DEPARTMENT OF INFORMATION TECHNOLOGY

**GOVERNMENT COLLEGE OF ENGINEERING,**

**AMRAVATI**

**B.TECH-THIRD SEMESTER**

**LAB RECORD FOR**

**ITU325**

**DIGITAL LOGIC DESIGN LABORATORY**

Submitted by

**Name of Student : Anushka Ganesh Narkhede**

**Registration ID : 20007020**

** Submitted By : Dilip R. Uike**

**Submitted to:**

**GOVERNMENT COLLEGE OF ENGINEERING,**

**AMRAVATI**

**GOVERNMENT COLLEGE OF ENGINEERING, AMRAVATI**

(An Autonomous Institute of Govt. of Maharashtra)

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**WINTER-2021**

**CERTIFICATE**

****

This is to certify that this lab record contains the bonafide lab work of ANUSHKA GANESH NARKHEDE having ID 20007020 of semester III of B.Tech in ***Information Technology*** during academic year 2021-22 for the Course Title:- **Digital Logic Design Lab** Course Code:-**ITU325**

Date:-

Head of Dept. Faculty

(Prof.A.W.Bhade) (Prof. Dilip Uike)

**Course Objectives :**

At the end students should

* To introduce the concepts and techniques associated with the number systems and codes.
* To minimize the logical expressions using Boolean postulates.
* To design various combinational and sequential circuits.
* To provide with an appreciation of applications for the techniques and mathematics used in this course.

**Table Of Contents**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Experiment No.** | **Date** | **Title Of Experiment** | **Page No.** | **Remark** |
| 1 |  | Introduction to VHDL. |  |  |
| 2 |  | Simulation of basic gates AND,OR,NOT in vhdl using dataflow modeling. |  |  |
| 3 |  | Simulation and implementation of gate (NAND, NOR, EXOR) in VHDL. |  |  |
| 4 |  | Simulation & Implementation of 8:1 Multiplexer |  |  |
| 5 |  | Design Half Adder and full adder circuit using VHDL. |  |  |
| 6 |  | Design & implement binary to gray code conversion in vhdl. |  |  |
| 7 |  | To simulate & implement 3:8 decoder using VHDL. |  |  |
| 8 |  | To design & implement the D- Flip Flop using VHDL. |  |  |
| 9 |  | To design and implement 3 bit synchronous up/down counter using VHDL. |  |  |

**PRACTICAL NO – 1**

Here we should specify the entity name for which we are writing the architecture body. The architecture statements should be inside the begin and end keyword. Architecture declarative part may contain variables, constants, or component declaration.

**Configuration:**

     If an entity contains many architectures and  any one of the possible architecture binding with its entity is done using configuration. It is used to bind the architecture body to its entity and a component with an entity.

Syntax:

**configuration** configuration\_name **of** entity\_name **is**

block\_configuration;

**end**  configuration\_name.

Block\_configuration defines the binding of components in a block. This can be written as

**for** block\_name

component\_binding;

**endfor**;

block\_name is the name of the architecture body. Component binding binds the components of the block to entities. This can be written as,

**for** component\_labels**:**component\_name

block\_configuration;

**endfor**;

**Package declaration:**

  Package declaration is used to declare components, types, constants, functions and so on.

Syntax:

**package** package\_name **is**  
       Declarations;  
**end** package\_name;

**Package body:**

   A package body is used to declare the definitions and procedures that are declared in corresponding package. Values can be assigned to constants declared in package in package body.

Syntax:

**packagebody** package\_name **is**        Function\_procedure definitions;  
**end** package\_name;

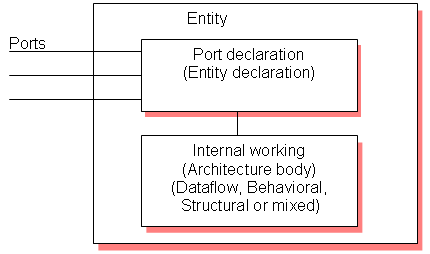
The internal working of an entity can be defined using different modeling styles inside architcture body. They are

1.      Dataflow modeling.

2.      Behavioral modeling.

3.      Structural modeling.

**Structure of an entity:**

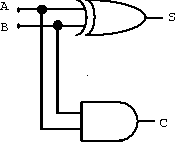


Let’s try to understand with the help of one example.

**Dataflow modeling:**

    In this style of modeling, the internal working of an entity can be implemented using concurrent signal assignment.

Let’s take half adder example which is having one XOR gate and a AND gate.



|  |
| --- |
| **Library** IEEE;  **use** IEEE.STD\_LOGIC\_1164.**all**;    **entity** ha\_en is  **port**(A,B**:inbit**;S,C**:outbit**);  **end** ha\_en;    **architecture** ha\_ar **of** ha\_en **is**  begin  S**<=**A **xor** B;  C**<=**A **and** B;    **end** ha\_ar; |

Here STD\_LOGIC\_1164 is an IEEE standard which defines a nine-value logic type, called STD\_ULOGIC. use is a keyword, which imports all the declarations from this package. The architecture body consists of concurrent signal assignments, which describes the functionality of the design. Whenever there is a change in RHS, the expression is evaluated and the value is assigned to LHS.

**Behavioral modeling:**

    In this style of modeling, the internal working of an entity can be implemented using set of statements.

      It contains:

* Process statements
* Sequential statements
* Signal assignment statements
* Wait statements

Process statement is the primary mechanism used to model the behavior of an entity. It contains sequential statements, variable assignment (:=) statements or signal assignment (<=) statements etc. It may or may not contain sensitivity list. If there is an event occurs on any of the signals in the sensitivity list, the statements within the process is executed.

