Logo

Description automatically generated

**SCHOOL OF ENGINEERING AND INFORMATION TECHNOLOGY**

**DIGITAL SYSTEM DESIGN LAB MANUAL**

**For**

**III Semester B. Tech CSE**

**Subject Name: DIGITAL SYSTEM DESIGN LAB**

**Subject Code:** **CSE 2162**

**Regulation: 2018**

**Submitted by**

**DEPARTMENT OF**

**COMPUTER SCIENCE AND ENGINEERING**

Logo

Description automatically generated

**SCHOOL OF ENGINEERING AND INFORMATION TECHNOLOGY**

**Certificate**

*Certified to be the bonafide record of the work done by Ms Fatima Khan Registration No 210101060*

*Branch Computer Science………Semester…3……….*

*in…Digital System Design...*

*laboratory, Bachelor of Technology program, during the year 2022-2023.*

Instructor Name…Suresha Sir

Submitted for End semester Practical Examination held on………

Examiner – I Examiner - II

**CSE 2162 DIGITAL SYSTEM DESIGN LAB [0-0-3-1]**

**Total number of Lab Sessions: 12**

**Course Overview:**

This course is a comprehensive study of principles and techniques of designingdigital systems. It teaches the fundamentals of digital systems applying the logicdesign and development techniques. Although a background in basic electronics is helpful, most of the material requires no training in electronics.

**Course Objectives:**

* To develop the skills of implementing logic circuits using Verilog.
* Simplify the logical expressions and implement using logic gates.
* Design and analyze combinational and sequential circuits, simple systems.
* Relate theoretical concepts to practical applications like multiplexer, encoder, decoder, code converter, counter, shift register applications.

|  |  |  |
| --- | --- | --- |
| **Course Learning outcome (CLO)** | | **Statement** |
| CLO1 | CSE 2162.1 | Simplify logical expressions and simulate using Verilog. |
| CLO2 | CSE 2162.2 | Design and analyze arithmetic circuits and combinational circuits using multiplexers, encoders, and decoders. |
| CLO3 | CSE 2162.3 | Construct decoders, encoders and employ them to suite various practical applications. |
| CLO4 | CSE 2162.4 | Design and simulate sequential circuits using Verilog. |
| CLO5 | CSE 2162.5 | Design and simulate simple processors. |

|  |  |  |
| --- | --- | --- |
| **Internal Assessmend and End Semester Exam Details (Lab)** | | |
| Continuous Assessment  (Max. Marks: 60) | Internal Test (Max. Marks: 20) | * Implement and demonstrate the practical questions given * Viva Voce |
| Assignment/Activity  (Max. Marks: 40) | * Lab Observation/ Demo/ Viva * Record book |
| End Semester Examination  (Max. Marks: 40) | Practical Examination (40% weightage)  (Max. Marks: 100) | |

**Important Instructions to the Students**

1. You should first study the solved problems and then try to solve all the exercise problems of the experiment in the lab.
2. Maintaining an observation copy is compulsory, where in the Verilog code of all the problems solved in the lab should be properly noted down after the completion.
3. Show your results and observation copies to the concerned staff.
4. You should maintain a folder of all the programs in the lab computer and save it by your name/roll no. You are also advised to keep a back-up of it.
5. Use of external storage media is not allowed.
6. Maintain the timings and discipline of the lab.
7. You will be evaluated in every lab you attend, based on your performance, observation copy and behavior in the lab.

|  |
| --- |
| C:\Users\suresha\AppData\Local\Microsoft\Windows\INetCache\Content.Outlook\8QLM4Q5S\MAHE Dubai_Logo1_2018(2570x774).jpg  **Digital Circuit Design Using Verilog Code**  **School of Engineering & Information Technology**  **Department of ECE and EEE** |

**HISTORY**

***Relationship between VLSI, VHDL, Verilog, Embedded systems, ASIC and FPGA***

Electronics started in the year 1906 with Vacuum Tubes. Later in the year 1947 transistor was invented. After that all electronic circuits were built using transistors. Later ICs were invented.  Initials days 10+ transistors were placed inside the IC, later 100+, 1000+ and 10000+ transistors were placed inside the IC.

|  |  |  |
| --- | --- | --- |
| 10+ | Transistors/IC | SSI |
| 100+ | Transistors/IC | MSI |
| 1000+ | Transistors/IC | LSI |
| 10000+ | Transistors/IC | VLSI |

Even ULSI and other names were used, VLSI got popular. So VLSI means more transistors /IC. Now a days 10,000,000,000 Transistors are placed inside IC. This IC is also called VLSI. This is all about Technology.  
Initial days electronics was implemented using Circuits. There circuits were put into the IC by IC manufacturers. But the MOQ (Minimum Order Quantity) was about 10000+. So PLD was invented. The circuits can be put inside IC buy the user himself. When an user buys the PLD IC, the logic gates inside the ICs are not connected. The user only connect. This process is called Configuration. 

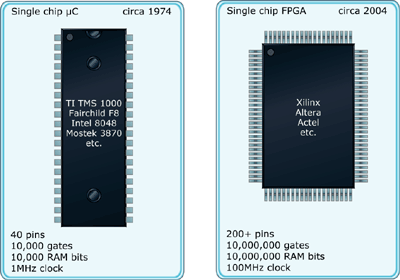
***Once the logic becomes complex, the circuits becomes big. Also testing the circuits become complex. To make it easy HDL (Hardware Description Language) was invented. There are two type of HDLs. VHDL and Verilog. These languages are used to configure the IC.***

**VLSI Engineer:** The engineer should have good Digital Electronics knowledge to work in HDLs. Today VLSI design means usage of Verilog or VHDL. VLSI verification means System Verilog. FPGA (Field Programmable Gate Array) is used to implement the Verilog (Verification + Logic) Or VHDL (Very High Speed Integrated Hardware Description Language). If you know VHDL or Verilog then you are called VLSI engineer.

**Embedded Engineer:** If you know Microprocessor programming, then you are called Embedded Engineer. Today Embedded industry prefer in-depth knowledge in C programming. The semiconductor industry has been experiencing an unprecedented growth fuelled by the developments in the VLSI industry. The VLSI industry has already crossed the quarter-micron threshold and devices with 0.18 micron feature-size are already in market. Thus, it becomes extremely difficult to design complex circuits with the help of conventional designing methods. The computer aided design tools introduced serve the purpose.

***To realize the design, programmable logic devices such as CPLDs (Complex Programmable Logic Devices) and FPGAs (Field Programmable Gate Arrays), come in very handy. Since these devices are programmable, the user can modify their designs very easily and conveniently.***

**FPGAs and Microprocessors**

**FPGAs and microprocessors are more similar than you may think. Here's a primer on how to program an FPGA and some reasons why you'd want to.** Small processors are, by far, the largest selling class of computers and form the basis of many embedded systems. The first single-chip microprocessors contained approximately 10,000 gates of logic and 10,000 bits of memory. Today, field programmable gate arrays (FPGAs) provide single chips approaching 10 million gates of logic and 10 million bits of memory.

**Figure 1: Comparison of first microprocessors to current FPGAs**

**ANALYSIS VERSUS SYNTHESIS**

A designer of digital systems is faced with two basic issues.

* For an existing logic network, it must be possible to determine the function performed by the network. This task is referred to as the ***analysis process***.
* The reverse task of designing a new network that implements a desired functional behavior is referred to as the ***synthesis process.*** The analysis process is rather straightforward and much simpler than the synthesis process.

