic := '0';
   signal d4 : std_logic := '0';
   signal d5 : std_logic := '0';
   signal d6 : std_logic := '0';
   signal d7 : std_logic := '0';
   signal s0 : std_logic := '0';
   signal s1 : std_logic := '0';
   signal s2 : std_logic := '0';

                                        --Outputs
   signal y : std_logic;


   constant clock_period : time := 10 ns;
                                        signal clock : std_logic;

BEGIN

                                        -- Instantiate the Unit Under Test
(UUT)
```

```
uut: eight_input_mux PORT MAP (
      d0 => d0,
      d1 => d1,
      d2 => d2,
      d3 => d3,
      d4 => d4,
      d5 => d5,
      d6 => d6,
      d7 => d7,
      s0 => s0,
      s1 => s1,
      s2 => s2,
      y => y
    );

-- Clock process definitions
clock_process :process
begin
                                    clock <= '0';
                                    wait for clock_period/2;
                                    clock <= '1';
                                    wait for clock_period/2;
end process;
```

```
   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;

      wait for clock_period*10;

      -- insert stimulus here
d0<= '1';
d1<= '0';
d2<= '0';
d3<= '0';
d4<= '0';
d5<= '0';
d6<= '0';
d7<= '0';
s0<= '0';
s1<= '0';
s2<= '0';
      wait for clock_period*10;
```

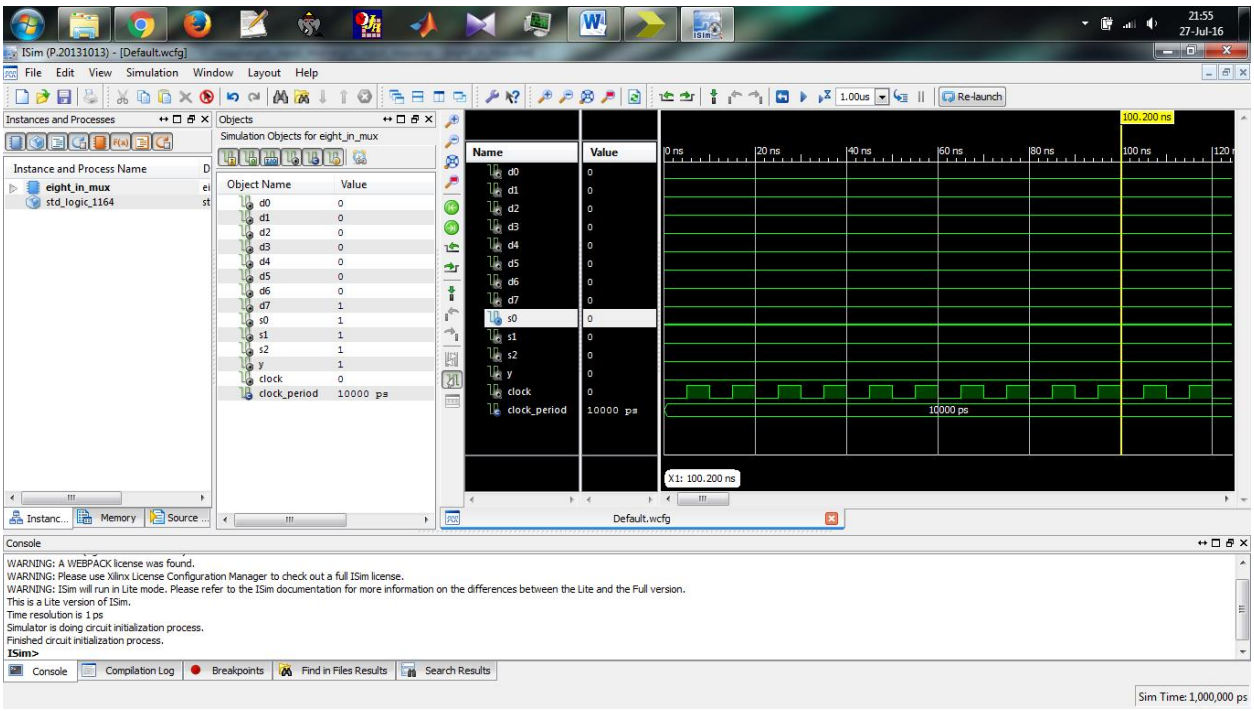
```
d0<= '0';
d1<= '1';
d2<= '0';
d3<= '0';
d4<= '0';
d5<= '0';
d6<= '0';
d7<= '0';
s0<= '0';
s1<= '0';
s2<= '1';
    wait for clock_period*10;

d0<= '0';
d1<= '0';
d2<= '1';
d3<= '0';
d4<= '0';
d5<= '0';
d6<= '0';
d7<= '0';
s0<= '0';
```

```
s1<= '1';
s2<= '0';
    wait for clock_period*10;

d0<= '0';
d1<= '0';
d2<= '0';
d3<= '1';
d4<= '0';
d5<= '0';
d6<= '0';
d7<= '0';
s0<= '0';
s1<= '1';
s2<= '1';
    wait for clock_period*10;
d0<= '0';
d1<= '0';
d2<= '0';
d3<= '0';
d4<= '1';
d5<= '0';
d6<= '0';
```

```
d7<= '0';
s0<= '1';
s1<= '0';
s2<= '0';
    wait for clock_period*10;
d0<= '0';
d1<= '0';
d2<= '0';
d3<= '0';
d4<= '0';
d5<= '1';
d6<= '0';
d7<= '0';
s0<= '1';
s1<= '0';
s2<= '1';
    wait for clock_period*10;
d0<= '0';
d1<= '0';
d2<= '0';
d3<= '0';
d4<= '0';
d5<= '0';
```