Inside the process the execution of statements will be sequential and if one entity is having two processes the execution of these processes will be concurrent. At the end it waits for another event to occur.

|  |
| --- |
| **library** IEEE;  **use** IEEE.STD\_LOGIC\_1164.**all**;    **entity** ha\_beha\_en **is**  **port**(  A **:inBIT**;  B **:inBIT**;  S **:outBIT**;   C **:outBIT**  );  **end** ha\_beha\_en;      **architecture** ha\_beha\_ar **of** ha\_beha\_en **is**  **begin**  process\_beh**:process**(A,B)  **begin**  S**<=** A **xor** B;  C**<=**A **and** B;  **endprocess** process\_beh;    **end** ha\_beha\_ar; |

 Here whenever there is a change in the value of a or b the process statements are executed.

**Structural modeling:**

   The implementation of an entity is done through set of interconnected components.

It contains:

* Signal declaration.
* Component instances
* Port maps.
* Wait statements.

 Component declaration:

   Syntax:

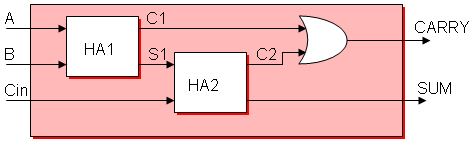
**component** component\_name [**is**]

List\_of\_interface ports;

**endcomponent** component\_name;

Before instantiating the component it should be declared using component declaration as shown above. Component declaration declares the name of the entity and interface of a component.

   Let’s try to understand this by taking the example of full adder using 2 half adder and 1 OR gate.



|  |
| --- |
| **library** IEEE;  **use** IEEE.STD\_LOGIC\_1164.**all**;  **entity** fa\_en **is**  **port**(A,B,Cin**:inbit**; SUM, CARRY**:outbit**);  **end** fa\_en;  **architecture** fa\_ar **of** fa\_en **is**  **component** ha\_en  **port**(A,B**:inbit**;S,C**:outbit**);  **endcomponent**;  **signal** C1,C2,S1**:bit**;  **begin**  HA1**:**ha\_en **portmap**(A,B,S1,C1);  HA2**:**ha\_en **portmap**(S1,Cin,SUM,C2);  CARRY**<=** C1 **or** C2;  **end** fa\_ar; |

The program we have written for half adder in dataflow modeling is instantiated as shown above. ha\_en is the name of the entity in dataflow modeling. C1, C2, S1 are the signals used for internal connections of the component which are declared using the keyword signal. Port map is used to connect different components as well as connect components to ports of the entity.

Component instantiation is done as follows.

    Component\_label: component\_name *port map* (signal\_list);

Signal\_list is the architecture signals which we are connecting to component ports. This can be done in different ways. What we declared above is positional binding. One  more type is the named binding. The above can be written as,

HA1:ha\_en port map(A => A,B => B, S => S1 ,C => C1 );

HA2:ha\_en port map(A => S1,B => Cin, S=> SUM, C => C2);

**Test bench:**

     The correctness of the above program can be checked by writing the test bench.The test bench is used for generating stimulus for the entity under test. Let’s write a simple test bench for full adder.

|  |
| --- |
| **library** IEEE;  **use** IEEE.STD\_LOGIC\_1164.**all**;  **entity** tb\_en **is**  **end** tb\_en;  **architecture** tb\_ar **of** tb\_en **is**  **signal** a\_i,b\_i,c\_i,sum\_i,carry\_i**:bit**;  **begin**  eut**:entity** work.fa\_en(fa\_ar)  **portmap**(A**=>**a\_i,B**=>**b\_i,Cin**=>**c\_i,SUM**=>**sum\_i,CARRY**=>**carry\_i);  stimulus**:process**  **begin**  a\_i**<=**'1';b\_i**<=**'1';c\_i**<=**'1';  **waitfor** 10ns;  a\_i**<=**'0';b\_i**<=**'1';c\_i**<=**'1';  **waitfor** 10ns;  a\_i**<=**'1';b\_i**<=**'0';c\_i**<=**'0';  **waitfor** 10ns;  **if** now**=**30ns **then**  **wait**;  **endif**;  **endprocess** stimulus;  **end** tb\_ar; |

Here now is a predefined function that returns the current simulation time

     What we saw upto this is component instantiation by positional and by name. In this test bench example the entity is directly instantiated. The direct entity instantiation syntax is:

Component\_label: *entity* entity\_name (architecture\_name)    
    *port map*(signal\_list);

**Result:** In this way we studied VHDL language.

**Viva Questions ?**

**1. What is an entity? What are the components involved in it?**

**2. What is architecture? What are the types of it?**

**3. What is structural modeling?**

**PRACTICAL NO – 2**

1. **AND Gate**

**Program:**

-- import std\_logic from the IEEE library

library IEEE;

use IEEE.std\_logic\_1164.all;

-- this is the entity

entity ANDGATE is

port (

IN1 : in std\_logic;

IN2 : in std\_logic;

OUT1: out std\_logic);

end ANDGATE;

architecture RTL of ANDGATE is

begin

OUT1 <= IN1 and IN2;

end RTL;

1. **OR Gate**

**Program:**

-- import std\_logic from the IEEE library

library IEEE;

use IEEE.std\_logic\_1164.all;

-- this is the entity

entity ORGATE is

port (

IN1 : in std\_logic;

IN2 : in std\_logic;

OUT1: out std\_logic);

end ORGATE;

architecture RTL of ORGATE is

begin

OUT1 <= IN1 or IN2;

end RTL;

