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

COMPONENT uadder PORT(

A : IN std\_logic\_vector(3 downto 0); B : IN std\_logic\_vector(3 downto 0); C : OUT std\_logic\_vector(3 downto 0)

);

END COMPONENT;

--Inputs

signal A : std\_logic\_vector(3 downto 0) := (others => '0'); signal B : std\_logic\_vector(3 downto 0) := (others => '0');

--Outputs

signal C : std\_logic\_vector(3 downto 0);

-- No clocks detected in port list. Replace <clock> below with

-- appropriate port name

--constant <clock>\_period : time := 10 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT) uut: uadder PORT MAP (

A => A,

B => B, C => C

);

-- 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. A<="0111";B<="0010"; wait for 100 ns;

--wait for <clock>\_period\*10;

-- insert stimulus here

wait;

end process;

END;

-- Code your design here library IEEE;

use IEEE.std\_logic\_1164.all;

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL; use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.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;

entity uadder is

Port ( A : in STD\_LOGIC\_VECTOR (3 downto 0); B : in STD\_LOGIC\_VECTOR (3 downto 0); C : out STD\_LOGIC\_VECTOR (3 downto 0));

end uadder;

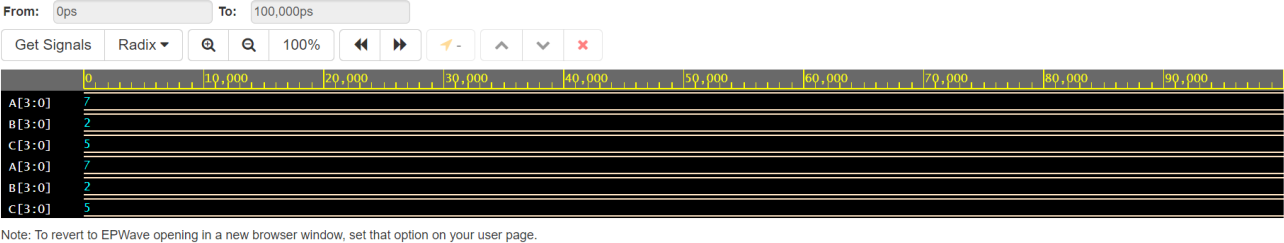
architecture Dataflow of uadder is

begin

C <= A - B;

end Dataflow;

Output :



Discussion :

Assignment 4: Implementation of binary to Gray converter and Gray to Binary converter using XILINX ISE

Aim: Software used:

|  |  |
| --- | --- |
| Property Name | XILINX ISE |
| Device family |  |
| Device |  |
| Package |  |
| Speed |  |
| Top-level source type |  |
| Synthesis Tool |  |
| Simulator |  |
| Preferred Language | VHDL |

Gray to Binary:

Theory :

The reflected binary code or Gray code is an ordering of the binary numeral system such that two successive values differ in only one bit (binary digit). This is very simple method to get Binary number from Gray code. These are following steps for *n*-bit binary numbers −

* The Most Significant Bit (MSB) of the binary code is always equal to the MSB of the given binary number.
* Other bits of the output binary code can be obtained by checking gray code bit at that index. If current gray code bit is 0, then copy previous binary code bit, else copy invert of previous binary code bit.

Truth table: (Gray to Binary)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Gray** | | | | **Binary** | | | |
| **G3** | **G2** | **G1** | **G0** | **B3** | **B2** | **B1** | **B0** |
| 0 | 0 | 0 | 0 |  |  |  |  |
| 0 | 0 | 0 | 1 |  |  |  |  |
| 0 | 0 | 1 | 0 |  |  |  |  |
| 0 | 0 | 1 | 1 |  |  |  |  |
| 0 | 1 | 0 | 0 |  |  |  |  |
| 0 | 1 | 0 | 1 |  |  |  |  |
| 0 | 1 | 1 | 0 |  |  |  |  |
| 0 | 1 | 1 | 1 |  |  |  |  |
| 1 | 0 | 0 | 0 |  |  |  |  |
| 1 | 0 | 0 | 1 |  |  |  |  |
| 1 | 1 | 1 | 0 |  |  |  |  |
| 1 | 0 | 1 | 1 |  |  |  |  |
| 1 | 0 | 1 | 0 |  |  |  |  |
| 1 | 1 | 0 | 0 |  |  |  |  |
| 1 | 1 | 0 | 1 |  |  |  |  |
| 1 | 1 | 1 | 1 |  |  |  |  |

Dataflow Model CODE:

****Design****  
  
library IEEE;  
use IEEE.STD\_LOGIC\_1164.ALL;  
  
  
entity gtb is  
    Port ( g : in  STD\_LOGIC\_VECTOR (3 downto 0);  
           b : inout  STD\_LOGIC\_VECTOR (3 downto 0));  
end gtb;  
  
architecture Dataflow of gtb is  
  
begin  
b(3)<=g(3);  
b(2)<=b(3) xor g(2);  
b(1)<=b(2) xor g(1);  
b(0)<=b(1) xor g(0);  
  
end Dataflow;

**TestBench**  
  
LIBRARY ieee;  
USE ieee.std\_logic\_1164.ALL;  
  
   
ENTITY tb\_gtb IS  
END tb\_gtb;  
   
ARCHITECTURE behavior OF tb\_gtb IS  
   
    COMPONENT gtb  
    PORT(  
         g : IN  std\_logic\_vector(3 downto 0);  
         b : INOUT  std\_logic\_vector(3 downto 0)  
        );  
    END COMPONENT;  
  
   signal g : std\_logic\_vector(3 downto 0) := (others => '0');  
  
   signal b : std\_logic\_vector(3 downto 0);  
BEGIN  
   
   uut: gtb PORT MAP (  
          g => g,  
          b => b  
        );  
  
   stim\_proc: process  
   begin         
      wait for 100 ns;     
  
     g<="1100";  
  
  
      wait;  
   end process;  
  
END;