```
d6<= '1';
d7<= '0';
s0<= '1';
s1<= '1';
s2<= '0';
     wait for clock_period*10;
d0<= '0';
d1<= '0';
d2<= '0';
d3<= '0';
d4<= '0';
d5<= '0';
d6<= '0';
d7<= '1';
s0<= '1';
s1<= '1';
s2<= '1';
     wait for clock_period*10;
     wait;
  end process;
END;
```

The Clock Pulse Simulation Diagram [0-100ns]:



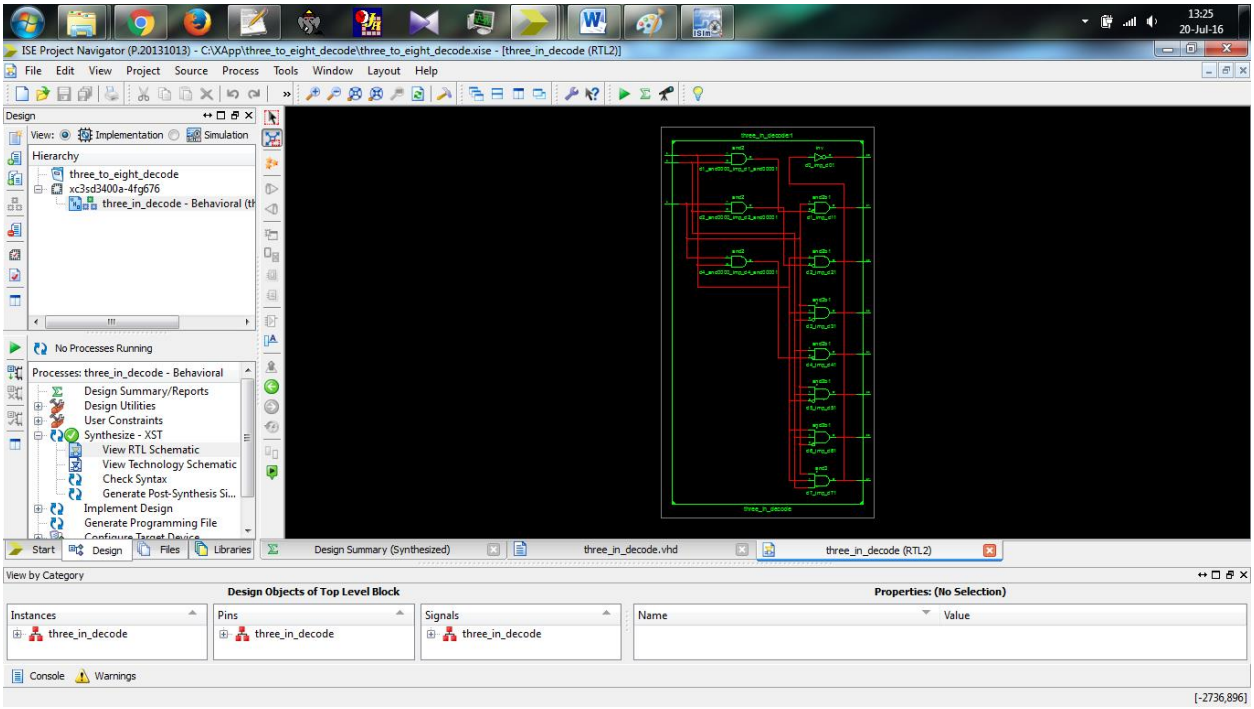
3:8 Decoder:

3:8 Decoder Behavioral:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity three_in_decode is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : in  STD_LOGIC;
           d0 : out  STD_LOGIC;
           d1 : out  STD_LOGIC;
           d2 : out  STD_LOGIC;
```

```
        d3 : out  STD_LOGIC;
        d4 : out  STD_LOGIC;
        d5 : out  STD_LOGIC;
        d6 : out  STD_LOGIC;
        d7 : out  STD_LOGIC);
end three_in_decode;

architecture Behavioral of three_in_decode is

begin
d0 <= not(a and b and c);
d1 <= not ( a and b) and c;
d2 <= not ( a and c) and b;
d3 <= not a and b and c;
d4 <= not ( b and c) and a;
d5 <= not b and a and c;
d6 <= not c and a and b;
d7 <= a and b and c;

end Behavioral;
```

The RTL-Schematic of 3:8 Decoder:



3:8 Decoder Simulation:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY three_to_eight_test IS
END three_to_eight_test;

ARCHITECTURE behavior OF three_to_eight_test IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT three_in_decode
```

```
 PORT(
      a : IN  std_logic;
      b : IN  std_logic;
      c : IN  std_logic;
      d0 : OUT  std_logic;
      d1 : OUT  std_logic;
      d2 : OUT  std_logic;
      d3 : OUT  std_logic;
      d4 : OUT  std_logic;
      d5 : OUT  std_logic;
      d6 : OUT  std_logic;
      d7 : OUT  std_logic
     );
 END COMPONENT;


--Inputs
signal a : std_logic := '0';
signal b : std_logic := '0';
signal c : std_logic := '0';

                                --Outputs
signal d0 : std_logic;
```

raktimchakraborty27@gmail.com

```
   signal d1 : std_logic;
   signal d2 : std_logic;
   signal d3 : std_logic;
   signal d4 : std_logic;
   signal d5 : std_logic;
   signal d6 : std_logic;
   signal d7 : std_logic;


   constant clock_period : time := 10 ns;
                                   signal clock : std_logic;

BEGIN

                                   -- Instantiate the Unit Under Test
(UUT)
   uut: three_in_decode PORT MAP (
         a => a,
         b => b,
         c => c,
         d0 => d0,
         d1 => d1,
         d2 => d2,
```

```
      d3 => d3,
      d4 => d4,
      d5 => d5,
      d6 => d6,
      d7 => d7
    );

-- Clock process definitions
clock_process :process
begin
                                    clock <= '0';
                                    wait for clock_period/2;
                                    clock <= '1';
                                    wait for clock_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   wait for 100 ns;
```

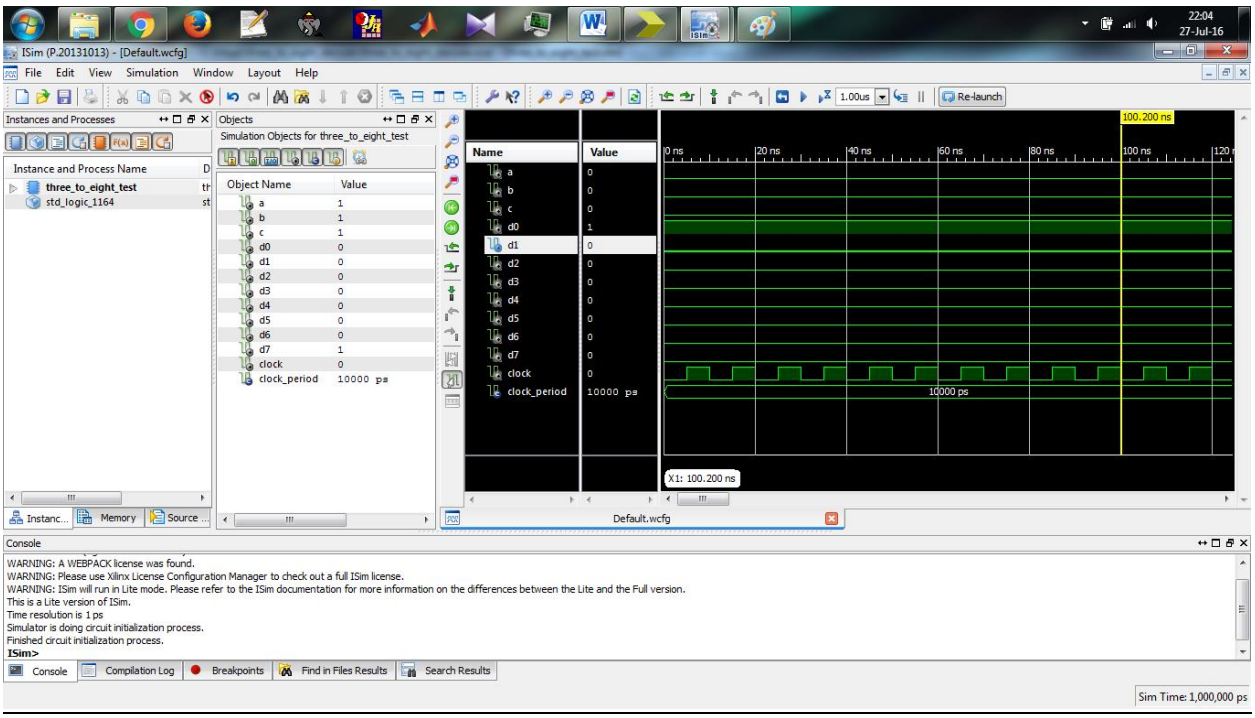
```
    wait for clock_period*10;