1. **NOT Gate**

**Program:**

-- import std\_logic from the IEEE library

library IEEE;

use IEEE.std\_logic\_1164.all;

-- this is the entity

entity NOTGATE is

port (

IN1 : in std\_logic;

OUT1: out std\_logic);

end NOTGATE;

architecture RTL of NOTGATE is

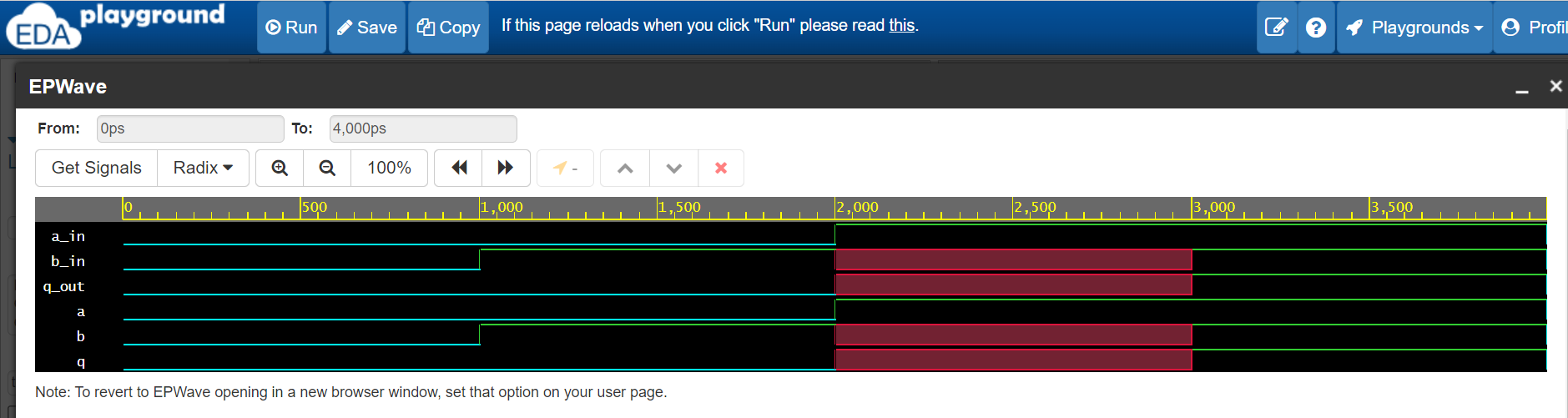
begin

OUT1 <= not IN1;

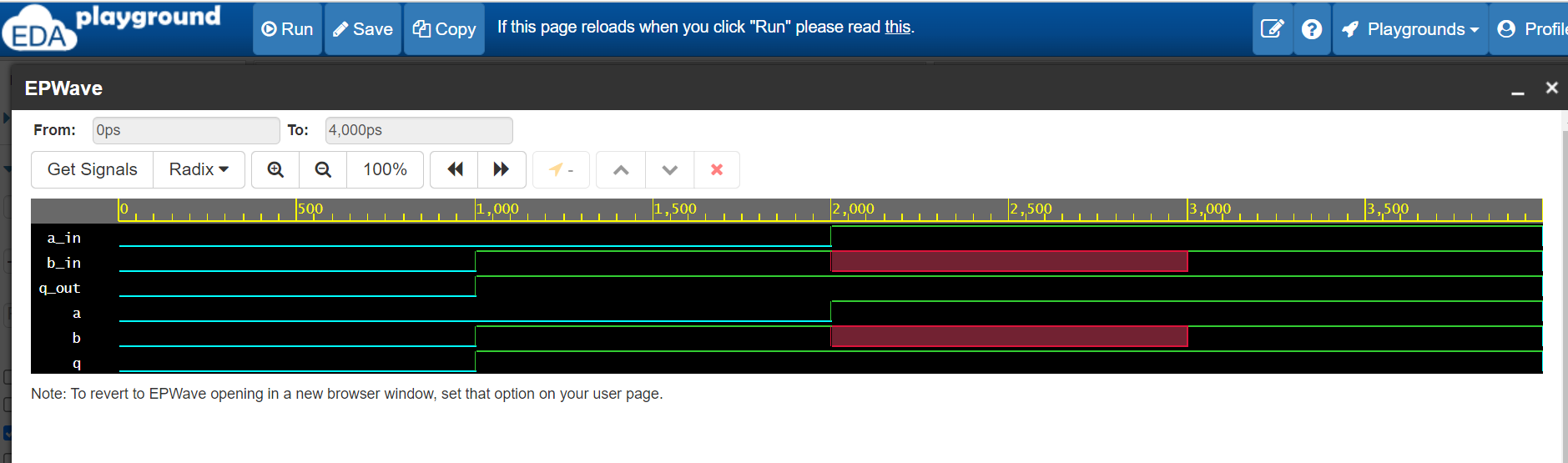
end RTL;