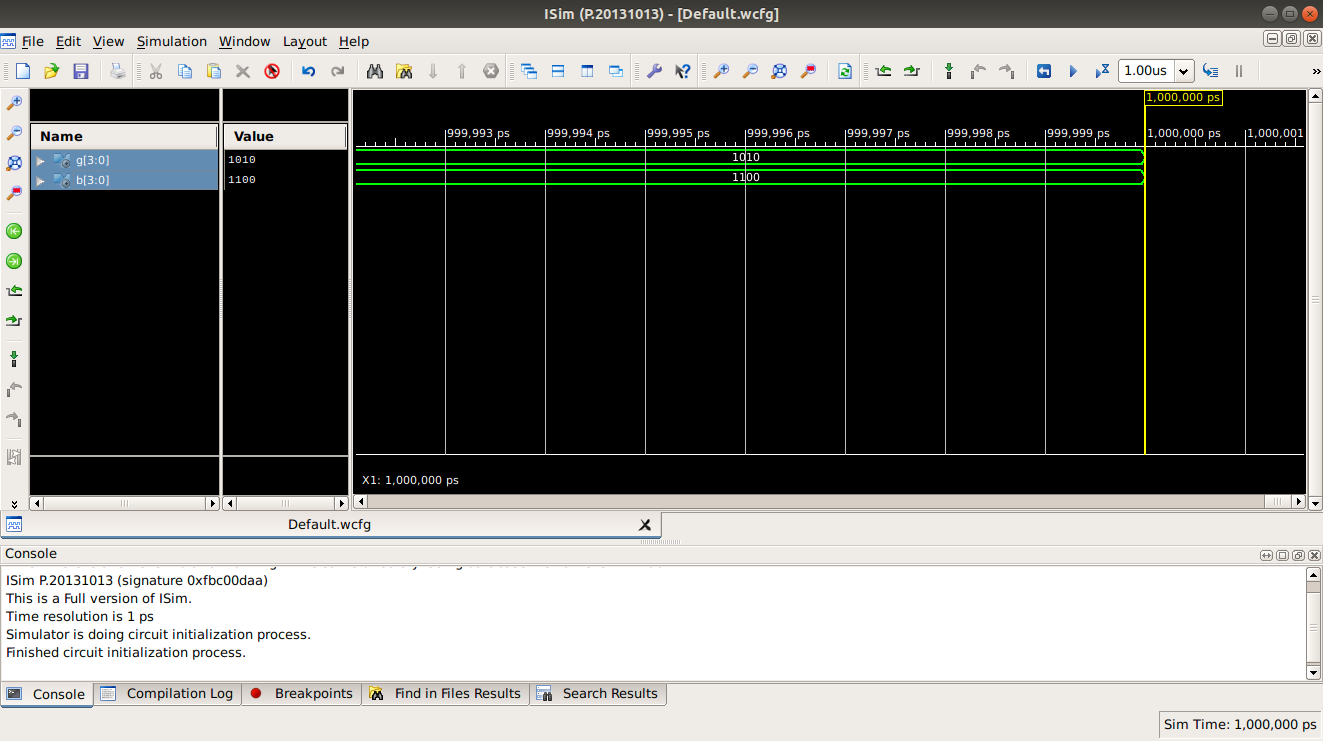
Code:

Behavioral Model code:

****Design****  
entity gtb\_bev is  
    Port ( g : in  STD\_LOGIC\_VECTOR (3 downto 0);  
           b : inout  STD\_LOGIC\_VECTOR (3 downto 0));  
end gtb\_bev;  
  
architecture Behavioral of gtb\_bev is  
  
begin  
process(g,b)  
begin  
b(3)<=g(3);  
for i in 2 downto 0 loop  
if(b(i+1)=g(i)) then  
b(i)<='0';  
else  
b(i)<='1';  
end if;  
end loop;  
 end process;  
  
end Behavioral;

**Testbench**  
  
LIBRARY ieee;  
USE ieee.std\_logic\_1164.ALL;  
  
ENTITY tb\_gtb\_bev IS  
END tb\_gtb\_bev;  
   
ARCHITECTURE behavior OF tb\_gtb\_bev IS  
   
    COMPONENT gtb\_bev  
    PORT(  
         g : IN  std\_logic\_vector(3 downto 0);  
         b : INOUT  std\_logic\_vector(3 downto 0)  
        );  
    END COMPONENT;  
      
  
   signal g : std\_logic\_vector(3 downto 0) := (others => '0');  
  
   signal b : std\_logic\_vector(3 downto 0);  
  
BEGIN  
  
   uut: gtb\_bev PORT MAP (  
          g => g,  
          b => b  
        );  
          -- Stimulus process  
   stim\_proc: process  
   begin         
      wait for 100 ns;     
    g<="1010";  
  
      wait;  
   end process;  
  
END;

Output :



Binary to Gray:

Theory :

This is very simple method to get Gray code from Binary number. These are following steps for n-bit binary numbers −

* The most significant bit (MSB) of the Gray code is always equal to the MSB of the given Binary code.
* Other bits of the output Gray code can be obtained by XORing binary code bit at the index and previous index.

Truth table: (Binary to Gray)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Binary** | | | | **Gray** | | | |  |
| **B3** | **B2** | **B1** | **B0** | **G3** | **G2** | **G1** | **G0** |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

Behavioral Model:

Here, the gate is designed, using the truth table. It simulates the behavior of the output of the gate, upon taking different Boolean variables as inputs. The Behavioral Architecture is used in this case.

Data flow Model:

Here, the gate is constructed, using the Dataflow Architecture. The gate is designed, using it’s Boolean function

Code:

Behavioral Model code:

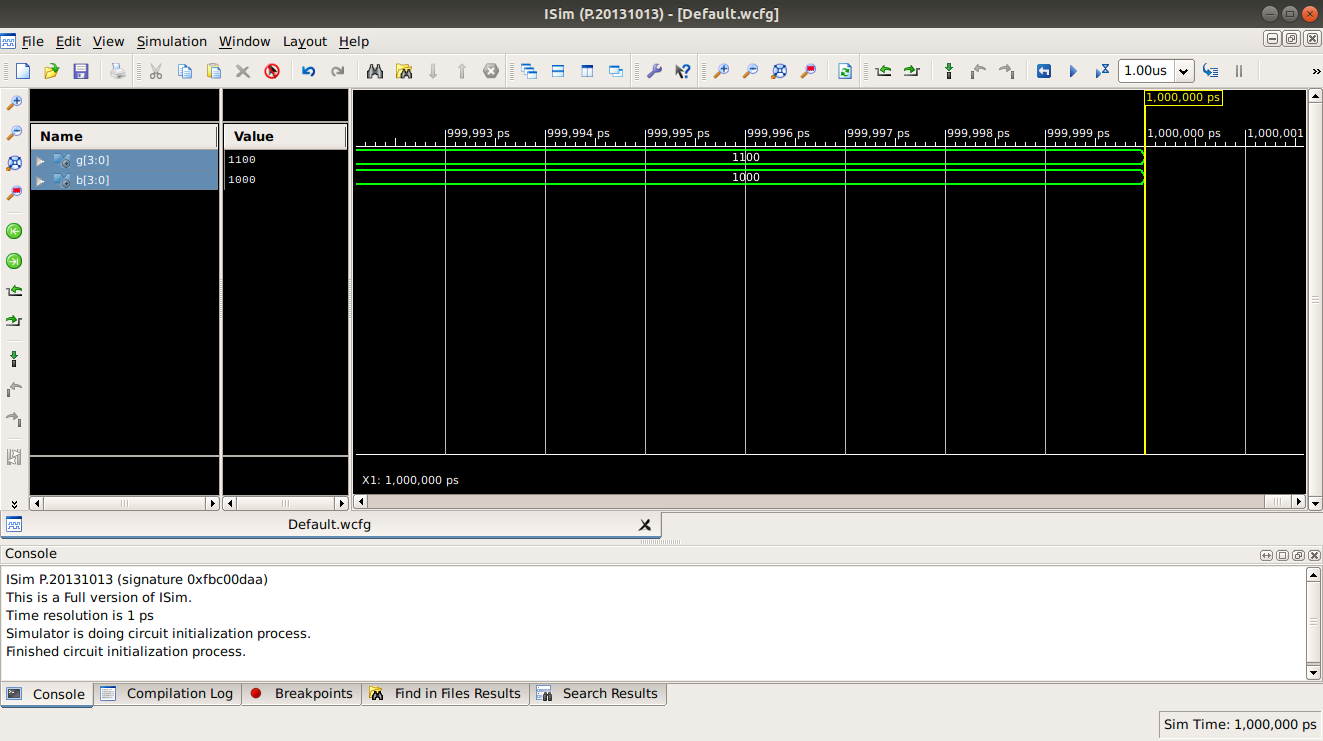
**Design**  
library IEEE;  
use IEEE.STD\_LOGIC\_1164.ALL;  
  
  
entity btg is  
    Port ( b : in  STD\_LOGIC\_VECTOR (3 downto 0);  
           g : out  STD\_LOGIC\_VECTOR (3 downto 0));  
end btg;  
  
architecture Behavioural of btg is  
  
begin  
process(b)  
begin  
g(3)<=b(3);  
for i in 0  to 2 loop  
if(b(i+1)=b(i)) then  
g(i)<='0';  
else  
g(i)<='1';  
end if;  
end loop;  
end process;  
  
end Behavioural;

**Testbench:**  
LIBRARY ieee;  
USE ieee.std\_logic\_1164.ALL;  
  
ENTITY tb\_btg IS  
END tb\_btg;  
   
ARCHITECTURE behavior OF tb\_btg IS  
  
    COMPONENT btg  
    PORT(  
         b : IN  std\_logic\_vector(3 downto 0);  
         g : OUT  std\_logic\_vector(3 downto 0)  
        );  
    END COMPONENT;  
  
   signal b : std\_logic\_vector(3 downto 0) := (others => '0');  
  
   signal g : std\_logic\_vector(3 downto 0);  
  
BEGIN  
  
   uut: btg PORT MAP (  
          b => b,  
          g => g  
        );  
  
   stim\_proc: process  
   begin         
  
      wait for 100 ns;     
  
      b<="1100";  
  
      wait;  
   end process;

END;

Dataflow Model Code:

****Design****  
  
library IEEE;  
use IEEE.STD\_LOGIC\_1164.ALL;  
  
  
entity btg is  
    Port ( b : in  STD\_LOGIC\_VECTOR (3 downto 0);  
           g : out  STD\_LOGIC\_VECTOR (3 downto 0));  
end btg;  
  
architecture Dataflow of btg is  
  
begin  
  
g(3)<=b(3);  
g(2)<=b(3) xor b(2);  
g(1)<=b(2) xor b(1);  
g(0)<=b(1) xor b(0);  
  
  
end Dataflow;  
  
  
  
****Testbench****  
LIBRARY ieee;  
USE ieee.std\_logic\_1164.ALL;  
  
ENTITY tb\_btg IS  
END tb\_btg;  
   
ARCHITECTURE behavior OF tb\_btg IS  
  
    COMPONENT btg  
    PORT(  
         b : IN  std\_logic\_vector(3 downto 0);  
         g : OUT  std\_logic\_vector(3 downto 0)  
        );  
    END COMPONENT;  
  
   signal b : std\_logic\_vector(3 downto 0) := (others => '0');  
  
   signal g : std\_logic\_vector(3 downto 0);  
  
BEGIN  
  
   uut: btg PORT MAP (  
          b => b,  
          g => g  
        );  
  
   stim\_proc: process  
   begin         
  
      wait for 100 ns;     
  
      b<="1000";  
  
      wait;  
   end process;  
  
END;

Output :

Discussion:

Gray codes are very useful in the normal sequence of binary numbers generated by the hardware that may cause an error or ambiguity during the transition from one number to the next. So, the Gray code can eliminate this problem easily since only one bit changes its value during any transition between two numbers.

Assignment 5: Implementation of 4bit comparator, 2:4 Decoder,3:8 Decoder, 8bit Multiplier using XILINX ISE

Aim: Software used:

|  |  |
| --- | --- |
| Property Name | XILINX ISE |
| Device family |  |
| Device |  |
| Package |  |
| Speed |  |
| Top-level source type |  |
| Synthesis Tool |  |
| Simulator |  |
| Preferred Language | VHDL |

4 bit Comparator: Theory:

Code:

Behavioral Model Code:

Output:

2:4 Decoder: Truth table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I1** | **I0** | **D3** | **D2** | **D1** | **D0** |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |

Dataflow Model:

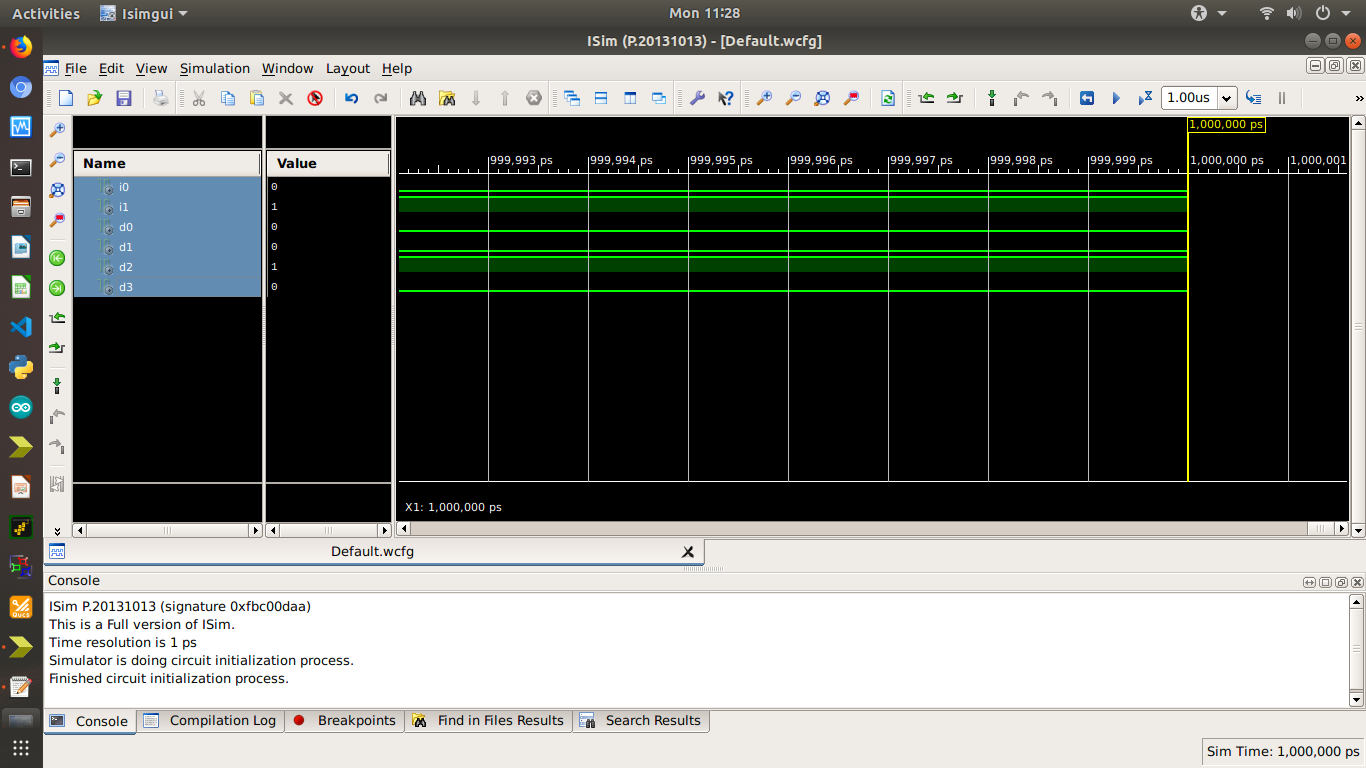
Here, the gate is constructed, using the Dataflow Architecture. The gate is designed, using it’s Boolean function

Code:

DATAFLOW:

****Design****  
library IEEE;  
use IEEE.STD\_LOGIC\_1164.ALL;  
  
entity decoder is  
    Port ( i0 : in  STD\_LOGIC;  
           i1 : in  STD\_LOGIC;  
           d0 : out  STD\_LOGIC;  
           d1 : out  STD\_LOGIC;  
           d2 : out  STD\_LOGIC;  
           d3 : out  STD\_LOGIC);  
end decoder;  
  
architecture Dataflow of decoder is  
  
begin  
d0<=(not i0)and(not i1);  
d1<=( i0)and(not i1);  
d2<=(not i0)and(i1);  
d3<=(i0)and( i1);  
  
end Dataflow;  
  
****Testbench****  
  
LIBRARY ieee;  
USE ieee.std\_logic\_1164.ALL;  
  
   
ENTITY tb\_decoder IS  
END tb\_decoder;  
   
ARCHITECTURE behavior OF tb\_decoder IS  
  
    COMPONENT decoder  
    PORT(  
         i0 : IN  std\_logic;  
         i1 : IN  std\_logic;  
         d0 : OUT  std\_logic;  
         d1 : OUT  std\_logic;  
         d2 : OUT  std\_logic;  
         d3 : OUT  std\_logic  
        );  
    END COMPONENT;  
  
   signal i0 : std\_logic := '0';  
   signal i1 : std\_logic := '0';  
  
   signal d0 : std\_logic;  
   signal d1 : std\_logic;  
   signal d2 : std\_logic;  
   signal d3 : std\_logic;  
  
   
BEGIN  
   
   uut: decoder PORT MAP (  
          i0 => i0,  
          i1 => i1,  
          d0 => d0,  
          d1 => d1,  
          d2 => d2,  
          d3 => d3  
        );  
  
   stim\_proc: process  
   begin         
  
      wait for 100 ns;     
        i0<='0';wait for 100ns;  
        i1<='1';  
    
      wait;  
   end process;  
  
END;

Output :

