

**ACHARYA INSTITUTE OF TECHNOLOGY
BANGALORE – 560107**

Approved by AICTE, New Delhi,
Affiliated to VTU, Belagavi & Recognized by Government of
Karnataka



**Laboratory Manual
on
DIGITAL DESIGN & COMPUTER ORGANIZATION
(BCS302)**

Prepared by
Ms. Y V KALYANI
Assistant Professor,
Department of Artificial Intelligence & Machine Learning

Vision of the Institute

Acharya Institute of Technology, committed to the cause of sustainable value-based education in all disciplines, envisions itself as a global fountainhead of innovative human enterprise, with inspirational initiatives for Academic Excellence.

Mission of the Institute

Acharya Institute of Technology strives to provide excellent academic ambience to the students for achieving global standards of technical education, foster intellectual and personal development, meaningful research, ethical, and sustainable service to societal needs.

Vision of the Department

To establish as a centre of excellence in moulding young professionals in the domain of Artificial Intelligence and Machine Learning who are capable of contributing to the wellness of the society as well as the industry.

Mission of the Department

1. To nurture the students with multi-disciplinary skills to be able to build sustainable solutions to engineering problems which in turn helps them to grab dynamic and promising career in the industry.
2. To mould energetic and committed engineers by inculcating moral values of professional ethics, social concerns, environment protection, sustainable solutions and life-long learning.
3. To facilitate research cooperation in the fields of artificial intelligence, machine learning, and data analytics by handholding with national and international research organizations, thus enabling the students to analyse and apply various levels of learning.

Program Educational Objectives (PEOs)

Engineering Graduates will

1. Have a strong foundation in key concepts and techniques related to artificial intelligence and its related fields, thus enabling them to analyze complex problems in various domains and apply AI and ML techniques to develop innovative and effective solutions.
2. Be conscious of the ethical implications of AI/ML technologies, be aware of potential biases and ensure fairness, transparency, and accountability and demonstrate a commitment to develop AI/ML applications for the betterment of society.
3. Possess effective communication skills and be capable of working collaboratively in multidisciplinary teams to address real-world challenges.
4. Have an entrepreneurial mindset and be equipped with the skills and knowledge to create innovative AI/ML solutions and potentially start their own ventures.
5. Demonstrate a commitment to lifelong learning and stay updated with advancements in AI and ML technologies and industry trends to emerge as leaders in their fields of expertise and domain that support service and economic development of the society.

Program Specific Outcomes (PSOs)

Engineering Graduates will be able to:

1. Apply the Artificial Intelligence and Data Analytics skills in the areas of Disaster Management, Health Care, Education, Agriculture, Transportation, Environment, Society and in other multi-disciplinary areas.
2. Analyse and demonstrate the knowledge of Human cognition, Artificial Intelligence, Machine Learning and Data engineering in terms of real-world problems to meet the challenges of the future.
3. Develop computational knowledge and project management skills using innovative tools and techniques to solve problems in the areas related to Deep Learning, Machine learning, Artificial Intelligence which will lead to lifelong learning.

Program Outcomes (POs)

Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. 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.
3. 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.
4. 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.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. 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.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. 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.
11. Project management and finance: Demonstrate knowledge and understanding of the 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 multidisciplinary environments.
12. 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.

Syllabus

Subject: DDCO

Subject Code: BCS302

Course Objectives

The objectives of this course are to make students to learn-

- 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

List of Experiments

Sl.N O	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

COS	Description
BCS302.6	Apply the fundamental concepts of Logical gates and design expressions.

BCS302.7	Apply fundamental concepts of flip flops and develop digital circuits
BCS302.8	Develop and Simulate Combinational circuit using Verilog HDL.

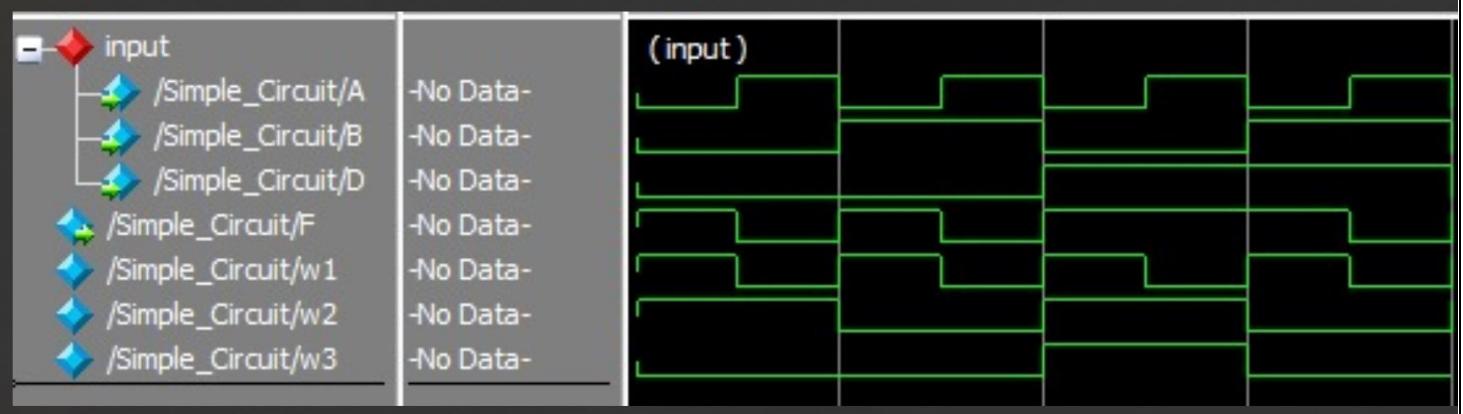
PROGRAM-1:

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

$$F(A, B, C, D) = \Sigma M(0, 1, 2, 3, 4, 5, 6, 7, 9, 11) + \Sigma d(8, 14)$$

A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	X
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	X
1	1	1	1	0

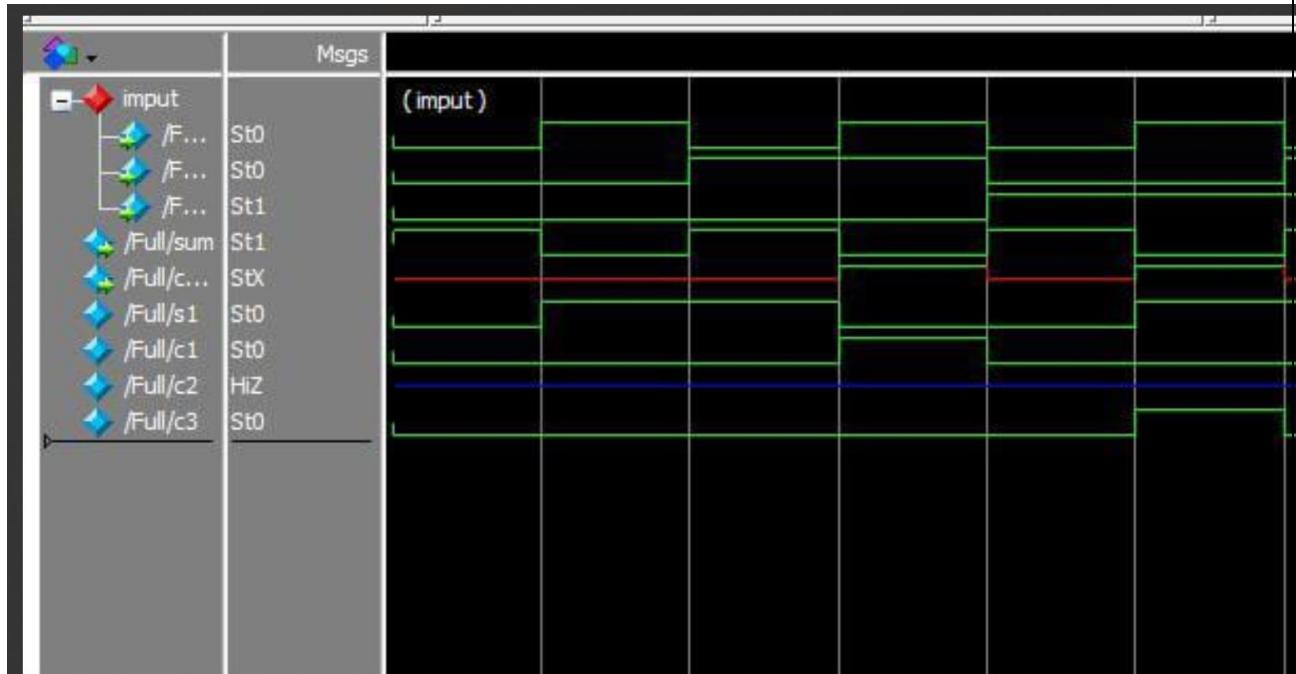
Simplified expression if $F(A, B, C, D) = A' + B'D$



PROGRAM 2:

- 1 . Design a 4 bit full adder and subtractor and simulate the same using basic gates

```
module FullAdder(a,b,cin,sum,cout);
    input a,b,cin;
    output wire sum,cout;
    wire s1,c1,c2,c3;
    xor(s1,a,b);
    xor(sum,s1,cin);
    and(c1,a,b);
    and(c2,b,cin);
    and(c3,a,cin);
    or(cout,c1,c2,c3);
endmodule
```



```
module FourBitSubtractor(
    input [3:0] A, // 4-bit input A
```

```

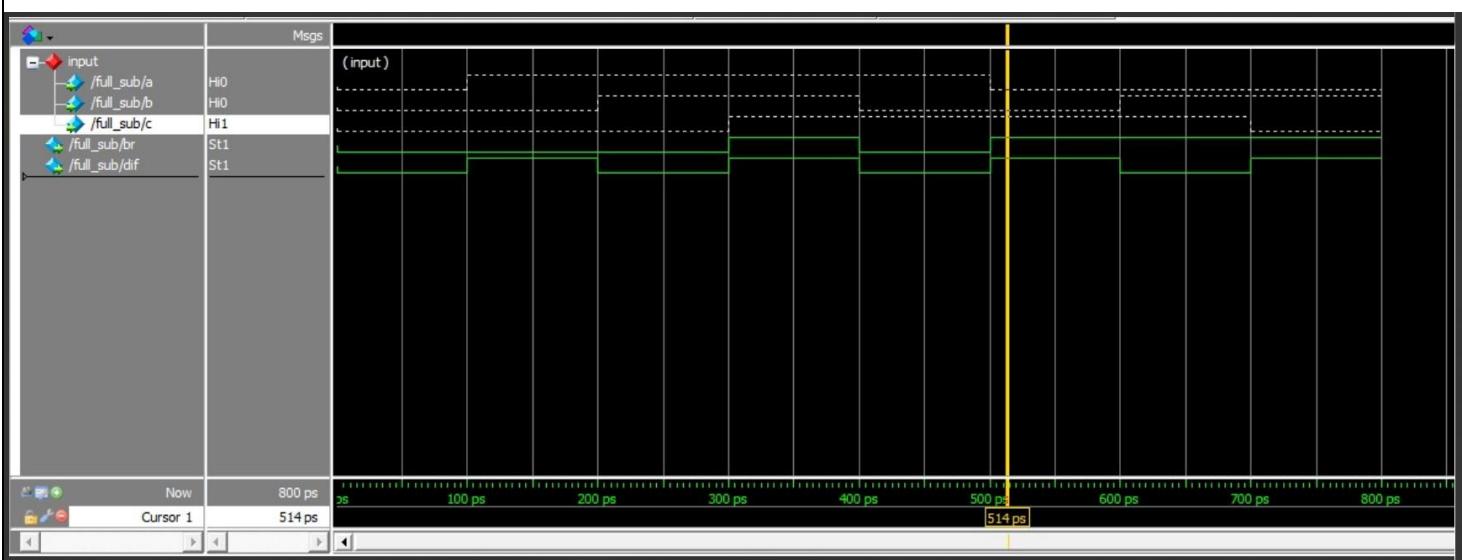
    input [3:0] B, // 4-bit input B
    output [3:0] Difference, // 4-bit output Difference
    output Bout // Borrow-out
);

wire [3:0] B_neg;
wire [4:0] Borrow;
wire [4:0] Sum;

assign B_neg = ~B + 1; // Two's complement of B
assign {Sum[0], Borrow} = A + B_neg;
assign Difference = Sum[3:0];
assign Bout = Borrow[4];

endmodule

```



PROGRAM 3:

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

Structural

```

module AND_2(output Y, input A, B);
and(Y, A, B);
endmodule

```

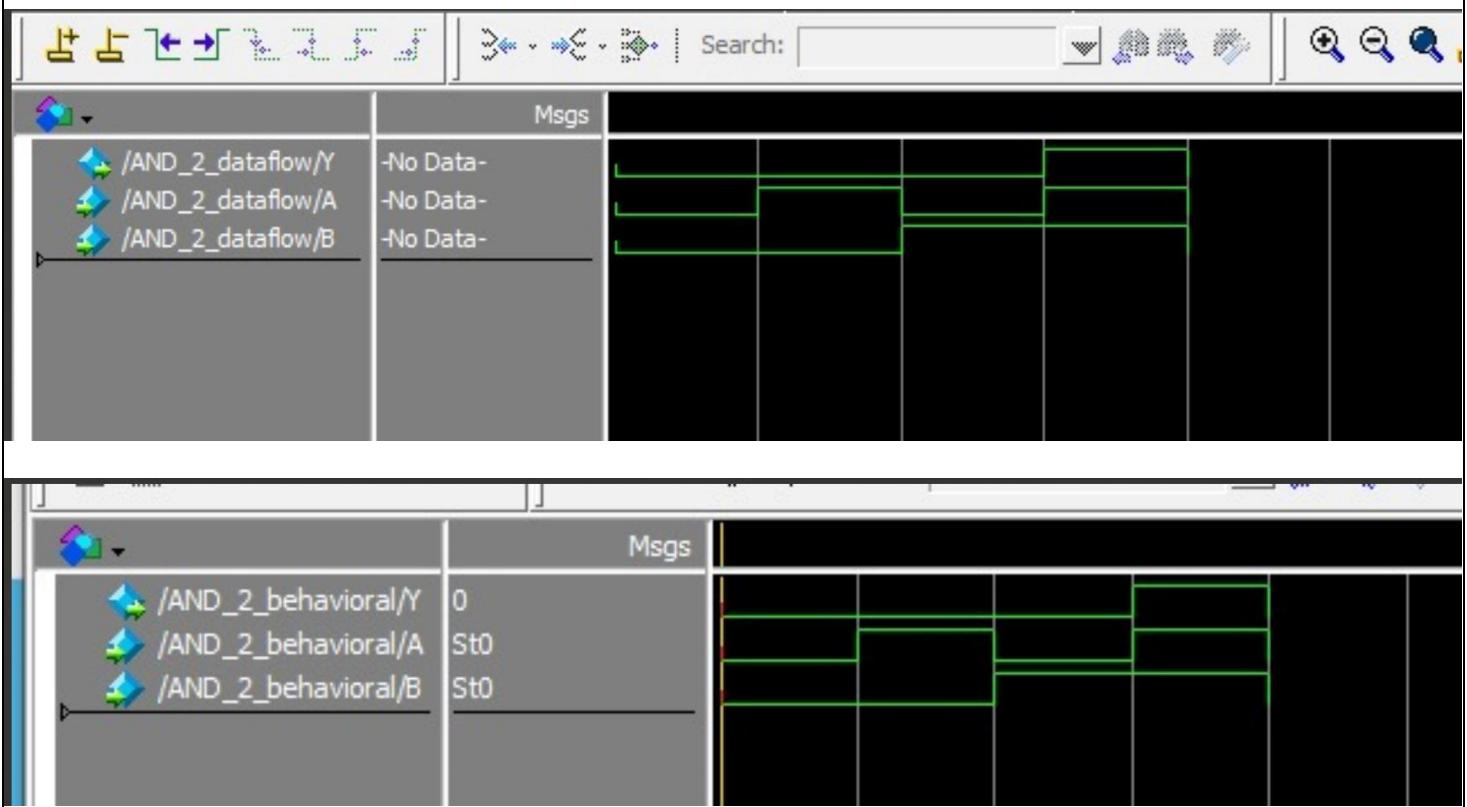
dataflow

```
module AND_2_data_flow (output Y, input A, B);  
  
assign Y = A & B;  
  
endmodule
```

