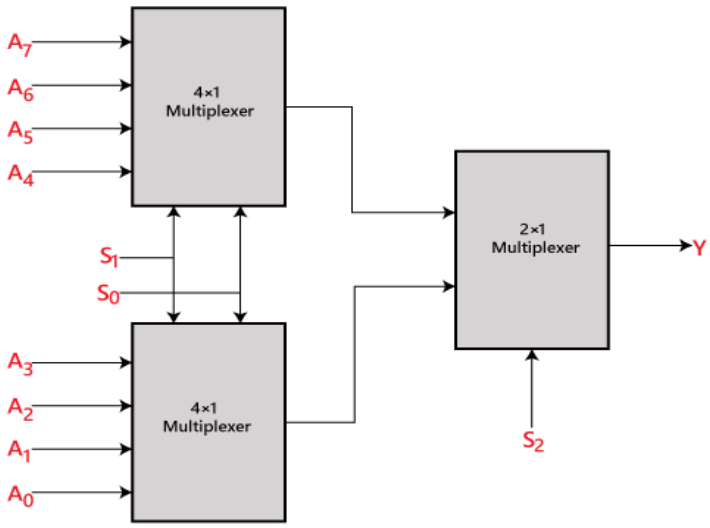
DIGITAL SYSTEM DESIGN – LAB 1

8:1 MUX using structural model, connection by name

Aim: The aim of this project is to design and implement an 8:1 multiplexer (mux) using the structural modeling approach in Verilog. The multiplexer will take 8 input signals and select one of them based on the control signal. The implementation will utilize connection by name method.

Procedure:

1. **Design Overview:** A multiplexer selects one of multiple input signals and passes it to the output based on the control signal. The 8:1 multiplexer will have 8 data inputs, 3 control inputs, and 1 output.
2. **Module Definition:** Create 3 Verilog modules named “mux_2”,”mux_4”, “mux_8” to represent the 2:1, 4:1, 8:1 multiplexers respectively. Define the input and output ports along with their data types.
3. **Internal Logic:** Inside the "mux_8" module, define the internal logic using structural modeling. The output should be determined based on the control inputs.



4. **Connection by Name:** In the instantiation of the multiplexer module, use connection by name to ensure clarity and accuracy in connecting the inputs and outputs. Explicitly connect each input and output using their port names.

5. **Testbench:** Create a testbench to verify the functionality of the multiplexer. Apply different combinations of input values and control signals to observe the output. Ensure that the multiplexer selects the correct input based on the control input.