    -- insert stimulus here
a <= '0';
b <= '0';
c <= '0';
wait for clock_period*10;

a <= '0';
b <= '1';
c <= '0';
wait for clock_period*10;

a <= '0';
b <= '0';
c <= '1';
wait for clock_period*10;

a <= '0';
b <= '1';
c <= '1';
wait for clock_period*10;
```

```
a <= '1';
b <= '0';
c <= '0';
wait for clock_period*10;

a <= '1';
b <= '0';
c <= '1';
wait for clock_period*10;

a <= '1';
b <= '1';
c <= '0';
wait for clock_period*10;

a <= '1';
b <= '1';
c <= '1';
wait for clock_period*10;

    wait;
  end process;
```

raktimchakraborty27@gmail.com

END;

The Clock Pulse Simulation Diagram [0-100 ns]:



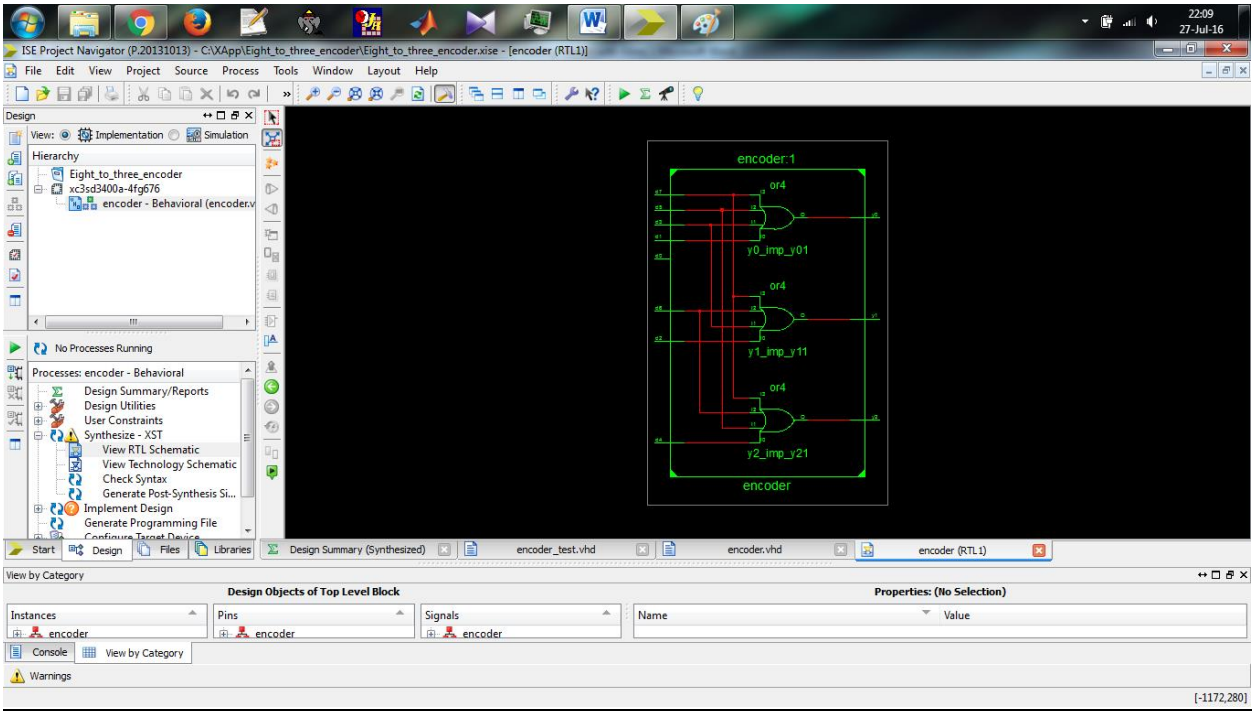
- **8:3 Encoder:**

8:3 Encoder Behavioral:

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


entity encoder is
    Port ( d0 : in  STD_LOGIC;
           d1 : in  STD_LOGIC;
           d2 : in  STD_LOGIC;
           d3 : in  STD_LOGIC;
           d4 : in  STD_LOGIC;
```

```
        d5 : in  STD_LOGIC;
        d6 : in  STD_LOGIC;
        d7 : in  STD_LOGIC;
        y0 : out  STD_LOGIC;
        y1 : out  STD_LOGIC;
        y2 : out  STD_LOGIC);
end encoder;

architecture Behavioral of encoder is

begin
y0 <= d1 or d3 or d5 or d7;
y1 <= d2 or d3 or d6 or d7;
y2 <= d4 or d5 or d6 or d7;

end Behavioral;
```

The RTL Schematic Diagram of 8:3 Encoder:



8:3 Encoder Simulation:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY encoder_test IS
END encoder_test;

ARCHITECTURE behavior OF encoder_test IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT encoder
```

```
 PORT(
      d0 : IN  std_logic;
      d1 : IN  std_logic;
      d2 : IN  std_logic;
      d3 : IN  std_logic;
      d4 : IN  std_logic;
      d5 : IN  std_logic;
      d6 : IN  std_logic;
      d7 : IN  std_logic;
      y0 : OUT  std_logic;
      y1 : OUT  std_logic;
      y2 : OUT  std_logic
     );
 END COMPONENT;