**Output:**

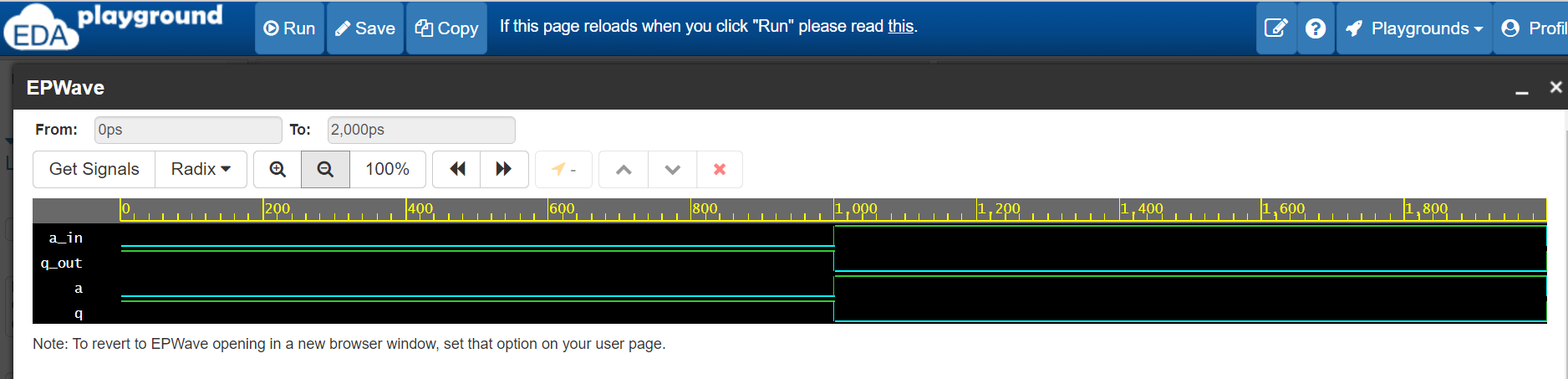
**A) AND Gate**

****

**B) OR Gate**

****

**C) NOT Gate**

****

**Result :**  In this way, we have studied the simulation of basic gates AND,OR,NOT in VHDL using dataflow modeling.

**Viva Questions ?**

1. What is logic gate?
2. What are universal gates?
3. Which of the two input logic gate can be used to implement an inverter circuit?
4. State De-Morgan’s theorem.

**PRACTICAL NO – 3**

**1. NAND Gate**:-

**Program:**

Entity N is,

port(A, B: in Bit; C: out Bit);

end;

Architecture AN of N is

begin

C<= A NAND B;

end;

**2. NOR Gate**:-

**Program:**

Entity Nr is

port(A, B: in Bit; C: out Bit);

end;

Architecture ANr of Nr is

begin

C<= A NOR B;

end;

**3. EXOR Gate**:-

**Program:**

Entity Ex is,

port(A, B: in Bit; C: out Bit);

end;

Architecture AEx of Ex is

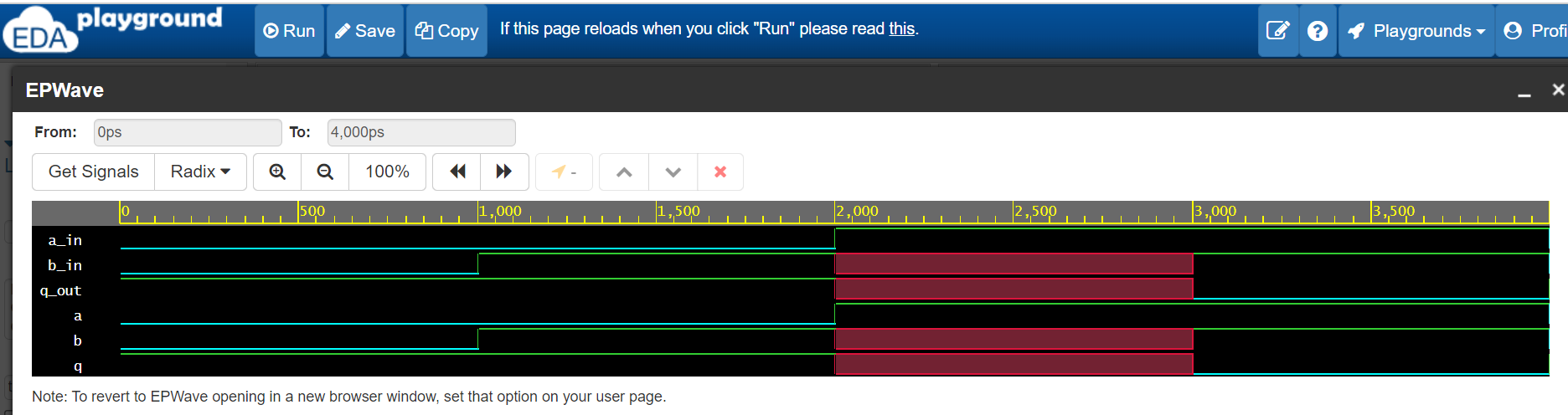
begin

C<= A XOR B;

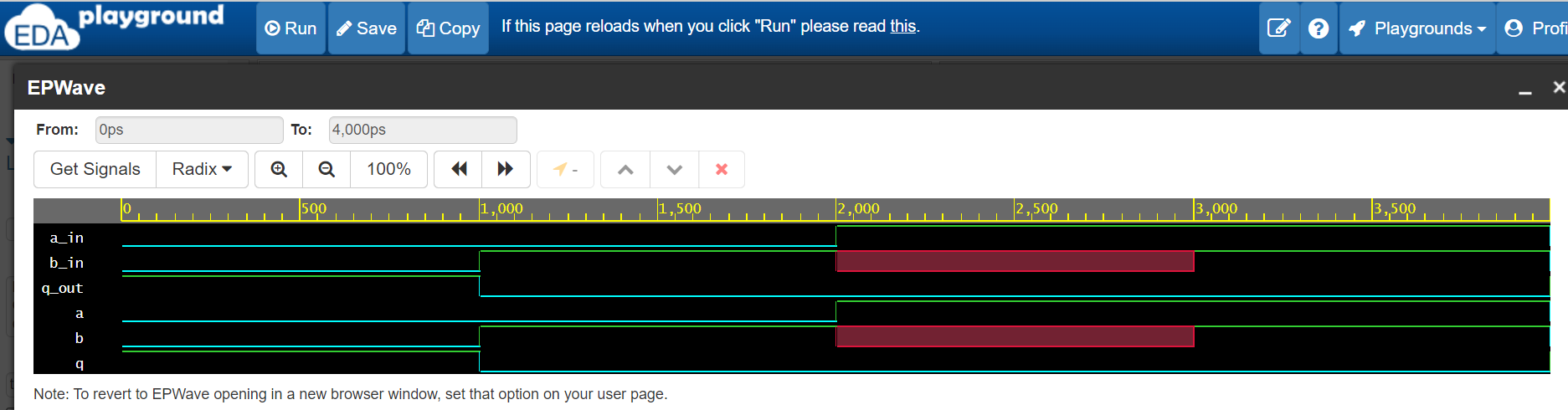
end;