****

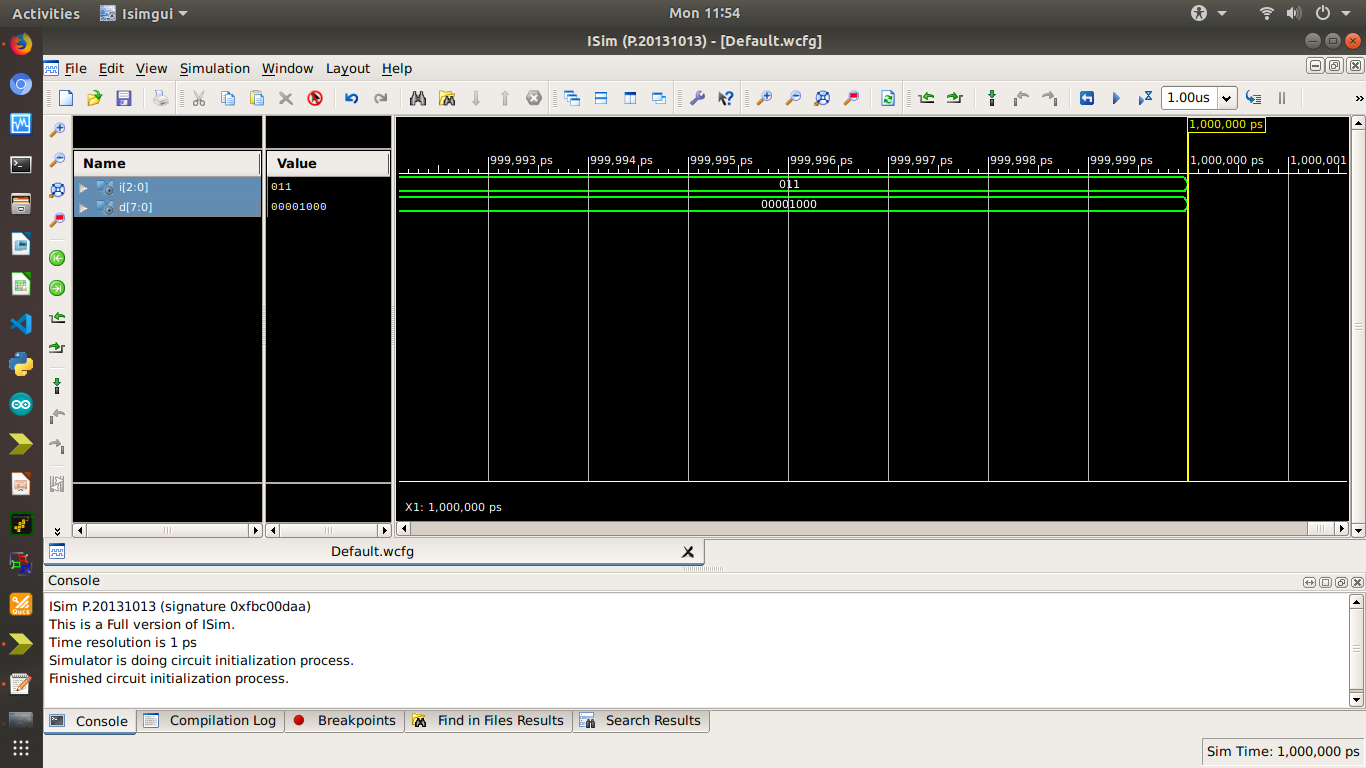
3:8 Decoder: Truth table:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **I2** | **I1** | **I0** | **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** |
|  |  |  |  |  |  |  |  |  |  |  |

Code:

Design  
  
library IEEE;  
use IEEE.STD\_LOGIC\_1164.ALL;  
USE ieee.std\_logic\_arith.all;  
USE ieee.std\_logic\_unsigned.all;  
  
  
entity decoder\_behav is  
    Port ( i : in  STD\_LOGIC\_VECTOR(2 downto 0);  
           d : out  STD\_LOGIC\_VECTOR (7 downto 0));  
end decoder\_behav;  
  
architecture Behavioral of decoder\_behav is  
begin  
process(i)  
variable s:integer;  
begin  
  
s:=CONV\_INTEGER(i);  
for i in 7 downto 0 loop  
if(i=s) then  
d(i)<='1';  
else  
d(i)<='0';  
end if;  
end loop;  
end process;  
  
end Behavioral;  
  
  
  
Testbench

LIBRARY ieee;  
USE ieee.std\_logic\_1164.ALL;  
  
   
ENTITY tb\_decoder IS  
END tb\_decoder;  
   
ARCHITECTURE behavior OF tb\_decoder IS  
  
   
    COMPONENT decoder\_behav  
    PORT(  
         i : IN  std\_logic\_vector(2 downto 0);  
         d : OUT  std\_logic\_vector(7 downto 0)  
        );  
    END COMPONENT;  
  
   signal i : std\_logic\_vector(2 downto 0) := (others => '0');  
  
   signal d : std\_logic\_vector(7 downto 0);  
  
BEGIN  
  
   uut: decoder\_behav PORT MAP (  
          i => i,  
          d => d  
        );  
   stim\_proc: process  
   begin         
  
      wait for 100 ns;     
  
  i<="011";wait for 100 ns;  
  
      wait;  
   end process;  
  
END;

Output :

8 Bit Multiplier: Code:

Output:

Discussion:

Assignment 6: Implementation of 2:4 Decoder (using case), 4:2 Encoder (Dataflow & using case), 8:3 Encoder (using loop)

Aim: Software used:

|  |  |
| --- | --- |
| Property Name |  |
| Device family |  |
| Device |  |
| Package |  |
| Speed |  |
| Top-level source type |  |
| Synthesis Tool |  |
| Simulator |  |
| Preferred Language |  |

2:4 Decoder: Theory:

Truth table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I1** | **I0** | **D3** | **D2** | **D1** | **D0** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Behavioral Model: (using case)

Output :

4:2 Encoder: Theory :

Truth Table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I3** | **I2** | **I1** | **I0** | **E1** | **E0** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Dataflow Model:

Dataflow Code:

Output :

Behavioral Code:

Output :

8:3 Encoder: Truth Table:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **I7** | **I6** | **I5** | **I4** | **I3** | **I2** | **I1** | **I0** | **E2** | **E1** | **E0** |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

Behavioral Code:

Output :

Discussion :

Assignment 7: Implementation of 2:1 MUX(Dataflow), 4:1 MUX (Dataflow, if-else and using Case) and 1:4 Demux (Using Dataflow and case) using XILINX ISE.