**The basic steps involved in designing of digital system,**

1. Specify the desired behavior of the circuit.
2. Synthesize the circuit.
3. Implement the circuit.
4. Test the circuit to check whether the desired specifications meet.

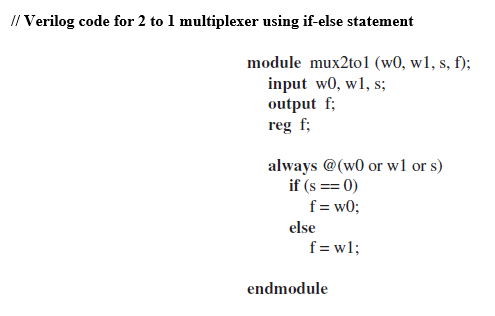
***Brief history of VERILOG***

* In the 1980s rapid advances in integrated circuit technology lead to efforts to develop standard design practices for digital circuits. Verilog was produced as a part of that effort.
* The original version of Verilog was developed by Gateway Design Automation, which was later acquired by Cadence Design System.
* In 1990 Verilog was put into the public domain, and it has since become the most popular language for describing digital circuits.
* In 1995 Verilog was adopted as an official IEEE Standard, called 1364-1995. An enhanced version of Verilog, called Verilog 2001, was adopted as IEEE Standard 1364-2001 in 2001.
* Verilog was originally intended for simulation and verification of digital circuits. Subsequently, with the addition of synthesis capability, Verilog has also become popular for use in design entry in CAD systems. The CAD tools are used to synthesize the Verilog code into a hardware implementation of the described circuit.

Verilog is a complex, sophisticated language. Learning all of its features is a daunting task. However, for use in synthesis only a subset of these features is important. To simplify the presentation, we will focus the discussion on the features of Verilog that are actually used in the examples in the notes. Verilog has a number of constructs similar to the C programming language.

**Behavioral Description (Procedural Assignment)**

**Rather than specifying a circuit by giving a Boolean expression, we can use if – else statements (or), case or while or for statements as a higher level of abstraction. Such statements are called procedural statements.**



**STRUCTURE OF VERILOG MODULE**

* The Verilog HDL describes a digital system as a set of modules. Each of these modules has an interface to each other modules to describe how they are interconnected. Each module consists of a **declaration and a body**.
* In the declaration, name, inputs and outputs of the module are listed. The body shows the relationship between the inputs and the outputs.

A module is a basic building block of Verilog HDL. Modules can represent pieces of hardware ranging from simple gate to complete systems. Eg, Microprocessor.

A Verilog input file in the Xilinx software environment consists of the following segments:

***Header:*** module name, list of input and output ports.

***Declarations:*** input and output ports, registers and wires.

***Logic Descriptions:*** equations, state machines and logic functions.

***End:*** endmodule

All your designs for this training must be specified in the above Verilog input format. Note that the *state diagram* segment does not exist for combinational logic designs.

The general structure of module is,

**module < module name > < port list > ;**

**< declares >**

**< module items >**

**endmodule**

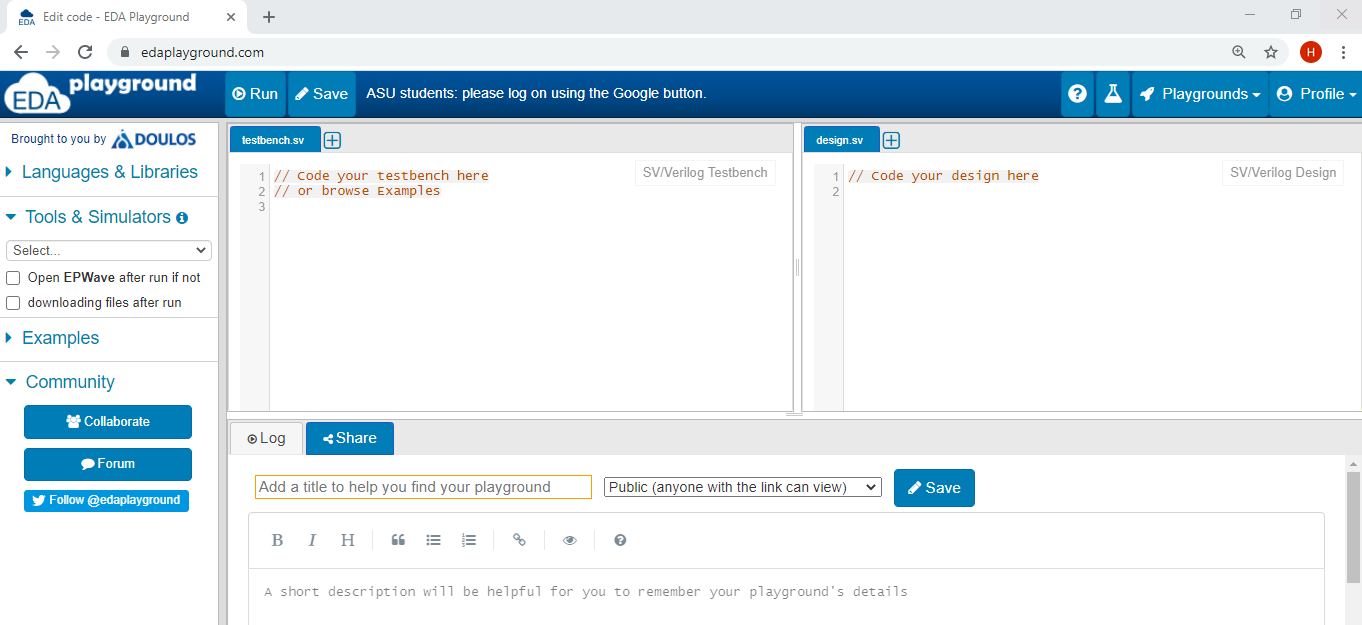
**Example:** Consider the circuit in shown in Figure.

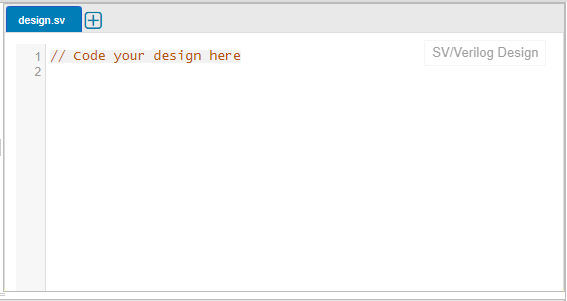
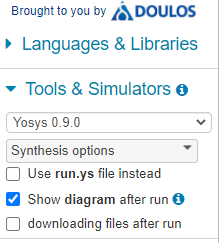
|  |  |
| --- | --- |
|  | // Verilog code (**gate level primitives**)  **module** logicfunction (*,,* *,*f);  **input** *,,* ;  **output** f;  **and** ( g, *,*);  **not** (k, );  **and** (h, k, );  **or** (f, g, h);  **endmodule** |

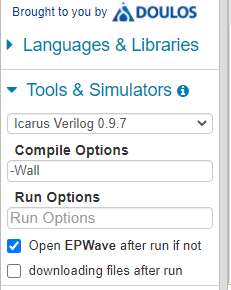
**SYNTHESIS USING AND, OR, AND NOT GATES**