--Inputs
signal d0 : std_logic := '0';
signal d1 : std_logic := '0';
signal d2 : std_logic := '0';
signal d3 : std_logic := '0';
signal d4 : std_logic := '0';
signal d5 : std_logic := '0';
```

```
   signal d6 : std_logic := '0';
   signal d7 : std_logic := '0';

                                        --Outputs
   signal y0 : std_logic;
   signal y1 : std_logic;
   signal y2 : std_logic;

   constant clock_period : time := 10 ns;
                                        signal clock : std_logic;

BEGIN

                                        -- Instantiate the Unit Under Test
(UUT)
   uut: encoder PORT MAP (
         d0 => d0,
         d1 => d1,
         d2 => d2,
         d3 => d3,
         d4 => d4,
         d5 => d5,
         d6 => d6,
```

```
      d7 => d7,
      y0 => y0,
      y1 => y1,
      y2 => y2
    );

-- Clock process definitions
clock_process :process
begin
                                    clock <= '0';
                                    wait for clock_period/2;
                                    clock <= '1';
                                    wait for clock_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
  -- hold reset state for 100 ns.
  wait for 100 ns;

  wait for clock_period*10;
```

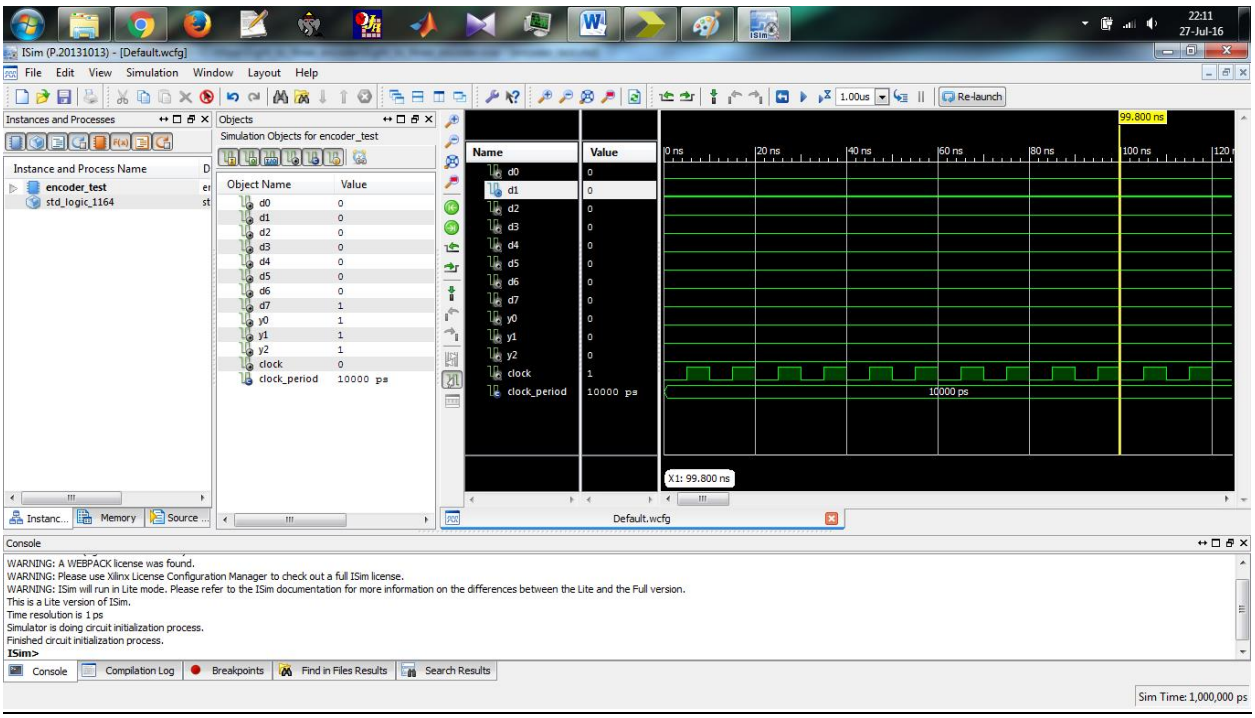
```
    -- insert stimulus here
d0 <= '1';
d1 <= '0';
d2 <= '0';
d3 <= '0';
d4 <= '0';
d5 <= '0';
d6 <= '0';
d7 <= '0';
wait for clock_period*10;

d0 <= '0';
d1 <= '1';
d2 <= '0';
d3 <= '0';
d4 <= '0';
d5 <= '0';
d6 <= '0';
d7 <= '0';
wait for clock_period*10;

d0 <= '0';
```

```
d1 <= '0';
d2 <= '1';
d3 <= '0';
d4 <= '0';
d5 <= '0';
d6 <= '0';
d7 <= '0';
wait for clock_period*10;

d0 <= '0';
d1 <= '0';
d2 <= '0';
d3 <= '1';
d4 <= '0';
d5 <= '0';
d6 <= '0';
d7 <= '0';
wait for clock_period*10;

d0 <= '0';
d1 <= '0';
d2 <= '0';
d3 <= '0';
```

```
d4 <= '1';
d5 <= '0';
d6 <= '0';
d7 <= '0';
wait for clock_period*10;

d0 <= '0';
d1 <= '0';
d2 <= '0';
d3 <= '0';
d4 <= '0';
d5 <= '1';
d6 <= '0';
d7 <= '0';
wait for clock_period*10;

d0 <= '0';
d1 <= '0';
d2 <= '0';
d3 <= '0';
d4 <= '0';
d5 <= '0';
d6 <= '1';
```

```
d7 <= '0';
wait for clock_period*10;

d0 <= '0';
d1 <= '0';
d2 <= '0';
d3 <= '0';
d4 <= '0';
d5 <= '0';
d6 <= '0';
d7 <= '1';
wait for clock_period*10;
      wait;
   end process;