**Output:**

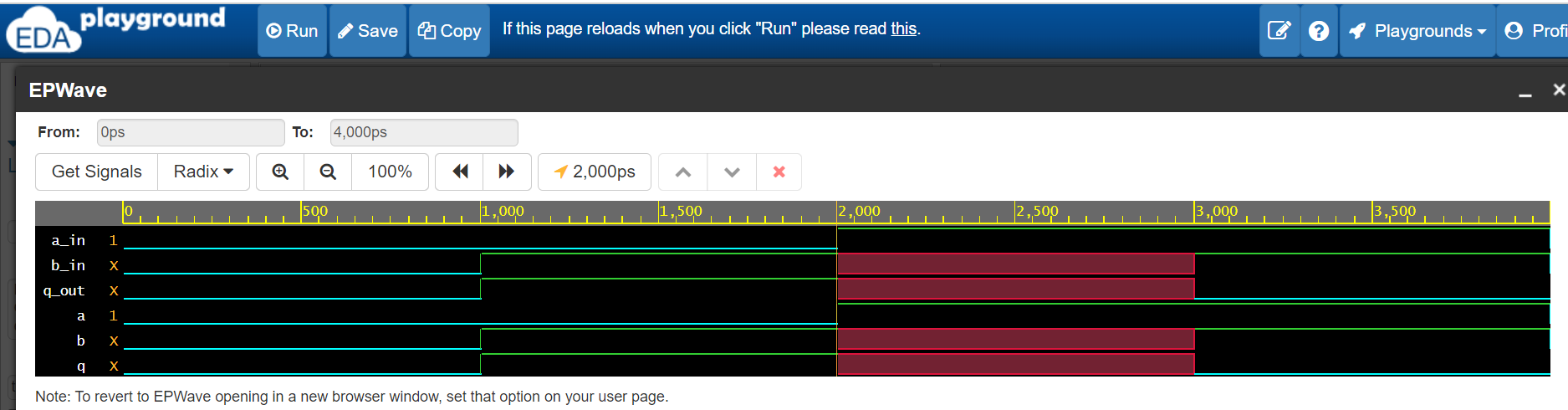
**A) NAND Gate:**

****

B) **NOR Gate:**

****

**EXOR Gate:**

****

**Result:** Thus we have seen simulation and implementation of NAND, NOR and XOR gate using VHDL

**Viva Questions :**

1. Which are the logic gates whose all output entries are logic 1 except for one entry

there is logic 0?

2. When the output of a NOR gate is high?

3. What is a combinational circuit?

**PRACTICAL NO – 4**

**Program:**

entity mux8to1 is

port ( s : in bit\_vector (2 downto 0);

d : in bit\_vector (7 downto 0);

y : out bit);

end mux8to1;

architecture equation of mux8to1 is

begin

with s select

y <= d(0) when "000",

d(1) when "001",

d(2) when "010",

d(3) when "011",

d(4) when "100",

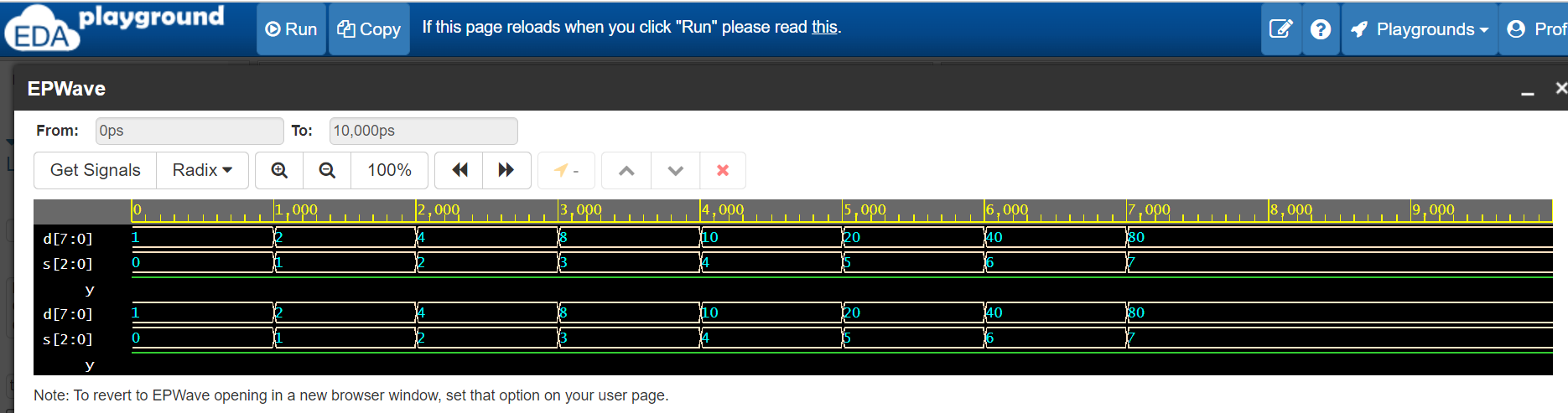
d(5) when "101",

d(6) when "110",

d(7) when others;

end equation;