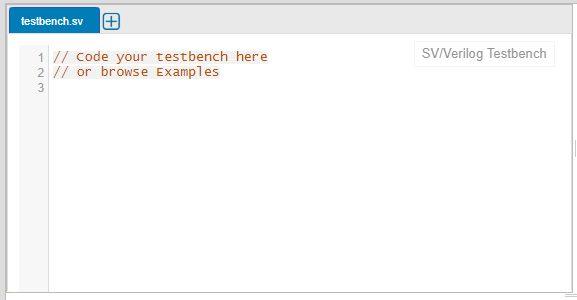
***The process whereby we begin with a description of the desired functional behavior and then generate a circuit that realizes this behavior is called synthesis.***

**Edaplay Ground**









**Experiment no 1: REALIZATION OF BOOLEAN FUNCTION SOP & POS (Algebraic Method)**

**The list of experiments/questions given below includes additional assignment questions given to students for practice.**

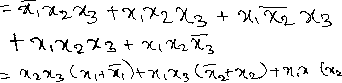
|  |  |
| --- | --- |
| Q.No | Questions |
| 1. | Design the simplest circuit that has 3 inputs which produces an output value of 1 whenever two or more of the input variables have the value 1; otherwise, the output has to be 0. Write the Verilog code for the simplified SOP expression using behavioral description. Simulate in Edaplay Ground. |
| 2. | Design the simplest circuit that has 3 inputs which produces an output value of 1 which produces an output value 1 whenever exactly one or two of the input variables have the value 1; otherwise, the output has to be 0. Write the Verilog code for the simplified POS expression using behavioral description. Simulate in Edaplay Ground. |
| 3 | **Case Study: Three-Way Light Control**  Assume that a large room has three doors and that a switch near each door controls a light in the room. It has to be possible to turn the light on or off by changing the state of any one of the switches. As a first step, let us turn this word statement into a formal specification using a truth table. Let 𝑥1, 𝑥2, 𝑎𝑛𝑑 𝑥3 be the input variables that denote the state of each switch. Assume that the light is off if all switches are open. Closing any one of the switches will turn the light on. Then turning on a second switch will have to turn off the light. Thus the light will be on if exactly one switch is closed, and it will be off if two (or no) switches are closed. If the light is off when two switches are closed, then it must be possible to turn it on by closing the third switch. If 𝑓 (𝑥1, 𝑥2, 𝑥3) represents the state of the light. Write the Verilog code for the simplified POS expression using behavioral description. Simulate in Edaplay Ground. |

**Exp no 1: Design the simplest circuit that has 3 inputs which produces an output value of 1 whenever two or more of the input variables have the value 1; otherwise, the output has to be 0. Write the Verilog code for the simplified SOP expression using behavioral description. Simulate in Edaplay Ground.**



**Truth table: SOP Expression**

***Truth table:***



|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

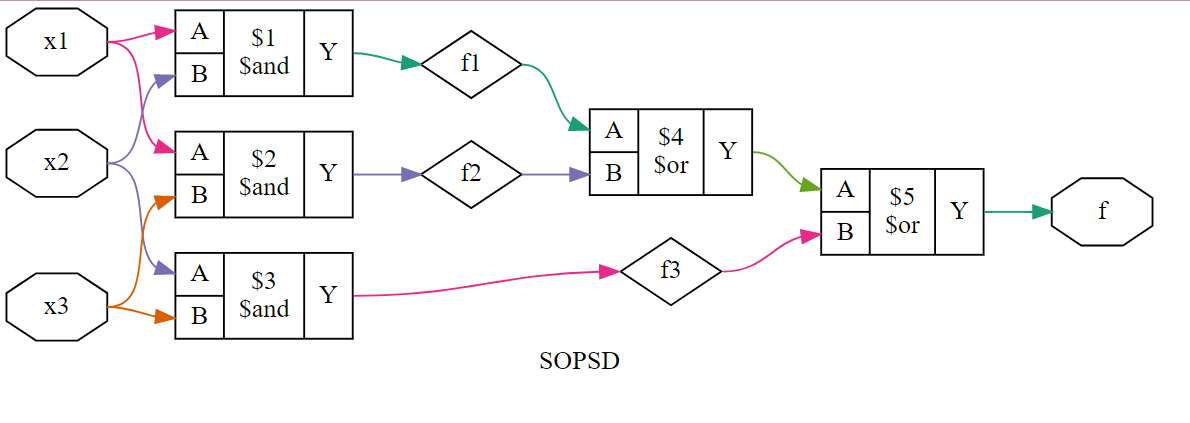


***Verilog code using Structural Description (gate level primitives) :***

// Verification of SOP using Gate level primitives - 210101060  
module SOP(x1,x2,x3,f);  
input x1,x2,x3;  
output wire f;  
and(f1, x2, x3);  
and(f2, x1, x3);  
and(f3, x1, x2);  
or(f, f1, f2, f3);  
endmodule

Text

Description automatically generated



*Testbench code:*

// Testbench  
module SOPTB;  
reg x1, x2, x3;  
wire f;  
SOPSD uut(.x1(x1), .x2(x2), .x3(x3), .f(f));  
initial begin  
x1 = 0; x2 = 0; x3 = 0; #100;  
x1 = 0; x2 = 0; x3 = 1; #100;  
x1 = 0; x2 = 1; x3 = 0; #100;  
x1 = 0; x2 = 1; x3 = 1; #100;  
x1 = 1; x2 = 0; x3 = 0; #100;  
x1 = 1; x2 = 0; x3 = 1; #100;  
x1 = 1; x2 = 1; x3 = 0; #100;  
x1 = 1; x2 = 1; x3 = 1; #100;  
end  
initial begin  
$dumpfile("dump.vcd");  
$dumpvars();  
end  
endmodule

Graphical user interface, text

Description automatically generated

Graphical user interface, application

Description automatically generated

***// Verilog Source Code using Behavioral description (Continuous assignment):***

// 210101060, continuous assignment

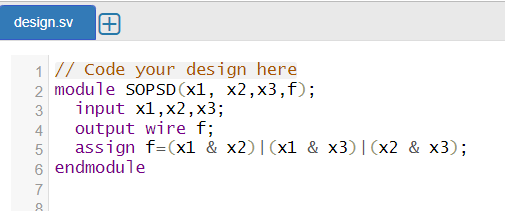
module SOPSD(x1, x2,x3,f);

input x1,x2,x3;

output wire f;

assign f=(x1 & x2)|(x1 & x3)|(x2 & x3);

endmodule



Diagram

Description automatically generated

*Testbench code:*

// Code your testbench here

module SOPSDTB;

reg x1, x2, x3;

wire f;

SOPSD uut(.x1(x1), .x2(x2), .x3(x3), .f(f));

initial begin

x1= 0; x2= 0; x3= 0; #100;

x1= 0; x2= 0; x3= 1; #100;

x1= 0; x2= 1; x3= 0; #100;

x1= 0; x2= 1; x3= 1; #100;

x1= 1; x2= 0; x3= 0; #100;

x1= 1; x2= 0; x3= 1; #100;

x1= 1; x2= 1; x3= 0; #100;

x1= 1; x2= 1; x3= 1; #100;

end

initial begin

$dumpfile("dump.vcd");

$dumpvars();

end

endmodule

Text

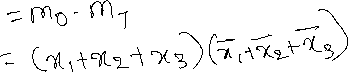
Description automatically generated with low confidence