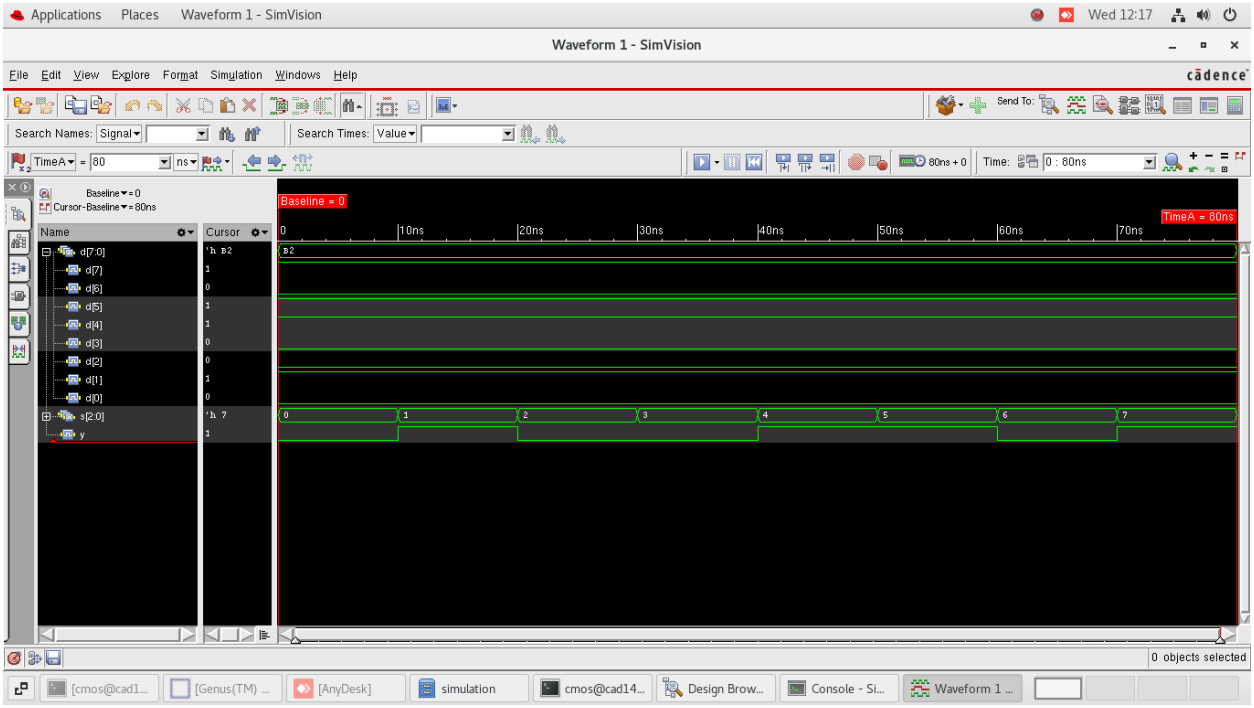
Design code mux_81.v

```
module mux_2(input wire a,b,s,output wire y);
wire n1,n2;
and a1(n1,~s,a);
and a2(n2,s,b);
or a3(y,n1,n2);
endmodule

module mux_4(input wire[3:0]d,input wire [1:0]s,output wire e);
wire l0,l1;
mux_2 m1(.a(d[0]),.b(d[1]),.s(s[0]),.y(l0));
mux_2 m2(.a(d[2]),.b(d[3]),.s(s[0]),.y(l1));
mux_2 m3(.a(l0),.b(l1),.s(s[1]),.y(e));
endmodule

module mux_8(input wire [7:0]d,input wire [2:0]s, output wire y);
wire p0,p1;
mux_4 m1(.d(d[3:0]),.s(s[1:0]),.e(p0));
mux_4 m2(.d(d[7:4]),.s(s[1:0]),.e(p1));
mux_2 m3(.a(p0),.b(p1),.s(s[2]),.y(y));
endmodule
```

Testbench code mux_81tb.v

```
module mux_8tb();

reg [7:0]d;

reg[2:0]s;

wire y;

mux_8 uut(.d(d),.s(s),.y(y));

initial begin

$monitor("D=%b,S=%b,Y=%b",d,s,y);

d = 8'b10110010;

s[2]=0;s[1]=0;s[0]=0;#10;  s[2]=0;s[1]=0;s[0]=1;#10;

s[2]=0;s[1]=1;s[0]=0;#10;  s[2]=0;s[1]=1;s[0]=1;#10;

s[2]=1;s[1]=0;s[0]=0;#10;  s[2]=1;s[1]=0;s[0]=1;#10;

s[2]=1;s[1]=1;s[0]=0;#10;  s[2]=1;s[1]=1;s[0]=1;#10;

$finish;

end

endmodule
```

Simulation Screenshot



Conclusion: In this project, we successfully designed and implemented an 8:1 multiplexer using the structural modeling approach in Verilog. The multiplexer selects one of 8 input signals based on the control signal and routes it to the output. Connection by name methodology was employed to ensure clear and accurate connections between module ports. The testbench was used to verify the functionality of the multiplexer under various scenarios. This project provides a fundamental understanding of multiplexers and their implementation in Verilog.

RCA Adder – structural model flow, with instantiation as connection by name

Aim: The objective of this project is to design and implement a Ripple Carry Adder (RCA) using Full Adder modules in Verilog. The project will follow a structural modeling approach, utilizing connection by name for instantiation. The aim is to understand the operation of a ripple carry adder and demonstrate its implementation using modular components.