**Output:**

****

**Result :** Thus we have seen Simulation & implementation of 8:1 Mux.

**Viva questions ?**

1. What is a sequential circuit?

2. Give the applications of multiplexer.

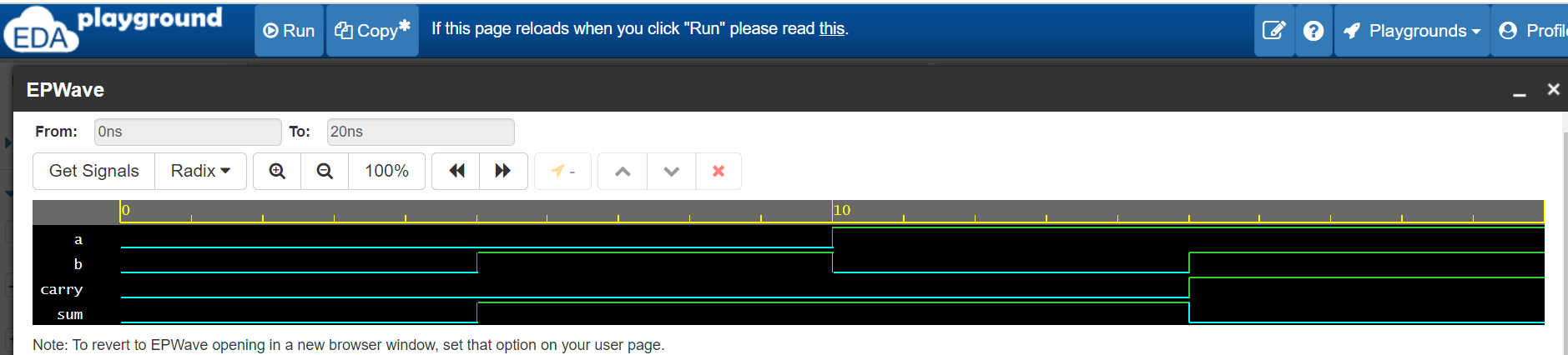
3. How many control inputs are there in 8:1 multiplexer?

4. What is a multiplexer?

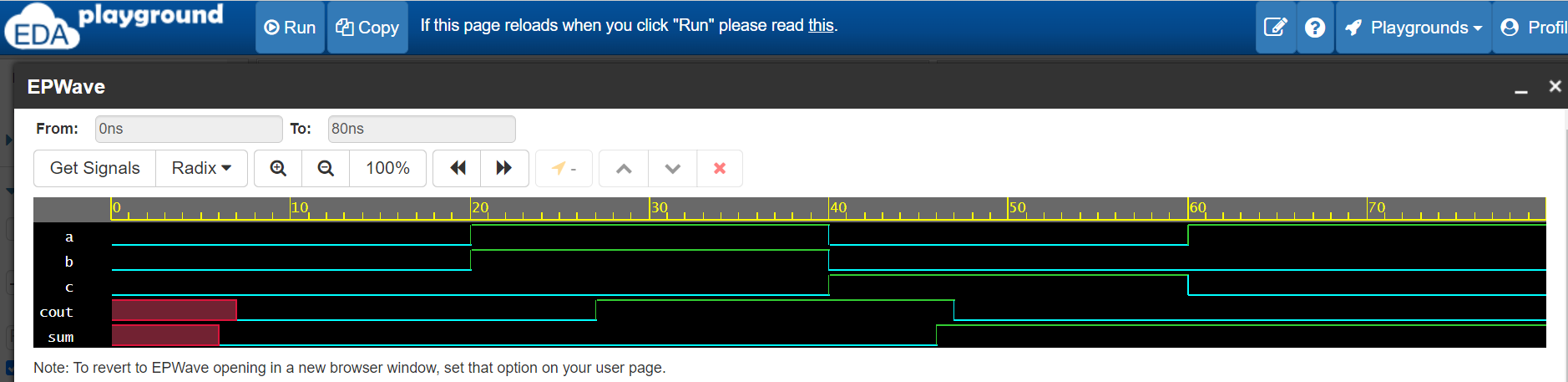
**PRACTICAL NO – 5**

**Output:**

**A) Half Adder:-**



**B) Full Adder :**

****

**Result:** Thus we are designed half adder and full adder circuit using VHDL successfully.

**Viva Questions ?**

1. What is sequential circuit?

2. Differentiate between combinational and sequential circuits.

3. What is a half adder?

4. What is a full adder?

5. What are the applications of adders?

**PRACTICAL NO. – 6**

**Program:-**

Entity code is

Port(b0,b1,b2,b3:in Bit; g(0),g(1),g(2),g(3):out Bit);

End;

Architecture gcode of code is

Begin

g(0)<=b0 XOR b1;