Graphical user interface, application

Description automatically generated

**Exp 2: Design the simplest circuit that has 3 inputs which produces an output value of 1 which produces an output value 1 whenever exactly one or two of the input variables have the value 1; otherwise, the output has to be 0. Write the Verilog code for the simplified POS expression using behavioral description. Simulate in Edaplay Ground.**

**Truth table: POS Expression**



|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

// reg. 210101060- POS karnaugh map- continuous assignment-brhavooral description

module POS (x1, x2, x3, f);

input x1, x2, x3;

output f;

wire f;



assign f=(x1|x2|x3) & (~x1|~x2|~x3);

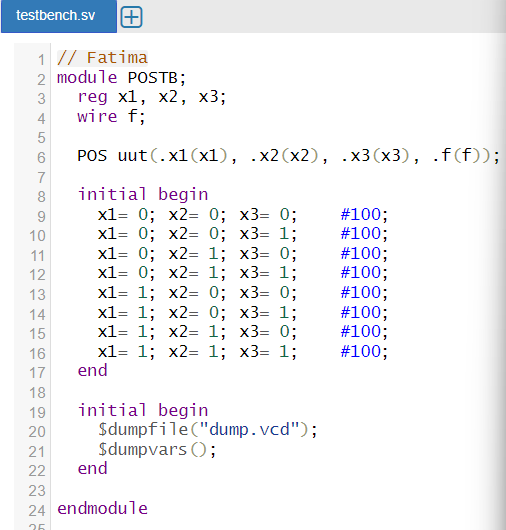
endmodule

Text, letter

Description automatically generated

Diagram

Description automatically generated



Graphical user interface, text, application

Description automatically generated

**Case Study: Three-Way Light Control**

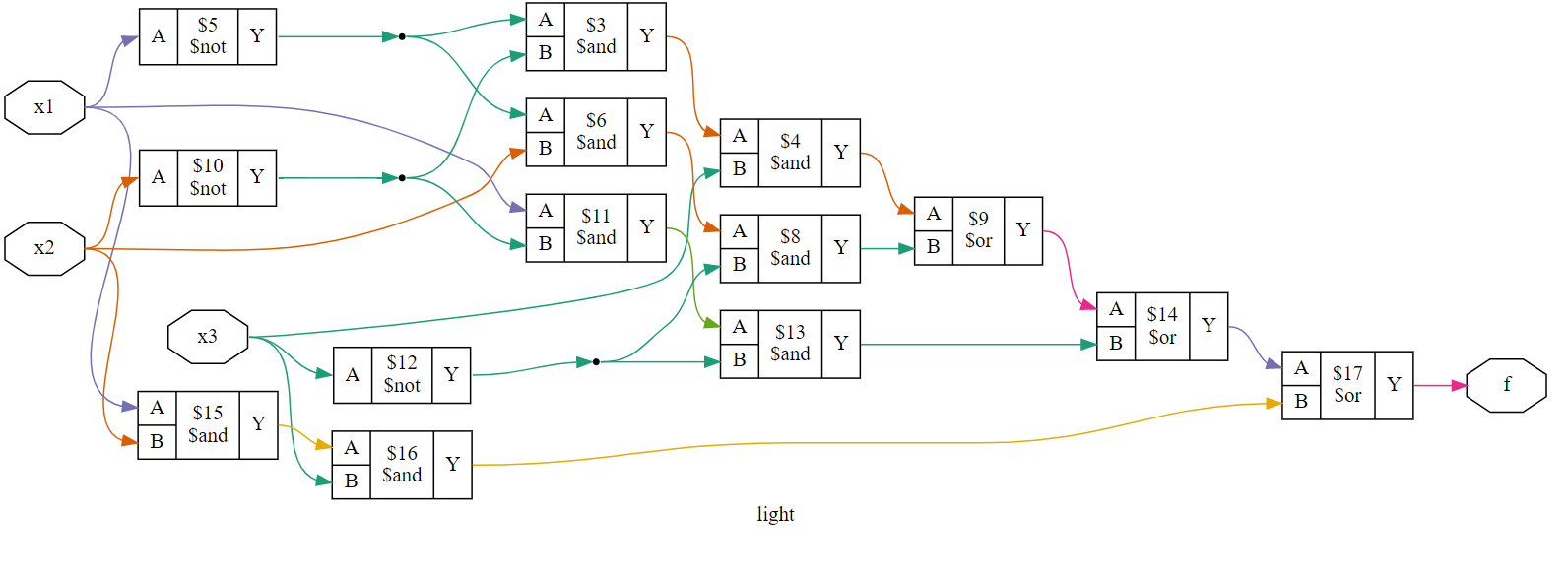
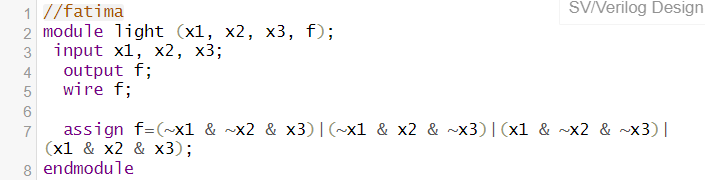
1. Assume that a large room has three doors and that a switch near each door controls a light in the room. It has to be possible to turn the light on or off by changing the state of any one of the switches. As a first step, let us turn this word statement into a formal specification using a truth table. Let 𝑥1, 𝑥2, 𝑎𝑛𝑑 𝑥3 be the input variables that denote the state of each switch. Assume that the light is off if all switches are open. Closing any one of the switches will turn the light on. Then turning on a second switch will have to turn off the light. Thus the light will be on if exactly one switch is closed, and it will be off if two (or no) switches are closed. If the light is off when two switches are closed, then it must be possible to turn it on by closing the third switch. If 𝑓 (𝑥1, 𝑥2, 𝑥3) represents the state of the light. Write the Verilog code for the simplified POS expression using behavioral description. Simulate in Edaplay Ground.

**Truth table: SOP Expression**



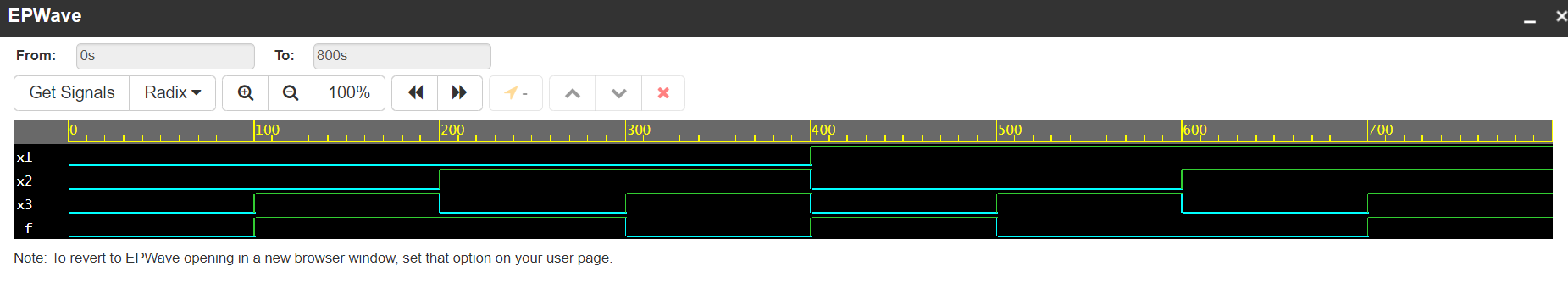
|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |





Table

Description automatically generated with medium confidence



**Realization of Boolean function (SOP and POS) (Karnaugh method)**

**The list of experiments/questions given below includes additional assignment questions given to students for practice.**

|  |  |
| --- | --- |
| Q.No | Questions |
| 1. | Determine the simplified SOP expression for the function using karnaugh map. Write the Verilog code for this implementation, synthesize and simulate using Edaplay ground. |
| 2. | Design the simplest circuit that has 3 inputs which produces an output value of 1 whenever two or more of the input variables have the value 1; otherwise, the output has to be 0. Write the Verilog code for the simplified SOP expression using behavioral description. Simulate in Edaplay Ground. |

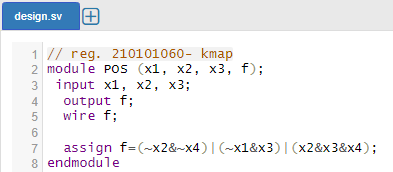
1. Determine the simplified SOP expression for the function using karnaugh map. Write the Verilog code for this implementation, synthesize and simulate using Edaplay ground.

