RN SHETTY TRUST®

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi
Approved by AICTE, New Delhi, Accredited by NAAC with 'A+' Grade
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph: (080) 28611880, 28611881 URL: www.rnsit.ac.in

Estd : 2001

Department of Computer Science & Engineering

Department of Computer Science & Engineering

RNSIT-CSE Bengaluru

LOGIC DESIGN & COMPUTER ORGANIZATION

LAB Work Book

(BCS302)

(As per Autonomous 2024 Scheme Course type- IPCC)

Compiled by

Department of Computer Science & Engineering

RNS Institute of Technology

Bengaluru-98

Name: _____

USN: _____

Semester: _____ **Section** _____

RN SHETTY TRUST®

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi
Approved by AICTE, New Delhi, Accredited by NAAC with 'A+' Grade
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph: (080) 28611880, 28611881 URL: www.rnsit.ac.in

Department of Computer Science & Engineering

VISION AND MISSION OF INSTITUTION

Vision

Building RNSIT into a World Class Institution

Mission

To impart high quality education in Engineering, Technology and Management with a Difference, Enabling Students to Excel in their Career by

1. Attracting quality Students and preparing them with a strong foundation in fundamentals so as to achieve distinctions in various walks of life leading to outstanding contributions
2. Imparting value based, need based, choice based and skill based professional education to the aspiring youth and carving them into disciplined, World class Professionals with social responsibility
3. Promoting excellence in Teaching, Research and Consultancy that galvanizes academic consciousness among Faculty and Students
4. Exposing Students to emerging frontiers of knowledge in various domains and make them suitable for Industry, Entrepreneurship, Higher studies, and Research & Development
5. Providing freedom of action and choice for all the Stake holders with better visibility

VISION AND MISSION OF CSE DEPARTMENT

Vision

Preparing better computer professionals for a real world

Mission

The Department of Computer Science and Engineering will make every effort to promote an intellectual and an ethical environment in which the strengths and skills of Computer Professionals will flourish by

1. Imparting Solid foundations and Applied aspects in both Computer Science Theory and Programming practices
2. Providing Training and encouraging R&D and Consultancy Services in frontier areas of Computer Science with a Global outlook
3. Fostering the highest ideals of Ethics, Values and creating Awareness on the role of Computing in Global Environment
4. Educating and preparing the graduates, highly Sought-after, Productive, and Well-respected for their work culture
5. Supporting and inducing Lifelong Learning practice

ACKNOWLEDGMENT

A material of this scope would not have been possible without the contribution of many people. We express our sincere gratitude to Mr. Satish R Shetty, Chairman and Mr. Karan R Shetty, CEO, RNS Group of Companies for providing magnanimous support in all our endeavours.

We are grateful to Dr. M K Venkatesha, Director and Dr. Ramesh Babu H S, Principal, Dr. Kavitha C, Dean and HoD, Dept. of CSE, RNSIT for extending their constant encouragement and support.

Our heartfelt thanks to Dr. Prasanna Kumar, Dr. Mallikarjun H M, Mr. Chetan Ghatage, Ms. Shyla N, Ms. Savitha T, Ms. Jeevitha for their unparalleled contribution throughout the preparation of this comprehensive manual. We also acknowledge our colleagues for their timely suggestions and unconditional support.

Department of CSE

Disclaimer

Estd : 2001

Edition: 2025- 26

Authors:	1. Dr. Prasanna Kumar M 2. Dr. Mallikarjun H M 3. Dr. Devaraju B M 4. Mr. Chetan Ghatage 5. Ms. Shyla N 6. Ms. Savitha T 7. Ms. Jeevitha M
Approved by:	Dr. Kavitha C, HoD, CSE Dr. Girijamma H A, Associate HoD, CSE Dr. M J Sudhamani, Associate HoD, CSE Dr. Sampada K S, Associate HoD, CSE
Department:	**Department of Computer Science & Engineering**

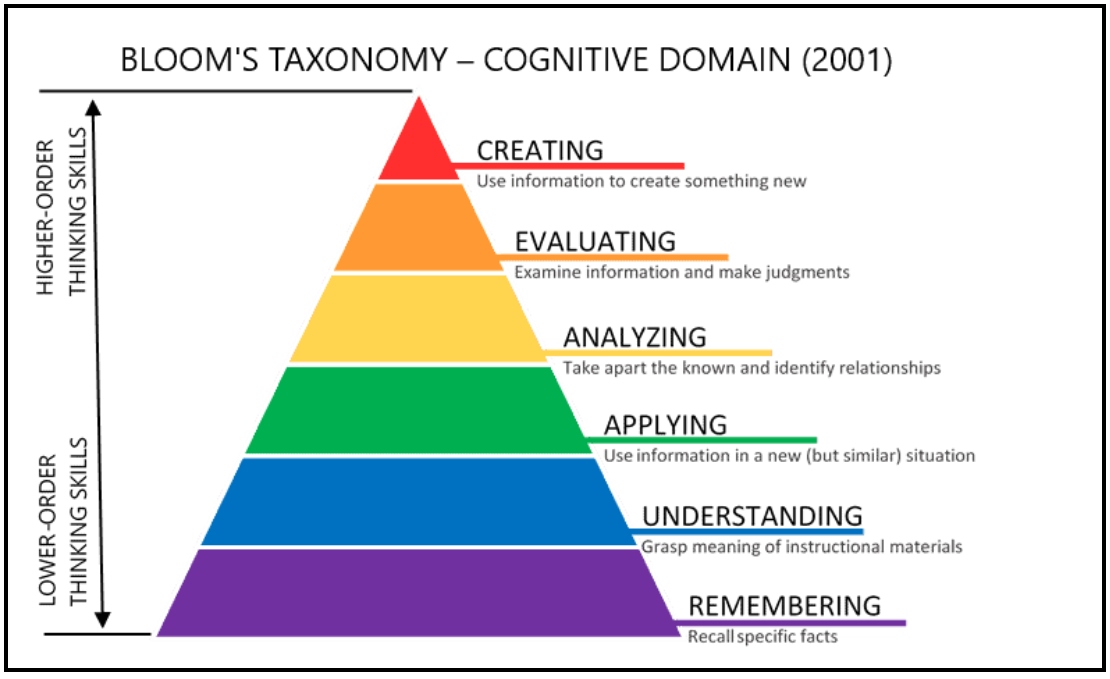
COURSE OUTCOMES

Course Outcomes: At the end of this course, students are able to:
CO1: Apply the K–Map techniques to simplify various Boolean expressions.
CO2: Design different types of combinational and sequential circuits along with Verilog programs.
CO3: Describe the fundamentals of machine instructions, addressing modes and Processor performance.
CO4: Explain the approaches involved in achieving communication between processor and I/O devices.

	WK	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PSO1	PSO2	PSO3
CO1	WK1	3	2	2	1	0							3	0	0
CO2	WK2, WK4	3	2	2	2	1							3	3	1
CO3	WK1, WK3	2	3	1	2	0							3	1	0
CO4	WK2, WK5	2	2	1	3	0							3	2	0

CO – Course Outcome, PO – Program Outcomes, WK – Knowledge Categories

REVISED BLOOMS TAXONOMY (RBT)



Mapping of 'Graduate Attributes' (GAs) and 'Program Outcomes' (POs)

Graduate Attributes (GAs) (As per Washington Accord Accreditation)	Program Outcomes (POs) (As per NBA New Delhi)
Engineering Knowledge	Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems
Problem Analysis	Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
Design/Development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate considerations for the public health and safety and the cultural, societal and environmental consideration.
Conduct Investigation of complex problems	Use research – based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.
Modern Tool Usage	Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
The engineer and society	Apply reasoning informed by the contextual knowledge to assess society, health, safety, legal and cultural issues and the consequential responsibilities relevant to the professional engineering practice.
Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental context and demonstrate the knowledge of and need for sustainable development.
Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
Individual and team work	Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings.
Communication	Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.
Project management & finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to ones won work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
Life Long Learning	Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

DESIGN & COMPUTER ORGANIZATION LABORATORY

INTERNAL EVALATION SHEET

EVALUATION (MAX MARKS 20)			
REGULAR EVALUATION A	**RECORD B**	**TEST C**	**TOTAL MARKS A+B+C**
09	**03**	**08**	**20**

A1: REGULAR LAB EVALUATION WRITE UP RUBRIC (MAX MARKS 03)	
Sl. No	**Parameters**
a	**Understanding of problem** statement while designing and implementing the program
b	**Writing program**- Program handles all possible conditions (1 marks)
c	**Result and documentation** (1 marks)

A2: REGULAR LAB EVALUATION VIVA RUBRIC (MAX MARKS 02)	
Sl.No.	**Parameter**
a	**Conceptual understanding of program-** Answers 80% of the viva questions asked (2

A3: REGULAR LAB PROGRAM EXECUTION RUBRIC (MAX MARKS 04)	
Sl.No	**Parameters**
a	**Design, implementation and demonstration -** Program follows syntax and semantics of Verilog programming language. Demonstrates the complete knowledge of the program written (2 marks)
b	**Result and documentation** All expected results are demonstrated successful, all errors are debugged with own practical knowledge and clear documentation according to the guidelines (2 marks)

B1: RECORD EVALUATION RUBRIC (MAX MARKS 03)	
Sl. No.	**Parameter**
a	**Documentation** Meticulous record writing including program, comments and expected output as per the guidelines mentioned (03 marks)

TEST /LAB INTERNALS MARKS (MAX MARKS 10)

TEST #	Write up 30	Execution 50	Viva 20	Total 100	Avg. 100	Final 08
TEST-1						
TEST-2						

FINAL MARKS OBTAINED			
A: REGULAR EVALUATION (09)		TOTAL (A+B+C)	
B: RECORD (03)			
C: TEST (08)		/20	Signature of Lab In Charge:

REGULAR LAB EVALUATION (MAX MARKS 5)

Program #	Date of Execution	Lab programs	A1 Write up (3)	A2 Viva (2)	A3 Exen. (4)	Total 9	Teacher Signature
1		Realization of Logic gates					
2		4-Variable logic expression					
3		Simple circuits in all models					
4		Half and Full Adder					
5		Half and Full Subtractor					
6		4 bit Binary Full adder Subtractor					
7		Multiplexer					
8		De-Multiplexer					
9		Flip-Flops					
10		4 bit Ripple counter					
Avg Marks							**9**

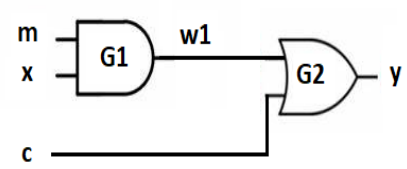
Assessment Type	Max Marks	Minimum Passing Marks	Evaluation Details
CIE Practical	12	5	Marks awarded for Weekly Conduction of Experiments and Submission of Laboratory records, scaled to 12 marks.
CIE Practical Test	8	3	Average of marks of Two tests, each conducted for 100 marks covering all experiments, scaled to 8.
Total CIE Practical (C)	**20**	**8**	**Marks of Experiments, Record and Test awarded for a maximum of 20 marks.**

PROGRAM LIST

Sl. No.	EXPERIMENTS
	Example Programs
	PART A
1	Realization of Logic gates AND, OR, NOT, NAND, NOR, XOR and XNOR using Verilog HDL
2	Given a 4-Variable logic expression, simplify it using appropriate technique and simulate the same using Basic gates in Verilog HDL.
3	Design Verilog HDL to implement simple circuits using Structural, Data flow and Behavioral model.
4	Design Verilog HDL to implement Half and Full Adder.
5	Design Verilog HDL to implement Half and Full Subtractor.
	PART B
6	Design Verilog HDL to implement 4 bit Binary Full adder, a 4 bit Binary Subtractor and simulate the same using basic gates.
7	Design Verilog HDL to implement Different types of Multiplexer - 2:1, 4:1 and 8:1.
8	Design Verilog HDL to implement Different types of De-Multiplexer- 1:2, 1:4 and 1:8
9	Design Verilog HDL for implementing various types of Flip-Flops such as JK and D
10	Design a 4 bit Ripple counter and implement with Verilog HDL
	Demo Experiment
1	Verilog implementation of AND, OR, and NOT logic gates using a MacCulloch-Pitts Perceptron model (single-layer Feedforward Neural Network).

Example Program i) Write a Verilog HDL code to realize y=m x + c

1. Structural




```
module equation_struct (
   input  m, x, c,   // inputs
   output y          // output
);
   wire w1;          // internal connection (net)
                     // Gate-level instantiations
   and G1 (w1, m, x);    // w1 = m AND x
   or  G2 (y,  w1, c);   // y  = w1 OR c
endmodule
```

2. Data Flow

```
module equation_df (
   input  m, x, c,   // inputs
   output y          // output
);
   // Dataflow modeling using continuous assignment
   assign y = (m & x) | c;  // y = (m AND x) OR c
endmodule
```

3. Behavioural

```
module equation_bh(
        input m,x,c,
        output y);
reg y;
always@(m,x,c)
begin
        y=(m & x )| c;
end
endmodule
```

Truth Table				
Inputs			**wire**	**Output**
m	**x**	**c**	**w1= (m & x)**	**y = (m & x) | c**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

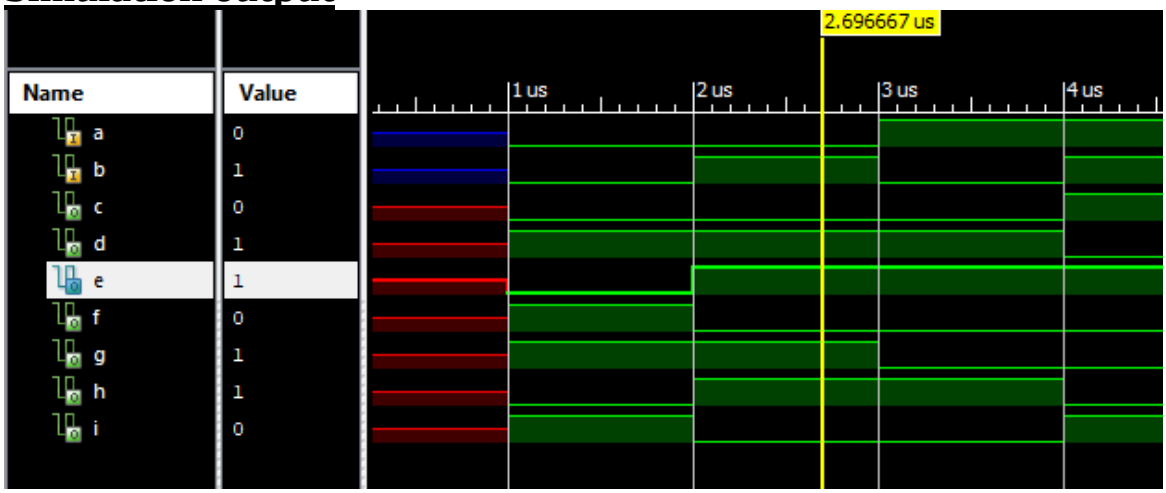
Example Program ii) Write a Verilog HDL code to realize all the logic gates.

Verilog code

```
module allgates (
   input  a, b,                  // two inputs
   output c, d, e, f, g, h, i    // outputs for each gate
);
assign c=a&b;
assign d=~(a&b);
assign e=a|b;
assign f=~(a|b);
assign g=~a;
assign h=a^b;
assign i=~(a^b);
endmodule
```

TRUTH TABLE								
INPUT		**OUTPUT**						
a	b	c(and)	d(nand)	e(or)	f(nor)	g(not) for i/p (a)	h(xor)	i(x-nor)
0	0	0	1	0	1	1	0	1
0	1	0	0	1	0	1	1	0
1	0	0	0	1	0	0	1	0
1	1	1	0	1	0	0	0	1

Simulation output



Signature of the Lab Incharge

PART A

Program 1:

Realization of Logic gates AND, OR, NOT, NAND, NOR, XOR and XNOR using Verilog HDL

1a) Structural Modeling

Structural modeling uses gate-level primitives (and, or, not, etc.) to connect components like a circuit schematic.

```
// Structural modeling of logic gates
module gates_structural (
        input A, B,
        output AND_out, OR_out, NOT_out, NAND_out, NOR_out, XOR_out, XNOR_out
);
        and (AND_out,  A, B);           // AND gate
        or (OR_out,  A, B);             // OR gate
        not (NOT_out, A);               // NOT gate
        nand (NAND_out, A, B);          // NAND gate
        nor (NOR_out,  A, B);           // NOR gate
        xor (XOR_out,  A, B);           // XOR gate
        xnor (XNOR_out, A, B);          // XNOR gate
endmodule
```

1b) Dataflow Modeling

Dataflow modeling uses continuous assignments (assign) with operators.

```
// Dataflow modeling of logic gates
module gates_dataflow (
        input A, B,
        output AND_out, OR_out, NOT_out, NAND_out, NOR_out, XOR_out, XNOR_out
);
        assign AND_out  = A & B;
        assign OR_out   = A | B;
        assign NOT_out  = ~A;
        assign NAND_out = ~(A & B);
        assign NOR_out  = ~(A | B);
        assign XOR_out  = A ^ B;
        assign XNOR_out = ~(A ^ B);
endmodule
```

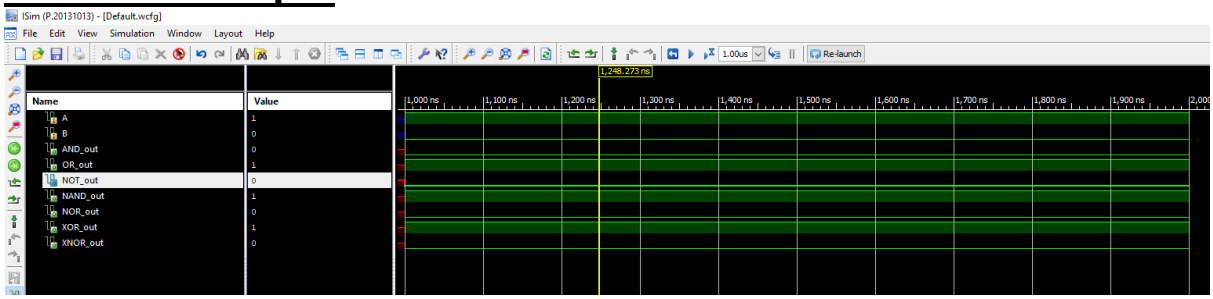
1c) Behavioral Modeling

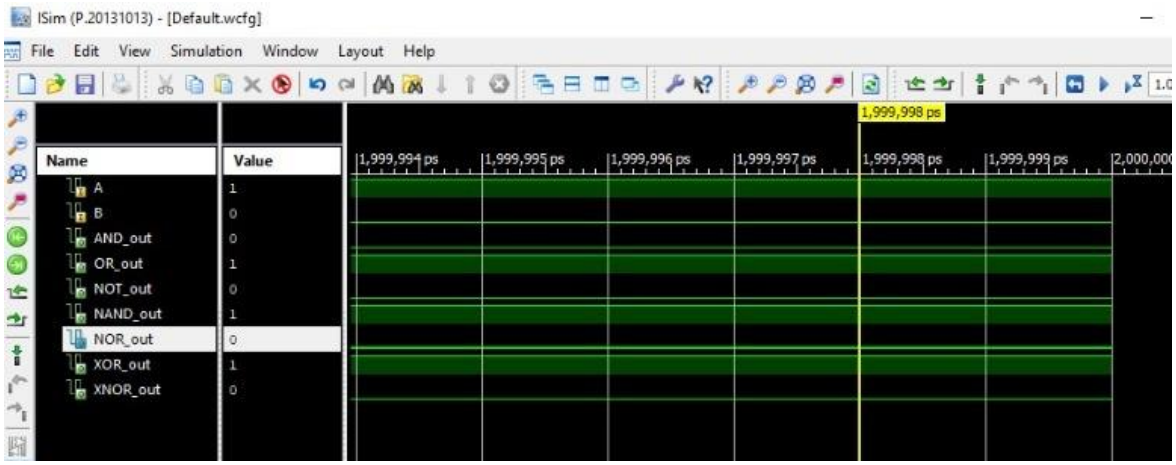
Behavioral modeling uses procedural blocks (always block with if/else or case).

```
// Behavioral modeling of logic gates
module gates_behavioral (
    input A, B,
    output reg AND_out,
    output reg OR_out,
    output reg NOT_out,
    output reg NAND_out,
    output reg NOR_out,
    output reg XOR_out,
    output reg XNOR_out
);

always @(A, B)
begin
    AND_out  = A & B;
    OR_out   = A | B;
    NOT_out  = ~A;
    NAND_out = ~(A & B);
    NOR_out  = ~(A | B);
    XOR_out  = A ^ B;
    XNOR_out = ~(A ^ B);
end

endmodule
```

TRUTH TABLE								
A	**B**	**AND (A·B)**	**OR (A+B)**	**NOT (~A)**	**NAND (~A·B)**	**NOR (~A+B)**	**XOR (A⊕B)**	**XNOR (~A⊕B)**
0	0	0	0	1	1	1	0	1
0	1	0	1	1	1	0	1	0
1	0	0	1	0	1	0	1	0
1	1	1	1	0	0	0	0	1

Simulation output





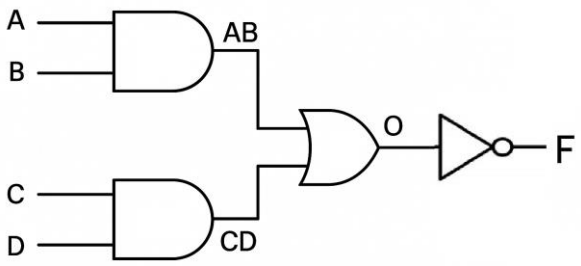
Viva Questions: Experiment 1		
Q. No.	**Question**	**Bloom's Level**
1	What are the truth tables for NAND and NOR gates?	Remembering
2	Why are NAND and NOR called universal gates?	Understanding
3	Write a simple Verilog code snippet for a 2-input XOR gate.	Applying
4	Compare the outputs of XOR and XNOR gates with an example.	Analyzing
5	Design a Verilog module that implements all 7 basic logic gates with a single input pair.	Creating

Draw Waveforms (for Different Combinations)

Signature of the Lab Incharge

Program 2a: Given a 4-variable logic expression, simplify it using appropriate using basic gates.

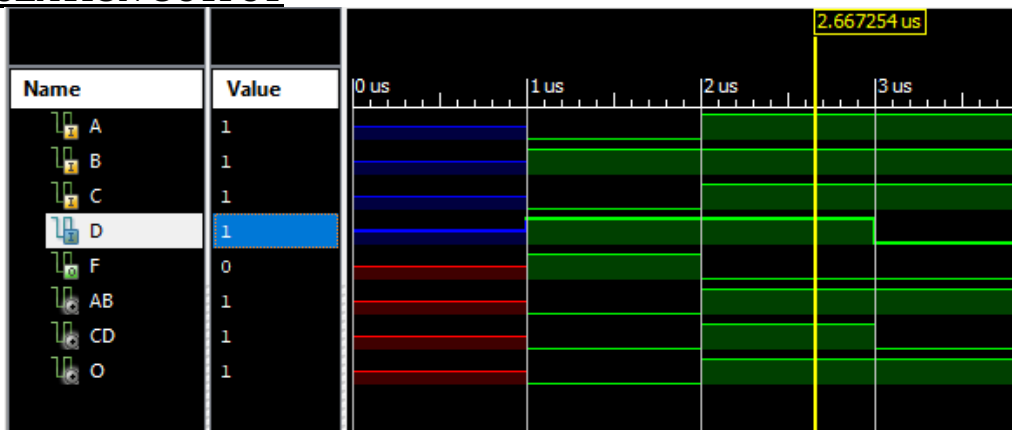
```
module 4variable (F, A, B, C, D);
        input A, B, C, D;
        output F;
        wire AB, CD, O;

        assign AB = A & B;
        assign CD = C & D;
        assign O = AB | CD;
        assign F = ~O;
endmodule
```



TRUTH TABLE

INPUT							OUTPUT
A	**B**	**C**	**D**	**AB**	**CD**	**O=AB|CD**	**F=~O**
0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	0	1
0	1	1	0	0	0	0	1
0	1	1	1	0	1	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	1
1	0	1	0	0	0	0	1
1	0	1	1	0	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	0
1	1	1	0	1	0	1	0
1	1	1	1	1	1	1	0

SIMULATION OUTPUT



Program 2b: Given a 4-variable logic expression, simplify it and write Verilog program.

f(A,B,C,D)=∑m(2,3,6,7,8,10,13,15)

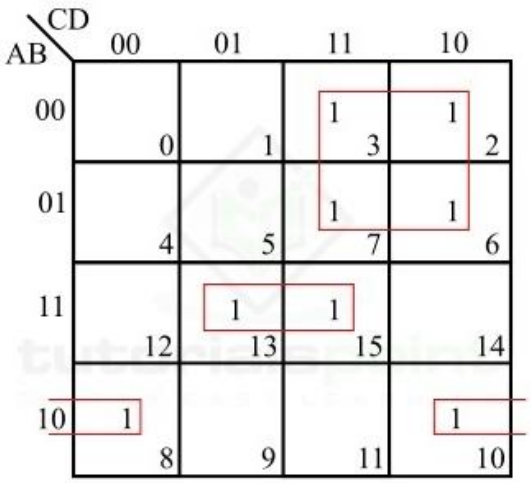


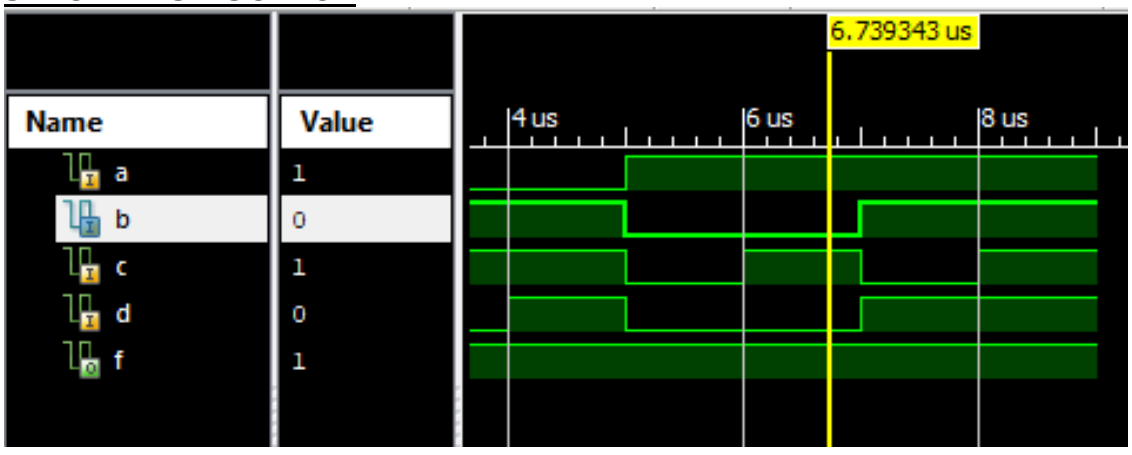
Figure 2 - SOP K-Map

TRUTH TABLE					
Decimal	**A**	**B**	**C**	**D**	**f**
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

f(A,B,C,D)= A'C +AB'D' +ABD

```
module expression(
        input a,b,c,d,
        output f
 );
assign f=(~a&c | a&~b&~d | a&b&d);
endmodule
```

SIMULATION OUTPUT



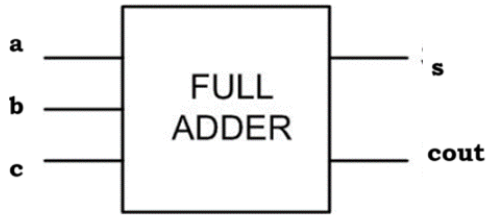
Viva Questions: Experiment 2		
Q. No.	**Question**	**Bloom's Level**
1	What are the basic Boolean algebra laws used for simplification?	Remembering
2	Explain the difference between Boolean algebra simplification and K-map simplification.	Understanding
3	Simplify F(A,B,C,D) = A'B + AB' + CD using K-map.	Applying
4	Why do we prefer K-map over Boolean algebra for 4-variable expressions?	Analyzing
5	Develop a Verilog code for the minimized expression obtained from a given logic function.	Creating

Activity/ Workbook

Signature of the Lab Incharge

Experiment 3: Design Verilog HDL to implement simple circuits using Structural, Data flow and Behavioral model

Block Diagram:



Truth Table				
Inputs			Outputs	
a	b	c	s (Sum)	cout (Carry)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

K Map Simplification:

S

a \ bc	00	01	11	10
0		1		1
1	1		1	

$s = a \oplus b \oplus c$

cout

a \ bc	00	01	11	10
0			1	
1		1	1	1

cout= (a.b) + (b.c) + (c.a)

So, Equations

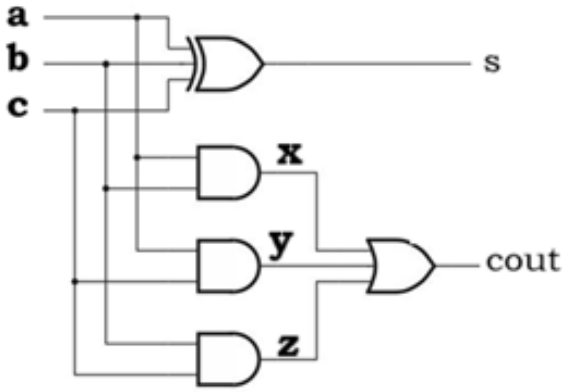
- Sum (s):

  $s = a \oplus b \oplus c$

- Carry-out (cout):

  cout= (a.b) + (b.c) + (c.a)

Circuit Diagram by using Logic Gates:



STRUCTURAL MODEL

```
module fadder_struct(a,b,c, s, cout);
input a,b,c;
output s,cout;
wire x,y,z;
        xor g1(s,a,b,c);                    //  s= a xor b  xor  c
        and g2(x,a,b);                      //  cout= ab + bc+ca
        and g3(y,b,c);
        and g4(z,a,c);
        or g5(cout,x,y,z);
endmodule
```

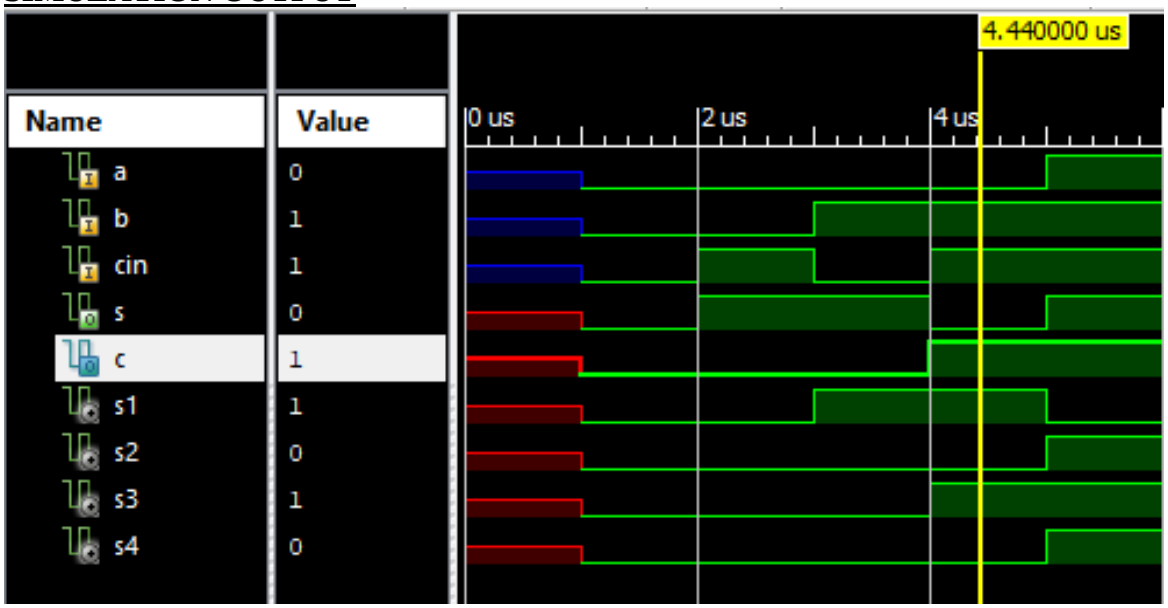
DATA FLOW MODEL

```
module fulladder_df(a,b,c, s,cout);
input a,b,c;
output s,cout;
        assign s = a ^ b ^ c;
        assign cout = (a & b) | (b & c) | (c & a);
endmodule
```

BEHAVIORAL MODULE

```
module fulladder_bh(a,b,c, s,cout);
        input a,b,c;
        output s,cout;
        reg s,cout;
        always @ (a,b,c)
        begin
                s= a ^ b ^ c;
                cout = (a & b) | (b & c) | (c & a);
        end
endmodule
```

SIMULATION OUTPUT



Viva Questions: Experiment 3		
Q. No.	**Question**	**Bloom's Level**
1	Define structural, dataflow, and behavioral modeling in Verilog.	Remembering
2	Differentiate between dataflow and behavioral modeling with examples.	Understanding
3	Write Verilog code for an AND gate in all three modeling styles.	Applying
4	Which modeling style is most efficient for designing complex systems, and why?	Analyzing
5	Implement a 2-bit comparator using all three modeling techniques.	Creating

Activity/ Workbook

Signature of the Lab Incharge

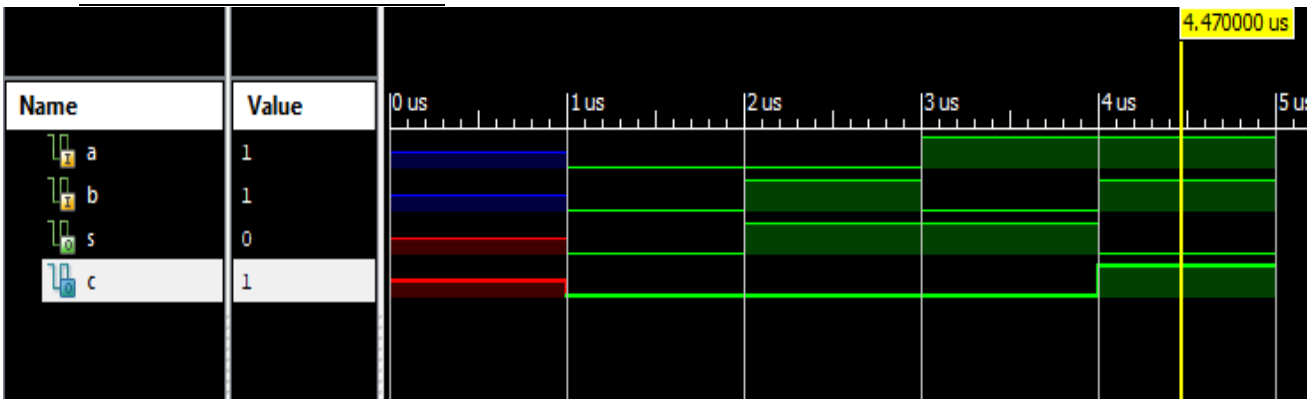
Program 4: Design Verilog HDL to implement Binary adder: Half and Full adder.

Half-Adder

```
module half_adder (input a,
        input b,
        output s,
        output c
);
        assign s = a ^ b;
         assign c = a & b;
endmodule
```

Truth Table			
INPUT		**OUTPUT**	
a	b	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

SIMULATION OUTPUT

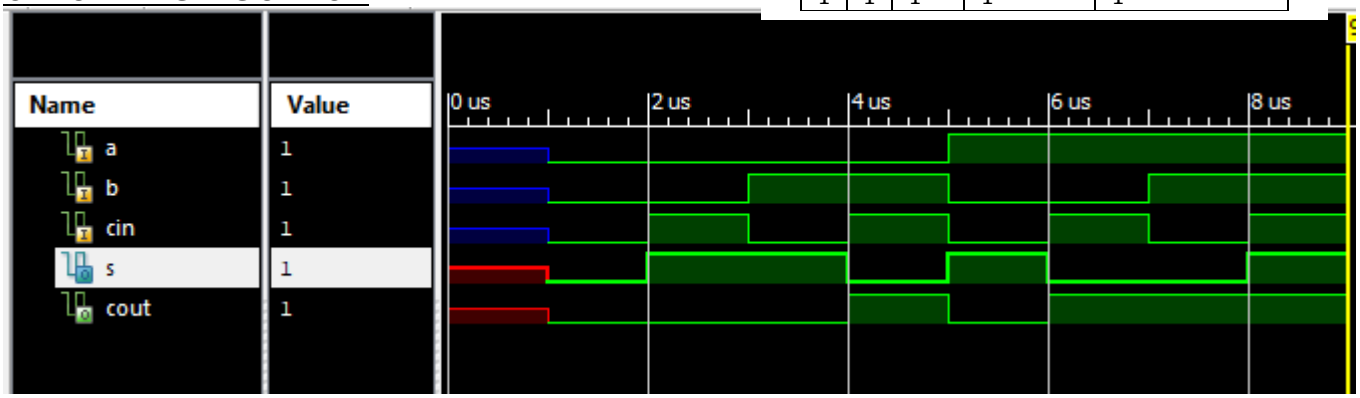


Full Adder

```
module fulladderd (
        input a,b,cin;
        output s,cout
);
assign s = a ^ b ^ cin;
assign cout = (a & b) | (b & cin) | (cin & a);
endmodule
```

Truth Table for Full Adder				
Inputs			**Outputs**	
a	**b**	**cin**	**s (Sum)**	**cout (Carry)**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

SIMULATION OUTPUT



Viva Questions: Experiment 4		
Q. No.	**Question**	**Bloom's Level**
1	State the logic expressions for half adder sum and carry.	Remembering
2	Why do we require a full adder if half adder exists?	Understanding
3	Implement a full adder using two half adders in Verilog.	Applying
4	Compare gate-level implementation of half adder and full adder.	Analyzing
5	Design an n-bit ripple carry adder using full adder modules.	Creating

Activity/ Workbook

Signature of the Lab Incharge

Program 5: Design Verilog HDL to implement Binary subtractor: Half and Full Subtractor.

Half-Subtractor

```
module half_subtractor (
        input a,
        input b,
        output diff,
        output borrow
);

        assign diff = a ^ b;
        assign borrow = ~a & b;

endmodule
```

Truth Table for Half-Subtractor			
Inputs		**Outputs**	
a	**b**	**diff (Difference)**	**borrow**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

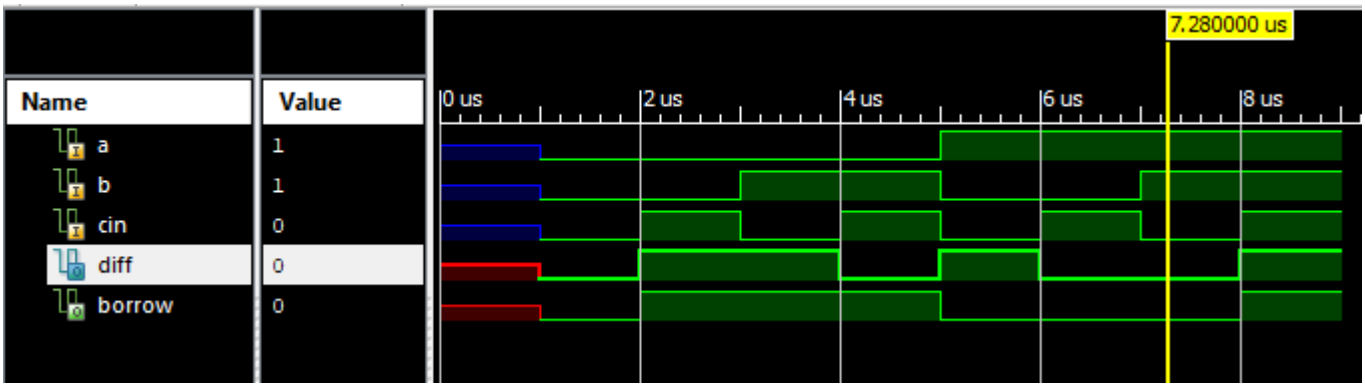
SIMULATION OUTPUT

Full subtractor

```
module fullsubtractor (a,b,c, diff,borrow);
        input a,b,c;
        output diff, borrow;
assign diff = a ^ b ^ c;
assign borrow = (~a & c) | (~a & b) | (b & c);
endmodule
```

Truth Table for fullsubtractor				
Inputs			**Outputs**	
a	**b**	**c (Bin)**	**diff (Difference)**	**borrow**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	1
1	1	1	1	1

Simulation output



Viva Questions: Experiment 5		
Q. No.	**Question**	**Bloom's Level**
1	Define the difference and borrow expressions for a half subtractor.	Remembering
2	How does a full subtractor differ from a full adder logically?	Understanding
3	Write Verilog code for a half subtractor.	Applying
4	Compare the borrow generation mechanism in half and full subtractor.	Analyzing
5	Design a 4-bit subtractor using full subtractor modules.	Creating

Activity/ Workbook

Signature of the Lab Incharge

PART B

Program 6: Design a 4-bit full adder and Subtractor and simulate the same using basic gates.

4-BIT FULL ADDER

```
module Adder(a,b,cin,sum,cout);
input [3:0]a,b;
input cin;
output  [3:0]sum;
output cout;
FullAdder FA1(a[0],b[0],cin,sum[0],cout1);
FullAdder FA2(a[1],b[1],cout1,sum[1],cout2);
FullAdder FA3(a[2],b[2],cout2,sum[2],cout3);
FullAdder FA4(a[3],b[3],cout3,sum[3],cout);
endmodule

module FullAdder (a,b,cin,s,cout);
input a,b,cin;
output s, cout;
assign s=a^b^cin;
assign cout = (a&b) | (b&cin) | (cin&a);
endmodule
```

TRUTH TABLE

a3	a2	a1	a0	b3	b2	b1	b0	cin	s3	s2	s1	s0	Cout	cout3	cout2	cout1
1	0	1	0	0	1	0	1	0	1	1	1	1	0	0	0	0
1	1	1	1	1	1	0	0	0	1	0	1	1	1	1	0	0
0	0	0	1	1	0	0	1	0	1	0	1	0	0	0	0	1
1	1	1	0	0	1	0	0	0	0	0	1	0	1	1	0	0
1	0	1	0	0	1	0	1	1	0	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	1	1	1	0	0	1	1	1	1
0	0	0	1	1	0	0	1	1	1	0	1	1	0	0	0	1
1	1	1	0	0	1	0	0	1	0	0	1	1	1	1	0	0

SIMULATION OUTPUT

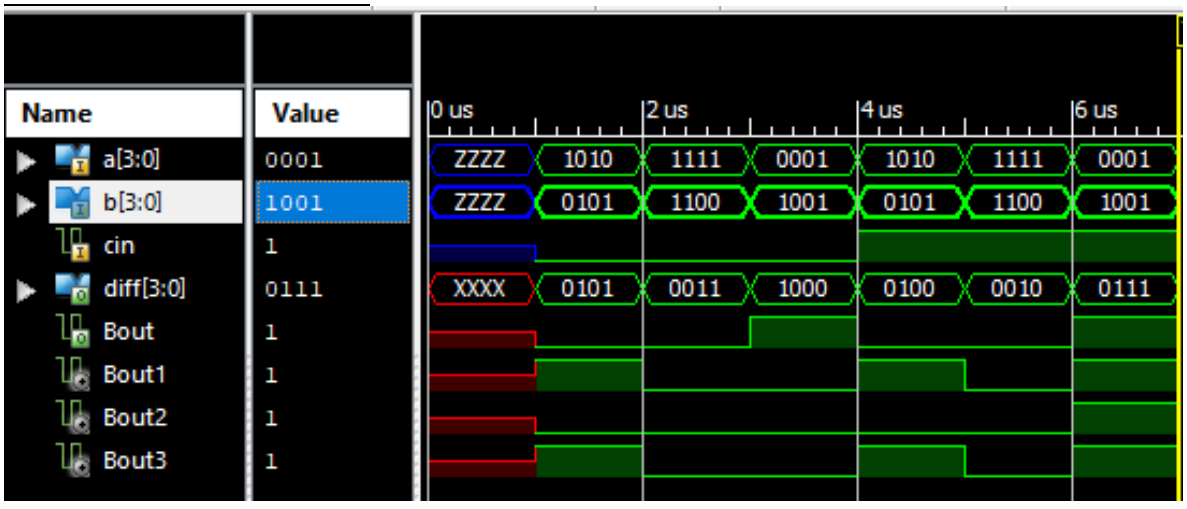
4-BIT FULL SUBTRACTOR

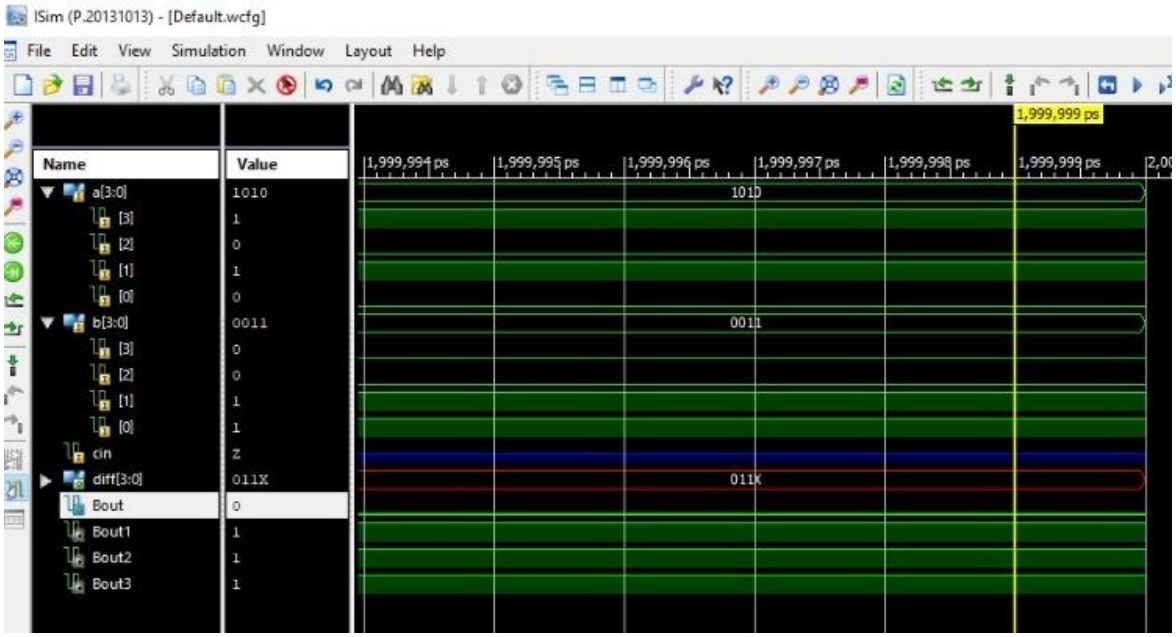
```
module subb(a,b,cin,diff,Bout);
input [3:0]a,b;
input cin;
output  [3:0]diff;
output Bout;
Fullsub FS1(a[0],b[0],cin,diff[0],Bout1);
Fullsub FS2(a[1],b[1],Bout1,diff[1],Bout2);
Fullsub FS3(a[2],b[2],Bout2,diff[2],Bout3);
Fullsub FS4(a[3],b[3],Bout3,diff[3],Bout);
endmodule

module Fullsub(a,b,cin, diff,Bout);
input a,b,cin;
output diff,Bout;
assign diff = a ^ b ^ cin;
assign Bout = (~a & cin) | (~a & b) | (b & cin);
endmodule
```

TRUTH TABLE

a3	a2	a1	a0	b3	b2	b1	b0	cin	diff3	diff2	diff1	diff0	Bout	Bout3	Bout2	Bout1
1	0	1	0	0	1	0	1	0	0	1	1	1	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	0	1	0	1	1	1	0	1	1	1	1
1	1	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0
1	0	1	0	0	1	0	1	1	0	1	1	0	0	0	0	1
1	1	1	1	1	1	0	0	1	0	0	1	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	1	0	0	1	0	0	1	0	0	1	1	0	0	0	1

SIMULATION OUTPUT





Viva Questions: Experiment 6		
Q. No.	**Question**	**Bloom's Level**
1	What is the role of carry in addition and borrow in subtraction?	Remembering
2	Explain how 2's complement is used in subtraction.	Understanding
3	Implement a 4-bit adder using full adder blocks in Verilog.	Applying
4	Compare the gate-level complexity of 4-bit adder vs 4-bit subtractor.	Analyzing
5	Design a single Verilog module that performs both addition and subtraction using a mode control input.	Creating

Activity/ Workbook

Signature of the Lab Incharge

Program 7: Design Verilog HDL to implement different types of multiplexers like 2:1, 4:1 and 8:1.

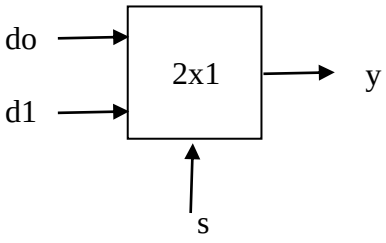
THEORY: A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. A multiplexer is also called as data selector, since it selects one of many inputs and steers the binary information to the output line. The selection of particular input line is controlled by a set of selection lines. Normally there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

Multiplexer (MUX) is a digital switch which connects data from one of n sources to the output. A number of select inputs determine which data source is connected to the output.

A 2 to 1 line multiplexer is shown in figure below, A & B are two inputs. S is Select line. Enable is used to enable or disable the mulitiplexer. Selection line S are decoded to select a particular input. Y is a output which selects any one of the input depending on the select lines. The truth table for the 2:1 mux is given in the table below.

a) 2 to 1 Multiplexer

y = s'd0+ sd1



2 to 1 Multiplexer (dataflow)

```
module m21_df(y, d0, d1, s);
output y;
input d0, d1, s;
assign y= (~s & d0 ) |  (s & d1);
endmodule
```

2 to 1 Multiplexer (behavioural)

```
module m21_bh(y, d0, d1, s);
output y;
input d0, d1, s;
reg y;
always@(s,d0,d1)
begin
        y= (~s & d0 ) |  (s & d1);
end
endmodule
```

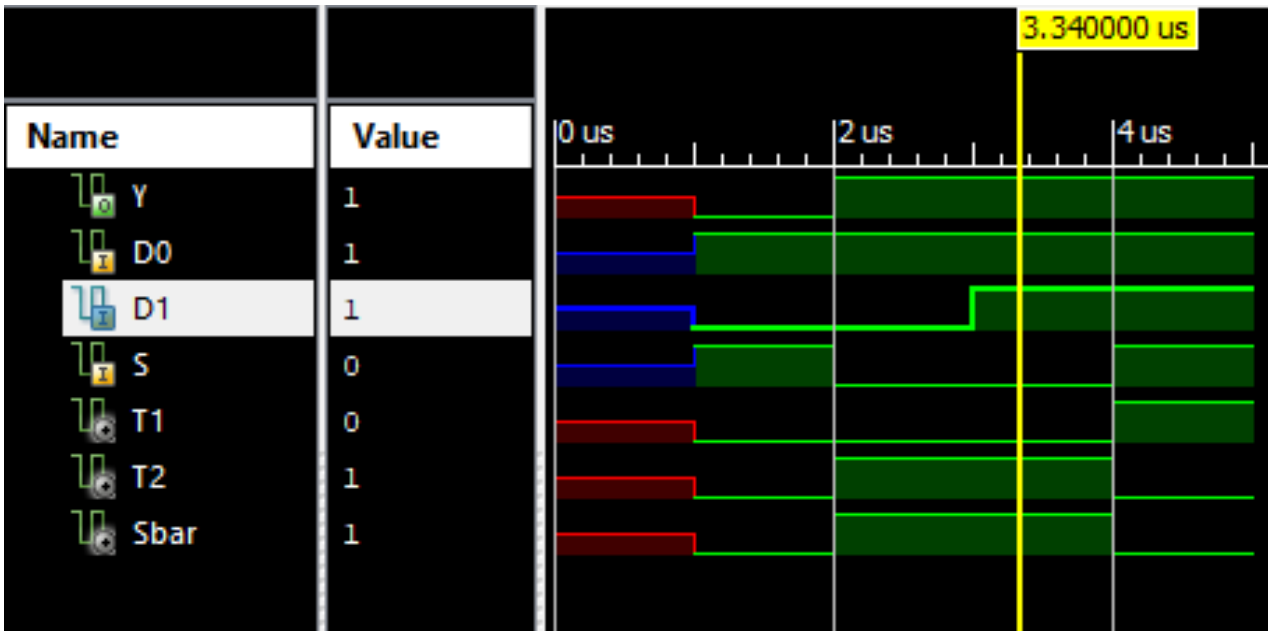
Truth Table for 2:1 Mux		
Input		Output
s	d	y
X	X	0
0	d0	1
1	d1	1

Truth Table for 4:1 Mux (In Brief)			
s (Select)	d0	d1	y (Output)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Equation:

$$y = s'd_0 + sd_1$$

SIMULATION OUTPUT



4 to 1 Multiplexer

```
module mux4to1_df(
   input [3:0] i,
   input [1:0] s,
   output y
   );
assign y=(~s[1]&~s[0]&i[0]  | ~s[1]& s[0]&i[1]  | s[1]&~s[0]&i[2]  | s[1]&s[0]&i[3]);
endmodule
```

TRUTH TABLE

Truth Table for 4:1 Mux			
I/P (Select lines)		**I/P**	**O/P**
S1	**S0**	**i3 i2 i1 i0**	**y**
0	0	0 0 0 0	0
0	0	0 0 0 1	1
0	1	0 0 1 0	1
1	0	0 1 0 0	1
1	1	1 0 0 0	1

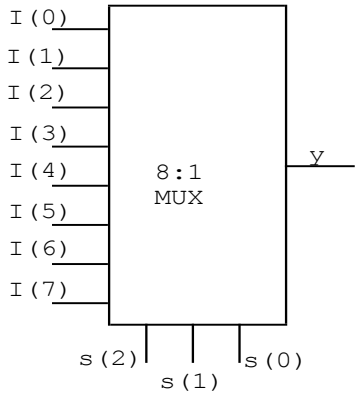
Equation:

$$y = s_1' s_0' i_0 + s_1' s_0 i_1 + s_1 s_0' i_2 + s_1 s_0 i_3$$

SIMULATION OUTPUT

c) 8 to 1 Multiplexer

Block diagram:



Truth Table for 8:1 Mux				
s2	s1	s0	Select Lines	y
0	0	0	i0	y = i0
0	0	1	i1	y = i1
0	1	0	i2	y = i2
0	1	1	i3	y = i3
1	0	0	i4	y = i4
1	0	1	i5	y = i5
1	1	0	i6	y = i6
1	1	1	i7	y = i7

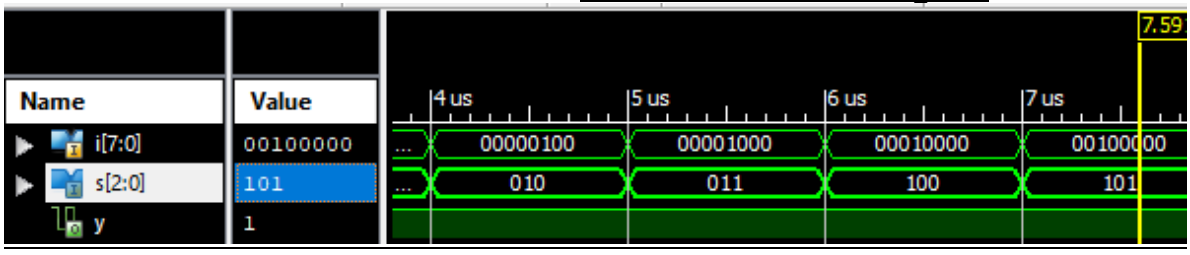
Equation:

$$y = s_2's_1's_0'i_0 + s_2's_1's_0i_1 + s_2's_1s_0'i_2 + s_2's_1s_0i_3 + s_2s_1's_0'i_4 + s_2s_1's_0i_5 + s_2s_1s_0'i_6 + s_2s_1s_0i_7$$

8:1 multiplexer behavioral Model

```
module mux_8*1(
        input [7:0] i,
        input [2:0] s,
        output y
);
reg y;
always @(i or s)
begin
        case (s)
        3'b000:y=i[0];
        3'b001:y=i[1];
        3'b010:y=i[2];
        3'b011:y=i[3];
        3'b100:y=i[4];
        3'b101:y=i[5];
        3'b110:y=i[6];
        3'b111:y=i[7];
        default: y=1'bx;
        end case
end
endmodule
```

Simulation output

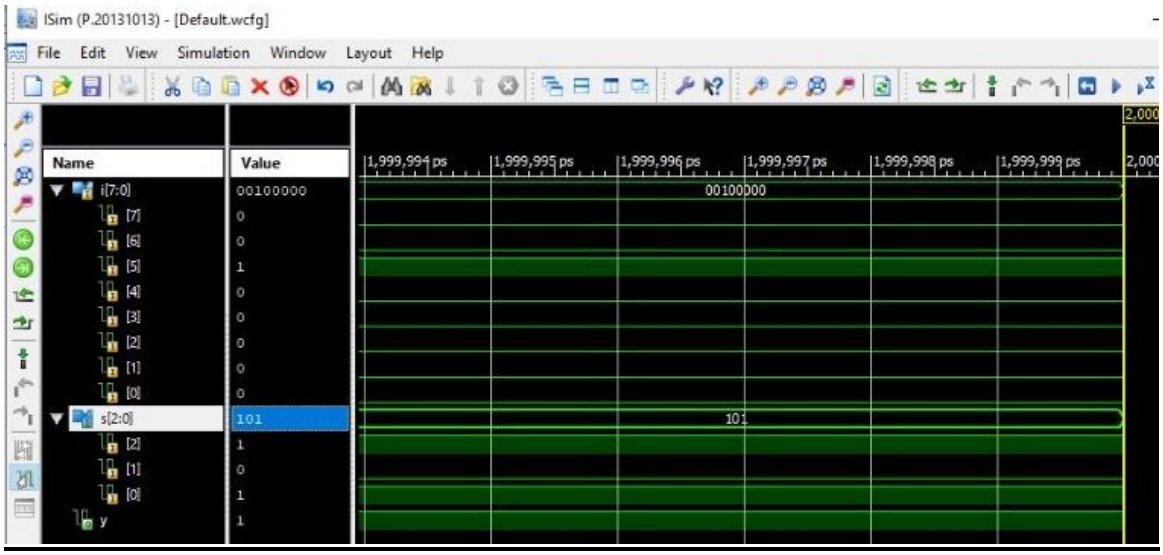


OR

8:1 multiplexer Data Flow Model

```
module mux8to1(
   input [7:0] i,
         input [2:0] s,
   output y
   );
assign y=(~s[2]&~s[1]&~s[0]&i[0] |~s[2]&~s[1]&s[0]&i[1]
                        |~s[2]&s[1]&~s[0]&i[2] |~s[2]&s[1]&s[0]&i[3]
                        |s[2]&~s[1]&~s[0]&i[4] | s[2]&~s[1]& s[0]&i[5]
                        |s[2]& s[1]&~s[0]&i[6] | s[2]& s[1]& s[0]&i[7]);
endmodule
```

SIMULATION OUTPUT



Viva Questions: Experiment 7		
Q. No.	**Question**	**Bloom's Level**
1	State the general equation for an n:1 multiplexer.	Remembering
2	Why are multiplexers called “data selectors”?	Understanding
3	Implement a 4:1 MUX using 2:1 MUXes in Verilog.	Applying
4	Compare structural and behavioral implementations of an 8:1 MUX.	Analyzing
5	Design a parameterized Verilog code for an N:1 MUX.	Creating

Activity/ Workbook

Signature of the Lab Incharge

Program 8: Design Verilog HDL to implement Different types of De-Multiplexer.

A De-Multiplexer (De-MUX) is a combinational logic circuit that takes a single input data line and routes it to exactly one of many output lines, based on the value of select lines. It performs the opposite operation of a multiplexer.

Inputs:

- Data input (D) – single bit to be distributed.
- Select lines (S) – binary value that chooses which output is active.

Outputs:

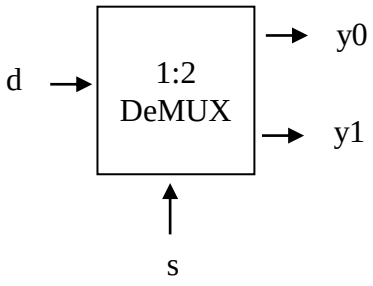
- Multiple output lines (Y0, Y1, …, Yn–1), where only one output is active at a time.

1:2 Demultiplexer

Inputs – Outputs

- Inputs: d (data), sel (select)
- Outputs: y0, y1

Block Diagram

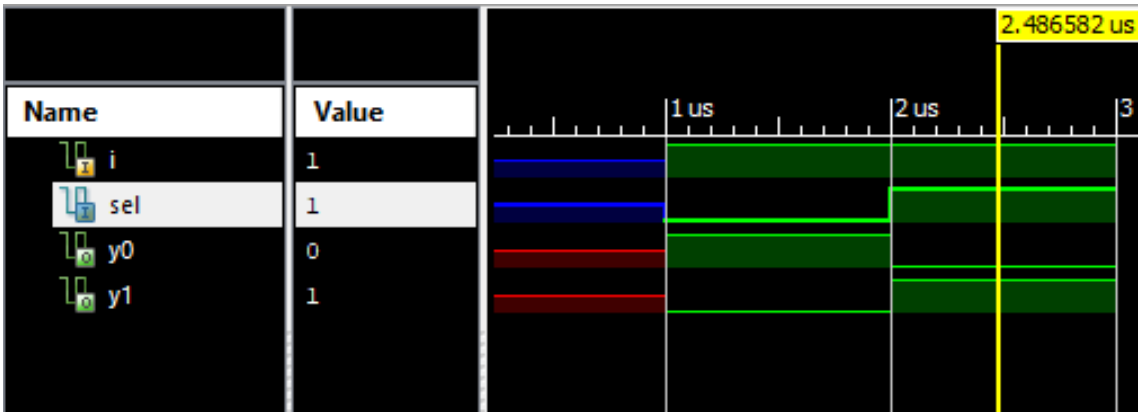


Equations
y0 = d . s'
y1 = d . s

Truth Tabl		
s	**y0**	**y1**
0	**d**	0
1	0	**d**

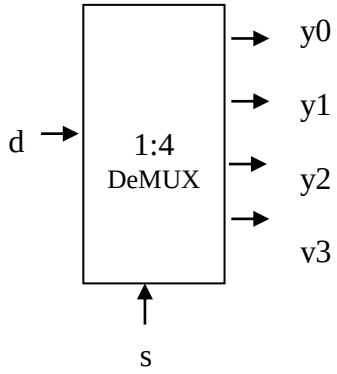
```
module demux1to2(
        input d, s;
        output y0, y1);
assign y0=(s)? 1'b0:d;
assign y1=(s)? d:1'b0;
endmodule
```

SIMULATION OUTPUT



1:4 Demultiplexer

Block Diagram

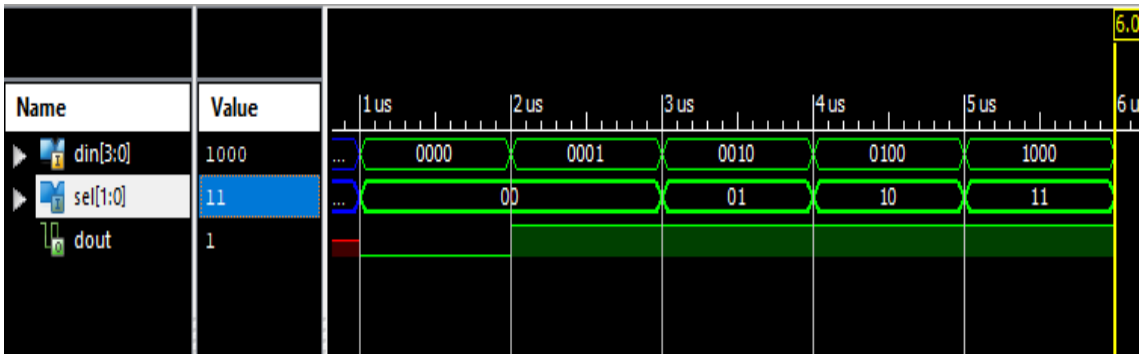


Equations
y0 = d . s1' . s0'
y1 = d . s1' . s0
y2 = d . s1 . s0'
y3 = d . s1 . s0

Truth Table					
s1	**s0**	**y0**	**y1**	**y2**	**y3**
0	0	**d**	0	0	0
0	1	0	**d**	0	0
1	0	0	0	**d**	0
1	1	0	0	0	**d**

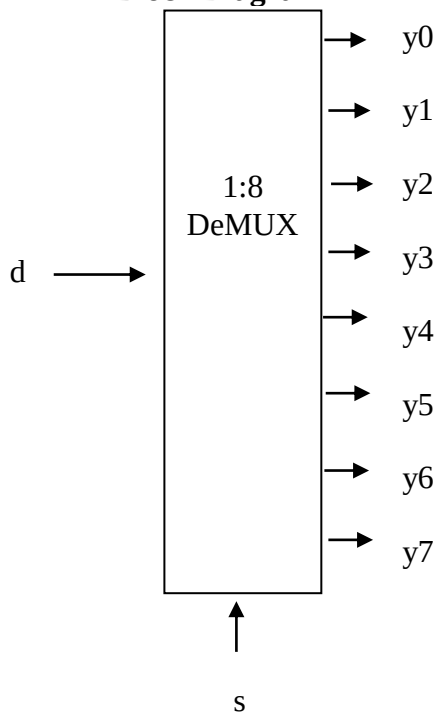
```
module demux1to4(Input d;
        input[1:0] s;
        output y0,y1,y2,y3
);
        assign y0=(s==2'b00)?d:1'b0;
        assign y1=(s==2'b01)?d:1'b0;
        assign y2=(s==2'b10)?d:1'b0;
        assign y3=(s==2'b11)?d:1'b0;
endmodule
```

SIMULATION OUTPUT



1:8 Demultiplexer

Block Diagram

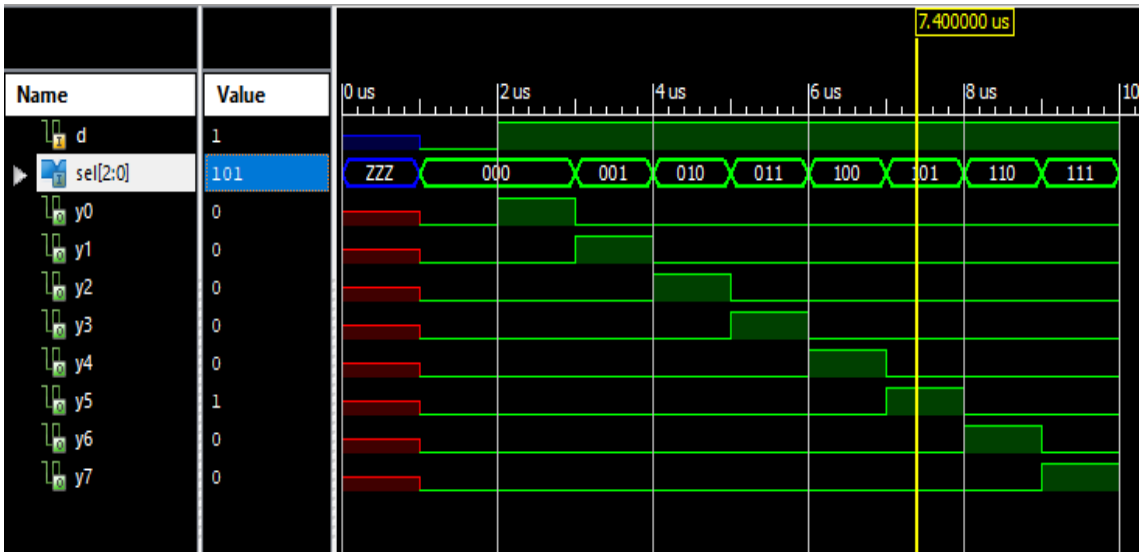


Equations
y0 = d . s2' . s1' · s0'
y1 = d . s2' .s1' . s0
y2 = d . s2' .s1 .s0'
y3 = d . s2'. s1 . s0
y4 = d . s2 . s1' . s0'
y5 = d . s2 . s1' . s0
y6 = d . s2 . s1 . s0'
y7 = d . s2 . s1 . s0

Truth Table for 1:8 De- MUX										
S2	**S1**	**S0**	**Y0**	**Y1**	**Y2**	**Y3**	**Y4**	**Y5**	**Y6**	**Y7**
0	0	0	**d**	0	0	0	0	0	0	0
0	0	1	0	**d**	0	0	0	0	0	0
0	1	0	0	0	**d**	0	0	0	0	0
0	1	1	0	0	0	**d**	0	0	0	0
1	0	0	0	0	0	0	**d**	0	0	0
1	0	1	0	0	0	0	0	**d**	0	0
1	1	0	0	0	0	0	0	0	**d**	0
1	1	1	0	0	0	0	0	0	0	**d**

```
module demux1to8(
        input d;
        input [2:0] s;
        output y0, y1,y2,y3,y4,y5,y6,y7
);
assign y0=(s==3'b000)?d:1'b0;
assign y1=(s==3'b001)?d:1'b0;
assign y2=(s==3'b010)?d:1'b0;
assign y3=(s==3'b011)?d:1'b0;
assign y4=(s==3'b100)?d:1'b0;
assign y5=(s==3'b101)?d:1'b0;
assign y6=(s==3'b110)?d:1'b0;
assign y7=(s==3'b111)?d:1'b0;
endmodule
```

SIMULATION OUTPUT



Viva Questions: Experiment 8		
Q. No.	**Question**	**Bloom's Level**
1	Define the truth table for a 1:4 demultiplexer.	Remembering
2	Differentiate between MUX and DEMUX applications.	Understanding
3	Write a Verilog program for a 1:8 DEMUX.	Applying
4	Explain how enable input affects DEMUX functionality.	Analyzing
5	Design a cascaded DEMUX system to implement a 1:16 DEMUX.	Creating

Activity/ Workbook

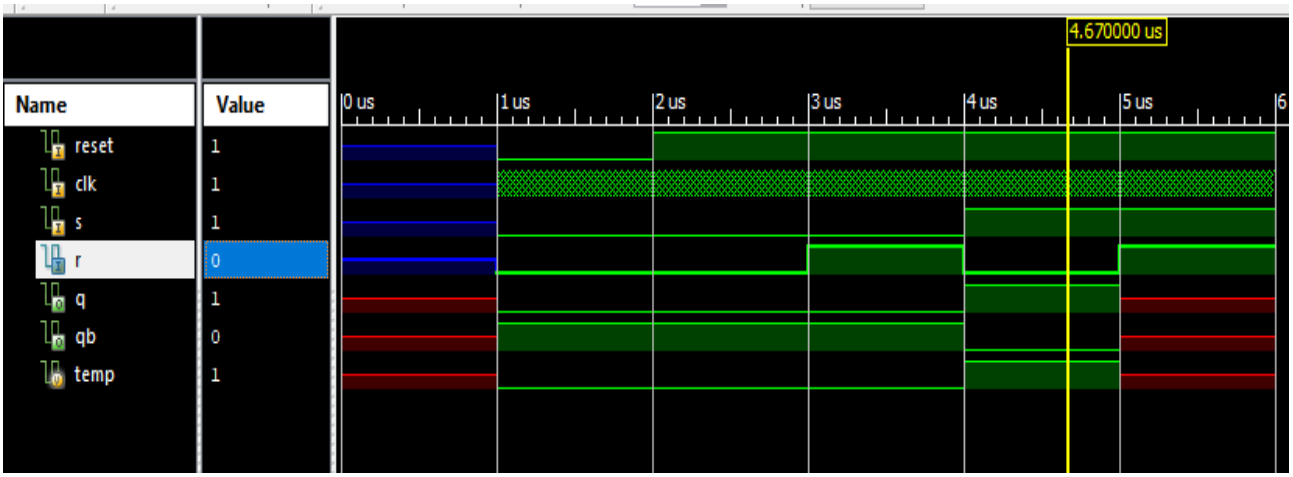
Signature of the Lab Incharge

Program 9: Design Verilog program for implementing various types of Flip-Flops such as SR, JK, and D Flip flop.

i) SR Flip-flop

```
module srff(reset, clk, s, r, q, qb);
input reset;
input clk;
input s;
input r;
output q;
output qb;
reg temp;
always@(negedge reset, posedge clk)
begin
        if(!reset)
                temp<=1'b0;
        else
        begin
                case({s,r})
                2'b01: temp<=1'b0;
                2'b10: temp<=1'b1;
                2'b00: temp<=temp;
                2'b11: temp<=1'bx;
                default: temp<=1'bz;
                end case
        end
end
assign q=temp;
assign qb=~q;
endmodule
```

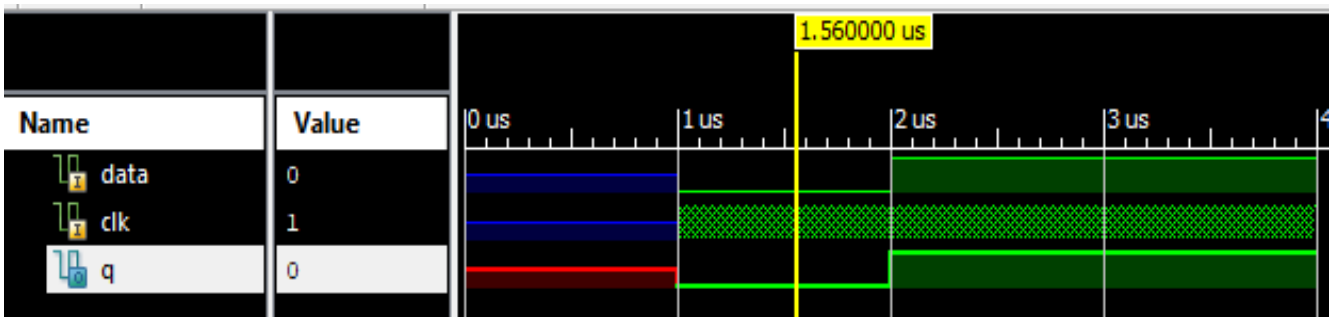
SIMULATION OUTPUT



ii) D Flip-flop:

```
module exp4_sync_dff(data,clk,reset,q);
input data,clk,reset;
output q;
reg q;
always@(posedge clk)
begin
        if(~reset)
                q=1'b1;
        else q=data;
end
endmodule
```

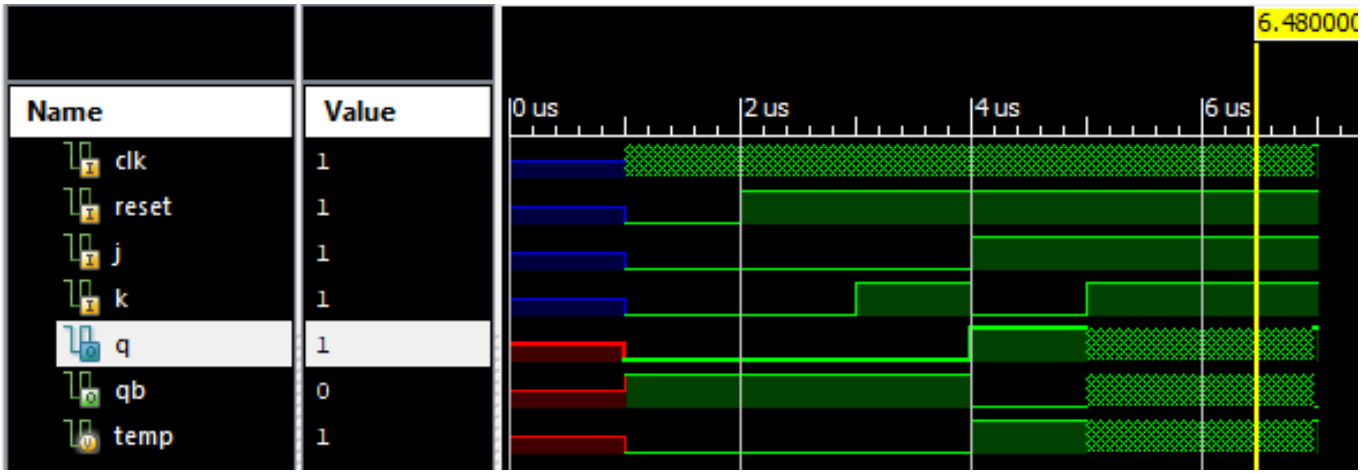
SIMULATION OUTPUT



iii) JK Flip-flop:

```
module jkff (clk, reset, j, k, q, qb);
input clk,reset;
input j;
input k;
output q;
output qb;
reg temp;
always@(negedge reset, posedge clk)
begin
        if(!reset)
                temp<=1'b0;
        else
        begin
                case({j,k})
                2'b01:temp=1'b0;
                2'b10:temp=1'b1;
                2'b11:temp=~temp;
                2'b00:temp=temp;
                default:temp<=1'bz;
                end case
        end
end
assign q=temp;
assign qb=~temp;
endmodule
```

SIMULATION OUTPUT



Viva Questions: Experiment 9		
Q. No.	**Question**	**Bloom's Level**
1	State the excitation table of JK flip-flop.	Remembering
2	Why is JK flip-flop called a “universal flip-flop”?	Understanding
3	Implement a D flip-flop in Verilog using behavioral modeling.	Applying
4	Compare timing diagrams of D and JK flip-flops.	Analyzing
5	Design a universal flip-flop module in Verilog that can act as D, T, or JK based on a control input.	Creating

Activity/ Workbook

Signature of the Lab Incharge

Program 10: Design a 4 bit Ripple counter and implement with Verilog HDL

Ring Counter Timing signals that control the sequence of operations in a digital system can be generated by a shift register or by a counter with a decoder. A ring counter is a circular shift register with only one flip-flop being set at any time; all others are cleared. The single bit is shifted from one flip-flop to the next to produce the sequence of timing signals. Figure (a) shows a four-bit shift register connected as a ring counter. The initial value of the register is 1000 and requires Preset/Clear flip-flops. The single bit is shifted right with every clock pulse and circulates back from T3 to T0. Each flip-flop is in the 1 state once every four clock cycles and produces one of the four timing signals shown in Fig. (b). Each output becomes a 1 after the negative-edge transition of a clock pulse and remains 1 during the next clock cycle.

For an alternative design, the timing signals can be generated by a two-bit counter that goes through four distinct states. The decoder shown in Fig. (c) decodes the four states of the counter and generates the required sequence of timing signals. To generate 2^n timing signals, we need either a shift register with 2^n flip-flops or an n -bit binary counter together with an n -to-2^n -line decoder. For example, 16 timing signals can be generated with a 16-bit shift register connected as a ring counter or with a 4-bit binary counter and a 4-to-16-line decoder. In the first case, we need 16 flip-flops. In the second, we need 4 flip-flops and 16 four-input AND gates for the decoder. It is also possible to generate the timing signals with a combination of a shift register and a decoder. That way, the number of flip-flops is less than that in a ring counter, and the decoder requires only two-input gates. This combination is called a Johnson counter .

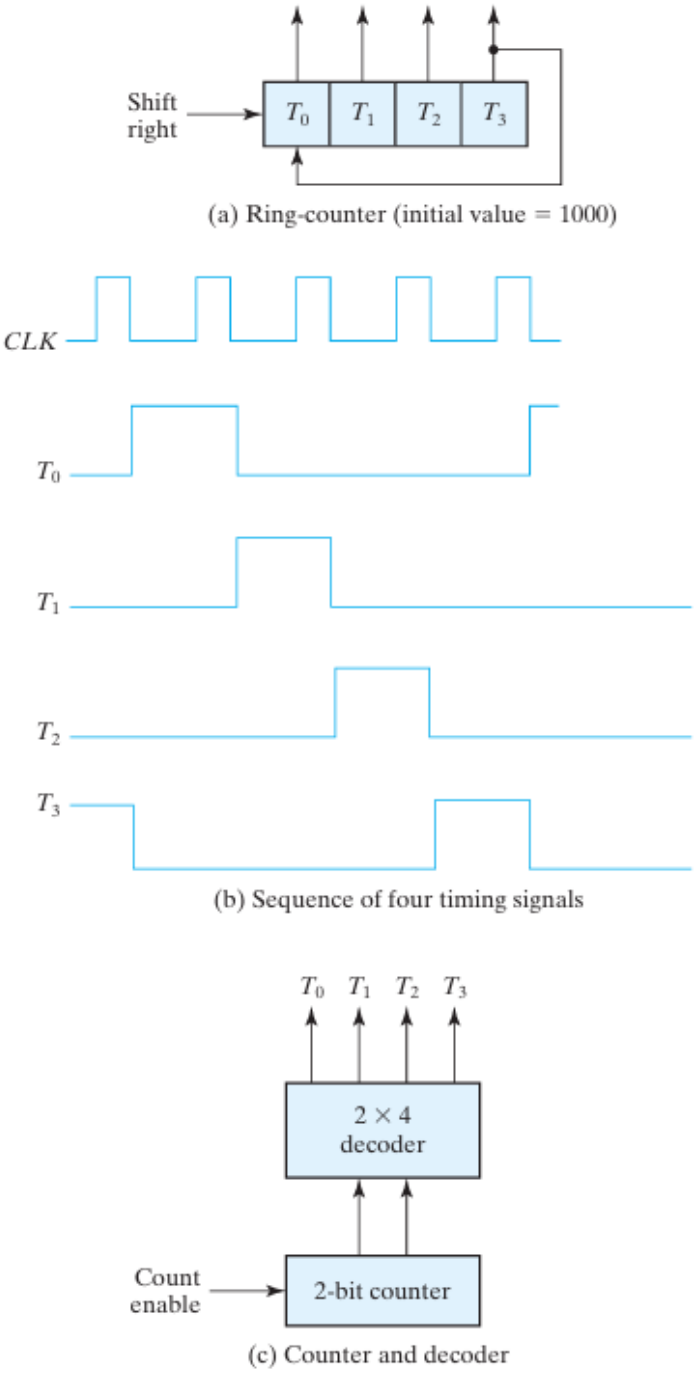
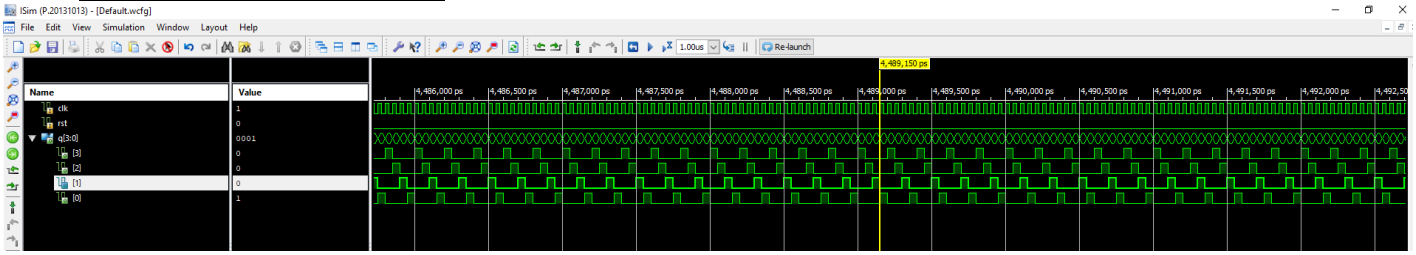


Fig: Generation of Timing Signals

```
// Simple 4-bit Ring Counter
module ring_counter (
    input  wire clk,
    input  wire rst,       // Active-high reset
    output reg [3:0] q
);

    always @(posedge clk or posedge rst) begin
        if (rst)
            q <= 4'b1000;                 // Initial state
        else
            q <= {q[0], q[3:1]};          // Shift right and wrap around
    end

endmodule
```

SIMULATION OUTPUT



Viva Questions: Experiment 10		
Q. No.	**Question**	**Bloom's Level**
1	What is the basic principle of a ripple counter?	Remembering
2	Why does a ripple counter have propagation delay issues?	Understanding
3	Implement a 4-bit ripple up counter in Verilog.	Applying
4	Compare ripple counter with synchronous counter in terms of speed.	Analyzing
5	Extend the ripple counter design to build a modulo-10 (decade) counter.	Creating

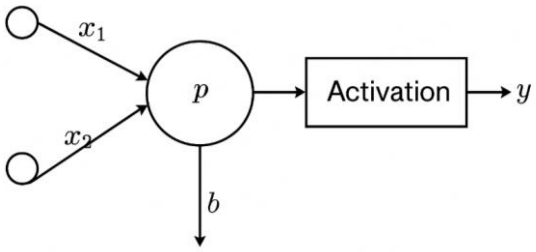
Activity/ Workbook

Signature of the Lab Incharge

Demo Experiment

Verilog implementation of AND, OR, and NOT logic gates using a MacCulloch-Pitts Perceptron model (single-layer feedforward Neural Network).

MacCulloch-Pitts Perceptron Basics: The MacCulloch-Pitts perceptron is one of the earliest models of a neural network, proposed by Warren McCulloch and Walter Pitts in 1943. It is a simple mathematical model of a biological neuron designed to simulate logical computation using binary input and output values. The perceptron consists of one or more input neurons (typically binary: 0 or 1), each associated with a weight, a bias term, and a threshold-based activation function. The core principle of the perceptron is to compute a weighted sum of the inputs, add a bias, and then pass the result through a step (threshold) activation function to determine the output. This output is usually 1 (active) if the result is equal to or exceeds the threshold, or 0 (inactive) otherwise. This model mimics the way a biological neuron fires when sufficient stimuli (input signals) are received. Although the original MacCulloch-Pitts model is limited to solving linearly separable problems and does not support learning (weights are manually set), it laid the theoretical foundation for modern neural networks and deep learning. The simplicity of the model makes it ideal for implementing basic logical operations like AND, OR, and NOT, demonstrating how computation can emerge from interconnected units using simple rules.



MacCulloch-Pitts Perceptron

The diagram consists of the following key components:

1. Inputs (x_1, x_2, ..., x_n):

These are binary signals (0 or 1) that represent stimuli or data being fed into the perceptron. Each input is associated with a specific weight that defines its influence on the output.

2. Weights (w_1, w_2, ..., w_n):

Each input is multiplied by a corresponding weight. Weights are numeric values (positive or negative) that determine the importance of each input. For example, if $w_1 = 1$ and $x_1 = 1$, the contribution of this input to the sum is 1.

3. Summation Unit (Σ):

This block calculates the weighted sum of all inputs:

$$p = \sum (x \,.\, w_i) + b$$

where b is the bias term. The bias acts like a threshold adjuster, shifting the activation point.

4. Bias (b):

The bias is a constant value added to the weighted sum. It helps the perceptron model a threshold behavior. A more negative bias increases the threshold needed for the neuron to activate.

5. Activation Function (Step Function):

The perceptron applies a step function (also called the Heaviside function) to the summation output (p). The step function checks:

- If $p \geq 0 \rightarrow$ output $y = 1$ (neuron fires)
- If $p < 0 \rightarrow$ output $y = 0$ (neuron does not fire)

6. Output (y):

The final output of the perceptron is binary—either 0 or 1—based on the result of the activation function. This output can represent a logical decision like true/false, yes/no, or fire/not fire.

Example Use:

Let's go through each logic gate using a perceptron model with two inputs (x1 and x2), weights (w1, w2), a bias term (b), and activation (usually a step function).

The perceptron calculates: $p = x1 \cdot w1 + x2 \cdot w2 + b$

$y = activation(p)$

Where activation is typically: $y = 1$ if $p \geq 0$, else 0

Assumptions:

- Inputs are binary (0 or 1).
- Weights and bias are modeled as integers.
- Threshold function is implemented via if-else (step function).

How It Works

- Each gate computes $p = \Sigma (w \cdot x) + b$.
- Output y is 1 if $p \geq 0$, else 0.
- This mimics the biological neuron logic of the MacCulloch-Pitts model.

1. AND Gate Design:

Goal: Output is 1 only when both inputs are 1.

- Inputs: $x_1, x_2 \in \{0, 1\}$
- Weights: $w_1 = 1, w_2 = 1$
- Bias: $b = -1.5$

x1	x2	w1	w2	b	p = x1·w1 + x2·w2 + b	y (step)
0	0	1	1	-1.5	0 + 0 - 1.5 = -1.5	0
0	1	1	1	-1.5	0 + 1 - 1.5 = -0.5	0
1	0	1	1	-1.5	1 + 0 - 1.5 = -0.5	0
1	1	1	1	-1.5	1 + 1 - 1.5 = 0.5	1

2. OR Gate Design:

Goal: Output is 1 if at least one input is 1.

- Inputs: $x_1, x_2 \in \{0, 1\}$
- Weights: $w_1 = 1, w_2 = 1$
- Bias: $b = -0.5$

x1	x2	w1	w2	b	p = x1·w1 + x2·w2 + b	y (step)
0	0	1	1	-0.5	0 + 0 - 0.5 = -0.5	0
0	1	1	1	-0.5	0 + 1 - 0.5 = 0.5	1
1	0	1	1	-0.5	1 + 0 - 0.5 = 0.5	1
1	1	1	1	-0.5	1 + 1 - 0.5 = 1.5	1

3. NOT Gate Design (1 input):

Goal: Output is the opposite of the input.

- Input: x ∈ {0, 1}
- Weight: w = –1
- Bias: b = 0.5

x	w	b	p = x·w + b	y (step)
0	-1	0.5	0 + 0.5 = 0.5	1
1	-1	0.5	-1 + 0.5 = -0.5	0

Verilog Code for AND Gate (Perceptron):

```
//========================================
// MacCulloch-Pitts Perceptron - AND Gate
//========================================
module perceptron_and(
   input wire x1,
   input wire x2,
   output reg y
);
   integer w1 = 1;
   integer w2 = 1;
   integer b  = -2; // Bias equivalent to -1.5, rounded for integer logic

   always @(*) begin
      integer p;
      p = x1 * w1 + x2 * w2 + b;
      if (p >= 0)
         y = 1;
      else
         y = 0;
   end
endmodule
```

Verilog Code for OR Gate (Perceptron):

```
//=========================================
// MacCulloch-Pitts Perceptron - OR Gate
//=========================================
module perceptron_or(
    input wire x1,
    input wire x2,
    output reg y
);
    integer w1 = 1;
    integer w2 = 1;
    integer b  = -1; // Bias equivalent to -0.5, rounded

    always @(*) begin
        integer p;
        p = x1 * w1 + x2 * w2 + b;
        if (p >= 0)
            y = 1;
        else
            y = 0;
    end
endmodule
```

Verilog Code for NOT Gate (Perceptron):

```
//=========================================
// MacCulloch-Pitts Perceptron - NOT Gate
//=========================================
module perceptron_not(
    input wire x,
    output reg y
);
    integer w = -1;
    integer b = 1; // Bias equivalent to 0.5, using integer logic

    always @(*)
    begin
        integer p;
        p = x * w + b;
        if (p >= 0)
            y = 1;
        else
            y = 0;
    end
endmodule
```

APPENDIX 1: Additional Programs

a) AND Gate

Structural Model	Data Flow Model	BehaviouralModel

moduleandstr(x,y,z);
inputx,y;
output z;
and g1(z,x,y);
endmodule
	moduleanddf(x,y,z);
	inputx,y;
	output z;
	assign z=(x&y);
	endmodule
		module andbeh(x,y,z);
		input x,y;
		output z;
		reg z;
		always @(x,y)
		z=x&y;
		endmodule

b) NAND Gate

Structural Model	Data Flow Model	BehaviouralModel

modulenandstr(x,y,z);
inputx,y;
output z;
nand g1(z,x,y);
endmodule
	modulenanddf(x,y,z);
	inputx,y;
	output z;
	assign z= !(x&y);
	endmodule
		module nandbeh(x,y,z);
		input x,y;
		output z;
		reg z;
		always @(x,y)
		z=!(x&y);
		endmodule

HALF ADDER:

Structural model	Dataflow model	Behaviouralmodel

modulehalfaddstr(sum,carry,a,b);
outputsum,carry;
inputa,b;
xor(sum,a,b);
and(carry,a,b);
endmodule
	modulehalfadddf(sum,carry,a,b);
	outputsum,carry;
	inputa,b;
	assign sum = a ^ b;
	assign carry=a&b;
	endmodule
		modulehalfaddbeh(sum,carry,a,b);
		outputsum,carry;
		inputa,b;
		regsum,carry;
		always @(a,b);
		sum = a ^ b;
		carry=a&b;
		endmodule

FULL ADDER:

Structural model	Dataflow model	Behaviouralmodel

module fulladdstr(sum,carry,a,b,c);
 	
outputsum,carry;
 	
inputa,b,c;
 	
xor g1(sum,a,b,c);
 	
and g2(x,a,b);
 	
and g3(y,b,c);
 	
and g4(z,c,a);
 	
or g5(carry,x,z,y);
 	
endmodule
	modulefulladddf(sum,carry,a,b,c);
	
	outputsum,carry;
	
	inputa,b,c;
	
	assign sum = a ^ b^c;
	
	assign carry=(a&b) | (b&c) | (c&a);
	
	endmodule
		modulefulladdbeh(sum,carry,a,b,c);
		
		outputsum,carry;
		
		inputa,b,c;
		
		regsum,carry;
		
		always @ (a,b,c)
		
		sum = a ^ b^c;
		
		carry=(a&b) | (b&c) | (c&a);
		
		endmodule

HALF SUBTRACTOR:

Structural model	Dataflow Model	BehaviouralModel

modulehalfsubtstr(diff,borrow,a,b);
 	
outputdiff,borrow;
 	
inputa,b;
 	
xor(diff,a,b);
 	
and(borrow,~a,b);
 	
endmodule
	modulehalfsubtdf(diff,borrow,a,b);
	
	outputdiff,borrow;
	
	inputa,b;
	
	assign diff = a ^ b;
	
	assign borrow=(~a&b);
	
	endmodule
		modulehalfsubtbeh(diff,borrow,a,b);
		
		outputdiff,borrow;
		
		inputa,b;
		
		regdiff,borrow;
		
		always @(a,b)
		
		diff = a ^ b;
		
		borrow=(~a&b);
		
		endmodule

FULL SUBTRACTOR:

Structural model	Dataflow Model	BehaviouralModel

module fullsubtstr(diff,borrow,a,b,c);
outputdiff,borrow;
inputa,b,c;
wire a0,q,r,s,t;
not(a0,a);
xor(x,a,b);
xor(diff,x,c);
and(y,a0,b);
and(z,~x,c);
or(borrow,y,z);
endmodule
	modulefullsubtdf(diff,borrow,a,b,c);
	outputdiff,borrow;
	inputa,b,c;
	assign diff = a^b^c;
	assign borrow=(~a&b)|(~(a^b)&c);
	endmodule
		modulefullsubtbeh(diff,borrow,a,b,c);
		outputdiff,borrow;
		inputa,b,c;
		outputdiff,borrow;
		always@(a,b,)
		diff = a^b^c;
		borrow=(~a&b)|(~(a^b)&c);
		endmodule

Multiplexers 2:1 MUX

Structural Model	Dataflow Model	BehaviouralModel

module mux21str(i0,i1,s,y);
input i0,i1,s;
output y;
wire net1,net2,net3;
not g1(net1,s);
and g2(net2,i1,s);
and g3(net3,i0,net1);
or g4(y,net3,net2);
endmodule
	module mux21df(i0,i1,s,y);
	input i0,i1,s;
	output y;
	assign y =(i0&(~s))|(i1&s);
	endmodule
		module mux21beh(i0,i1,s,y);
		input i0,i1,s;
		output y;
		reg y;
		always@(i0,i1)
		begin
		if(s==0) y=i1;
		if(s==1)y=i0;
		end
		endmodule

4:1 MUX

Structural Model	Dataflow Model	BehaviouralModel

module mux41str(i0,i1,i2,i3,s0,s1,y);
input i0,i1,i2,i3,s0,s1;
 	
wire a,b,c,d;
 	
output y;
 	
and g1(a,i0,s0,s1);
 	
and g2(b,i1,(~s0),s1);
 	
and g3(c,i2,s0,(~s1));
 	
and g4(d,i3,(~s0),(~s1));
 	
or(y,a,b,c,d);
	module mux41df(i0,i1,i2,i3,s0,s1,y);
	
	input i0,i1,i2,i3,s0,s1;
	
	output y;
	
	assign y=((i0&(~(s0))&(~(s1)))|
	(i1&(~(s0))&s1)|
	|(i2&s0&(~(s1)))|
	(i3&s0&s1);
	
	endmodule
		module mux41beh(in,s,y);
		output y ;
		input [3:0] in ;
		input [1:0] s ;
		reg y;
		always @ (in,s)
		
		begin
		if (s[0]==0&s[1]==0)
		y = in[3];
		else if (s[0]==0&s[1]==1)
		y = in[2];
		else if (s[0]l==1&s[1]==0)
		y = in[1];
		else
		y = in[0];
		end
		endmodule

1:4 DEMUX

Structural Model	Dataflow Model	BehaviouralModel

module demux14str(in,d0,d1,d2,d3,s0,s1);
 	
output d0,d1,d2,d3;
 	
input in,s0,s1;
 	
and g1(d0,in,s0,s1);
 	
and g2(d1,in,(~s0),s1);
 	
and g3(d2,in,s0,(~s1));
 	
and g4(d3,in,(~s0),(~s1));
 	
endmodule
	module demux14df(
	in,d0,d1,d2,d3,s0,s1);
	
	output d0,d1,d2,3;
	
	input in,s0,s1;
	assign s0 = in & (~s0) & (~s1);
	
	assign d1= in & (~s0) & s1;
	
	assign d2= in & s0 & (~s1);
	assign d3= in & s0 & s1;
	
	endmodule
		module demux14beh(
		din,sel,dout);
		output [3:0] dout ;
		reg [3:0] dout ;
		input din ;
		wire din ;
		input [1:0] sel ;
		wire [1:0] sel ;
		always @ (din or sel) begin
		case (sel)
		0 : dout = {din,3'b000};
		1 : dout = {1'b0,din,2'b00};
		2 : dout = {2'b00,din,1'b0};
		default : dout = {3'b000,din};
		endcase
		
		end
		
		endmodule

VIVA QUESTIONS & ANSWERS

1. What are logic gates?
   A programming function that, processes true and false signals.
2. What are combinational logic circuits?
   In digital circuit theory, combinational logic circuit is a type of digital logic which is implemented by Boolean circuits, where the output is a pure function of the present input only.
3. What does VHDL stands for?
   VHDL stands for VHSIC (Very High Speed Integrated Circuit) Hardware Description Language.
4. Types of models in VHDL?
   There are 3 types: Data flow Description, Behavioural Description and Structural Description.
   Data-flow description: A digital circuit can be described at the data-flow level by giving the logic equation of that circuit.
   Behavioural description: A digital circuit can be described at the behavioural level in terms of Its function or behaviour, without giving any implementation details.
   Structural description: A digital circuit can be described at the structural level by specifying the interconnection of the gates or flip-flops that comprise the circuit.
5. What is meant by embedded system?
   An embedded system is a computer system designed for specific control functions within a larger system, often with real-time computing constraints.
6. What is an I.C ?
   Integrated circuit. A miniaturized electronic circuit that combines a variety of components like transistors, resistors, capacitors, and diodes all into one incredibly small piece.
7. What is multiplexer and mention its applications ?
   In electronics, a multiplexer or mux is a device that selects one of several analog or digital input signals and forwards the selected input into a single line. Multiplexers are mainly used to increase the amount of data that can be sent over the network within a certain amount of time and bandwidth. A multiplexer is also called a data selector.
8. What is de-multiplexer and mention its applications ?
   A De-multiplexer or demux is a device taking a single input signal and selecting one of many data-output-lines, which is connected to the single input.
9. What are flip flops?
   Flip flops are sequential circuits used to store one bit of information at a time. Which has two stable states i.e., present and past previous state.
10. Define combinational logic
    When logic gates are connected together to produce a specified output for certain specified combinations of input variables, with no storage involved, the resulting circuit is called combinational logic.

11. Define half adder and full adder
    The logic circuit that performs the addition of two bits is a half adder. The circuit that performs the addition of three bits is a full adder.
12. Define Decoder.
    A decoder is a multiple - input multiple output logic circuits that converts coded inputs into coded outputs where the input and output codes are different.
13. What is binary decoder?
    A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2n out puts lines.
14. Define Encoder?
    An encoder has 2n input lines and n output lines. In encoder the output lines generate the binary code corresponding to the input value.
15. What is priority Encoder?
    A priority encoder is an encoder circuit that includes the priority function. In priority encoder, if 2 or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.
16. Define multiplexer?
    Multiplexer is a digital switch. It allows digital information from several sources to be routed onto a single output line.
17. Which gate is equal to AND-invert Gate?
    NAND gate.
18. Which gate is equal to OR-invert Gate?
    NOR gate.
19. Bubbled OR gate is equal to---------- NAND gate
    Bubbled AND gate is equal to---------- NOR gate
20. What is the classification of sequential circuits?
    The sequential circuits are classified on the basis of timing of their signals into two types. They are, 1) Synchronous sequential circuit.2) Asynchronous sequential circuit.
21. Define Flip flop.
    The basic unit for storage is flip flop. A flip-flop maintains its output state either at 1 or 0 until directed by an input signal to change its state.
22. What are the different types of flip-flop?
    There are various types of flip flops. Some of them are mentioned below they are RS flip-flop,SR flip-flop,D flip-flop, JK flip-flop and, T flip-flop.
23. Define race around condition.
    In JK flip-flop output is fed back to the input. Therefore, change in the output results change in the input. Due to this in the positive half of the clock pulse if both J and K are high then output toggles continuously. This condition is called 'race around condition'.
24. What is a master-slave flip-flop?
    A master-slave flip-flop consists of two flip-flops where one circuit serves as a master and the other as a slave.
25. Define sequential circuit?

In sequential circuits the output variables dependent not only on the present input Variables but they also depend up on the past history of these input variables.

26. Give the comparison between combinational circuits and sequential circuits.
    Combinational circuits Sequential circuits. Memory unit is not required - -Memory unity is required Parallel adder is a combinational circuit Serial adder is a sequential circuit
27. Define binary logic?
    Binary logic consists of binary variables and logical operations. The variables aredesignated by the alphabets such as A, B, C, x, y, z, etc., with each variable having only two distinct values: 1 and 0. There are three basic logic operations: AND, OR, and NOT.
28. What are the basic digital logic gates?
    The three basic logic gates are AND gate, OR gate, NOT gate
29. What is a Logic gate?
    Logic gates are the basic elements that make up a digital system. The electronic gate is a circuit that can operate on several binary inputs to perform a particular logical function.
30. Which gates are called as the universal gates? What are its advantages?
    The NAND and NOR gates are called as the universal gates. These gates are used to perform any type of logic application.

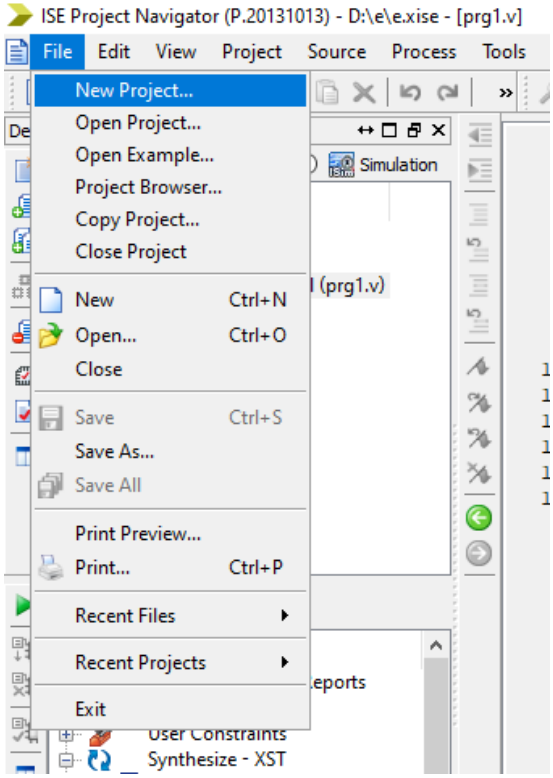
Simulation Steps:

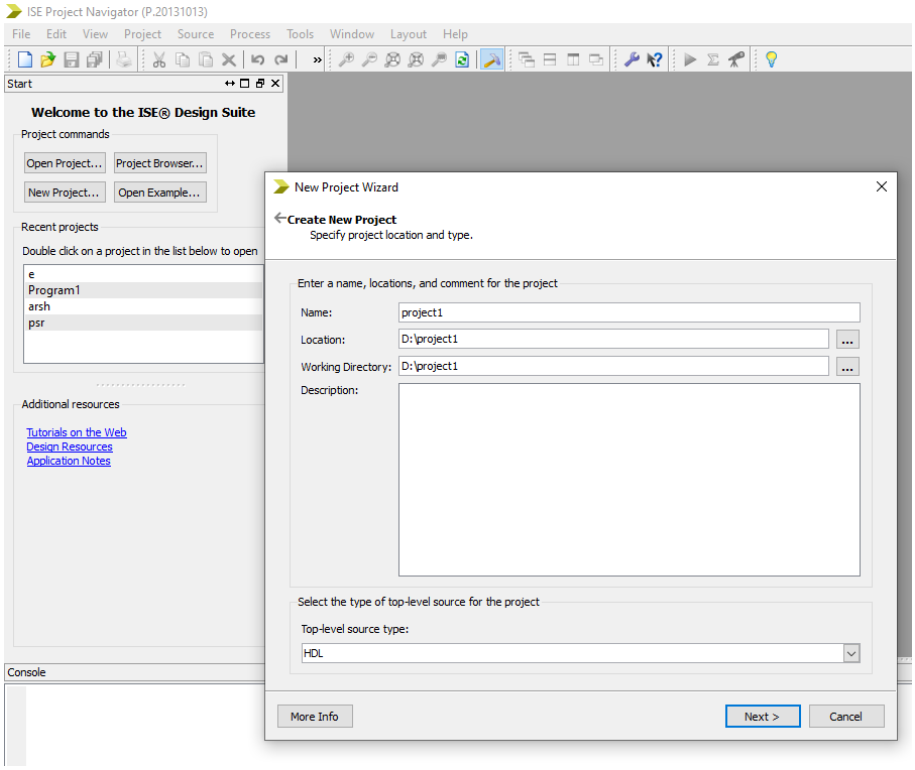
1. Start The Xilinx Project Navigator By Using The Desktop Shortcut or By using the Start →Programs →Xilinx ISE (14.7).
2. Create a New project Select File menu and Then Select New Project.
3. Specify the project Name and Location in pop up Window and click next.
4. To Create New Verilog file Right click on the device name and Select NEW SOURCE.

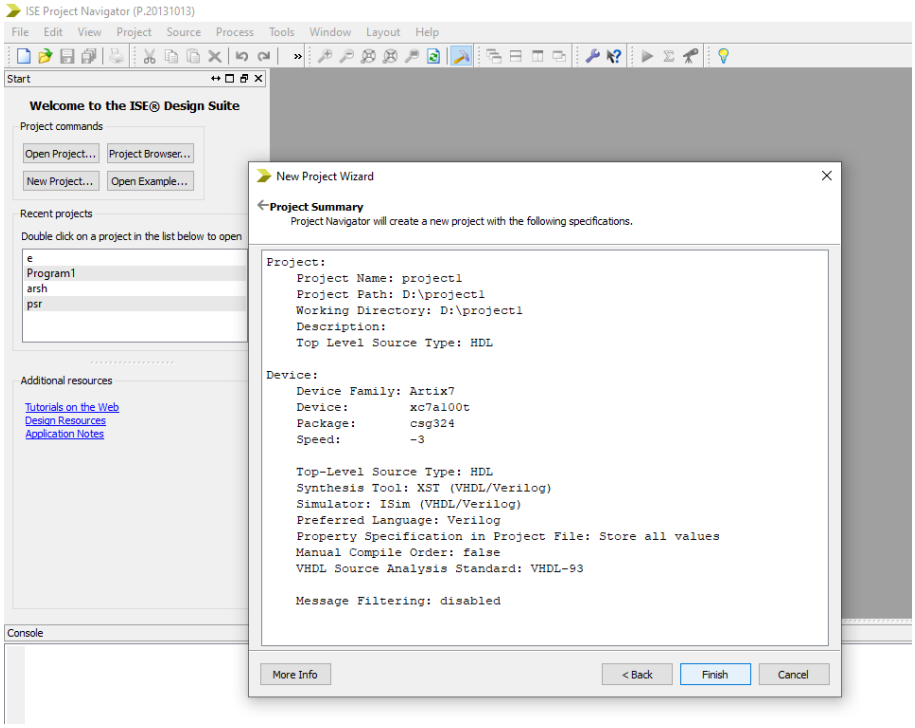
   Select Verilog module in New Source Wizard and Give Suitable name for the Project Click

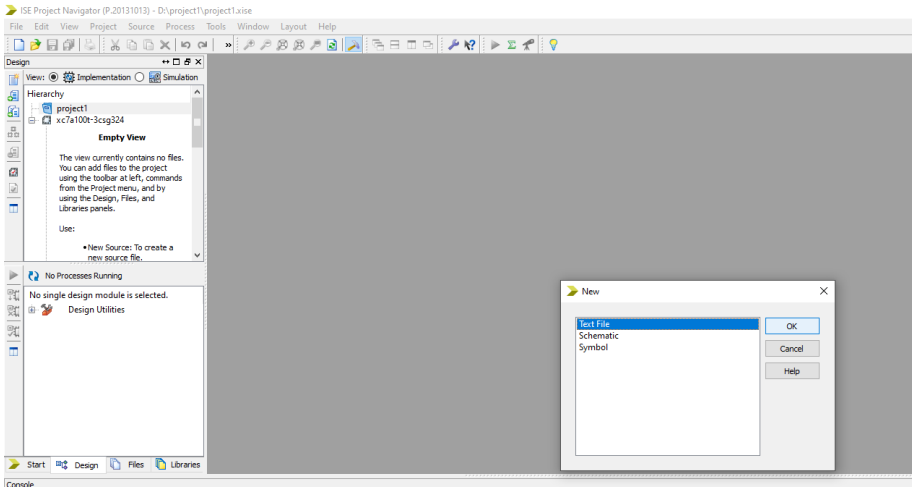
   NEXT for the Define Module Window.
5. Write Behavioral Verilog code in Verilog Editor.

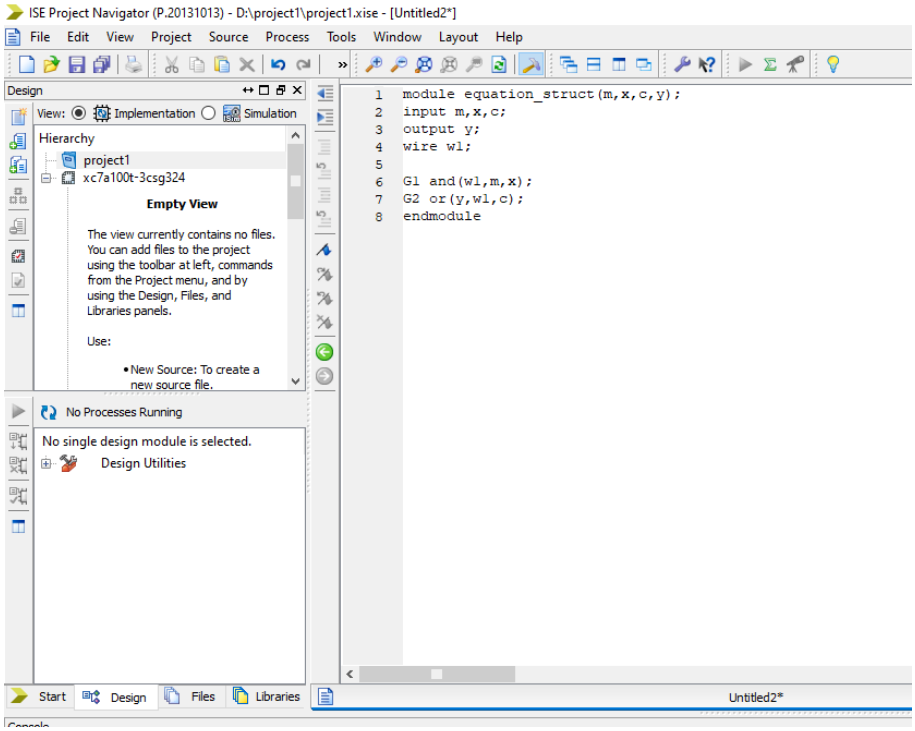
Steps Screenshots Shown below

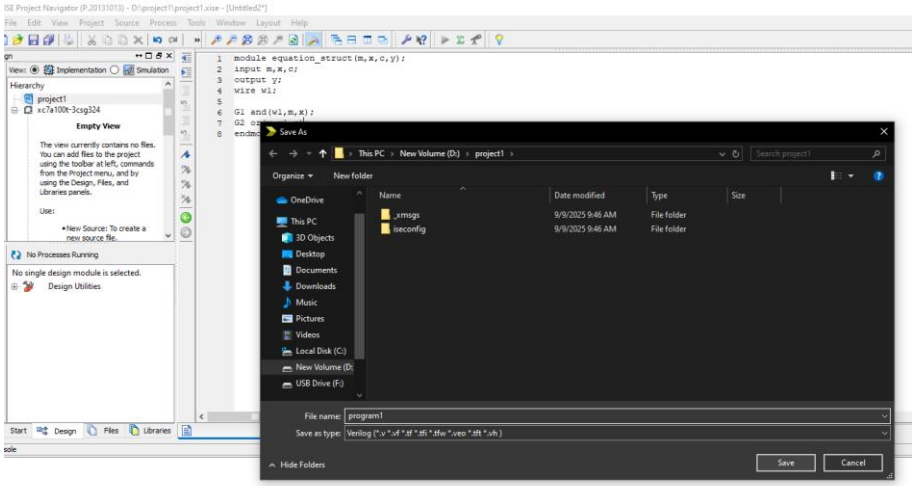


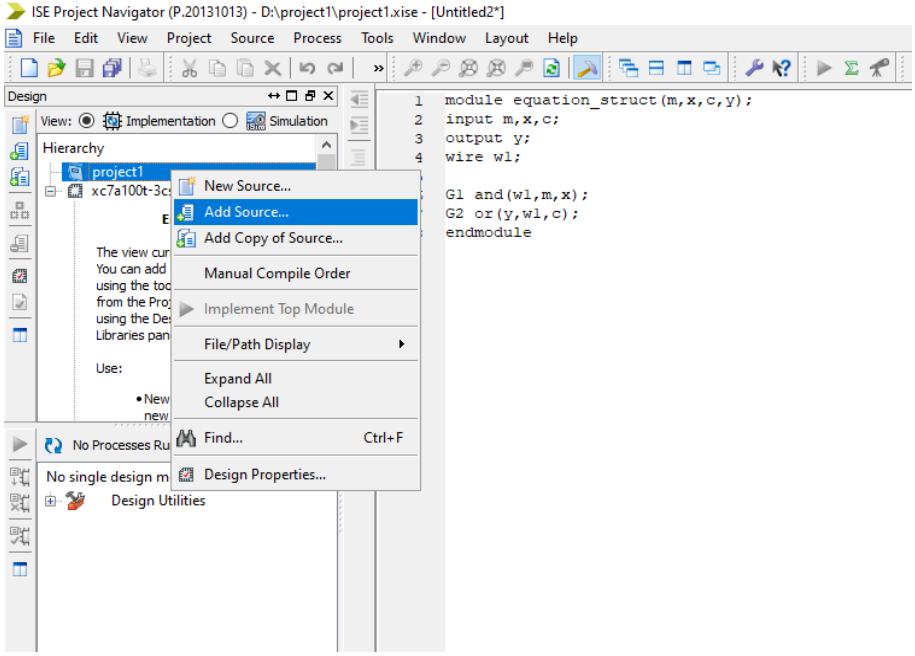

ISE Project Navigator (P.20131013)
File Edit View Project Source Process Tools Window Layout Help
Start
Welcome to the ISE® Design Suite
Project commands
Open Project...
Project Browser...
New Project...
Open Example...
Recent projects
Double click on a project in the list below to open
e
Program1
arsh
psr
Additional resources
Tutorials on the Web
Design Resources
Application Notes
Console
New Project Wizard
Create New Project
Specify project location and type.
Enter a name, locations, and comment for the project
Name: project1
Location: D:\project1
Working Directory: D:\project1
Description:
Select the type of top-level source for the project
Top-level source type:
HDL
More Info
Next >
Cancel

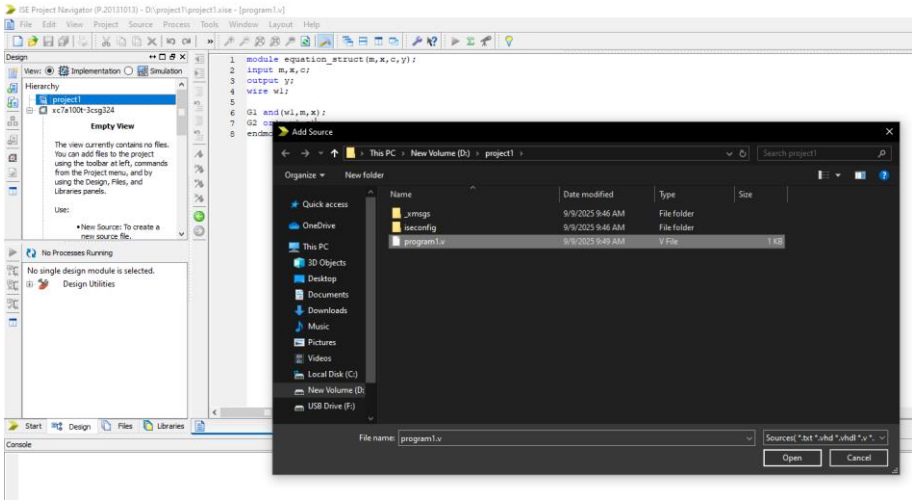

ISE Project Navigator (P.20131013)
File Edit View Project Source Process Tools Window Layout Help
Start
Welcome to the ISE® Design Suite
Project commands
Open Project...
Project Browser...
New Project...
Open Example...
Recent projects
Double click on a project in the list below to open
e
Program1
arsh
psr
Additional resources
Tutorials on the Web
Design Resources
Application Notes
Console
New Project Wizard
Project Summary
Project Navigator will create a new project with the following specifications.
Project:
Project Name: project1
Project Path: D:\project1
Working Directory: D:\project1
Description:
Top Level Source Type: HDL
Device:
Device Family: Artix7
Device: xc7a100t
Package: csg324
Speed: -3
Top-Level Source Type: HDL
Synthesis Tool: XST (VHDL/Verilog)
Simulator: ISim (VHDL/Verilog)
Preferred Language: Verilog
Property Specification in Project File: Store all values
Manual Compile Order: false
VHDL Source Analysis Standard: VHDL-93
Message Filtering: disabled
More Info
< Back
Finish
Cancel

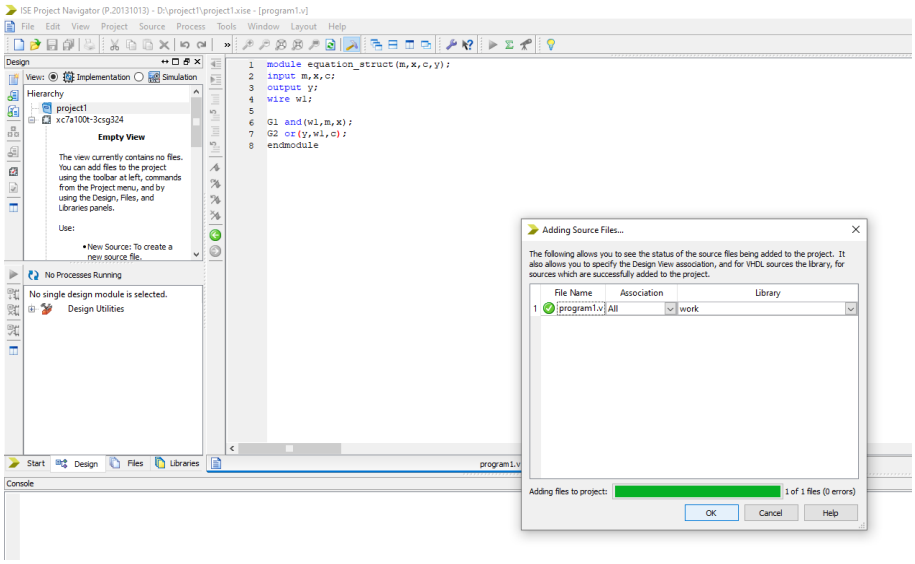

ISE Project Navigator (P.20131013) - D:\project1\project1.xise
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
project1
xc7a100t-3csg324
Empty View
The view currently contains no files. You can add files to the project using the toolbar at left, commands from the Project menu, and by using the Design, Files, and Libraries panels.
Use:
•New Source: To create a new source file.
No Processes Running
No single design module is selected.
Design Utilities
Start Design Files Libraries
Console
New
Text File
Schematic
Symbol
OK
Cancel
Help

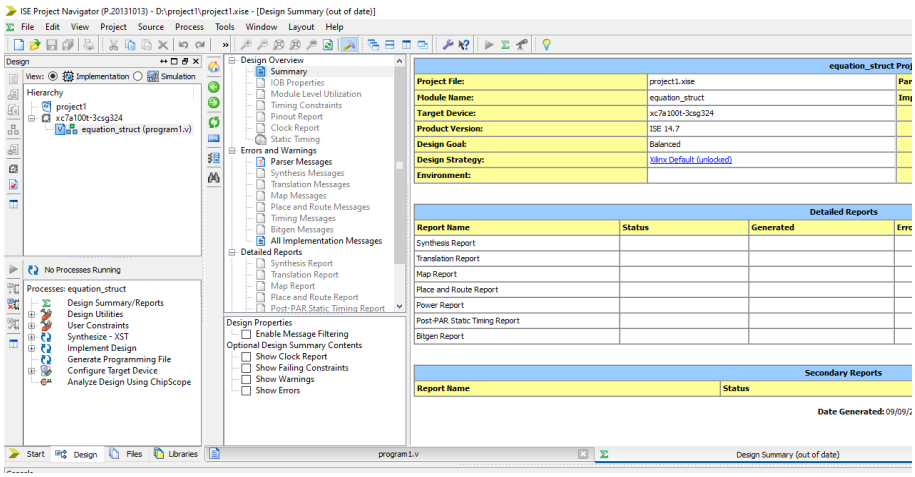

ISE Project Navigator (P.20131013) - D:\project1\project1.xise - [Untitled2*]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
project1
xc7a100t-3csg324
Empty View
The view currently contains no files. You can add files to the project using the toolbar at left, commands from the Project menu, and by using the Design, Files, and Libraries panels.
Use:
•New Source: To create a new source file.
No Processes Running
No single design module is selected.
Design Utilities
1 module equation_struct(m,x,c,y);
2 input m,x,c;
3 output y;
4 wire w1;
5
6 G1 and(w1,m,x);
7 G2 or(y,w1,c);
8 endmodule
Start Design Files Libraries
Untitled2*

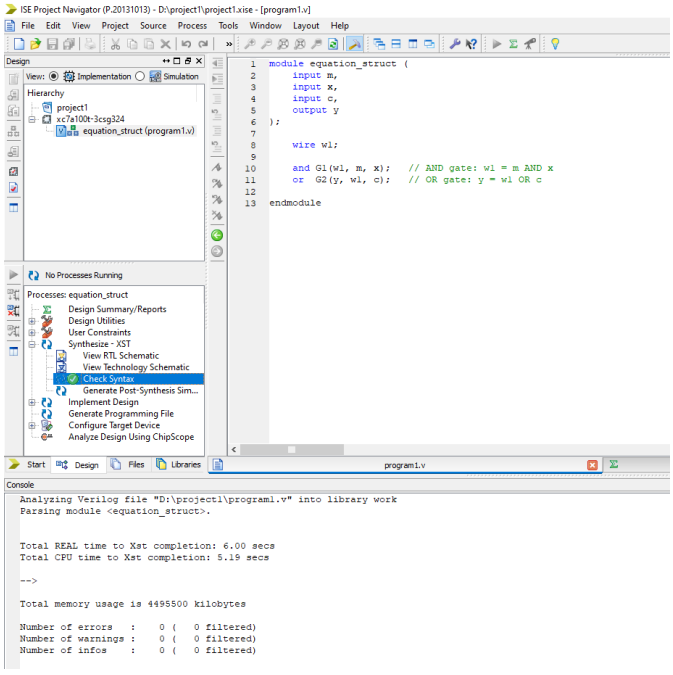

module equation_struct(m,x,c,y);
input m,x,c;
output y;
wire w1;
G1 and(w1,m,x);
Save As
program1
Save
Cancel

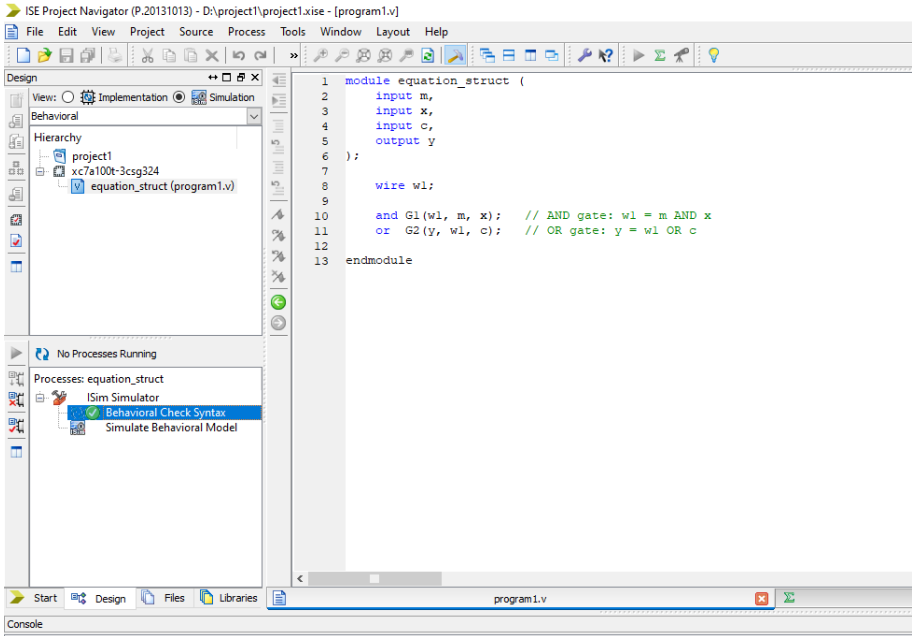

ISE Project Navigator (P.20131013) - D:\project1\project1.xise - [Untitled2*]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
project1
New Source...
Add Source...
Add Copy of Source...
Manual Compile Order
Implement Top Module
File/Path Display
Expand All
Collapse All
Find... Ctrl+F
Design Properties...
Design Utilities
module equation_struct(m,x,c,y);
input m,x,c;
output y;
wire w1;
G1 and(w1,m,x);
G2 or(y,w1,c);
endmodule


Add Source
program1.v
Open
Cancel

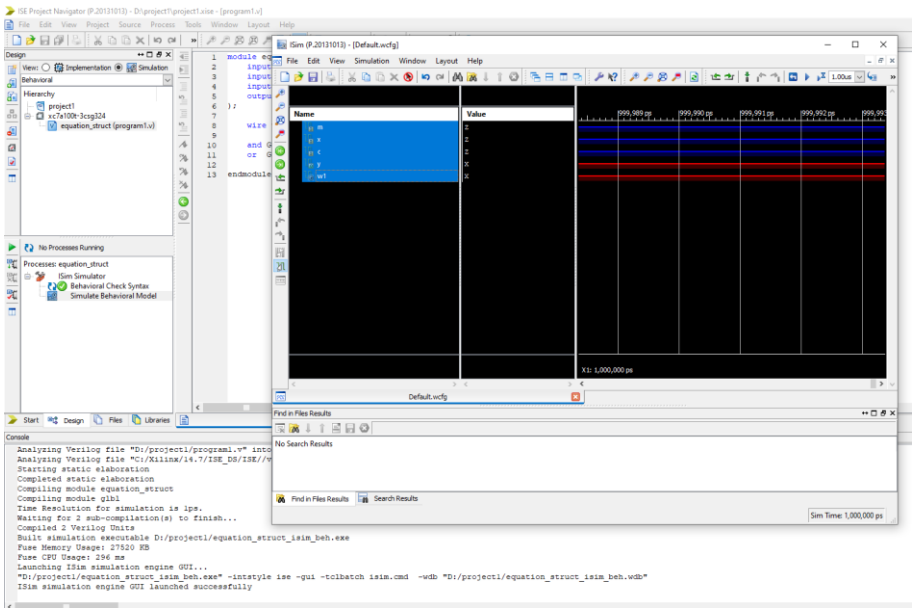

ISE Project Navigator (P.20131013) - D:\project1\project1.xise - [program1.v]
module equation_struct(m,x,c,y);
input m,x,c;
output y;
wire w1;
G1 and(w1,m,x);
G2 or(y,w1,c);
endmodule
Adding Source Files...
The following allows you to see the status of the source files being added to the project. It also allows you to specify the Design View association, and for VHDL sources the library, for sources which are successfully added to the project.
File Name
Association
Library
program1.v
All
work
Adding files to project:
1 of 1 files (0 errors)
OK
Cancel
Help

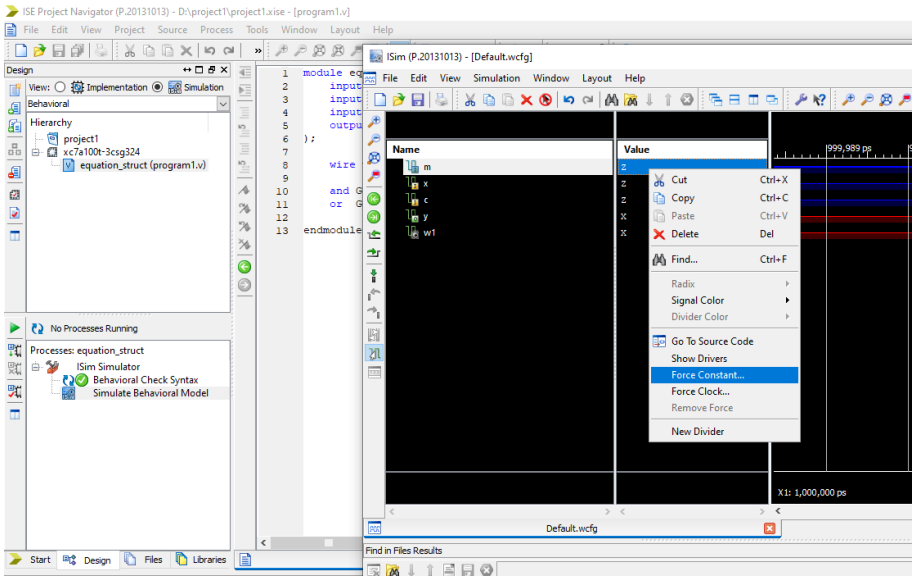

ISE Project Navigator (P.20131013) - D:\project1\project1.xise - [Design Summary (out of date)]
Design Overview
Summary
Errors and Warnings
Detailed Reports
equation_struct Proj
Project File:
project1.xise
Module Name:
equation_struct
Target Device:
xc7a100t-3csg324
Product Version:
ISE 14.7
Design Goal:
Balanced
Design Strategy:
Xilinx Default (unlocked)
Environment:
Detailed Reports
Report Name
Status
Generated
Synthesis Report
Translation Report
Map Report
Place and Route Report
Power Report
Post-PAR Static Timing Report
Bitgen Report
Secondary Reports
Report Name
Status
Date Generated:

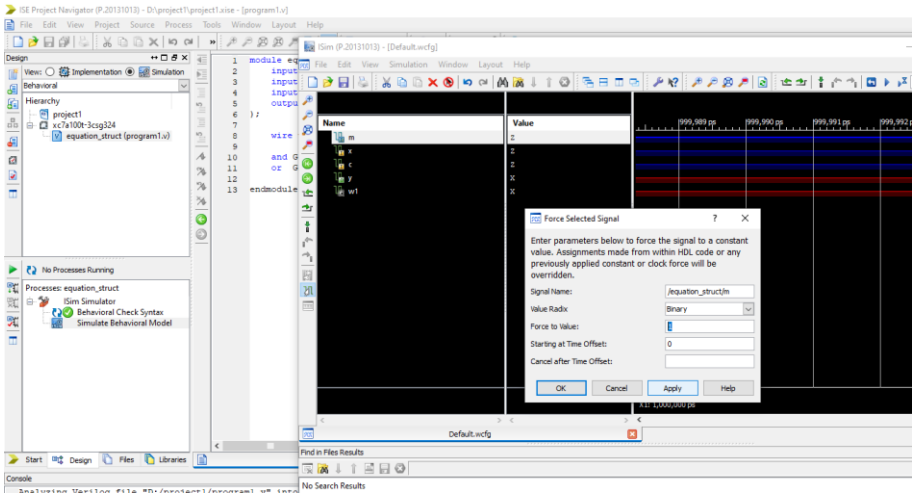

ISE Project Navigator (P.20131013) - D:\project1\project1.xise - [program1.v]
module equation_struct (
input m,
input x,
input c,
output y
);
wire w1;
and G1(w1, m, x); // AND gate: w1 = m AND x
or G2(y, w1, c); // OR gate: y = w1 OR c
endmodule
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Sim...
Analyzing Verilog file "D:\project1\program1.v" into library work
Parsing module <equation_struct>.
Total REAL time to Xst completion: 6.00 secs
Total CPU time to Xst completion: 5.19 secs
Total memory usage is 4495500 kilobytes
Number of errors : 0 (0 filtered)
Number of warnings : 0 (0 filtered)
Number of infos : 0 (0 filtered)




```
module equation_struct (
    input m,
    input x,
    input c,
    output y
);

    wire w1;

    and G1(w1, m, x);   // AND gate: w1 = m AND x
    or  G2(y, w1, c);   // OR gate: y = w1 OR c

endmodule
```




ISE Project Navigator (P.20131013) - D:\project1\project1.xise - [program1.v]
ISim (P.20131013) - [Default.wcfg]
Hierarchy
project1
xc7a100t-3csg324
equation_struct (program1.v)
No Processes Running
Processes: equation_struct
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
Name
Value
Cut
Copy
Paste
Delete
Find...
Radix
Signal Color
Divider Color
Go To Source Code
Show Drivers
Force Constant...
Force Clock...
Remove Force
New Divider
X1: 1,000,000 ps
Default.wcfg
Find in Files Results


Force Selected Signal
Enter parameters below to force the signal to a constant value. Assignments made from within HDL code or any previously applied constant or clock force will be overridden.
Signal Name:
/equation_struct/m
Value Radix
Binary
Force to Value:
Starting at Time Offset:
Cancel after Time Offset:
OK
Cancel
Apply
Help
No Search Results