behavioral

```
module AND_2_behavioral (output reg Y, input A, B);  
  
always @ (A or B) begin  
  
if (A == 1'b1 & B == 1'b1) begin  
  
Y = 1'b1;  
  
end  
  
else  
  
Y = 1'b0;  
  
end  
  
endmodule
```





PROGRAM 4:

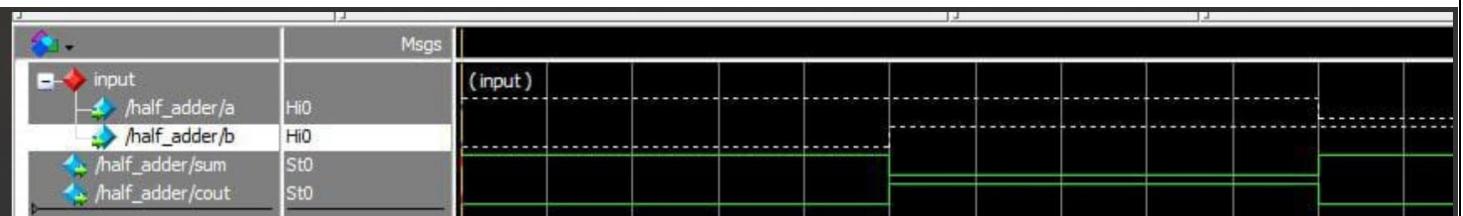
1. Design Verilog HDL to implement Binary Adder-Subtractor – Half and Full Adder, Half and Full Subtractor.

```
Half adder:
module half_adder(
input a,b,
output sum,cout
);

assign sum = a ^ b ;
assign cout = a & b;

endmodule
```

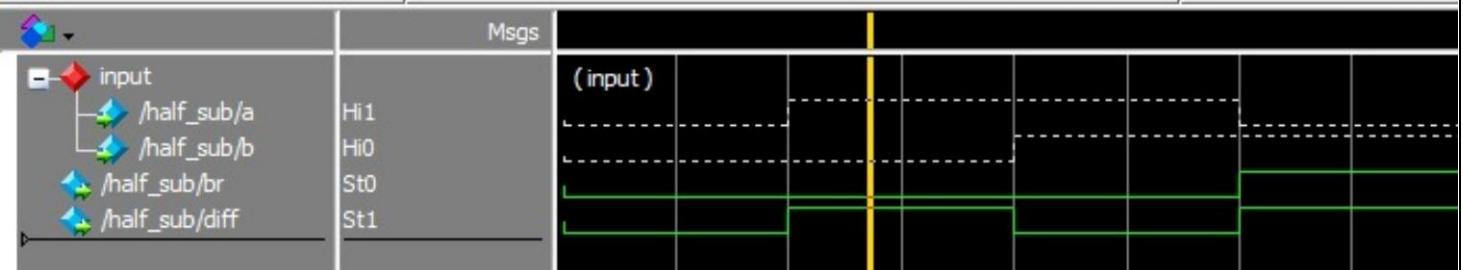
Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Half subtractor:

```
module half_sub(
input a,b,
output br,dif
);
assign dif = a ^ b;
assign br = ~a & b;
endmodule
```