**Karnaugh method:**

Diagram

Description automatically generated

//**Verilog code using Behavioral modelling**

****

// 210101060

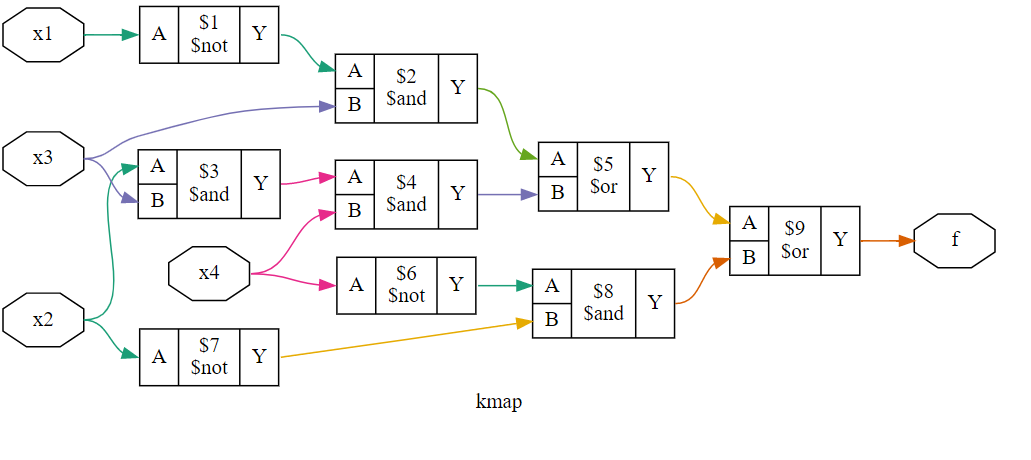
module kmap(x1, x2, x3, x4, f );

input x1, x2, x3, x4;

output wire f;

assign f = (~x1 & x3) | (x2 & x3 & x4) | (~x4 & ~x2);

endmodule



Text

Description automatically generated

//fatima

module KMAPTB;

reg x1, x2 ,x3, x4;

wire f;

kmap uut(.x1(x1), .x2(x2), .x3(x3), .x4(x4) , .f(f));

initial begin

$monitor("time=%d, x1=%b, x2=%b, x3=%b, x4=%b, f=%b\n", $time, x1, x2, x3, x4, f);

x1=0;x2=0;x3=0;x4=0;#100;

x1=0;x2=0;x3=0;x4=1;#100;

x1=0;x2=0;x3=1;x4=0;#100;

x1=0;x2=0;x3=1;x4=1;#100;

x1=0;x2=1;x3=0;x4=0;#100;

x1=0;x2=1;x3=0;x4=1;#100;

x1=0;x2=1;x3=1;x4=0;#100;

x1=0;x2=1;x3=1;x4=1;#100;

x1=1;x2=0;x3=0;x4=0;#100;

x1=1;x2=0;x3=0;x4=1;#100;

x1=1;x2=0;x3=1;x4=0;#100;

x1=1;x2=0;x3=1;x4=1;#100;

x1=1;x2=1;x3=0;x4=0;#100;

x1=1;x2=1;x3=0;x4=1;#100;

x1=1;x2=1;x3=1;x4=0;#100;

x1=1;x2=1;x3=1;x4=1;#100;

end

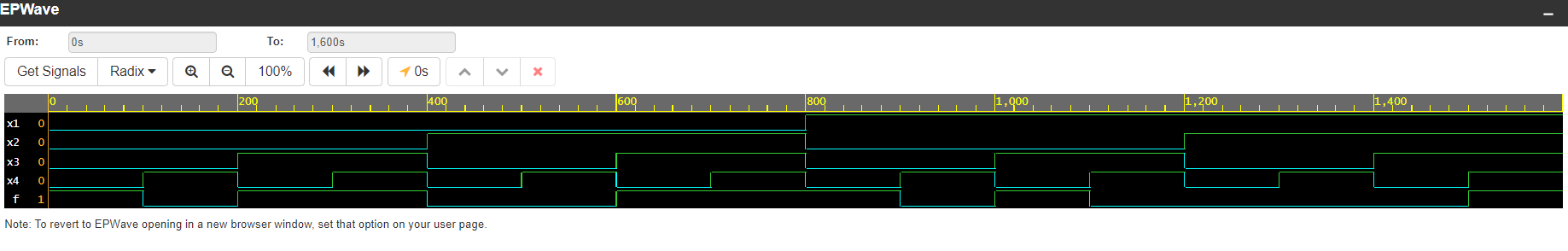
initial begin

$dumpfile("dump.vcd");

$dumpvars();

end

endmodule

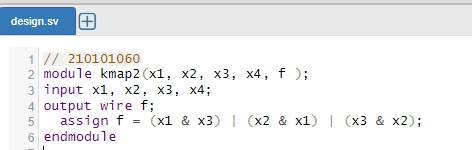


**2. Design the simplest circuit that has 3 inputs 𝑥1, 𝑥2 𝑎𝑛𝑑 𝑥3, which produces an output value of 1  
whenever two or more of the input variables have the value 1; otherwise, the output has to be 0.  
Write the Verilog code for the simplified SOP expression using behavioral description. Simulate in  
Edaplay Ground. Use Karnaugh map.**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |





// 210101060

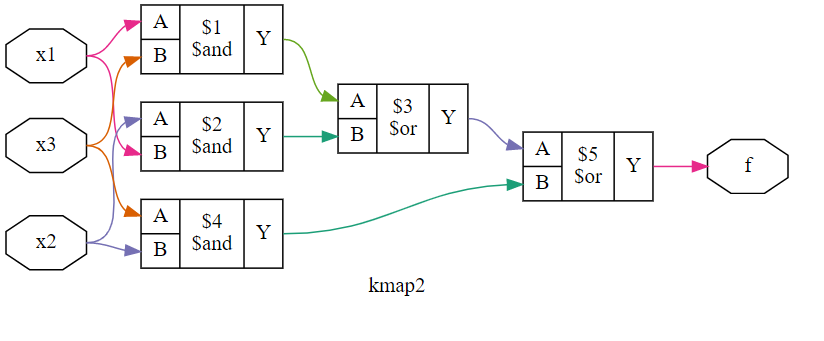
module kmap2(x1, x2, x3, x4, f );

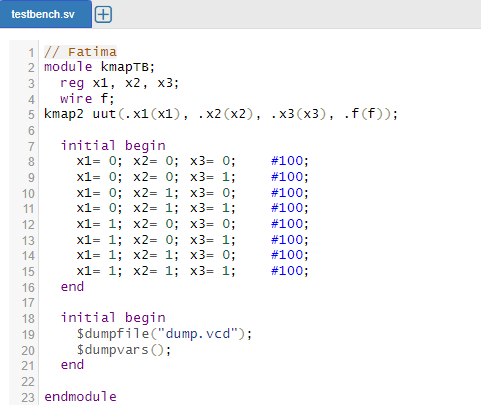
input x1, x2, x3, x4;

output wire f;

assign f = (x1 & x3) | (x2 & x1) | (x3 & x2);

endmodule

****

****

**Graphical user interface, application

Description automatically generated**

**REALIZATION OF MULTILEVEL NAND AND NOR CIRCUITS**

1. Consider a multilevel circuit shown in figure. Find the functionally of the circuit. Write the Verilog code for this implementation, synthesize and simulate using Edaplay ground.