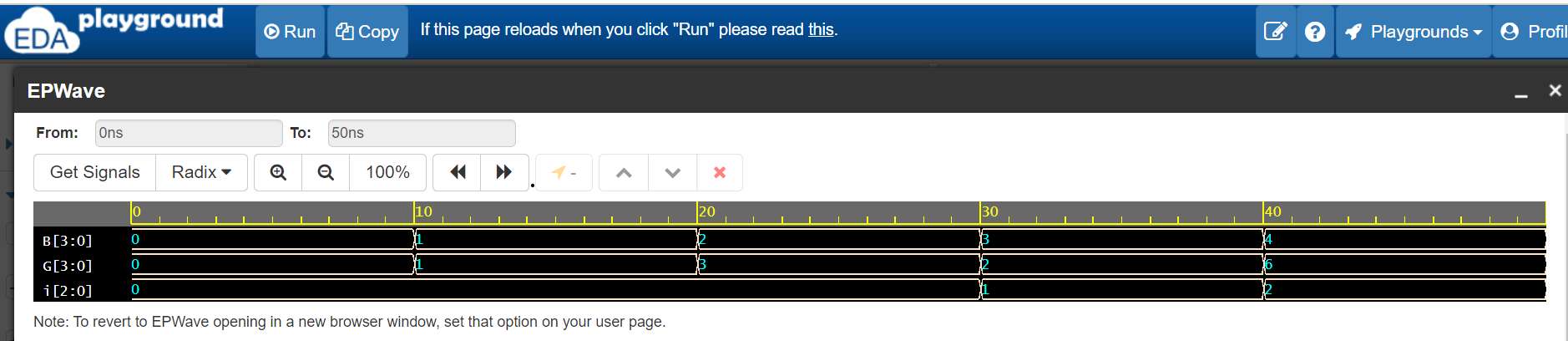
g(1)<=b1 XOR b2;

g(2)<=b2 XOR b3;

g(3)<=b3;

End;

**Output:**

****

**Result:** Thus, we have design & implemented binary to gray code using VHDL.

**Viva Questions ?**

1. What are code converters?

2. What is the necessity of code conversions?

3. What is gray code?

4. Realize the Boolean expressions for - Binary to gray code conversion.

**PRACTICAL NO – 7**

**PROGRAM:-**

Entity dec is

Port(a,b,c: in Bit;d:out Bit\_vector(0 to 7));

End;

Architecture pdec of dec is

Begin

d(0)<=(not a) and(not b) and (not c) ;

d(1)<=(not a) and(not b) and c ;

d(2)<=(not a) and b and (not c) ;

d(3)<=(not a) and b and c ;

d(4)<= a and (not b) and (not c) ;

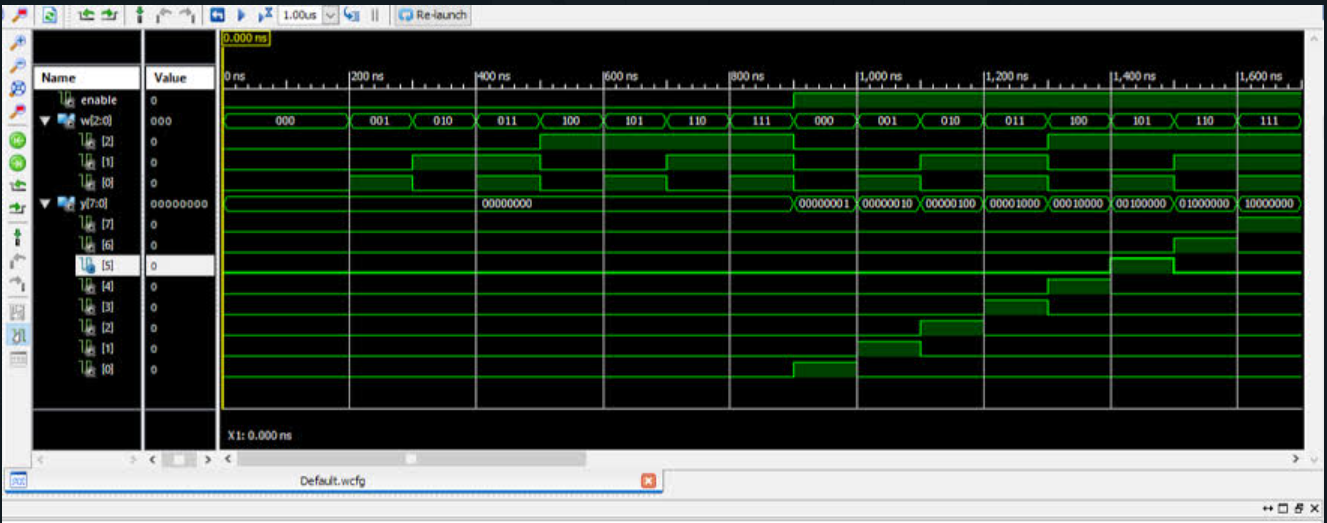
d(5)<=a and (not b) and c ;

d(6)<= a and b and (not c) ;

d(7)<= a and b and c ;

End architecture;

**Output:**

****

**Result:** Thus, we have designed and implemented 3:8 decoder successfully.

**Viva Questions ?**

1. What are the applications of decoder?

2. What is the difference between decoder & encoder?

3. Using 3:8 decoder and associated logic, implement a full adder?

**PRACTICAL NO. – 8**

**Program:**

Library ieee:

Use ieee std\_logic\_1164.all;

Entity af is

Port( idlock : In std\_logic ; Q: out std\_logic);

End;

Architecture aaf of af is

Begin

Process (clock)

Begin

If clock event and clock=’1’ then

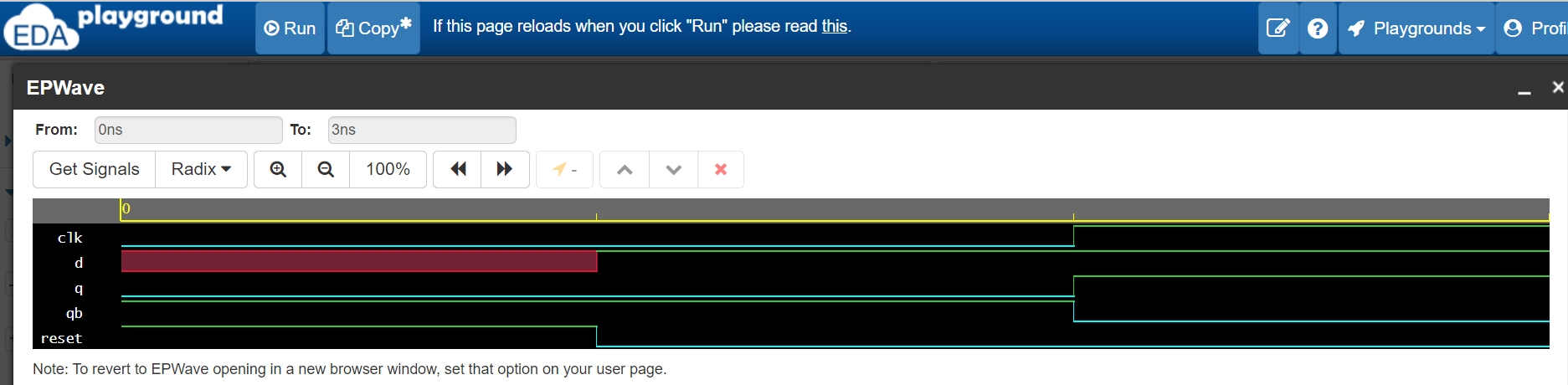
Q<=D;

End if;

End process;

End aaf;

**Output:**

****

**Result:** Thus we have successfully designed and implemented D-flip flop.

**Viva Questions ??**

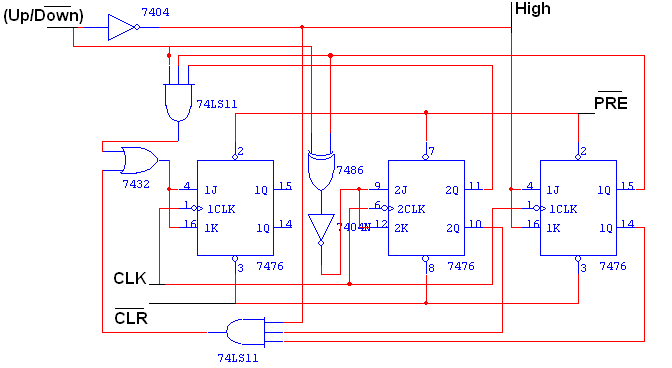
1. What are the applications of different Flip-Flops?

2. What are the different flip-flop?

3. What is race around in flip-flop & how to overcome it?

**PRACTICAL NO. 9**

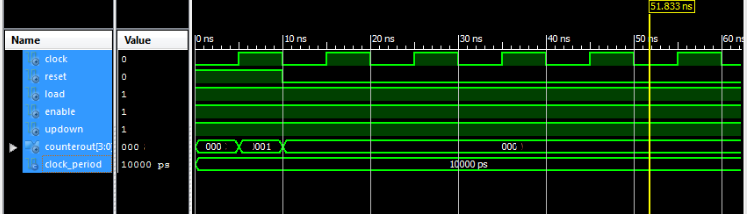
**LOGIC DIAGRAM:**



**TRUTH TABLE:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input**  **Up/Down** | **Present State**  **QA QB QC** | **Next State**  **QA+1 Q B+1 QC+1** | **A**  **JA KA** | **B**  **JB KB** | **C**  **JC KC** |
| **0** | **0 0 0** | **1 1 1** | **1 X** | **1 X** | **1 X** |
| **0** | **1 1 1** | **1 1 0** | **X 0** | **X 0** | **X 1** |
| **0** | **1 1 0** | **1 0 1** | **X 0** | **X 1** | **1 X** |
| **0** | **1 0 1** | **1 0 0** | **X 0** | **0 X** | **X 1** |
| **0** | **1 0 0** | **0 1 1** | **X 1** | **1 X** | **1 X** |
| **0** | **0 1 1** | **0 1 0** | **0 X** | **X 0** | **X 1** |
| **0** | **0 1 0** | **0 0 1** | **0 X** | **X 1** | **1 X** |
| **0** | **0 0 1** | **0 0 0** | **0 X** | **0 X** | **X 1** |
| **1** | **0 0 0** | **0 0 1** | **0 X** | **0 X** | **1 X** |
| **1** | **0 0 1** | **0 1 0** | **0 X** | **1 X** | **X 1** |
| **1** | **0 1 0** | **0 1 1** | **0 X** | **X 0** | **1 X** |
| **1** | **0 1 1** | **1 0 0** | **1 X** | **X 1** | **X 1** |
| **1** | **1 0 0** | **1 0 1** | **X 0** | **0 X** | **1 X** |
| **1** | **1 0 1** | **1 1 0** | **X 0** | **1 X** | **X 1** |
| **1** | **1 1 0** | **1 1 1** | **X 0** | **X 0** | **1 X** |
| **1** | **1 1 1** | **0 0 0** | **X 1** | **X 1** | **X 1** |

**Output:**

****

**Procedure:**

1. Connections are given as per circuit diagram.
2. Logical inputs are given as per circuit diagram.
3. Observe the output and verify the truth table.

**Result:** Thus we have studied to design and implement 3 bit synchronous up/down counter.

**Viva Questions???**

1. What are synchronous counters?

2. What are the advantages of synchronous counters?

3. What is an excitation table?