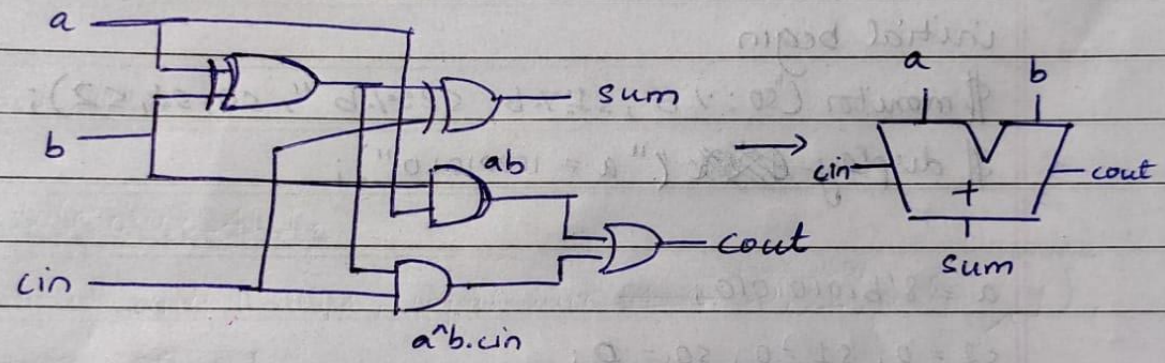
Procedure:

1. **Design Overview:** Ripple Carry Adder is a digital circuit that adds two n-bit binary numbers together. The operation is performed by sequentially adding corresponding bits from the two input numbers along with the carry generated from the previous stage. The least significant bit is added first, and the carry propagates to the next stage.
2. **Module Definition:** Define a Full Adder module that takes three inputs (A, B, and Cin) and produces two outputs (Sum and Cout). This module will be the basic building block for the ripple carry adder. Each stage of the ripple carry adder will consist of one Full Adder.
3. **Ripple Carry Adder:** Create a new Verilog module named "Rca" to represent the entire ripple carry adder. Define the input ports A and B (the numbers to be added), and the output ports Sum and Cout (the result and carry-out).
4. **Structural Modeling:** Inside the "Rca" module, instantiate the required number of Full Adder modules based on the bit width of the numbers being added. Connect the inputs and outputs of each Full Adder module appropriately, ensuring a sequential connection for the carry propagation.
5. **Connection by Name:** In the instantiation of the Full Adder modules within the "Rca" module, utilize connection by name methodology. Explicitly connect each input and output using their port names.
6. **Testbench:** Develop a testbench to verify the functionality of the ripple carry adder. Apply different input values to observe the sum and carry-out results. Verify that the ripple carry adder correctly adds the input numbers.

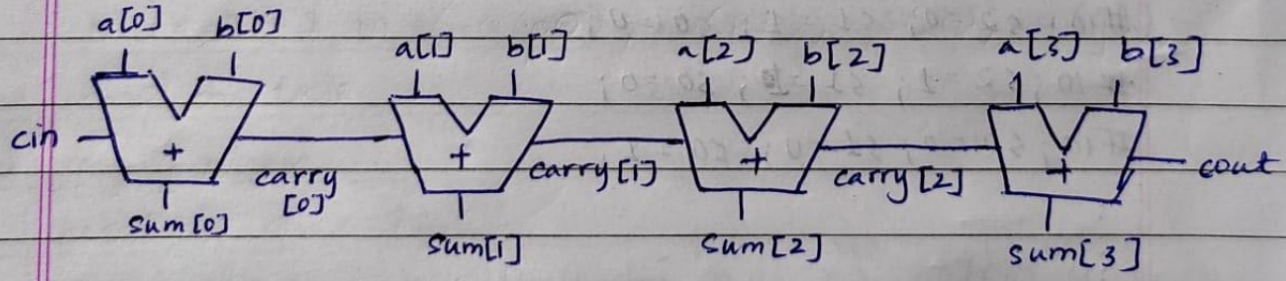
Sum = a^b^cin

cout = a^b.cin + ab

Full Adder a+b+cin = sum + cout



RCA a[3:0] + b[3:0] + cin (MSB LSB)