Diagram

Description automatically generated

|  |  |
| --- | --- |
| Table  Description automatically generated | // **Structural Specification Of Logic Circuits**  **//210101060**  **// Verilog Code**  **module** multi (x1, x2, x3, x4, x5, f );  **input** x1, x2, x3, x4, x5;  **output f;**  **assign f = (~x1 & ~x5) | (x2 & ~x5) | (~x3 & ~x5) | (~x4 & ~x5);**  **endmodule** |

Text, letter

Description automatically generated

Diagram

Description automatically generated

Graphical user interface, text, application

Description automatically generated

A picture containing background pattern

Description automatically generated

**SYNTHESIZE & SIMULATION OF ARITHMETIC CIRCUITS IN Edaplay Ground.**

1. Design Half adder logic circuit. Write the Verilog code for this implementation, synthesize and simulate using Edaplay ground.

**DESIGN OF HALF ADDER CIRCUIT**

**Half adder:** ***The circuit which implements the addition of only two bits, is called a half-adder.***

|  |  |
| --- | --- |
| A picture containing graphical user interface  Description automatically generated | A picture containing table  Description automatically generated |
| Diagram, schematic  Description automatically generated | Diagram  Description automatically generated |
| **// Verilog code for HA using gate level primitives.** | **// Verilog code for HA using continuous**  **// assignment.** |
| **module** halfadd ();  **input** ;  **output** ;  **xor** ();  **and** ();    **endmodule** | **module** halfadd ();  **input** ;  **output** ;  **assign** ;  **assign** ;    **endmodule** |

**Text

Description automatically generated**// Verilog code for HA using continuous

// 210101060

module halfadd (x,y,s,c);

input x,y;

output s,c;

assign s = x ^ y;

assign c = x & y;

endmodule

Graphical user interface, text

Description automatically generated**A picture containing clock

Description automatically generated**

// Fatima

module halfTB;

reg x, y;

wire s, c;

halfadd uut(.x(x), .y(y), .s(s), .c(c));

initial begin

x=0;y=0;#100;

x=0;y=1;#100;

x=1;y=0;#100;

x=1;y=1;#100;

end

initial begin

$dumpfile("dump.vcd");

$dumpvars();

end

endmodule

**Graphical user interface

Description automatically generated**

**Full adder: *A logic circuit that accepts two 1-bit signals and carry in as inputs and produces their sum and carry as output is called full adder.***

1. Design full adder logic circuit. Write the Verilog code for this implementation, synthesize and simulate using Edaplay ground.

|  |  |
| --- | --- |
| ***C:\idraw20\25.TIF***  **Truth table**    **Boolean expression**  ***Carry out + XY*** | **Logic diagram**  **C:\idraw20\26.TIF** |

Text

Description automatically generated// Verilog code for FA using continuous

// 210101060

module fulladd (Cin, a, b, Sout, Cout);

input Cin,a,b;

output wire Sout, Cout;

assign Sout = (a^b)^Cin;

assign Cout = (a^b)&Cin|(a&b);

endmodule

Graphical user interface, text

Description automatically generated// Fatima

module FATB;

reg a, b, Cin;

wire Sout, Cout;

fulladd uut(.a(a), .b(b), .Cin(Cin), .Sout(Sout), .Cout(Cout));

initial begin

a=0;b=0;Cin=0;#100;

a=0;b=0;Cin=1;#100;

a=0;b=1;Cin=0;#100;

a=0;b=1;Cin=1;#100;

a=1;b=0;Cin=0;#100;

a=1;b=0;Cin=1;#100;

a=1;b=1;Cin=0;#100;

a=1;b=1;Cin=1;#100;

end

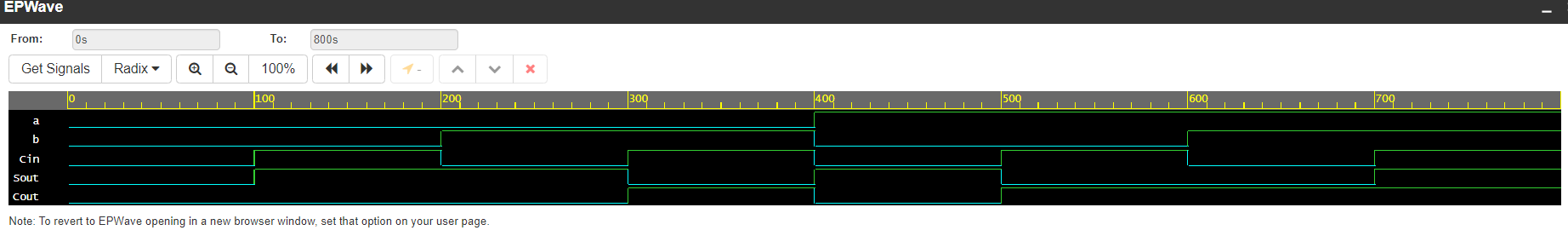
initial begin

$dumpfile("dump.vcd");

$dumpvars();

end

endmodule

****

**Decomposed Full-Adder**

1. Design full adder logic circuit using half adders. Write the Verilog code for this implementation, synthesize and simulate using Edaplay ground.

In view of the names used for the circuits, one can expect that a full-adder can be constructed using half-adders. This can be accomplished by creating a multilevel circuit given in figure. It uses two half-adders to form a full-adder.

Text, letter

Description automatically generatedDiagram

Description automatically generated

// Verilog code for HA using continuous

// 210101060

module halfadd (x,y,s,c);

input x,y;

output s,c;

assign s = x ^ y;

assign c = x & y;

endmodule

module FA(a,b,Cin, Sout, Cout);

input a, b, Cin;

output wire Sout, Cout;

wire f1, f2, f3;

halfadd stage0(.x(a), .y(b), .s(f1), .c(f2));

halfadd stage1(.x(Cin), .y(f1), .s(Sout), .c(f3));

assign Cout=f2|f3;