END;
```

The Simulated Clock Pulse Diagram of 8:3 Encoder [0-100 ns]:



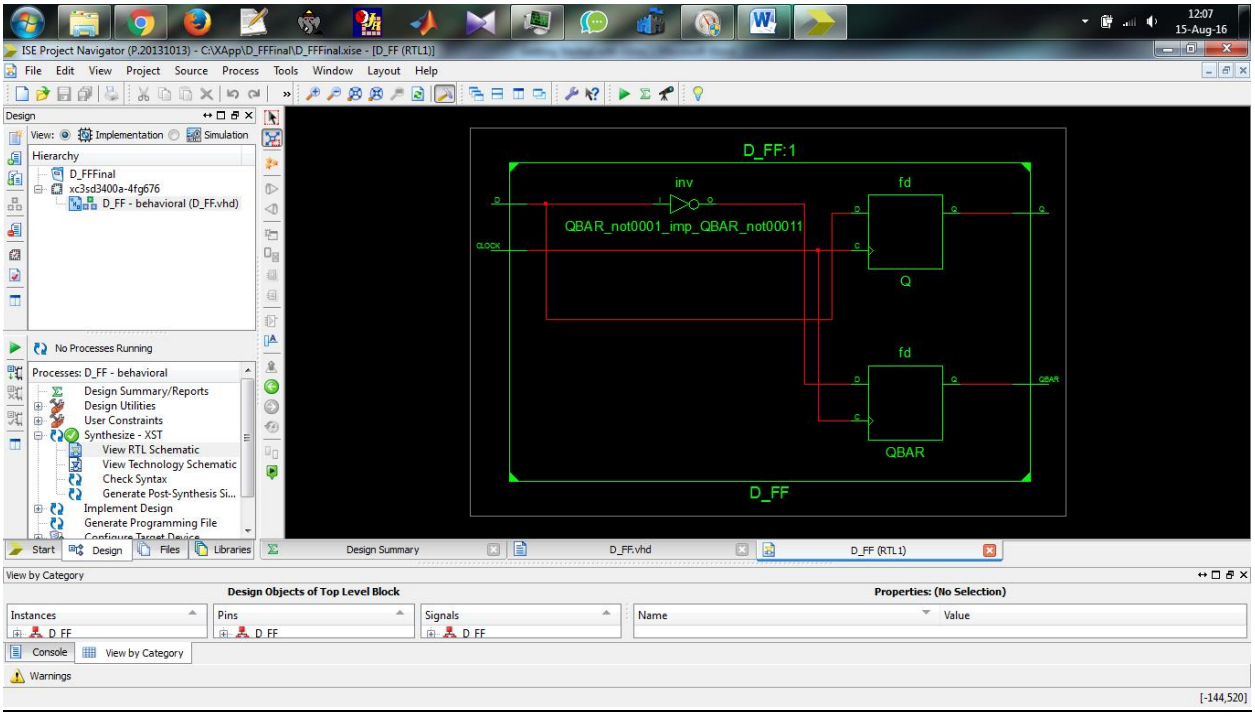
❖ D-Flip-Flop:

D-Flip-Flop Behavioral:

```
library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;

entity D_FF is
PORT( D,CLOCK: in std_logic;
Q,QBAR: out std_logic);
end D_FF;
```

```
architecture behavioral of D_FF is
begin
process(CLOCK)
begin
if(CLOCK='1' and CLOCK'EVENT) then
Q <= D;
QBAR<= NOT D;
end if;
end process;
end behavioral;
```

The RTL Schematic Diagram of D-Flip-Flop:



raktimchakraborty27@gmail.com

❖ **D-Flip-Flop Simulation :**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY DFFTEST IS
END DFFTEST;

ARCHITECTURE behavior OF DFFTEST IS
   COMPONENT D_FF
   PORT(
        D : IN  std_logic;
        CLOCK : IN  std_logic;
        Q : OUT  std_logic
       );
   END COMPONENT;

  --Inputs
  signal D : std_logic := '0';
  signal CLOCK : std_logic := '0';

                                    --Outputs
  signal Q : std_logic;
```

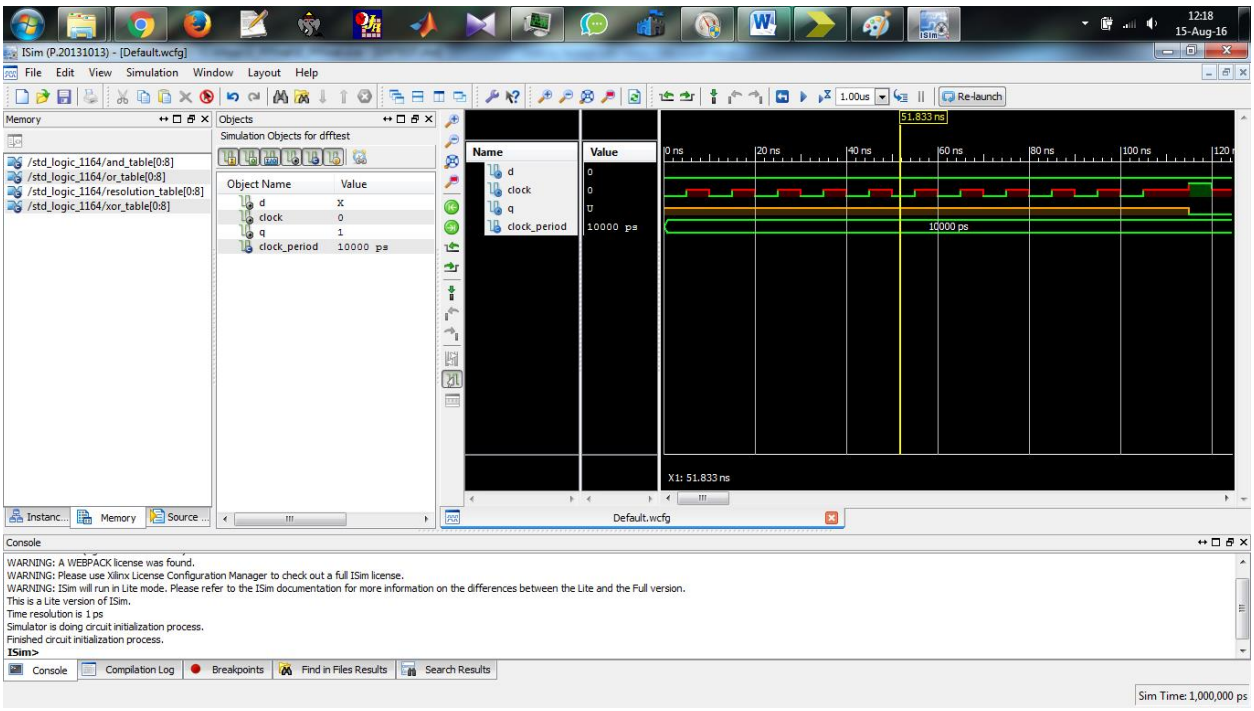
raktimchakraborty27@gmail.com

```
  -- Clock period definitions
  constant CLOCK_period : time := 10 ns;

BEGIN

                                    -- Instantiate the Unit Under Test
(UUT)
  uut: D_FF PORT MAP (
        D => D,
        CLOCK => CLOCK,
        Q => Q
      );

  -- Clock process definitions
  CLOCK_process :process
  begin
                                    CLOCK <= '0';
                                    wait for CLOCK_period/2;
                                    CLOCK <= '1';
                                    wait for CLOCK_period/2;
  end process;
```

```
   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 10 ns;

      wait for CLOCK_period*10;

      -- insert stimulus here
CLOCK<='1';
D<='0';
wait for CLOCK_period*10;

CLOCK<='1';
D<='1';
wait for CLOCK_period*10;

CLOCK<='0';
D<='X';
wait for CLOCK_period*10;

      wait;
   end process;
```