Inputs		Outputs	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



Full adder:

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

$$c = ab + bc + ac$$

$$s = a \oplus b \oplus c$$

Inputs			Outputs	
A	B	C _{in}	Sum	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



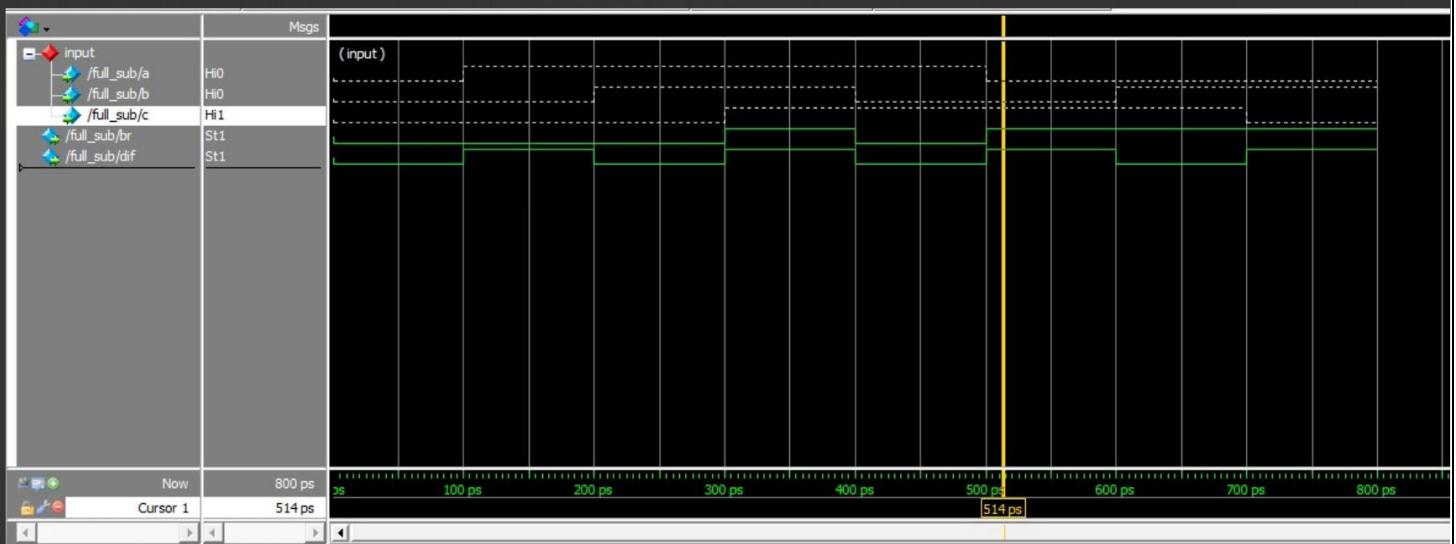
Full subtractor

```
module full_sub(
input a,b,c,
output br,dif
);

assign dif = (a ^ b ^ c );
assign br = (~a & b ) | (b & c) | (~a & c);

endmodule
```

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



PROGRAM 5:

1. Design Verilog HDL to implement Decimal adder

```
//module declaration with inputs and outputs
module bcd_adder(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
```



PROGRAM 6:

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

2:1 MUX

```
module m21(D0, D1, S, Y);

output Y;

input D0, D1, S;

assign Y=(D0 & ~S) | (D1 & S);

endmodule
```

OR

```
assign Y = (S) ? D1:D0;
```

S	Y
0	D0
1	D1



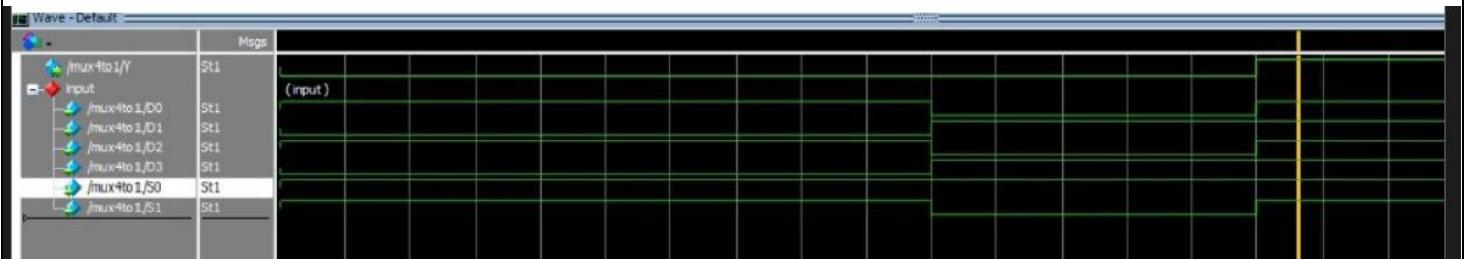
4:1 MUX

```
module mux4to1(output Y, input D0, D1, D2, D3, S0, S1);

assign Y = (D0 & ~S0 & ~S1) | (D1 & ~S0 & S1) | (D2 & S0 & ~S1) | (D3 & S0 & S1);

endmodule
```

S0	S1	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3



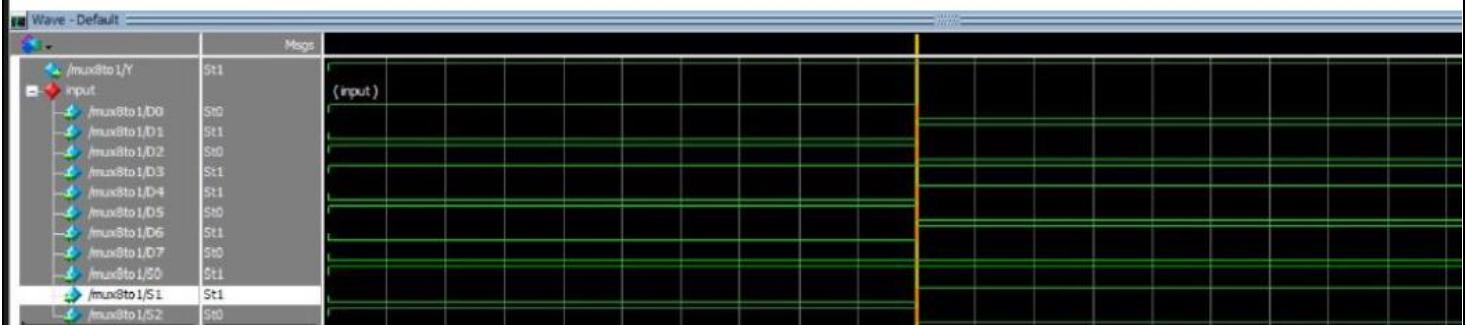
8:1MUX

```
module mux8to1(output Y, input D0, D1, D2, D3, D4, D5, D6, D7, S0, S1, S2);

assign Y = (D0 & ~S0 & ~S1 & ~S2) | (D1 & ~S0 & ~S1 & S2) | (D2 & ~S0 & S1 & ~S2) | (D3 & ~S0 & S1 & S2) | (D4 & S0 & ~S1 & ~S2) | (D5 & S0 & ~S1 & S2) | (D6 & S0 & S1 & ~S2) | (D7 & S0 & S1 & S2);

endmodule
```

S0	S1	S2	Y
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7



PROGRAM 7:

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

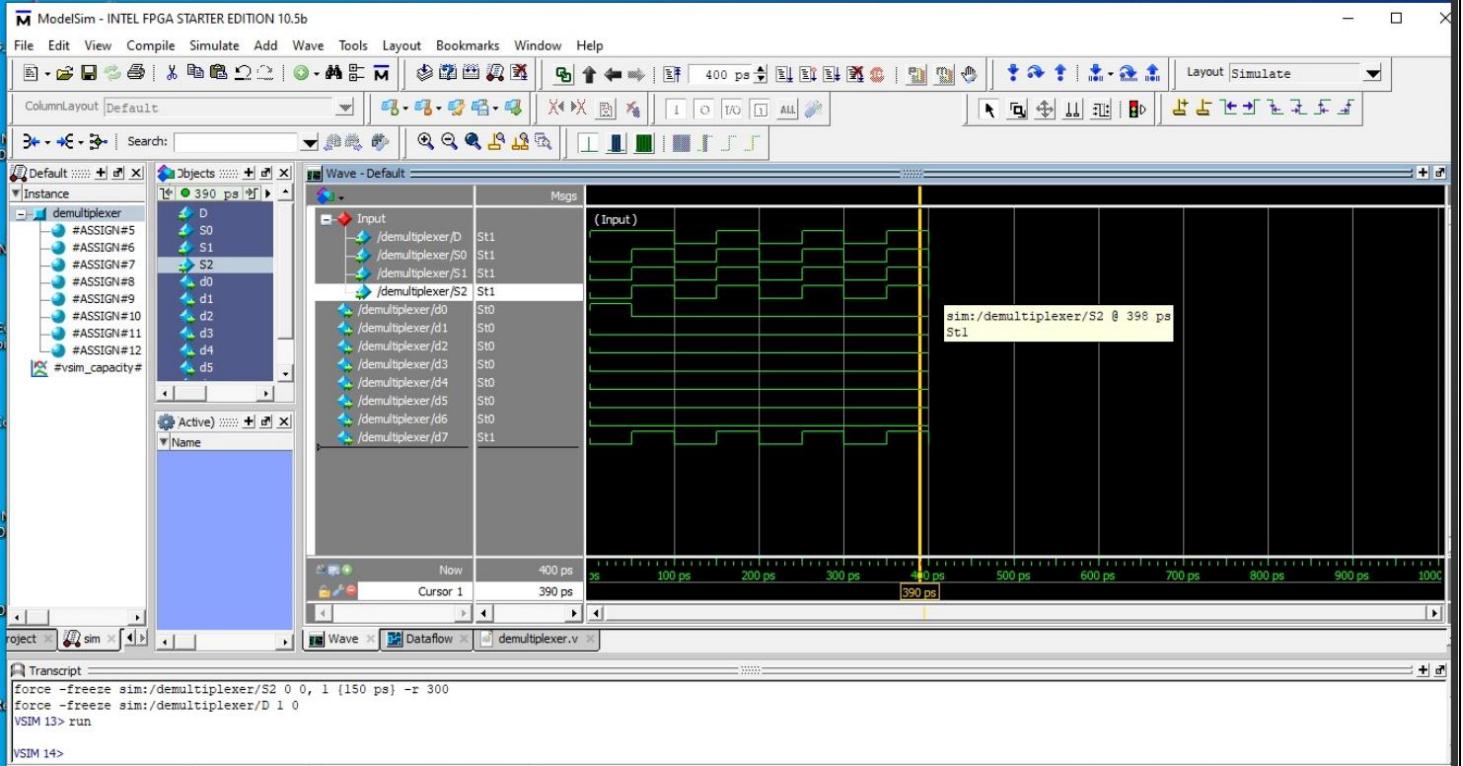
1:8 DEMUX

```

module Demultiplexer(D,S0,S1,S2,d0,d1,d2,d3,d4,d5,d6,d7);
input D,S0,S1,S2;
output d0,d1,d2,d3,d4,d5,d6,d7;
assign d0 = (D & ~S0 & ~S1 & ~S2),
d1=(D & ~S0 & ~S1 & S2),
d2=(D & ~S0 & S1 & ~S2),
d3=(D & ~S0 & S1 & S2),
d4=(D & S0 & ~S1 & ~S2),
d5=(D & S0 & ~S1 & S2),
d6=(D & S0 & S1 & ~S2),
d7=(D & S0 & S1 & S2);
endmodule

```

I/P	S0	S1	S2	O/P
D	0	0	0	D0
D	0	0	1	D1
D	0	1	0	D2
D	0	1	1	D3
D	1	0	0	D4
D	1	0	1	D5
D	1	1	0	D6
D	1	1	1	D7



1:4 demux

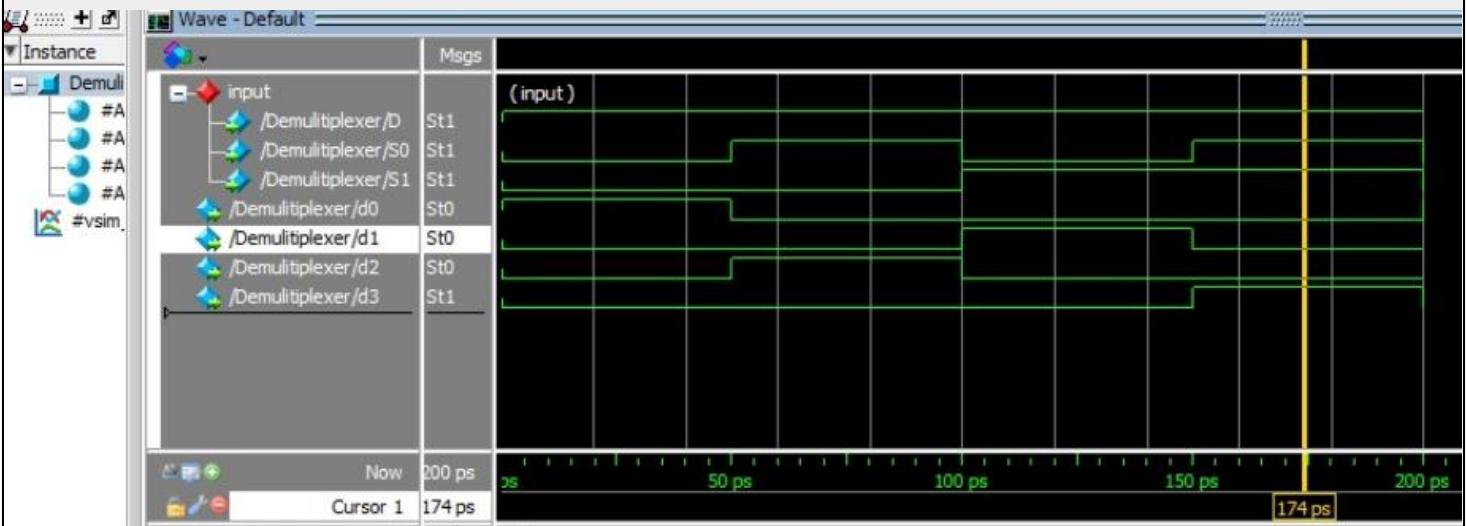
```

module Demultiplexer(D,S0,S1,d0,d1,d2,d3);
input D,S0,S1;
output d0,d1,d2,d3;
assign d0 = (D & ~S0 & ~S1),
d1=(D & ~S0 & S1),
d2=(D & S0 & ~S1),
d3=(D & S0 & S1);
endmodule

```

I/P	S0	S1	O/P
D	0	0	D0
D	0	1	D1
D	1	0	D2

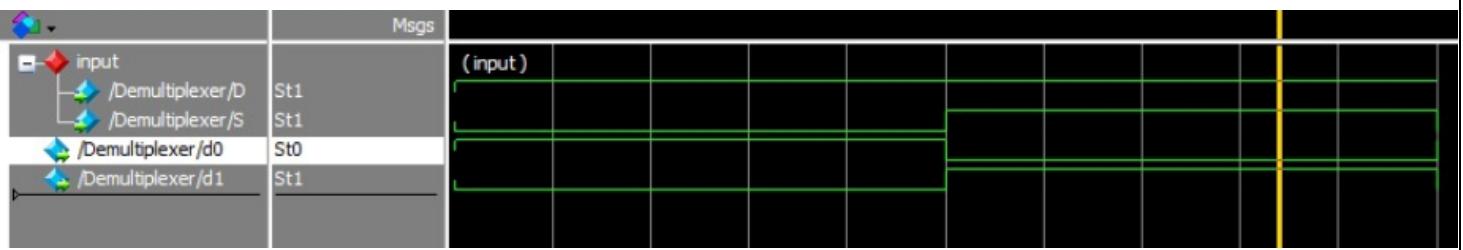
D	1	1	D3
---	---	---	----



1:2 demUX

```
module Demultiplexer(D,S,d0,d1);
input D,S;
output d0,d1;
assign d0 = (D & ~S),
d1=(D & S);
endmodule
```

I/P	S	O/P
D	0	D0
D	1	D1



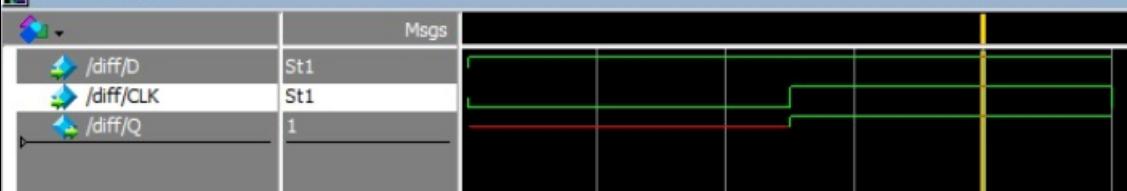
PROGRAM 8:

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

```
D Flip flop(POSITIVE EDGE TRIGRRED FLIP FLOP)
```

```
module dff(D,CLK,Q);
input D, CLK;
output Q;
reg Q;
always @ (posedge CLK)
begin
    Q <= D;
end
endmodule
```

CLK	D	Q	COMMENTS
↑	D	D	Q=D
X	D	Q	NO CHANGE



SR FLIPFLOP

```
module srff(clk,s,r,q);
input clk,s,r;
output q;
reg q;

always @ (posedge clk)
begin
    case({s,r})
        2'b00:

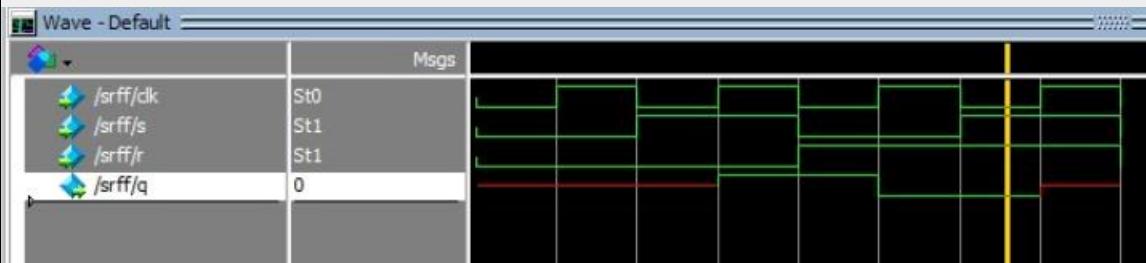
```

```

q <= q;
2'b01:
q <= 0;
2'b10:
q <= 1;
2'b11:
q <= 1'bx;
default:
q <= q;
endcase
end
endmodule

```

clk	S	r	q	COMMENTS
↑	0	0	Q	NO CHANGE
↑	0	1	0	RESET
↑	1	0	1	SET
↑	1	1	x	FORBIDDEN



JK FLIP FLOP

clk	j	k	q	COMMENTS
↑	0	0	Q	NO CHANGE
↑	0	1	0	0
↑	1	0	1	1
↑	1	1	q'	TOGGLE

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

endmodule
```