endmodule

A screenshot of a computer

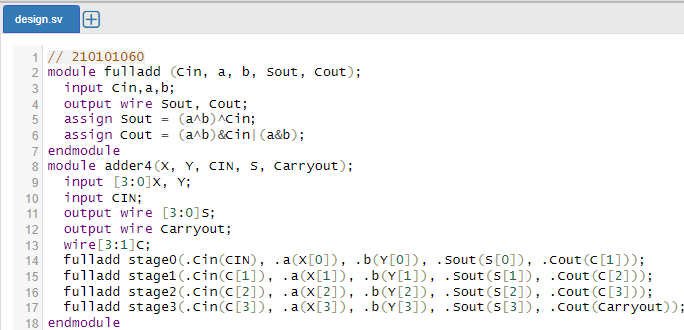
Description automatically generated

**DESIGN OF FOUR BIT ADDER**

1. Write Verilog code to implement using four bit adder circuit. Simulate using Edaplay Ground.

Chart, box and whisker chart

Description automatically generated



// 210101060

module fulladd (Cin, a, b, Sout, Cout);

input Cin,a,b;

output wire Sout, Cout;

assign Sout = (a^b)^Cin;

assign Cout = (a^b)&Cin|(a&b);

endmodule

module adder4(X, Y, CIN, S, Carryout);

input [3:0]X, Y;

input CIN;

output wire [3:0]S;

output wire Carryout;

wire[3:1]C;

fulladd stage0(.Cin(CIN), .a(X[0]), .b(Y[0]), .Sout(S[0]), .Cout(C[1]));

fulladd stage1(.Cin(C[1]), .a(X[1]), .b(Y[1]), .Sout(S[1]), .Cout(C[2]));

fulladd stage2(.Cin(C[2]), .a(X[2]), .b(Y[2]), .Sout(S[2]), .Cout(C[3]));

fulladd stage3(.Cin(C[3]), .a(X[3]), .b(Y[3]), .Sout(S[3]), .Cout(Carryout));

endmodule

Graphical user interface, text

Description automatically generated

module adder4TB;

reg [3:0]X;

reg [3:0]Y;

reg CIN;

wire [3:0]S;

wire Carryout;

adder4 uut(.X(X), .Y(Y), .CIN(CIN), .S(S), .Carryout(Carryout));

initial begin

$dumpvars(1, adder4TB);

#5;

X=0;Y=0;

#5;

X=0;Y=13;

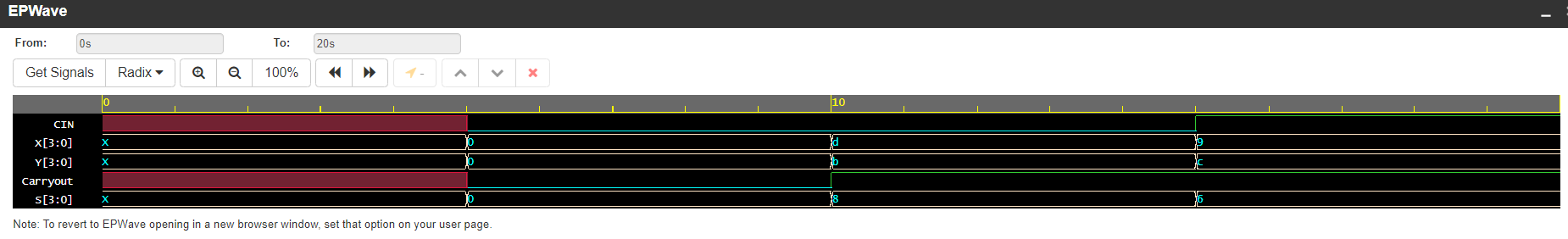
#5;

X=2;Y=12;

#5 $finish;

end

endmodule



Diagram

Description automatically generated

**DESIGN OF CODE CONVERTER (BINARY TO GRAY / GRAY TO BINARY)**

**Design 3 bit binary to grey code converter circuit. Write the Verilog code using continuous assignment, synthesize and simulate using Edaplay ground.**

**BINARY CODE TO GRAY CODE CONVERTER Truth Table:**

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

**Boolean expression:**

G2 = B2

G1 = B2 B1

G0 = B1 B0

Text, letter

Description automatically generated

// 210101060

module binary(B0, B1, B2, G0, G1, G2,);

input B0, B1, B2;

output wire G0, G1, G2;

assign G2=B2;

assign G1=B2^B1;

assign G0=B1^B0;

endmodule

**Graphical user interface, text, application, email

Description automatically generated**

**Diagram

Description automatically generated**

**Graphical user interface, application

Description automatically generated**

**GRAY CODE TO BINARY CODE CONVERTER**

1. Design 3 bit grey to binary code converter circuit Write the Verilog code using continuous assignment, synthesize and simulate using Edaplay ground.

**Truth Table:**

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

**Boolean expression:**

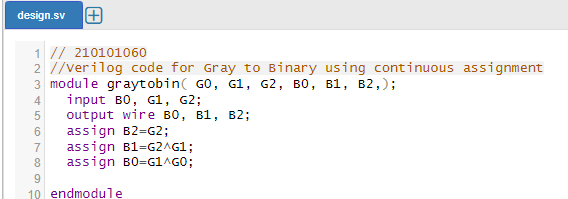
B2 = G2

B1 = G2 G1

B0 = G2 G1 G0

// 210101060

//Verilog code for Gray to Binary using continuous assignment

module graytobin( G0, G1, G2, B0, B1, B2,);

input G0, G1, G2;

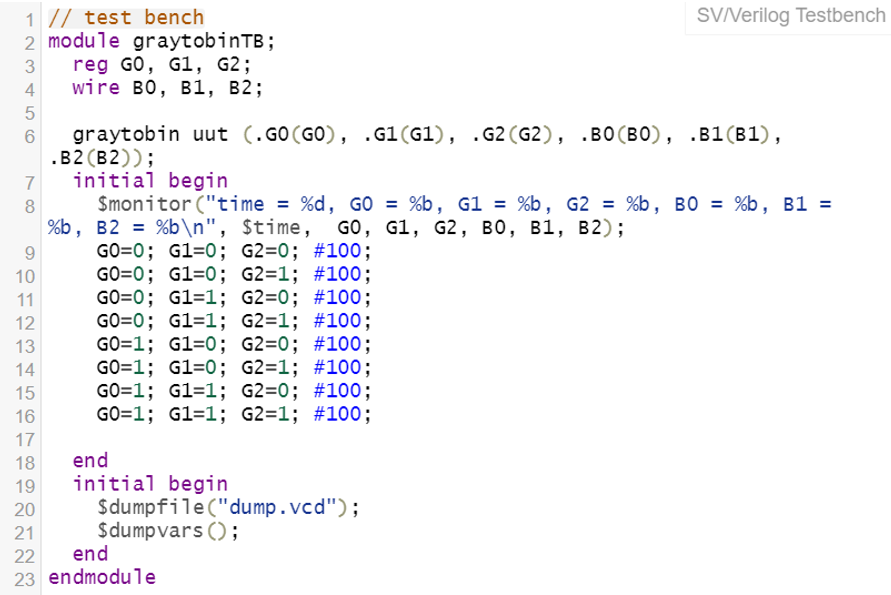
output wire B0, B1, B2;

assign B2=G2;

assign B1=G2^G1;

assign B0=G1^G0;

endmodule

****

A screenshot of a computer

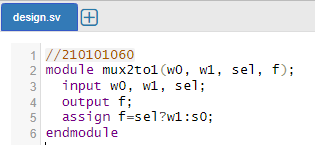
Description automatically generated

**DESIGN OF MULTIPLEXER CIRCUIT**

**// Verilog code for 2 to 1 multiplexer using conditional operator**

1. Design a 2 to 1 multiplexer using basic gates. Write the Verilog code using conditional operator, synthesize and simulate using Edaplay ground.

|  |  |
| --- | --- |
| Diagram  Description automatically generated | Text, letter  Description automatically generated |

****

**Diagram

Description automatically generatedGraphical user interface, text

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

1. **Write the Verilog code using conditional operator, synthesize and simulate using Edaplay ground.**

**Done above.**

1. **TO 1 MULTIPLEXER**:



1. Design a 4 to 1 multiplexer using basic gates. Write the Verilog code using conditional operator, synthesize and simulate using Edaplay ground.

|  |  |
| --- | --- |
| Diagram  Description automatically generated | Diagram  Description automatically generated |

**// Verilog code for 4 to 1 multiplexer using conditional operator**

**Graphical user interface, text, application

Description automatically generated**

Diagram

Description automatically generated

Text

Description automatically generated

Graphical user interface, website

Description automatically generated

**// Verilog code for 4 to 1 multiplexer using 2 to 1 multiplexer**

1. Design a 4 to 1 multiplexer using 2 to 1 multiplexer. Write the Verilog code using structural modelling, synthesize and simulate using Edaplay ground.