Software used:

|  |  |
| --- | --- |
| Property Name |  |
| Device family |  |
| Device |  |
| Package |  |
| Speed |  |
| Top-level source type |  |
| Synthesis Tool |  |
| Simulator |  |
| Preferred Language |  |

Theory :

2:1 MUX:

Truth table:

|  |  |  |  |
| --- | --- | --- | --- |
| **INPUT** | | | **OUTPUT** |
| **S** | **I1** | **I0** | **M** |
|  |  |  |  |
|  |  |  |  |

Dataflow Model:

Code:

Dataflow Model:

4:1 MUX:

Truth Table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **INPUT** | | | | | | **OUTPUT** |
| **S1** | **S0** | **I3** | **I2** | **I1** | **I0** | **M** |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Dataflow Mode

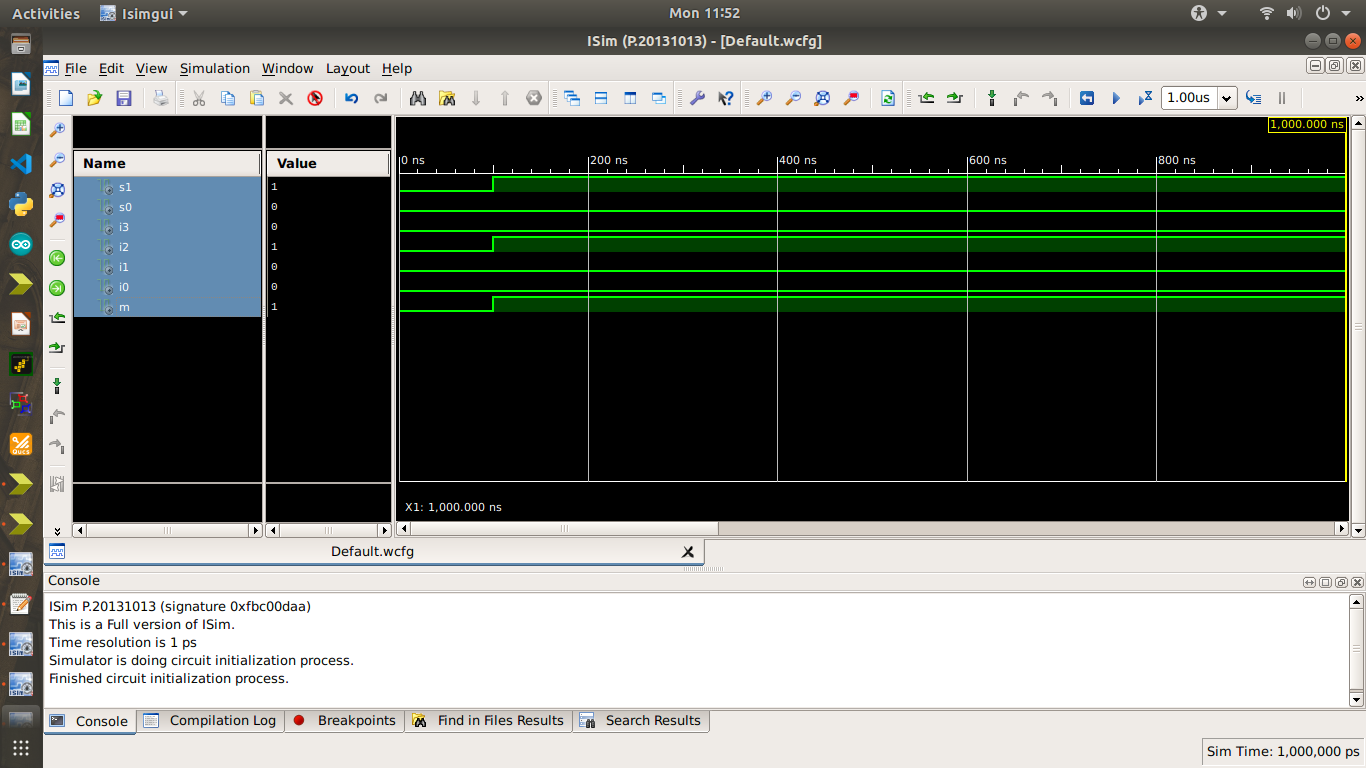
Dataflow Model Code:

library IEEE;  
use IEEE.STD\_LOGIC\_1164.ALL;  
  
  
entity mux4x1 is  
    Port ( s1 : in  STD\_LOGIC;  
           s0 : in  STD\_LOGIC;  
           i3 : in  STD\_LOGIC;  
           i2 : in  STD\_LOGIC;  
           i1 : in  STD\_LOGIC;  
           i0 : in  STD\_LOGIC;  
           m : out  STD\_LOGIC);  
end mux4x1;  
  
architecture Dataflow of mux4x1 is  
  
begin  
m<=((not s0) and (not s1) and i0) or ((not s1) and s0 and i1)  
 or ( s1 and (not s0) and i2) or ( s0 and s1 and i3);  
end Dataflow;

TESTBENCH

LIBRARY ieee;  
USE ieee.std\_logic\_1164.ALL;  
  
ENTITY tb\_mux4x1 IS  
END tb\_mux4x1;  
   
ARCHITECTURE behavior OF tb\_mux4x1 IS  
   
    -- Component Declaration for the Unit Under Test (UUT)  
   
    COMPONENT mux4x1  
    PORT(  
         s1 : IN  std\_logic;  
         s0 : IN  std\_logic;  
         i3 : IN  std\_logic;  
         i2 : IN  std\_logic;  
         i1 : IN  std\_logic;  
         i0 : IN  std\_logic;  
         m : OUT  std\_logic  
        );  
    END COMPONENT;  
  
   signal s1 : std\_logic := '0';  
   signal s0 : std\_logic := '0';  
   signal i3 : std\_logic := '0';  
   signal i2 : std\_logic := '0';  
   signal i1 : std\_logic := '0';  
   signal i0 : std\_logic := '0';  
  
   signal m : std\_logic;  
  
BEGIN  
  
   uut: mux4x1 PORT MAP (  
          s1 => s1,  
          s0 => s0,  
          i3 => i3,  
          i2 => i2,  
          i1 => i1,  
          i0 => i0,  
          m => m  
        );  
  
   stim\_proc: process  
   begin         
  
      wait for 100 ns;     
s1<='1';s0<='0';i3<='0';i2<='1';i1<='0';i0<='0';  
  
      wait;  
   end process;  
  
END;

