SIR M VISVESVARAYA INSTITUTE OF TECHNOLOGY

(Affiliated to VTU, Recognized by AICTE and Accredited by NBA, NAAC and an ISO 9001-2008 Certified Institution)

Bengaluru – 562157

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

DIGITAL DESIGN AND COMPUTER ORGANIZATION LABORATORY

CHOICE BASED CREDIT SYSTEM

BCS302 – III Semester B.E
(Academic Year 2023-2024)

Compiled and Prepared by:
Mr. Raghav and Ms. Sowjanya Lakshmi A
Associate Professor, Assistant Professor
Dept. of ISE

Under the Guidance of:
Dr. G. C. Bhanu Prakash
Professor & Head
Dept. of ISE

Department Vision and Mission

VISION

To empower students with knowledge and skills to develop the competency in the emerging areas of Information Technology.

MISSION

- To train the students to have Professional career in IT industry and Higher studies through Quality Education.
- To provide outstanding Teaching and Research environment by implementing innovative Teaching and Research Methodologies for Quality Education and Research.

PROGRAM OUTCOMES

PO's	PO Description
PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complexengineering problems.
PO2	**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.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations 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. • That cannot be solved by straightforward application of knowledge, theories and techniques applicable to the engineering discipline as against problems given at the end of chapters in a typical text book that can be solved using simple engineering theories and techniques • That may not have a unique solution. For example, a design problem can be solved in many ways and lead to multiple possible solutions; • That require consideration of appropriate constraints / requirements not explicitly given in the problem statement such as cost, power requirement, durability, product life, etc.; which need to be defined (modelled) within appropriate mathematical framework; and • That often require use of modern computational concepts and tools
PO5	**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.
PO6	**The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequentresponsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a memberor leader in diverse teams, and in multidisciplinary settings.
PO10	**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.
PO11	**Project management and finance**: Demonstrate knowledge and understanding ofthe engineering and management principles and apply these to one's own work, as a member and leader in a team to manage projects and in multidisciplinaryenvironments.
PO12	**Life-long learning:** Recognize the need for, and have the preparation and ability toengage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES

PSO's	PSO Description
PSO1	An ability to design and analyze algorithms by applying theoretical concepts to build complex and computer- based systems in the domain of System Software, Computer Networks & Security, Web technologies, Data Science and Analytics.
PSO2	Be able to develop various software solutions by applying the techniques of Data Base Management, Complex Mathematical Models, Software Engineering practices and Machine Learning with Artificial Intelligence.

Digital Design and Computer Organization		Semester	3
Course Code	BCS302	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	3:0:2:0	SEE Marks	50
Total Hours of Pedagogy	40 hours Theory + 20 Hours of Practicals	Total Marks	100
Credits	04	Exam Hours	3
Examination nature (SEE)	Theory		

Course objectives:

- To demonstrate the functionalities of binary logic system
- To explain the working of combinational and sequential logic system
- To realize the basic structure of computer system
- To illustrate the working of I/O operations and processing unit

Teaching-Learning Process (General Instructions)
These are sample Strategies; that teachers can use to accelerate the attainment of the various course outcomes.

1. Chalk and Talk
2. Live Demo with experiments
3. Power point presentation

MODULE-1 **8 Hr**

Introduction to Digital Design: Binary Logic, Basic Theorems And Properties Of Boolean Algebra, Boolean Functions, Digital Logic Gates, Introduction, The Map Method, Four-Variable Map, Don't-Care Conditions, NAND and NOR Implementation, Other Hardware Description Language – Verilog Model of a simple circuit.

Text book 1: 1.9, 2.4, 2.5, 2.8, 3.1, 3.2, 3.3, 3.5, 3.6, 3.9

MODULE-2 **8 Hr**

Combinational Logic: Introduction, Combinational Circuits, Design Procedure, Binary Adder- Subtractor, Decoders, Encoders, Multiplexers. HDL Models of Combinational Circuits – Adder, Multiplexer, Encoder. **Sequential Logic**: Introduction, Sequential Circuits, Storage Elements: Latches, Flip-Flops.

Text book 1: 4.1, 4.2, 4.4, 4.5, 4.9, 4.10, 4.11, 4.12, 5.1, 5.2, 5.3, 5.4.

MODULE-3 **8 Hr**

Basic Structure of Computers: Functional Units, Basic Operational Concepts, Bus structure, Performance – Processor Clock, Basic Performance Equation, Clock Rate, Performance Measurement.**Machine Instructions and Programs:** Memory Location and Addresses, Memory Operations, Instruction and Instruction sequencing, Addressing Modes.

Text book 2: 1.2, 1.3, 1.4, 1.6, 2.2, 2.3, 2.4, 2.5

MODULE-4 **8 Hr**

Input/output Organization: Accessing I/O Devices, Interrupts – Interrupt Hardware, Enabling and Disabling Interrupts, Handling Multiple Devices, Direct Memory Access: Bus Arbitration, Speed, size and Cost of memory systems. Cache Memories – Mapping Functions.

Text book 2: 4.1, 4.2.1, 4.2.2, 4.2.3, 4.4, 5.4, 5.5.1

MODULE-5	8 Hr
Basic Processing Unit: Some Fundamental Concepts: Register Transfers, Performing ALU operations, fetching a word from Memory, Storing a word in memory. Execution of a Complete Instruction. **Pipelining:** Basic concepts, Role of Cache memory, Pipeline Performance. **Text book 2: 7.1, 7.2, 8.1**	

PRACTICAL COMPONENT OF IPCC

Sl.NO	Experiments Simulation packages preferred: Multisim, Modelsim, PSpice or any other relevant
1	Given a 4-variable logic expression, simplify it using appropriate technique and simulate the same using basic gates.
2	Design a 4 bit full adder and subtractor and simulate the same using basic gates.
3	Design Verilog HDL to implement simple circuits using structural, Data flow and Behavioural model.
4	Design Verilog HDL to implement Binary Adder-Subtractor – Half and Full Adder, Half and Full Subtractor.
5	Design Verilog HDL to implement Decimal adder.
6	Design Verilog program to implement Different types of multiplexer like 2:1, 4:1 and 8:1.
7	Design Verilog program to implement types of De-Multiplexer.
8	Design Verilog program for implementing various types of Flip-Flops such as SR, JK and D.

Course outcomes (Course Skill Set):
At the end of the course, the student will be 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.
CO5:Analyze internal Organization of Memory and Impact of cache/Pipelining on Processor Performance.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

CIE for the theory component of the IPCC (maximum marks 50)

- IPCC means practical portion integrated with the theory of the course.
- CIE marks for the theory component are **25 marks** and that for the practical component is **25 marks**.
  25 marks for the theory component are split into **15 marks** for two Internal Assessment Tests (Two Tests, each of 15 Marks with 01-hour duration, are to be conducted) and **10 marks** for other assessment methods mentioned in 22OB4.2. The first test at the end of 40-50% coverage of the syllabus and the second test after covering 85-90% of the syllabus.
- Scaled-down marks of the sum of two tests and other assessment methods will be CIE marks for the theory component of IPCC (that is for **25 marks)**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the theory component of IPCC.

CIE for the practical component of the IPCC

- **15 marks** for the conduction of the experiment and preparation of laboratory record, and **10 marks** for the test to be conducted after the completion of all the laboratory sessions.
- On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce and marks shall be awarded on the same day.
- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to **15 marks**.
- The laboratory test **(duration 02/03 hours)** after completion of all the experiments shall be conducted for 50 marks and scaled down to **10 marks.**
- Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for **25 marks**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

SEE for IPCC

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the course (**duration 03 hours**)

1. The question paper will have ten questions. Each question is set for 20 marks.
2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), **should have a mix of topics** under that module.
3. The students have to answer 5 full questions, selecting one full question from each module.
4. Marks scored by the student shall be proportionally scaled down to 50 Marks

- **The theory portion of the IPCC shall be for both CIE and SEE, whereas the practical portion will have a CIE component only. Questions mentioned in the SEE paper may include questions from the practical component**.
-

Suggested Learning Resources:

Books

1. M. Morris Mano & Michael D. Ciletti, Digital Design With an Introduction to Verilog Design, 5e, Pearson Education.
2. Carl Hamacher, ZvonkoVranesic, SafwatZaky, Computer Organization, 5th Edition, Tata McGraw Hill.

Web links and Video Lectures (e-Resources):

https://cse11-iiith.vlabs.ac.in/

Activity Based Learning (Suggested Activities in Class)/ Practical Based learning

Assign the group task to Design the various types of counters and display the output accordingly

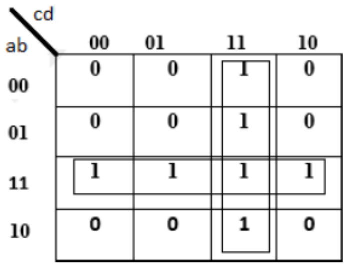
Assessment Methods

- Lab Assessment (25 Marks)
- GATE Based Aptitude Test

1. **Given a 4-variable logic expression, simplify it using appropriate technique and simulate the same using basic gates.**

Design:

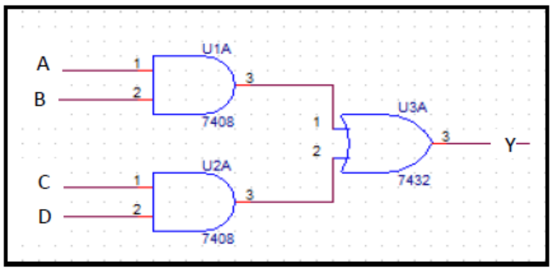
Consider the function f(a,b,c,d) =$\sum m$ (3,7,11,12,13,14,15)



ab \ cd	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	1	1	1	1
10	0	0	1	0

Y = ab + cd

Circuit Diagram



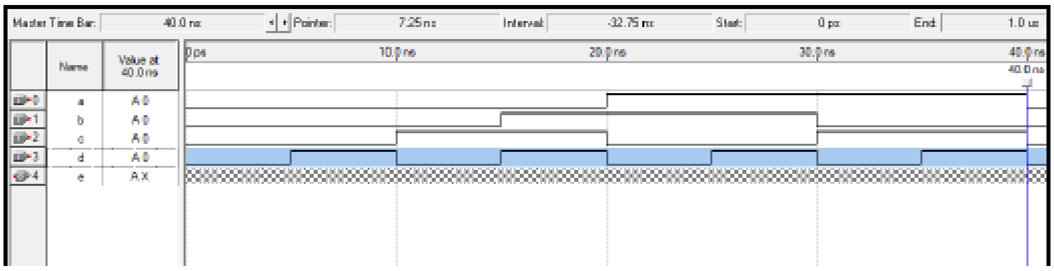
Truth Table

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Verilog Code

```
module f(a,b,c,d,e);
            Input a,b,c,d;
            Output e;
            Assign e=a&b|c&d;
endmodule
```

Output



The four variable logic expression is simplified using K-map and simplified the same using basic gates.

2. **Design a 4 bit full adder and subtractor and simulate the same using basic gates.**

Design for Full adder

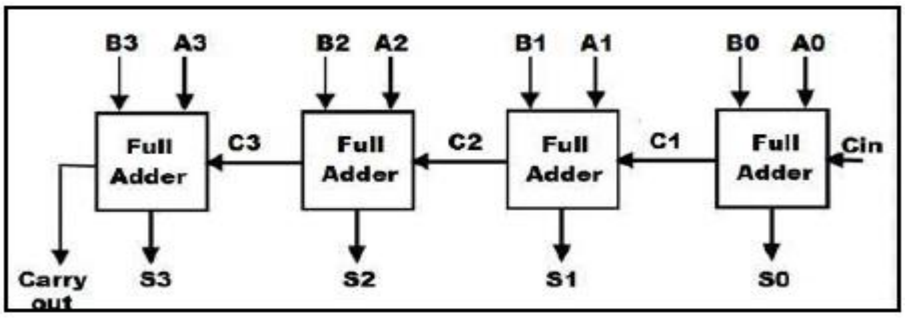
Truth Table

A	B	Cin	Sum	Cout
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

Sum = A’B’C + A’BC’ + AB’C’ + ABC

Cout = A’BC + AB’C + ABC’ + ABC

Circuit Diagram of 4-bit Full adder

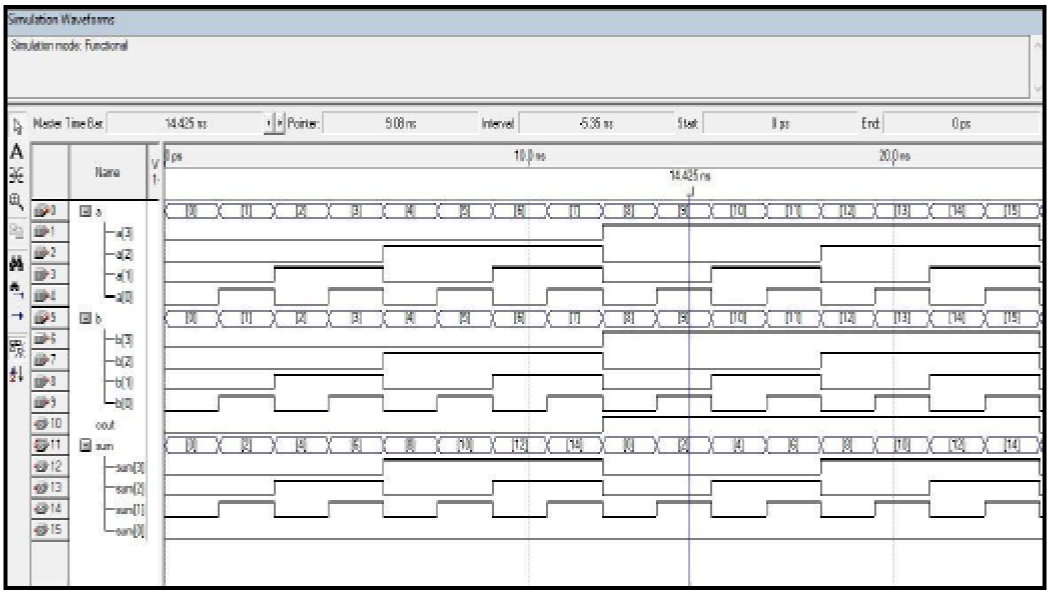


Verilog code

```
module fa(a,b,cin,sum,cout);
        input a,b,cin;
        output sum,cout;
        assign sum = ((!a & !b & cin) | (!a & b & !c) | (a & !b & !c) | (a & b & c));
        assign cout = ((!a & b & c) | (a & !b & c) | (a & b & !c) | (a & b & c));
endmodule

module fourbitadder(a,b,sum,c);
        input [3:0]a;
        input [3:0]b;
        output [3:0]sum;
        output c;
        wire c1,c2,c3;
        fa g0(a[0],b[0],0,sum[0],c1);
        fa g1(a[1],b[1],c1,sum[1],c2);
        fa g2(a[2],b[2],c2,sum[2],c3);
        fa g3(a[3],b[3],c3,sum[3],c);
endmodule
```

Output



Design for Full subtractor

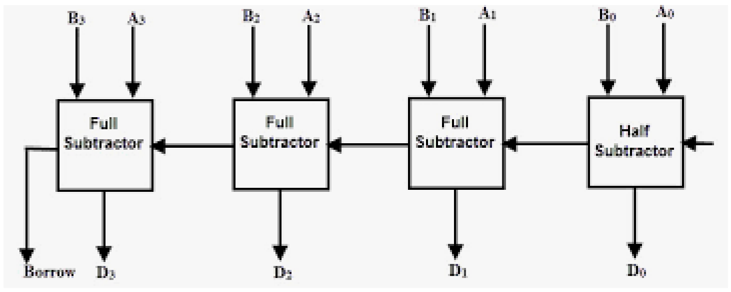
Truth Table

A	B	Cin	Diff	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	0
1	1	1	1	1

Sum = A’B’C + A’BC’ + AB’C’ + ABC

Cout = A’B’C + A’BC’ + A’BC + ABC

Circuit diagram of four bit full Subtractor



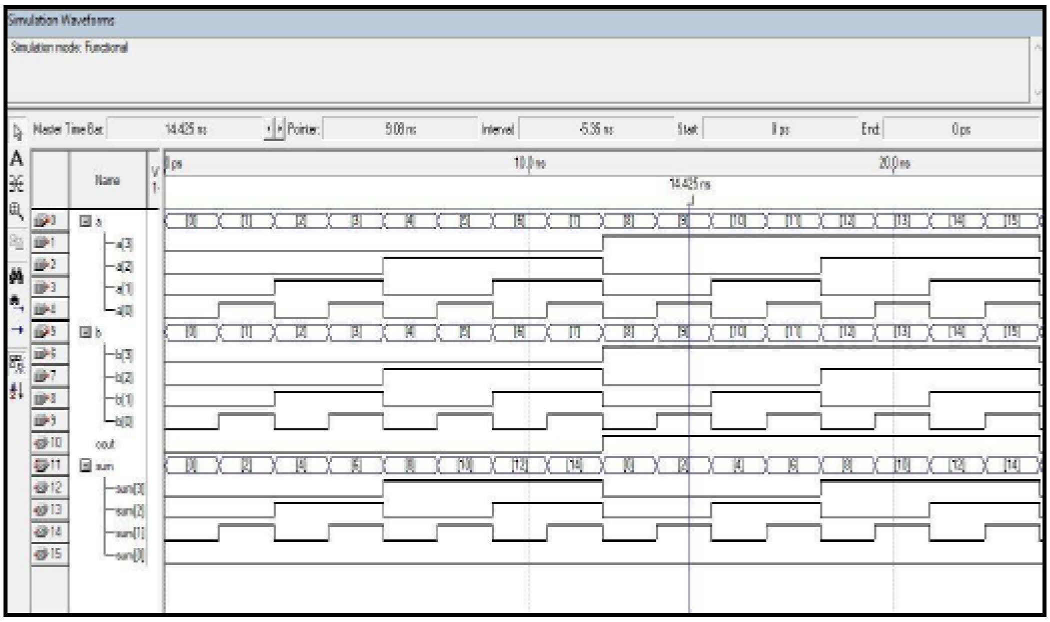
Verilog Program

```
module fs(a,b,c,d,bout);
        input a,b,c;
        output d,bout;
        assign d = ((!a&!b&c) | (!a&b&!c) | (a&!b&!c) | (a&b&c));
        assign bout = ((!a&!b&c) | (!a&b&!c) | (!a&b&c) | (a&b&c));
endmodule

module fourbitsub(a,bin,diff,bout);
        input [3:0]a;
        input [3:0]bin;
        output [3:0]diff;
        output bout;
        wire b1,b2,b3;

        fs g0(a[0],bin[3],0,diff[0].b1);
        fs g1(a[1],bin[2],b1,diff[1],b2);
        fs g2(a[2],bin[1],b2,diff[2],b3);
        fs g3(a[3],bin[0],b3,diff[3],bout);
endmodule
```

Output


Simulation Waveforms

Simulation mode: Functional

Master Time Bar: 14.425 ns Pointer: 5.08 ns Interval: -5.35 ns Start: 0 ps End: 0 ps

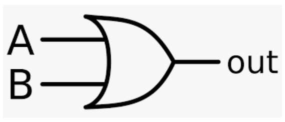
The four bit full adder and Subtractor is designed and simulated the same using basic gates

3. Design Verilog HDL to implement simple circuits using structural, Data flow and Behavioural model.

Design of a simple OR Gate

Truth table

A	B	Y=A + B
0	0	0
0	1	1
1	0	1
1	1	1



Structural model

```
module structural(a,b,y);
        input a,b;
        output y;
        or g1(y,a,b);
endmodule
```

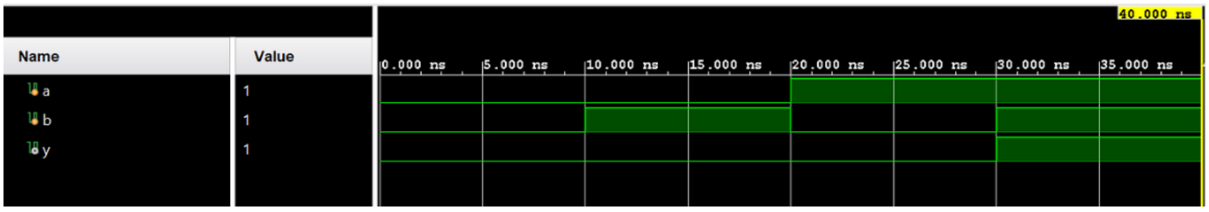
Dataflow model

```
module dataflow(a,b,y);
        input a,b;
        output y;
        assign y=a|b;
endmodule
```

Behavioural model

```
module beh(a,b,y);
        input a,b;
        output y;
        reg y;
        always @(a or b)
                begin
                        if((a==0)&&(b==0))
                                y=0;
                        else
                                y=1;
                end
endmodule
```

Output



The simple OR gate is implemented using Structural, Dataflow and Behavioural model

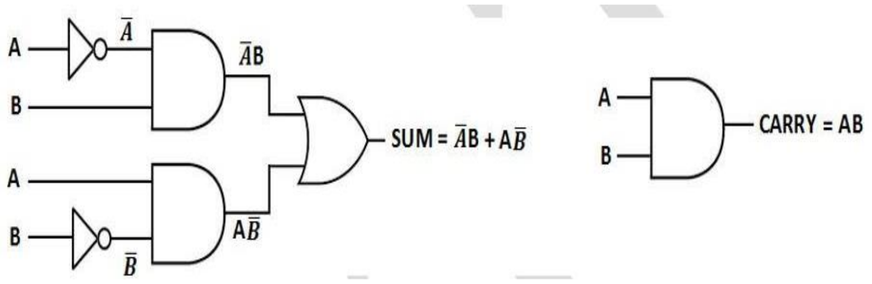
4. Design Verilog HDL to implement Binary Adder-Subtractor – Half and Full Adder, Half and Full Subtractor.

Design of Half Adder

INPUTS		OUTPUTS	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S=\bar{A}B+A\bar{B}=A\oplus B$$
$$C=A\,B$$

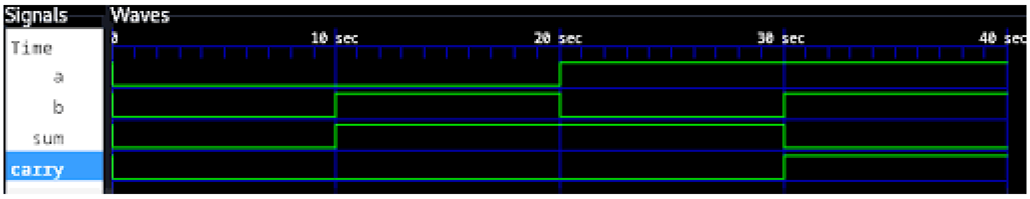
Circuit Diagram



Verilog Code

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

Output



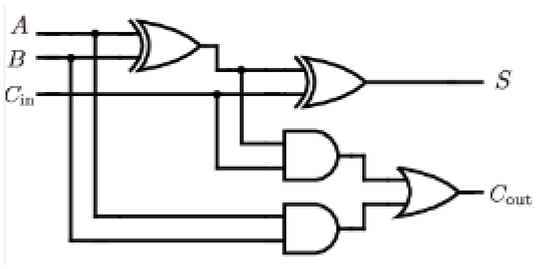
Design of Full adder

INPUTS			OUTPUTS	
A	B	Cin	S	C
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

$S = A \oplus B \oplus Cin$

$C = Cin\,(A \oplus B) + AB$

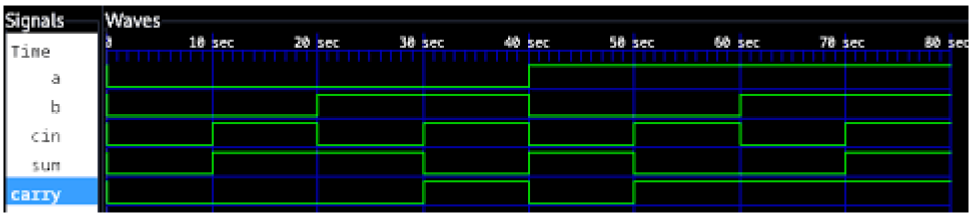
Circuit Diagram



Verilog Code

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

Output



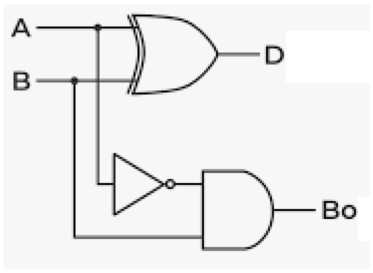
Design of Half Subtractor

A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$D=\overline{A}.B+A.\overline{B}$$

$$B_o=\overline{A}.B$$

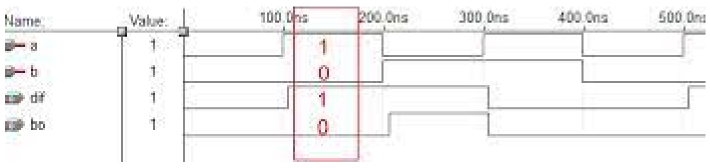
Circuit diagram



Verilog code

```
module hs(a,b,d,bout)
            input a,b;
            output d,bout;
            assign d=a^b;
            assign bout=!a&b;
endmodule
```

Output

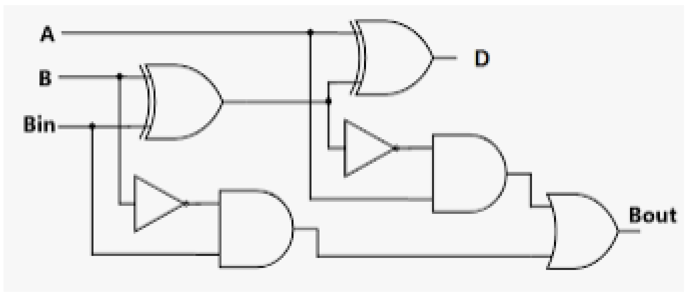


Design of full Subtractor

Inputs			Outputs	
A	B	Borrow_{in}	Diff	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	0
1	1	1	1	1

$$d = A \oplus B \oplus C$$
$$b = C(A \odot B) + \overline{A}B$$

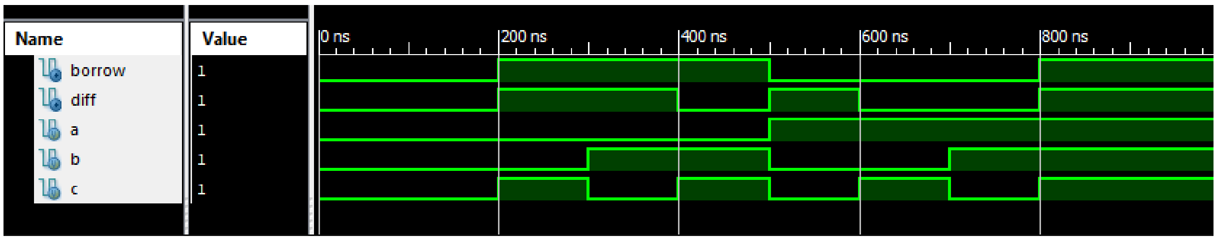
Circuit diagram



Verilog Code

```
module fs(a,b,bin,d,bout);
        input a,b,bin;
        output d,bout;
        assign d=a^b^bin;
        assign bout=(!a&(b^bin))|(b&bin);
endmodule
```

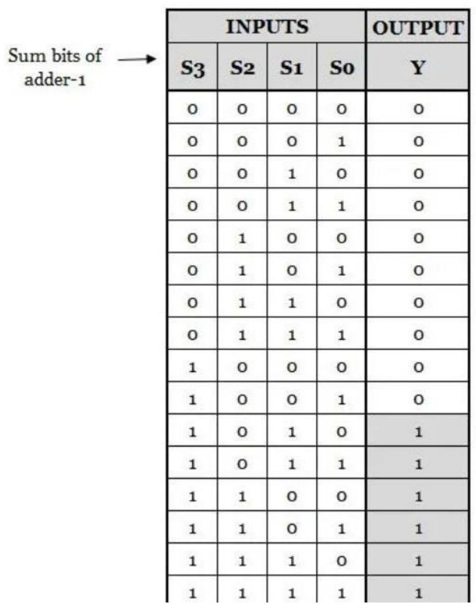
Output



The truth table of half adder, full adder, half Subtractor, and full Subtractor are verified.

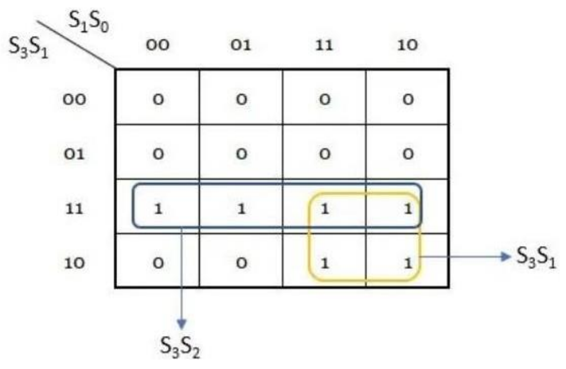
5. Design Verilog HDL to implement Decimal adder.

Design



Sum bits of adder-1

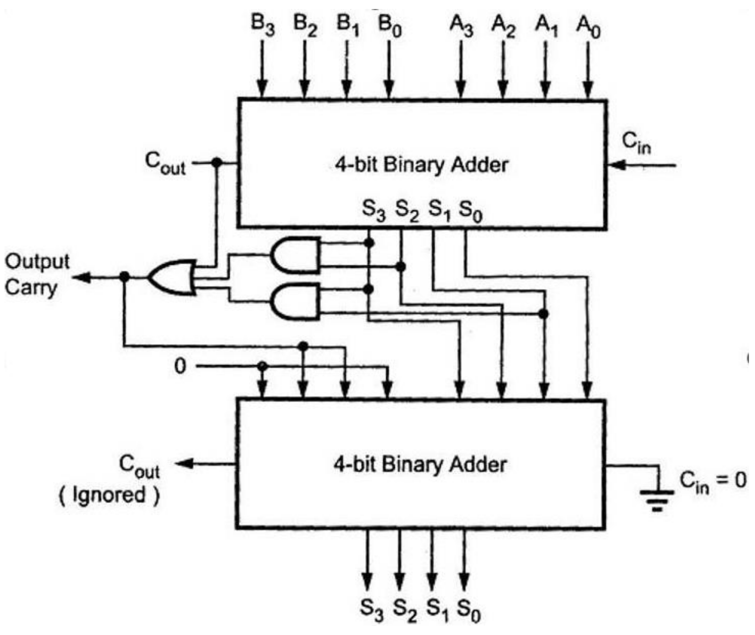
INPUTS				OUTPUT
S3	S2	S1	S0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



S_3S_1 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$$\mathbf{Y = S_3S_2 + S_3S_1}$$

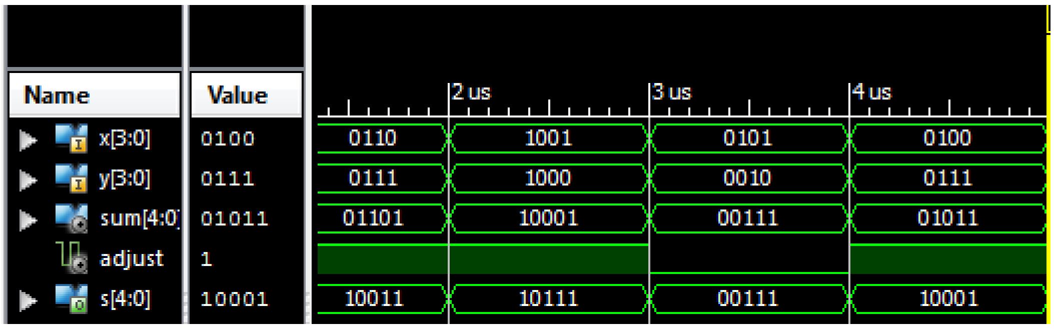
Circuit diagram



Verilog code

```
module bcd(a,b,carry_in,sum,carry);
//declare the inputs and outputs of the module with their sizes.
        input [3:0] a,b;
        input carry_in;
        output [3:0] sum;
        output carry;
//Internal variables
        reg [4:0] sum_temp;
        reg [3:0] sum; reg carry;
//always block for doing the addition
        always @(a,b,carry_in)
        begin
                sum_temp = a+b+carry_in; //add all the inputs
                if(sum_temp > 9)
                        begin
                                sum_temp = sum_temp+6; //add 6, if result is more than 9.
                                carry = 1; //set the carry output
                                sum = sum_temp[3:0];
                        end
                else
                        begin
                                carry = 0;
                                sum = sum_temp[3:0];
                        end
        end
endmodule
```

Output

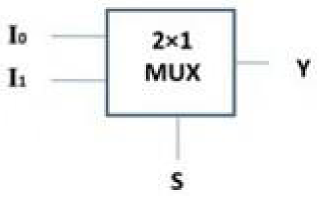

Name
Value
2 us
3 us
4 us
x[3:0] 0100 | 0110 | 1001 | 0101 | 0100
y[3:0] 0111 | 0111 | 1000 | 0010 | 0111
sum[4:0] 01011 | 01101 | 10001 | 00111 | 01011
adjust 1
s[4:0] 10001 | 10011 | 10111 | 00111 | 10001

Decimal Adder is implemented using Verilog code and simulated and its functioning correctly for all possible values.

6. Design Verilog program to implement Different types of multiplexer like 2:1, 4:1 and 8:1.

Design of 2:1 multiplexer

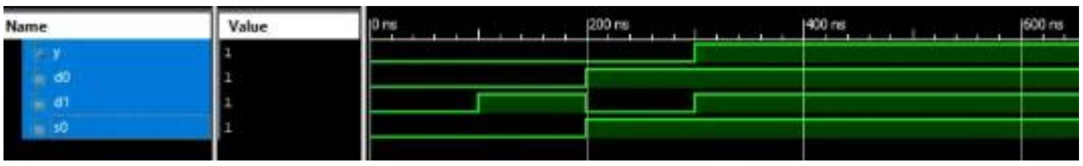
S	Y
0	I_0
1	I_1



Verilog code

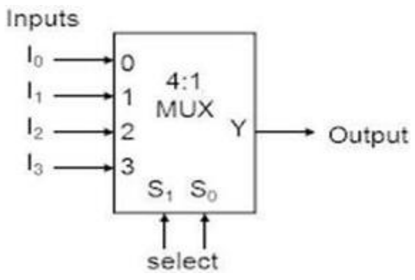
```
module mux2_1( I, sel, y);
        input [1:0] I;
        input sel;
        output y;
        reg y;
        always@ (sel , I)
        begin
                case (sel)
                        1'b0: y = I [0];
                        1'b1: y = I [1];
                endcase
        end
endmodule
```

Output



Design of 4:1 Multiplexer

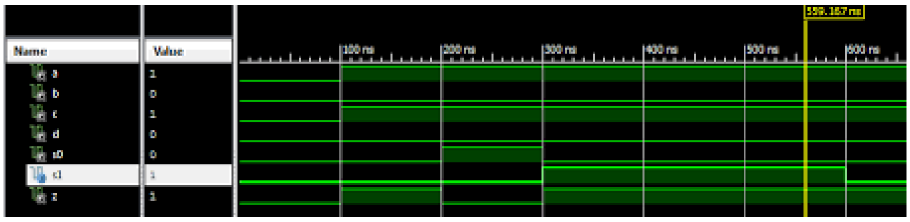
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



Verilog code

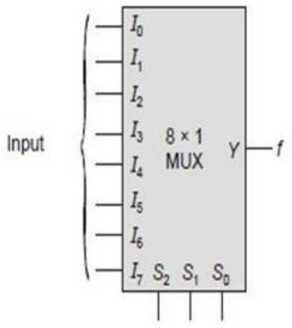
```
module mux_4_1 (I, sel, y);
        input [3:0] I;
        input [1:0] sel;
        output y;
        reg y;
        always@ (sel, I)
                begin
                        case (sel)
                                2'b00: y = I [0];
                                2'b01: y = I [1];
                                2'b10: y = I [2];
                                default: y = I [3];
                        endcase
                end
endmodule
```

Output



Design of 8:1 Multiplexer

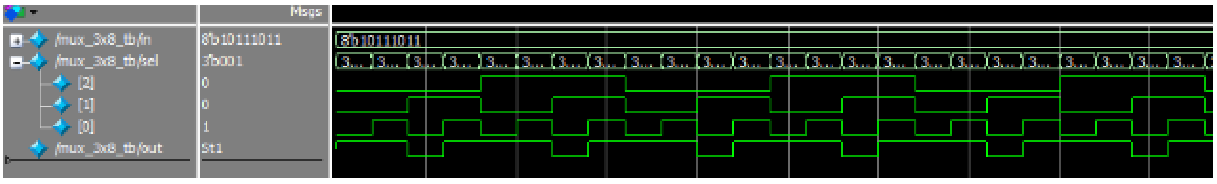
S_2	S_1	S_0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7



Verilog code

```
module mux_8_1 (I, sel, y);
        input [7:0] I;
        input [2:0] sel;
        output y;
        reg y;
        always@ (sel, I )
                begin
                        case (sel)
                                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];
                                default: y = I [7];
                        endcase
                end
endmodule
```

Output

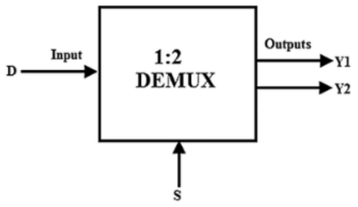


```
                       Msgs
/mux_3x8_tb/in         8'b10111011
/mux_3x8_tb/sel        3'b001
  [2]                  0
  [1]                  0
  [0]                  1
/mux_3x8_tb/out        St1
```

7. Design Verilog program to implement types of De-Multiplexer.

Design of 1:2 Demultiplexer

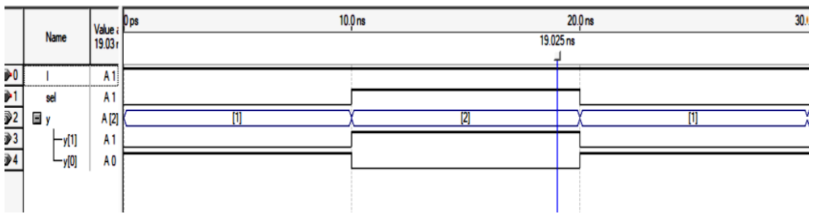
Select	Input	Outputs	
S	D	Y_2	Y_1
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0



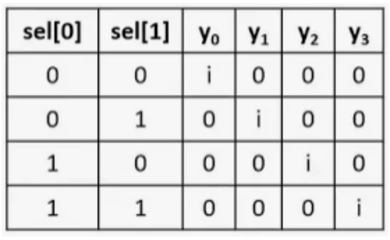
Verilog Code

```
module de_mux_1_2 (I, sel, y);
          input I;
          input sel;
          output reg [1:0] y;
          always@ (sel, I)
                    begin
                            case (sel)
                                    1'b0: begin y [0] = I; y [1] = 1'b0; end
                                    1'b1: begin y [0] = 1'b0 ; y [1] = I; end
                            endcase
                    end
endmodule
```

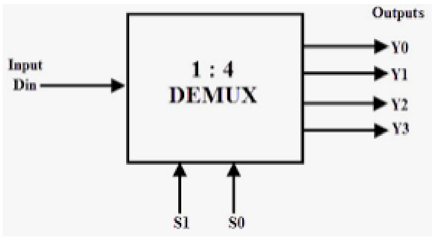
Output



Design of 1:4 demultiplexer



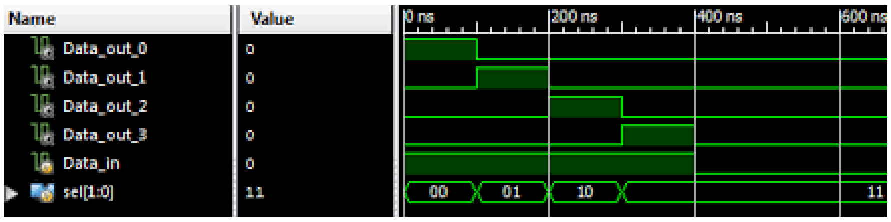
sel[0]	sel[1]	y_0	y_1	y_2	y_3
0	0	i	0	0	0
0	1	0	i	0	0
1	0	0	0	i	0
1	1	0	0	0	i



Verilog Code

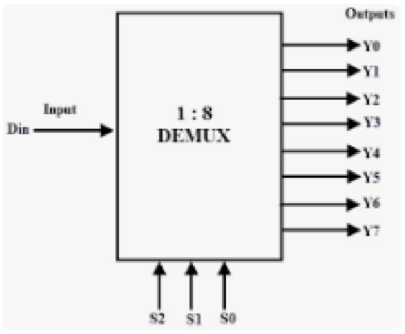
```
module de_mux_1_4 (I, sel, y);
        input I;
        input [1:0] sel;
        output reg [3:0] y;
        always@ (sel, I)
                begin case (sel)
                        2'b00: begin y [0] = I; y [1] = 0; y [2] = 0; y [3] = 0; end
                        2'b01: begin y [0] = 0; y [1] = I; y [2] = 0; y [3] = 0; end
                        2'b10: begin y [0] = 0; y [1] = 0; y [2] = I; y [3] = 0; end
                        2'b11: begin y [0] = 0; y [1] = 0; y [2] = 0; y [3] = I; end
                      endcase
                end
endmodule
```

Output



Design of 1:8 Demultiplexer

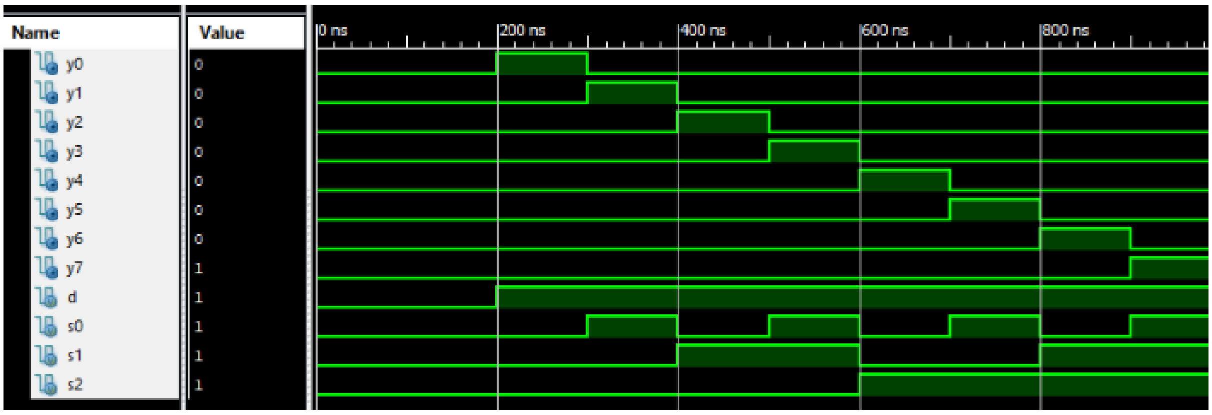
Input	Select lines			Output lines							
I	S_2	S_1	S_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
I	0	0	0	I	0	0	0	0	0	0	0
I	0	0	1	0	I	0	0	0	0	0	0
I	0	1	0	0	0	I	0	0	0	0	0
I	0	1	1	0	0	0	I	0	0	0	0
I	1	0	0	0	0	0	0	I	0	0	0
I	1	0	1	0	0	0	0	0	I	0	0
I	1	1	0	0	0	0	0	0	0	I	0
I	1	1	1	0	0	0	0	0	0	0	I



Verilog code

```
module de_mux_1_8 (I, sel, y);
        input I;
        input [2:0] sel;
        output reg [7:0] y;
        always@(sel ,I)
                case (sel)
                 3'b000:begin y[0] = I; y[1] = 0 ;y[2] = 0;y[3] = 0 ;y[4] = 0;y[5] = 0;y[6] = 0;y[7] = 0;end
                 3'b001:begin y[0] = 0; y[1] = I; y[2] = 0;y[3] = 0;y[4] = 0;y[5] = 0;y[6] = 0;y[7] = 0;end
                 3'b010:begin y[0] = 0; y[1] = 0;y[2] = I; y[3] = 0;y[4] = 0; y[5] = 0;y[6] = 0;y[7] = 0;end
                 3'b011:begin y[0] = 0;y [1] = 0; y[2] = 0;y[3] = I ;y[4]= 0;y[5] = 0 ;y[6] = 0;y[7] = 0;end
                 3'b100:begin y[0] = 0; y[1] = 0;y[2] = 0;y[3] = 0;y[4] = I; y[5] = 0;y[6] = 0;y[7] = 0; end
                 3'b101:begin y[0] = 0; y[1] = 0;y[2] = 0;y[3] = 0;y[4] = 0;y[5] = I; y[6] = 0;y[7] = 0; end
                 3'b110:begin y[0] = 0; y[1] = 0;y[2] = 0;y[3] = 0;y[4] = 0;y[5] = 0;y[6] = I; y[7] = 0; end
                 3'b111:begin y[0] = 0; y[1] = 0;y[2] = 0;y[3] = 0;y[4] = 0;y[5] = 0;y[6] = 0;y[7] = I; end
                 default: y = 8'bxxxxxxx;
                endcase
endmodule
```

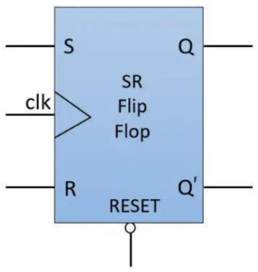
Output



The truth table of different types of de-mux 2:1, 4:1 and 8:1 are verified and simulated.

8. Design Verilog program for implementing various types of Flip-Flops such as SR, JK, and D.

Design of SR Flipflop



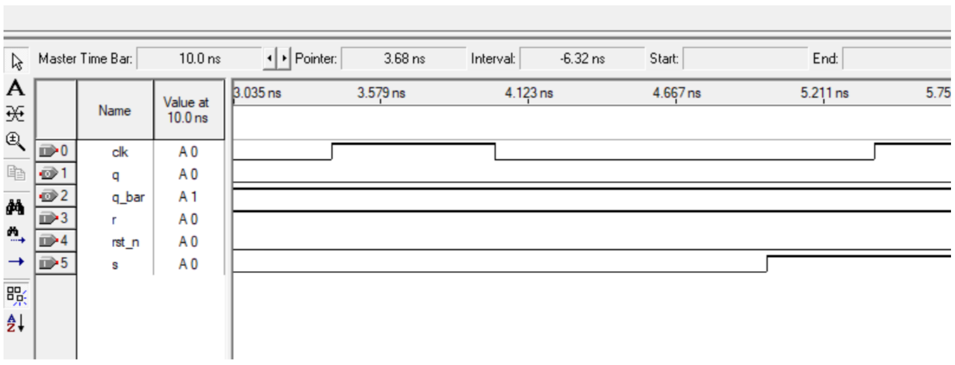
SR Flip Flop

S	R	Q_{n+1}
0	0	Q_n(No Change)
0	1	0
1	0	1
1	1	x

Verilog Code

```
module sr(input clk, rst_n, input s,r, output reg q, output q_bar);
//always@(posedge clk or negedge rst_n) // for asynchronous reset
always@(posedge clk) begin // for synchronous reset
        if(!rst_n) q <= 0;
        else begin
                case({s,r})
                  2'b00: q <= q; // No change
                  2'b01: q <= 1'b0; // reset
                  2'b10: q <= 1'b1; // set
                  2'b11: q <= 1'bx; // Invalid inputs
                endcase
        end
        end
assign q_bar = ~q;
endmodule
```

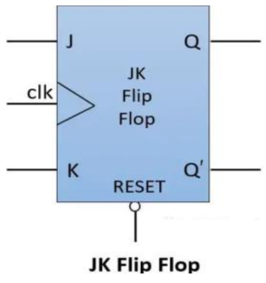
Output



Master Time Bar: 10.0 ns Pointer: 3.68 ns Interval: -6.32 ns Start: End:

	Name	Value at 10.0 ns
0	clk	A 0
1	q	A 0
2	q_bar	A 1
3	r	A 0
4	rst_n	A 0
5	s	A 0

Design of JK Flipflop



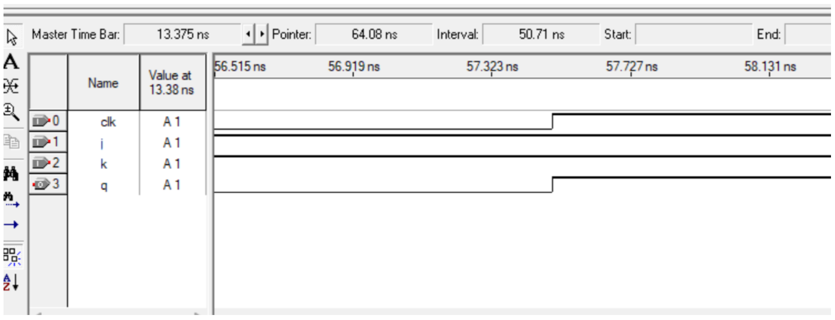
JK Flip Flop

J	K	Q_{n+1}
0	0	Q_n(No Change)
0	1	0
1	0	1
1	1	$\overline{Q_n}$(Toggles)

Verilog code

```
module jk( input j, input k, input clk, output reg q);
always @ (posedge clk)
        case ({j,k})
                2'b00 : q <= q;
                2'b01 : q <= 0;
                2'b10 : q <= 1;
                2'b11 : q <= ~q;
        endcase
endmodule
```

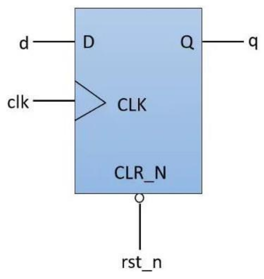
Output



Master Time Bar: 13.375 ns Pointer: 64.08 ns Interval: 50.71 ns Start: End:

	Name	Value at 13.38 ns
0	clk	A 1
1	j	A 1
2	k	A 1
3	q	A 1

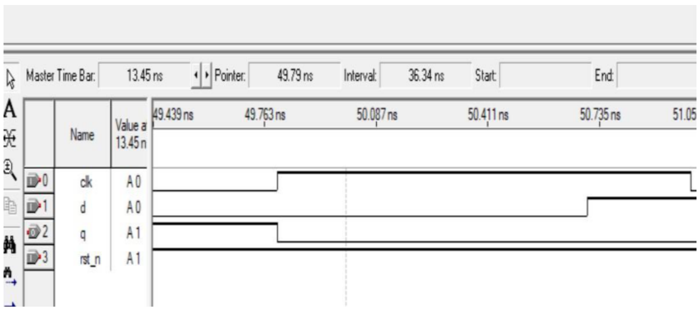
Design of D- Flipflop



Verilog code:

```
module dd (
input clk, rst_n, input d,
output reg q
);
always@(posedge clk or negedge rst_n)
begin
        if(!rst_n) q <= 0;
        else q <= d;
        end
endmodule
```

Output


Master Time Bar: 13.45 ns
Pointer: 49.79 ns
Interval: 36.34 ns
Start:
End:
Name
Value at 13.45 ns
49.439 ns
49.763 ns
50.087 ns
50.411 ns
50.735 ns
51.05
0 clk A 0
1 d A 0
2 q A 1
3 rst_n A 1