For example, the 4-to-1 multiplexer can be built using three 2-to-1 multiplexers as illustrated in Figure 4.3.

|  |  |
| --- | --- |
| Diagram  Description automatically generated  **Graphical symbol & Truth table of 4:1 multiplexer** | A picture containing text, clock  Description automatically generated  **Figure 4.3 Using 2-to-1 multiplexers to build a 4-to-1 multiplexer.** |

**Text

Description automatically generated**

Text

Description automatically generatedGraphical user interface

Description automatically generated

**// Verilog code for 2 to 1 multiplexer using if-else statement**

1. Design a 2 to 1 multiplexer using basic gates. Write the Verilog code using if else statement, synthesize and simulate using Edaplay ground.

**Text

Description automatically generated with medium confidence**

Diagram

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

**SYNTHESIZE & SIMULATION OF FLIP FLOPS IN XILINX ENVIRONMENT.**

1. Write Verilog code to implement using SR Flip Flop.

**SR Flip Flop**

Table

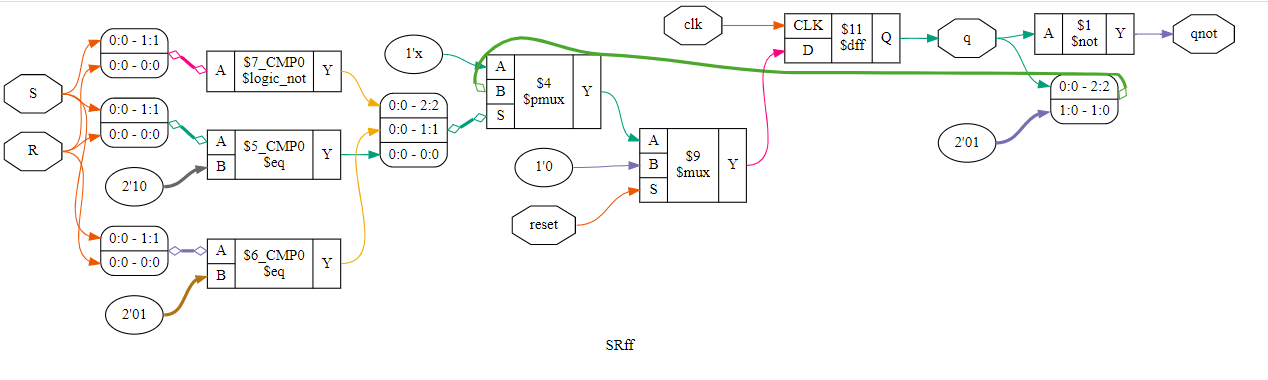
Description automatically generatedDiagram

Description automatically generated

**// Verilog code for SR Flip flop using Behavioral description**

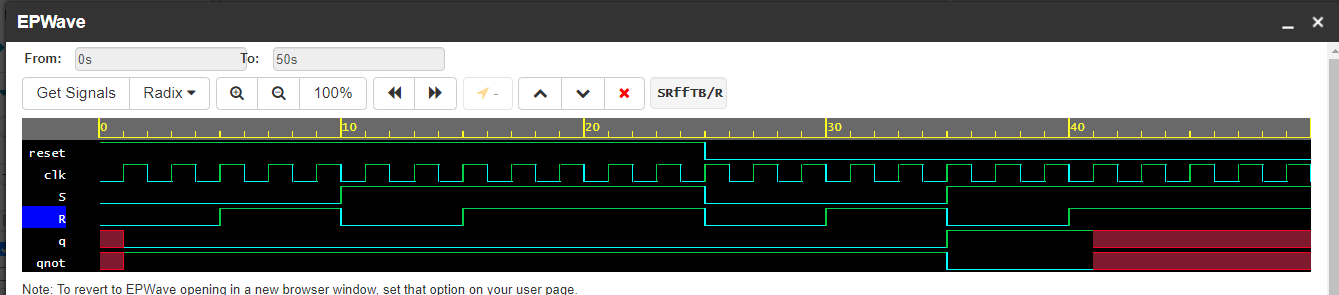
Graphical user interface, text, application

Description automatically generated



Graphical user interface, text, application

Description automatically generated



.**JK Flip Flop**

1. Write Verilog code to implement using JK Flip Flop.

Diagram, schematic

Description automatically generated

Table

Description automatically generated **//Verilog code for JK Flip flop using Behavioral description**

Graphical user interface, text, application

Description automatically generated

Diagram

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

**D Flip flop**

1. Write Verilog code to implement using D Flip Flop.

|  |  |
| --- | --- |
| [Image result for d ff using jk ff](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwiR9O2CuJbdAhUCNhoKHR3mBUAQjRx6BAgBEAU&url=https://electrosome.com/sr-d-t-flip-flop-using-jk/&psig=AOvVaw2UoSCjRqEd-W1-AvfhFf89&ust=1535775583601747)**Circuit Diagram** | **Characteristic table**  Table  Description automatically generated with medium confidence  **Graphical symbol**  Diagram  Description automatically generated |

Text

Description automatically generated

Diagram

Description automatically generated

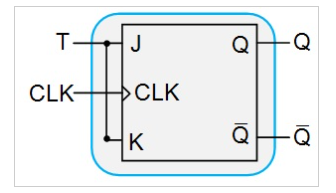
Graphical user interface, text, application, email

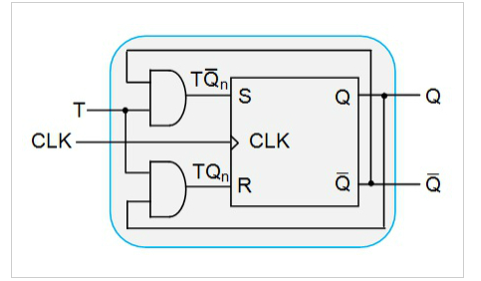
Description automatically generated

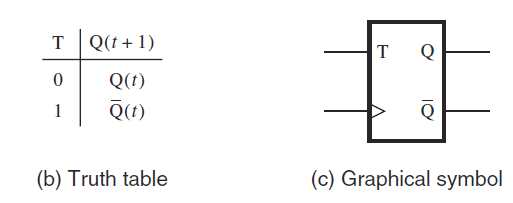
Graphical user interface, text, application

Description automatically generated

**T Flip flop**

1. Write Verilog code to implement using T Flip Flop.





Text

Description automatically generated

Diagram

Description automatically generated

Text

Description automatically generated