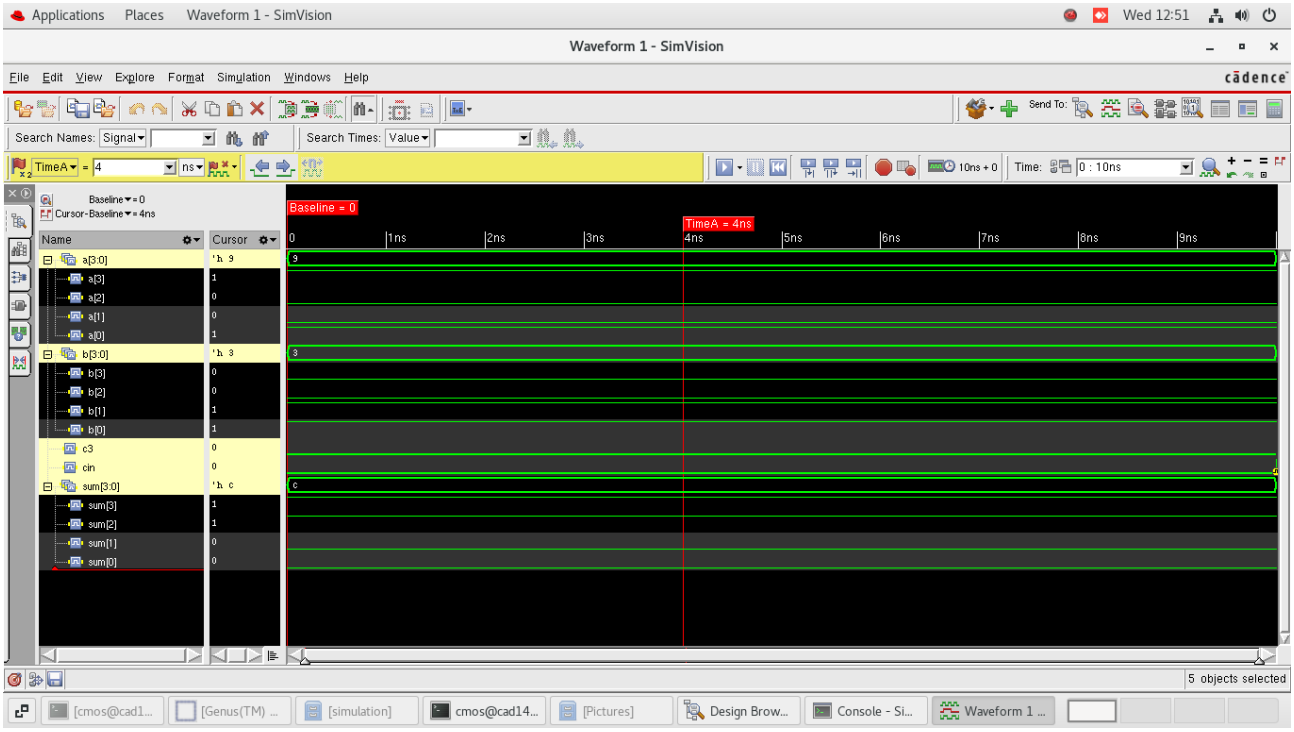
Design – rca.v

```
module fulladd(input wire a,b,cin,output wire cout,sum);
wire p1,p2,p3;
xor x1(p1,a,b);
xor x2(sum,p1,cin);
and a1(p2,p1,cin);
and a2(p3,a,b);
or(cout,p2,p3);
endmodule

module rca(input wire [3:0]a,b,input wire cin,output wire c3,output wire [3:0]S);
wire c0,c1,c2;
fulladd f1(.a(a[0]),.b(b[0]),.cin(cin),.cout(c0),.sum(S[0]));
fulladd f2(.a(a[1]),.b(b[1]),.cin(c0),.cout(c1),.sum(S[1]));
fulladd f3(.a(a[2]),.b(b[2]),.cin(c1),.cout(c2),.sum(S[2]));
fulladd f4(.a(a[3]),.b(b[3]),.cin(c2),.cout(c3),.sum(S[3]));
endmodule
```

Testbench – rcatb.v

```
module rcatb();
reg [3:0]a,b; reg cin;
wire c3; wire [3:0]sum;
rca uut(.a(a),.b(b),.cin(cin),.c3(c3),.S(sum));
initial begin
cin=0; a=4'b1001;b=4'b0011;#10; cin=1;
$finish;
end
endmodule
```

Simulation Screenshot



Conclusion: The project successfully demonstrated the implementation of a Ripple Carry Adder using Full Adders with a structural modeling approach in Verilog. The Full Adder modules were used to perform bit-wise addition, and the ripple carry was propagated through stages to achieve the final result. The testbench verified the functionality of the ripple carry adder for various input combinations. This project provided insight into the operation of ripple carry adders and their modular implementation in Verilog.