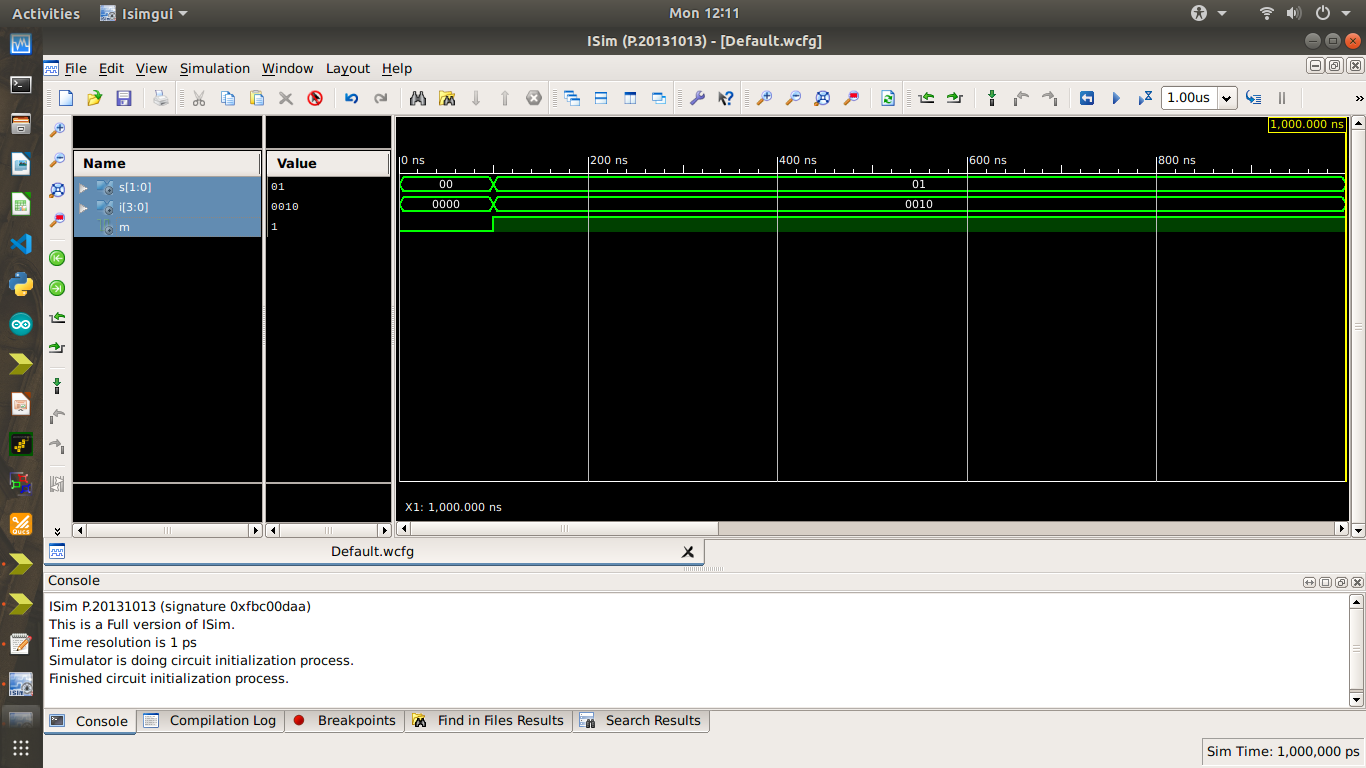
Output :



Behavioral Model code: (Using if-else)

entity mux4x1\_be is  
    Port ( s : in  STD\_LOGIC\_VECTOR (1 downto 0);  
           i : in  STD\_LOGIC\_VECTOR (3 downto 0);  
           m : out  STD\_LOGIC);  
end mux4x1\_be;  
  
architecture Behavioral of mux4x1\_be is  
  
begin  
process(i,s)  
begin  
if(s(0)=s(1)) then  
if(s(0)='0') then  
m<=i(0);  
else  
m<=i(3);  
end if;  
else  
if(s(0)='1') then  
m<=i(1);  
else  
m<=i(2);  
end if;  
end if;  
end process;  
  
end Behavioral;  
  
testbench  
  
LIBRARY ieee;  
USE ieee.std\_logic\_1164.ALL;  
  
ENTITY tb\_mux4x1 IS  
END tb\_mux4x1;  
   
ARCHITECTURE behavior OF tb\_mux4x1 IS  
  
    COMPONENT mux4x1\_be  
    PORT(  
         s : IN  std\_logic\_vector(1 downto 0);  
         i : IN  std\_logic\_vector(3 downto 0);  
         m : OUT  std\_logic  
        );  
    END COMPONENT;  
  
   signal s : std\_logic\_vector(1 downto 0) := (others => '0');  
   signal i : std\_logic\_vector(3 downto 0) := (others => '0');  
  
   signal m : std\_logic;  
  
BEGIN  
  
   uut: mux4x1\_be PORT MAP (  
          s => s,  
          i => i,  
          m => m  
        );  
  
   stim\_proc: process  
   begin         
  
      wait for 100 ns;     
  
      s<="01";i<="0010";  
  
      wait;  
   end process;  
  
END;

Output:



1:4 De mux: Theory :

Truth Table:

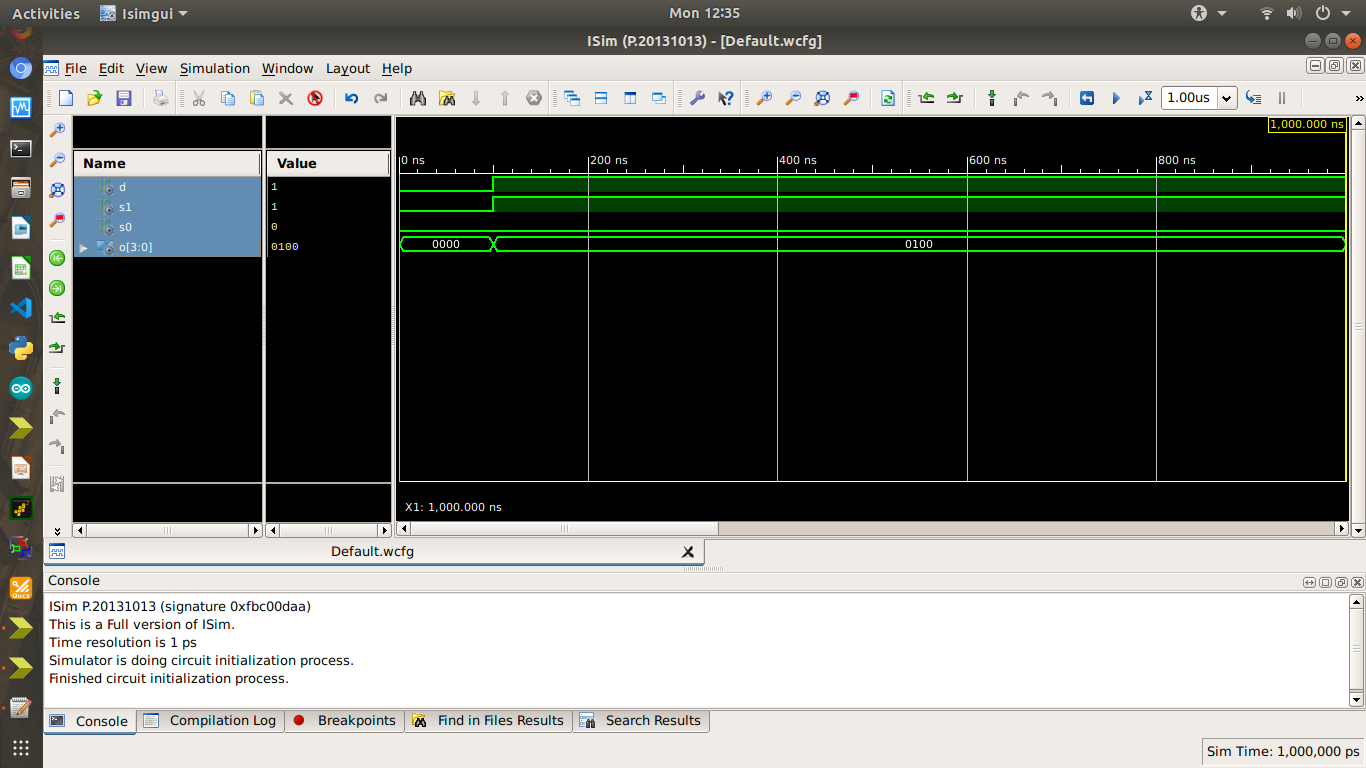
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Data Input** | **Select Input** | | **Output** | | | |
| **D** | **S1** | **S0** | **Y3** | **Y2** | **Y1** | **Y0** |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Dataflow Model:

Dataflow Code:

library IEEE;  
use IEEE.STD\_LOGIC\_1164.ALL;  
  
  
  
entity dmux1x4 is  
    Port ( d : in  STD\_LOGIC;  
           s1 : in  STD\_LOGIC;  
           s0 : in  STD\_LOGIC;  
           o : out  STD\_LOGIC\_VECTOR (3 downto 0));  
end dmux1x4;  
  
architecture Dataflow of dmux1x4 is  
  
begin  
o(0)<=d and (not s0) and (not s1);  
o(1)<=d and (not s1) and  s0;  
o(2)<=d and  s1 and (not s0);  
o(3)<=d and  s0 and  s1;  
  
  
end Dataflow;  
  
  
LIBRARY ieee;  
USE ieee.std\_logic\_1164.ALL;  
   
  
   
ENTITY tb\_demux IS  
END tb\_demux;  
   
ARCHITECTURE behavior OF tb\_demux IS  
   
    -- Component Declaration for the Unit Under Test (UUT)  
   
    COMPONENT dmux1x4  
    PORT(  
         d : IN  std\_logic;  
         s1 : IN  std\_logic;  
         s0 : IN  std\_logic;  
         o : OUT  std\_logic\_vector(3 downto 0)  
        );  
    END COMPONENT;  
  
   signal d : std\_logic := '0';  
   signal s1 : std\_logic := '0';  
   signal s0 : std\_logic := '0';  
  
   signal o : std\_logic\_vector(3 downto 0);  
  
BEGIN  
  
   uut: dmux1x4 PORT MAP (  
          d => d,  
          s1 => s1,  
          s0 => s0,  
          o => o  
        );  
  
   stim\_proc: process  
   begin         
  
      wait for 100 ns;     
  
     d<='1'; s1<='1';s0<='0';  
  
  
  
      wait;  
   end process;  
  
END;

Output:



Discussion:

In this assignment we learned about multiplexer and de multiplexer and learned about the implementation in both dataflow and behavioral.

Assignment 8: Implementation of Full Adder using Half Adder (using Structural Method), 4 bit Parallel Adder (using Structural Method).

Aim : Software used:

|  |  |
| --- | --- |
| Property Name |  |
| Device family |  |
| Device |  |
| Package |  |
| Speed |  |

|  |  |
| --- | --- |
| Top-level source type |  |
| Synthesis Tool |  |
| Simulator |  |
| Preferred Language |  |

Full Adder: Theory:

Truth Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INPUT** | | | **OUTPUT** | |
| **A** | **B** | **Cin** | **S** | **Cout** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Structural Model

Code:

Output:

4 Bit Parallel Adder: Theory:

Structural Model:

Code:

Output:

Discussion:

Assignment 9: Implementation of 2:1 MUX (using Structural Method) & 4:1 MUX using 2:1 MUX (using Structural Method) in XILINX ISE

Aim :

Software used:

|  |  |
| --- | --- |
| Property Name |  |
| Device family |  |
| Device |  |
| Package |  |
| Speed |  |
| Top-level source type |  |
| Synthesis Tool |  |
| Simulator |  |
| Preferred Language |  |

Theory :

2:1 MUX

Truth table:

|  |  |  |  |
| --- | --- | --- | --- |
| **INPUT** | | | **OUTPUT** |
| **S** | **I1** | **I0** | **Y** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Structural Model:

Code : (Structural method)

Output :

4:1 MUX

Truth table:

|  |  |  |
| --- | --- | --- |
| **S1** | **S0** | **Y** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Structural Model:

Code : (Structural)

Output :

Discussion :

Assignment 10: Implementation of SR flip flop, JK flip flop, D flip flop, T flip flop (using if

–else) in XILINX ISE Aim :

Software used:

|  |  |
| --- | --- |
| Property Name |  |
| Device family |  |
| Device |  |
| Package |  |
| Speed |  |
| Top-level source type |  |
| Synthesis Tool |  |
| Simulator |  |
| Preferred Language |  |

Theory :

SR flip flop: Truth table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CLK** | **S** | **R** | **Q** | **Q’** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Dataflow model:

Code :

Output :

JK flip flop:

Truth table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CLK** | **J** | **K** | **Q** | **Q’** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Dataflow model:

Code :

Output :

D flip flop: Truth table:

|  |  |  |  |
| --- | --- | --- | --- |
| **CLK** | **D** | **Q** | **Q’** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Dataflow model:

Code :

Output :

T flip flop: Truth table:

|  |  |  |  |
| --- | --- | --- | --- |
| **CLK** | **T** | **Q** | **Q’** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Dataflow model:

Code :

Output :

Discussion: