Laboratory Report Cover Sheet

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
College of Engineering and Technology
Department of Electronics and Communication Engineering

18ECC206J VLSI Design
VI Semester, 2022-2023 (Even Semester)

Name :

Register No. :

Title of Experiment : REALIZATION OF COMBINATIONAL AND SEQUENTIAL CIRCUITS USING GATE LEVEL AND DATAFLOW MODELING

Date of Conduction:

Date of Submission:

Particulars	**Max. Marks**	**Marks Obtained**
Pre-Lab and Post Lab	10	
Design, HDL Code, In-Lab Performance	15	
Output verification &viva	10	
Lab Report	05	
Total	**40**	

REPORT VERIFICATION

Staff Name :

Signature :

Lab Experiment 1

Realization of Combinational and Sequential circuits using Gate-Level and Dataflow modeling

1.1 Objective: To learn the design of combinational and sequential circuits using GateLevel and Dataflow modeling in Verilog then simulating and synthesizing using EDA tools

1.2 Software tools Requirement:

Equipment's:
Computer with Xilinx and Modelsim Software Specifications:
HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk
Softwares: Synthesis tool: Xilinx ISE
Simulation tool: ModelSim Simulator

1.3 Prelab Questions

(write pre lab Q & A in an A4 sheet)

1.List the types of design methodologies for digital design with an example?
2. Give the difference between module and module instance.
3. What are built in gate primitivies?
4. Give the use of net, reg and wire data types.
5. Declare the following variables in Verilog:
a. An 8-bit vector net called *a_in*.
b. An integer called *count*.
c. An array called delays. Array contains 20 elements of the type integer.
d. A parameter cache_size equal to 512.

1.4.1 Problem 1: Write a Verilog code to implement Ripple Carry Adder.

The following points should be taken care of:

1. Use assign statement to design sub module half adder
2. Use gate primitives to design full adder using half adder
3. Construct the top module 4-bit ripple carry adder using 1-bit full adder

1.4.2 Problem 2: Write a Verilog code to implement Multiplexer

The following points should be taken care of:

1. Use data flow model to design 2x1 Multiplexer
2. Use conditional statement to design 4x1 Multiplexer
3. Use a gate level model to design 8x1 Mux using two 4x1 Mux and one 2x1 Mux.

1.4.3 Problem 3: Write a Verilog code to implement Flip flops.

The following points should be taken care of:

1. Use a gate level model to design a positive edge triggered SR flip flop.
2. Use Gate-level model to design positive edge triggered JK flip flop.

1.5 Post lab

1. Draw the block diagram of 4x16 decoder. Use 3x8 decoder to design 4x16 decoder using gate level model.

Problem 1:

Program (i): Use assign statement to design sub module half adder

Half adder:

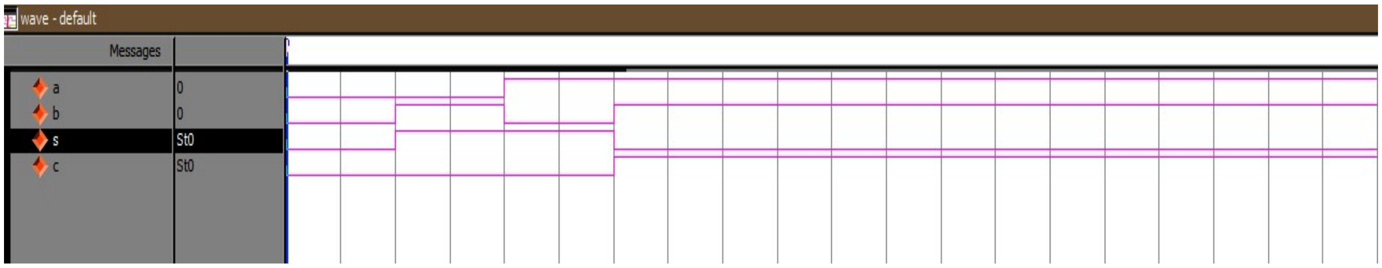
```
module ha_df (s,c,a,b);
input a,b;
output s, c;
assign s-a^b;
assign c-a&b;
endmodule
```

Half adder: TESTBENCH

```
module ha_tb_v;
// Inputs
reg a;
reg b;
// Outputs
wire s;
wire c;
// Instantiate the Unit Under Test (UUT)
ha uut (
.s(s),
.c(c),
.a(a),
.b(b)
);
initial begin
// Initialize Inputs
a = 0;
b = 0;
// Wait 100 ns for global reset to finish
#100; a=0; b=1;
    #100; a=1; b=0;
#100; a=1; b=1;
// Add stimulus here
end
 endmodule
```

Waveform:

Halfadder:



Program (ii): Use gate primitives to design full adder using half adder

Full adder:

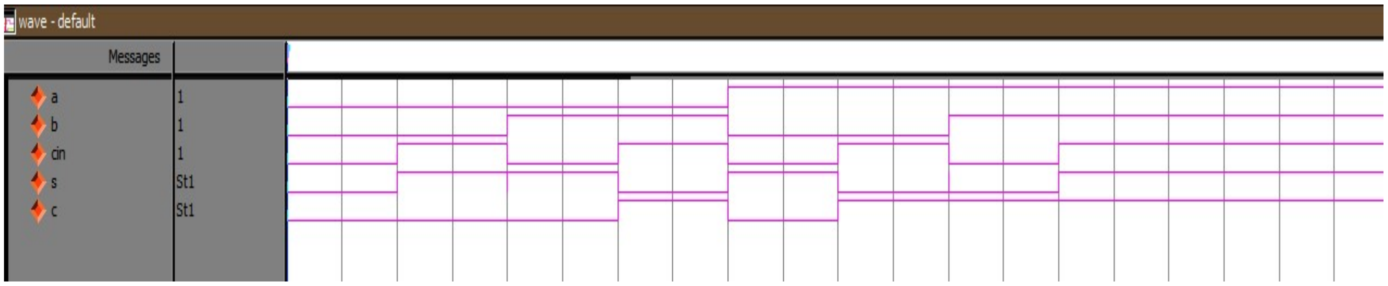
```
module fa(s,c,a,b,cin);

input a, b, cin ;

output s,c;

wire sl, c1,c2;
ha_df h1 (s1,c1,a,b);

ha_df h2 (s,c2,s1,cin);

endmodule
```

Full adder: TESTBENCH

```
module fa_tb_v;
// Inputs
reg a;
reg b;
reg cin;
// Outputs
wire s;
wire c;
// Instantiate the Unit Under Test (UUT)
fa uut (
.s(s),
.c(c),
.a(a),
.b(b),
.cin(cin)
);
initial begin
// Initialize Inputs
a = 0;
b = 0;
cin = 0;
// Wait 100 ns for global reset to finish
#100;    a = 0; b = 0; cin = 1;
    #100;         a = 0; b = 1; cin = 0;
```

```
#100;    a = 0; b = 1; cin = 1;
#100;    a = 1; b = 0; cin = 0;
#100;    a = 1; b = 0; cin = 1;
#100;    a = 1; b = 1; cin = 0;
#100;    a = 1; b = 1; cin = 1;
// Add stimulus here
End
```

Waveform:
Full adder:



Program (iii): Construct the top module 4-bit ripple carry adder using 1-bit full adder

<u>Ripple Carry Adder:</u>

```
module rca(s,c,a,b,cin);

input[3:0]a,b;

input cin;

output[3:0]s;

output c;

wire c0,c1,c2;

fa g1(s[0],c0,a[0],b[0],cin);

fa g2(s[1],c1,a[1],b[1],c0);

fa g3(s[2],c2,a[2],b[2],c1);

fa g4(s[3],c,a[3],b[3],c2);

endmodule
```

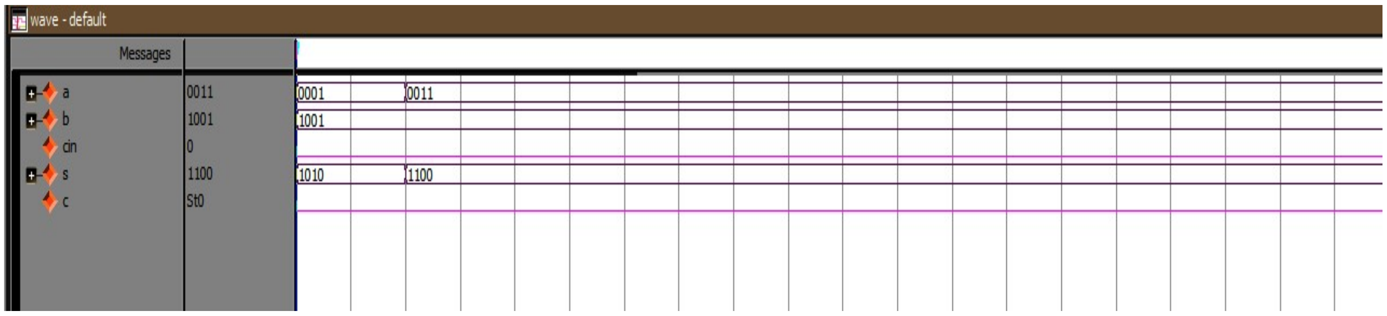
<u>Ripple Carry adder:TESTBENCH</u>

```
module rca_tb_v;
// Inputs
reg [3:0] a;
reg [3:0] b;
reg cin;
// Outputs
wire [3:0] s;
wire c;
```

```
// Instantiate the Unit Under Test (UUT)
rca uut (
.s(s),
.c(c),
.a(a),
.b(b),
.cin(cin)
);
initial begin
// Initialize Inputs
a = 4'b0001;
b = 4'b1001;
cin = 0;
// Wait 100 ns for global reset to finish
#100; a = 4'b0011;
b = 4'b1001;
cin = 0;
// Add stimulus here
end
endmodule
```

Waveform:

Ripple carry adder:



```
a     0011   0001  0011
b     1001   1001
cin   0
s     1100   1010  1100
c     St0
```