END;

Clock pulse diagram of D-Flip-Flop [0-100 ns]:



2s Complement Adder/Subtractor:

Behavioral:

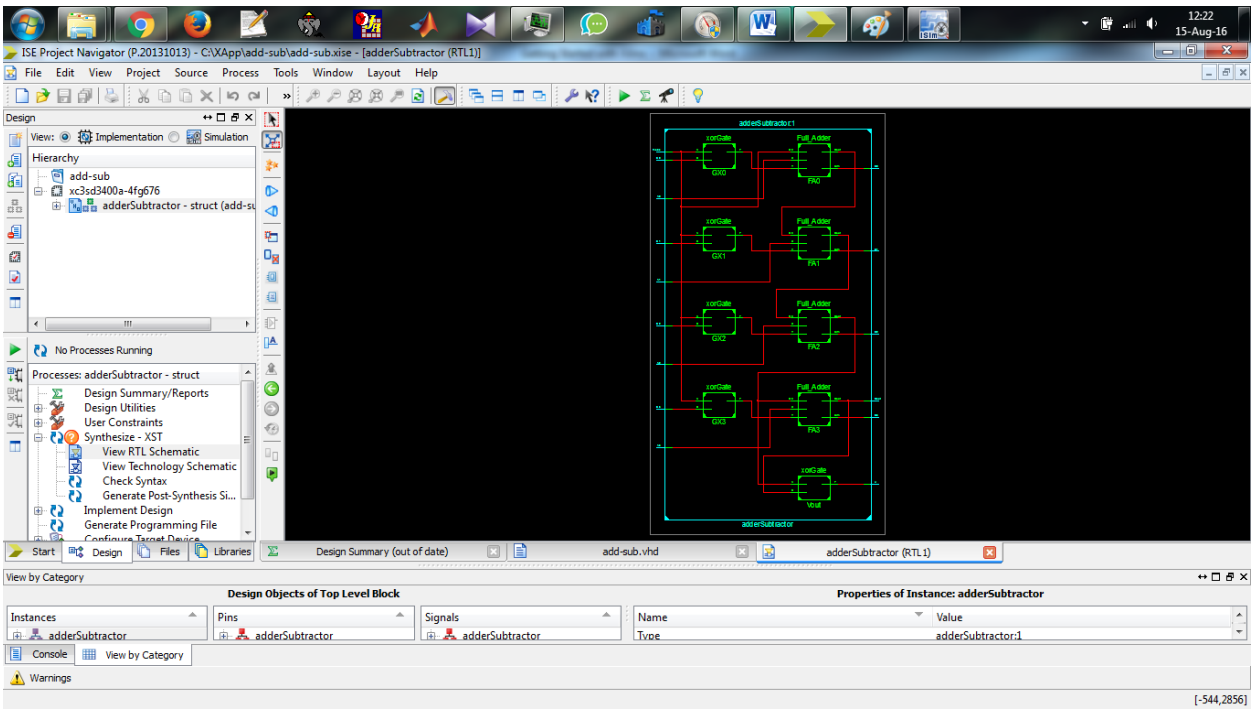
```
library ieee;
use ieee.std_logic_1164.all;
--
entity xorGate is
   port( A, B : in std_logic;
         F : out std_logic);
end xorGate;
architecture func of xorGate is
```

```
begin
   F <= A xor B;
end func;
--*============================
                    -- This is the FULL ADDER
library ieee;
use ieee.std_logic_1164.all;
entity Full_Adder is
   port( X, Y, Cin : in std_logic;
         sum, Cout : out std_logic);
end Full_Adder;
architecture func of Full_Adder is
begin
   sum <= (X xor Y) xor Cin;
   Cout <= (X and (Y or Cin)) or (Cin and Y);
end func;
--
*============================*============================
======


--Now we build the four bit Adder Subtractor
library ieee;
use ieee.std_logic_1164.all;
```

```
entity adderSubtractor is
  port( mode            : in std_logic;
        A3, A2, A1, A0  : in std_logic;
        B3, B2, B1, B0  : in std_logic;
        S3, S2, S1, S0  : out std_logic;
              Cout, V : out std_logic);
end adderSubtractor;
--Structural architecture
architecture struct of adderSubtractor is

  component xorGate is            --XOR component
     port( A, B : in std_logic;
           F : out std_logic);
  end component;

  component Full_Adder is                --FULL ADDER component
    port( X, Y, Cin : in std_logic;
          sum, Cout : out std_logic);
  end component;

  --interconnecting wires
  signal C1, C2, C3, C4: std_logic; --intermediate carries
  signal xor0, xor1, xor2, xor3 : std_logic; --xor outputs
```

```
begin
   GX0: xorGate port map(mode, B0, xor0);
   GX1: xorGate port map(mode, B1, xor1);
   GX2: xorGate port map(mode, B2, xor2);
   GX3: xorGate port map(mode, B3, xor3);

   FA0: Full_Adder port map(A0, xor0, mode,  S0, C1);-- S0
   FA1: Full_Adder port map(A1, xor1, C1,  S1, C2);  -- S1
   FA2: Full_Adder port map(A2, xor2, C2,  S2, C3);  -- S2
   FA3: Full_Adder port map(A3, xor3, C3,  S3, C4);  -- S3

   Vout: xorGate port map(C3, C4, V);              -- V
   Cout <= C4;                                  -- Cout
end struct;
```

The RTL Schematic Diagram of 2s Complement Adder/Subtractor:



Simulation of 2s Complement Adder/Subtractor:

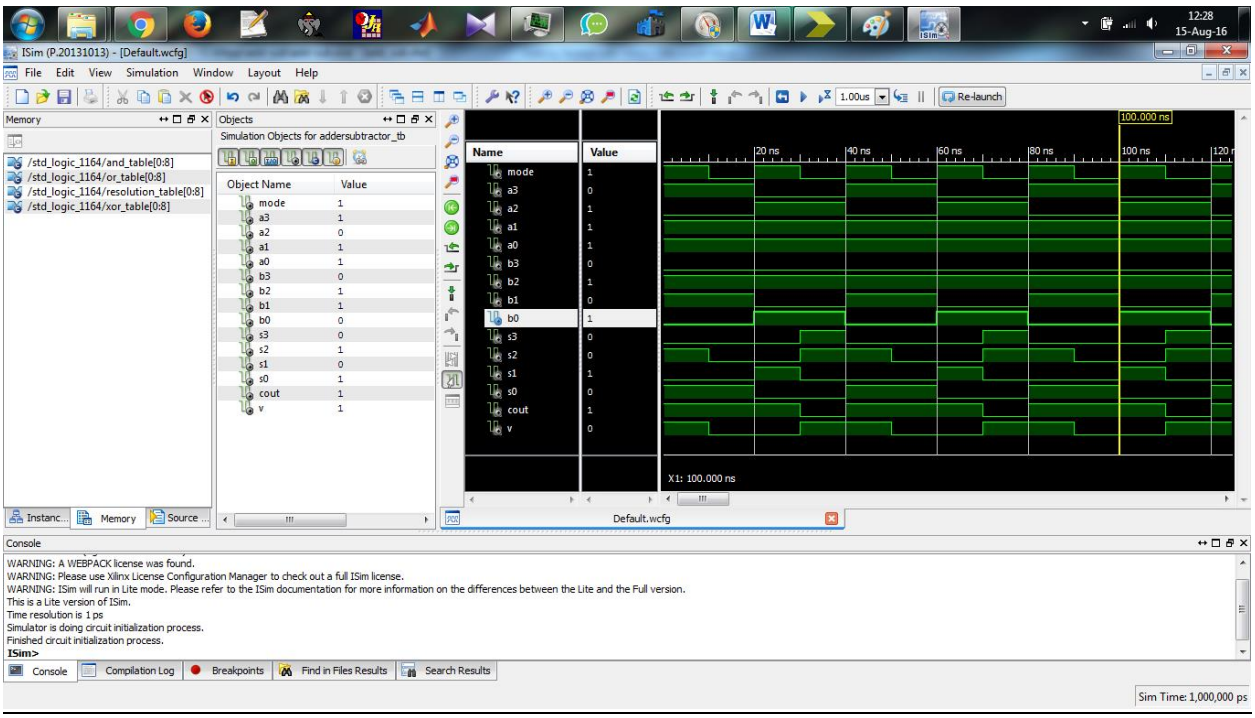
```
library ieee;
use ieee.std_logic_1164.all;
entity adderSubtractor_tb is
end adderSubtractor_tb;
--
architecture testbench of adderSubtractor_tb is
  component adderSubtractor is
    port( mode        : in std_logic;
       A3, A2, A1, A0 : in std_logic;
       B3, B2, B1, B0  : in std_logic;
```

```
        S3, S2, S1, S0  : out std_logic;
              Cout, V  : out std_logic);
    end component;

    signal mode, A3, A2, A1, A0 : std_logic;
    signal B3, B2, B1, B0 : std_logic;
    signal S3, S2, S1, S0, Cout, V : std_logic;

begin
    mapping: adderSubtractor port map(
                              mode,
                        A3, A2, A1, A0,
                        B3, B2, B1, B0,
                        S3, S2, S1, S0,
                             Cout, V);
    process
    begin
      mode <= '1'; -- do subtraction
      wait for 10 ns;
      mode <= '0'; -- do addition
      wait for 10 ns;
    end process;
```

```
process
   variable errCnt : integer :=0;
begin
   --TEST 1
   A3 <= '1';
   A2 <= '0';
   A1 <= '1';
   A0 <= '1';
   --
   B3 <= '0';
   B2 <= '1';
   B1 <= '1';
   B0 <= '0';
   --
   wait for 20 ns;
       if(Cout /= '1' or V /= '0') then
      errCnt:= errCnt + 1;
   end if;

   --TEST 2
   A3 <= '0';
   A2 <= '1';
   A1 <= '1';
```

```
    A0 <= '1';
    --
    B3 <= '0';
    B2 <= '1';
    B1 <= '0';
    B0 <= '1';
    --
    wait for 20 ns;

  if(Cout /= '0' or V /= '1') then
      errCnt:= errCnt + 1;
    end if;
  end process;
end testbench;
```

The Clock Pulse Diagram of 2s Complement Adder/Subtractor:



❖ 8x3 EPROM :

Behavioral:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Eight_Three_Rom is
port (a,b,c : in Std_logic;
    f,g,h : out std_logic);

end Eight_Three_Rom;

architecture Behavioral of Eight_Three_Rom is
```

begin

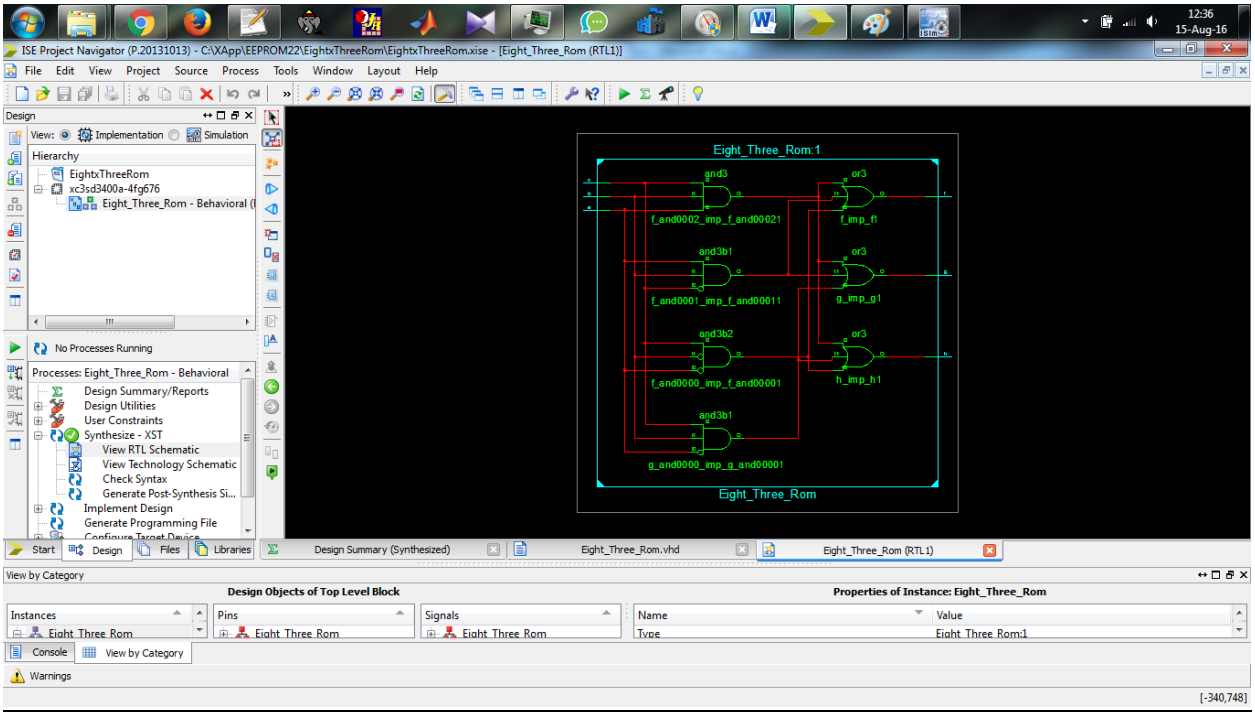
f <= ((not a) and (not b) and c) or (a and b and (not c)) or (a and b and c);

g <= ((not a) and b and c) or (a and b and (not c)) or (a and b and c) ;

h <= ((not a) and (not b) and c) or ((not a) and b and c) or (a and b and c);

end Behavioral;

The RTL Schematic Diagram of 8x3 EPROM:



The Simulation 8x3 EPROM:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY epromtest IS
END epromtest;

ARCHITECTURE behavior OF epromtest IS

   -- Component Declaration for the Unit Under Test (UUT)

   COMPONENT Eight_Three_Rom
   PORT(
        a : IN  std_logic;
        b : IN  std_logic;
        c : IN  std_logic;
        f : OUT  std_logic;
        g : OUT  std_logic;
        h : OUT  std_logic
       );
   END COMPONENT;
```

```
   --Inputs
   signal a : std_logic := '0';
   signal b : std_logic := '0';
   signal c : std_logic := '0';

                                        --Outputs
   signal f : std_logic;
   signal g : std_logic;
   signal h : std_logic;

   constant clock_period : time := 10 ns;
                                        signal clock : std_logic;

BEGIN

                                        -- Instantiate the Unit Under Test
(UUT)
   uut: Eight_Three_Rom PORT MAP (
         a => a,
         b => b,
         c => c,
         f => f,
         g => g,
```

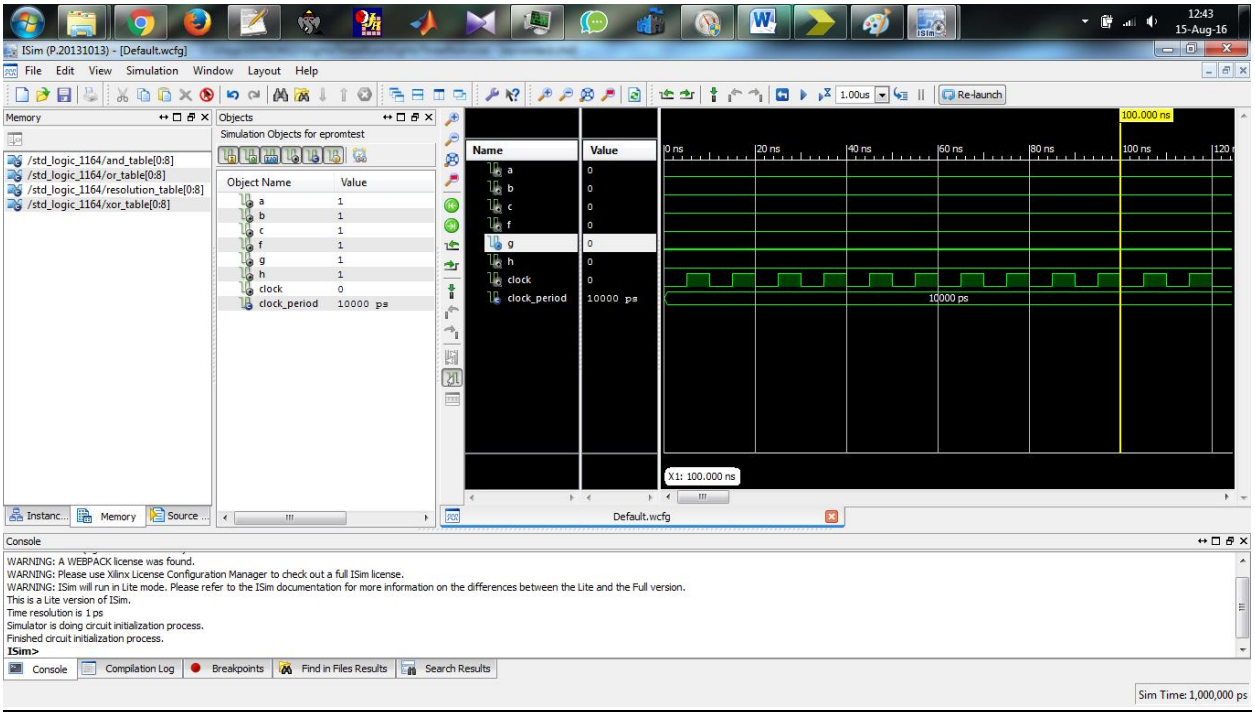
```
        h => h
      );

   -- Clock process definitions
   clock_process :process
   begin
                                        clock <= '0';
                                        wait for clock_period/2;
                                        clock <= '1';
                                        wait for clock_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;

      wait for clock_period*10;

      -- insert stimulus here
a <= '0';
```

raktimchakraborty27@gmail.com

```
b <= '0';
c <= '1';
  wait for clock_period*10;

 a <= '0';
b <= '1';
c <= '1';
  wait for clock_period*10;

a <= '1';
b <= '0';
c <= '1';
  wait for clock_period*10;

a <= '1';
b <= '1';
c <= '0';
  wait for clock_period*10;

a <= '1';
b <= '1';
c <= '1';
  wait for clock_period*10;
```

```
    wait;
  end process;

END;
```

The Clock Pulse Diagram of 8x3 EPROM:



raktimchakraborty27@gmail.com

References:

•[1] Engineering Digital Design by Richard F. Tinder.

•[2] VHDL Programing by Example by Douglas L. Perry

•[3] FPGA Architecture for the Challenge from toronto.edu.

•[4]http://www.altera.com/literature/hb/cyc2/cyc2_cii51002.pdf

•[5] Documentation: Stratix IV Devices Altera.com. 2008-06-11.Retrieved s2013-05-01.

•[6]http://www.xilinx.com/support/documentation/user_guides/ug070.pdf

•[7] Xilinx aims 65-nm FPGAs at DSP applications. EETimes.

•[8] Digital Circuits and Design: 4th Edition by S Arivazhagan, S Salivahanan

Thank You

Happy Coding!!!!!!!

raktimchakraborty27@gmail.com