University of Engineering and Technology, Peshawar

Department of Computer Systems Engineering.

Course Lab: CSE-308 Digital System Design

Zubair Khan **19** PWCSE **1797**

Section A

Batch 21 (Spring_2022)

Submitted to Engr. Madiha Sher

LAB 02 Title

Introduction To Modelsim And Gate Level Modeling

Objectives(s):

This lab will enable students to:

- Learn top down and bottom up design methodologies
- Data flow level modeling
- Data gate level modeling

Theory

Data Gate Level Modeling

Designing circuits using basic logic gates is known as gate-level modeling. A digital circuit is implemented using logic gates and interconnections between these gates.

Data Flow Level Modeling

Dataflow modeling uses a number of operators that act on operands to produce the desired results. Verilog HDL provides about 30 operator types like &, |, ^, ==, >, <,? etc. Dataflow modeling uses continuous assignments and the keyword ***assign***. It is called continuous assignment because it remains active all the time.

assign out = a & b;

The left hand side variable must be of type wire. At the right hand side, there can be a single variable or expression and of type reg or wire.

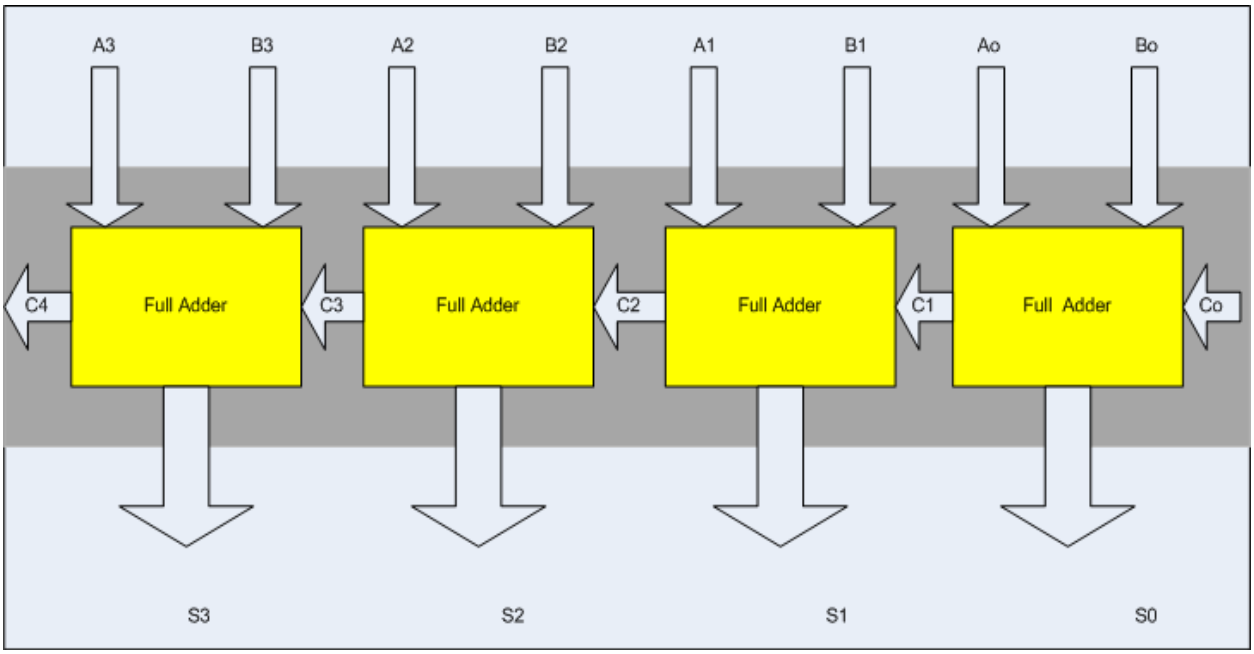
Operators used:

Arithmetic: *+,-,*,/*
Logical: *&&, ||*
Bitwise: *&, |, ~, ^*
Reduction: *~|, ~&, ~^*
Shift: *<<, >>*
Equality*:* *==, !=, (case equality: ===, !==)*
Conditional: *? , :*
Concatenation: *{}*
Example: a=01;b=110; c={a,b};
Replication: *{{}}*
Example: A=111111000000
x={6{1}}; y={6{0}};
A={x,y} or A={6{1},6{0}};

LAB Task 01

4-bit Ripple Carry Adder

Block Diagram



I/O Connection:

Ground “Co” Permanently and connect S0-S3 with four LEDs also connect C4 to another LED.

Steps for task 1a:

The following steps should be performed while designing a 4 bit RCA adder

ModelSim:

1- First implement a Full adder using data gate level modeling.

Data Gate Level Modeling.

Code (Full Adder):

```
1 //Structural Model : Half Adder
2 module half_adder(output S,C,input A,B);
3  xor(S,A,B);
4  and(C,A,B);
5 endmodule
6
7 //Structural Model : Full Adder
8 module full_adder(S,Cout,A,B,Cin);
9  input A,B,Cin;
10  output S,Cout;
11  wire s1,c1,c2;
12  half_adder HA1(s1,c1,A,B);
13  half_adder HA2(S,c2,s1,Cin);
14  or o1(Cout,c1,c2);
15 endmodule
```

2- Simulate the Full adder with a test bench.

Code (Test Bench For Full Adder):

```
17 module test_FA(); //test bench to test FA (Full Adder)
18         // Inputs are regs here
19         reg A, B, Cin;
20         // Outputs are wires here
21         wire S, Cout;
22         // Instantiate the Unit/Design Under Test (U/DUT)
23         // FA dut (S, Cout, A, B, Cin);  // Positional association
24         full_adder  fa(.A(A), .B(B), .Cin(Cin), .S(S), .Cout(Cout));  // Explicit association
25
26         initial begin
27                 // Initialize Inputs
28                 A = 0; B = 0; Cin = 0;
29                 #10
30                 Cin = 1;
31                 #10
32                 B = 1;
33                 #10
34                 A = 1;
35                 #10
36                 A = 0;
37                 #10
38                 Cin = 0;
39         end
40
41         initial begin
42                 $monitor("A=%b,B=%b,Cin=%b--->Sum=%b,Cout=%b  \n",A,B,Cin,S,Cout);
43         end
44
45 endmodule
```

Output (Full Adder):

```
# Compile of fulladder.v was successful.
vsim work.test_FA
# vsim work.test_FA
# Loading work.test_FA
# Loading work.full_adder
# Loading work.half_adder
run
# A=0,B=0,Cin=0--->Sum=0,Cout=0
#
# A=0,B=0,Cin=1--->Sum=1,Cout=0
#
# A=0,B=1,Cin=1--->Sum=0,Cout=1
#
# A=1,B=1,Cin=1--->Sum=1,Cout=1
#
# A=0,B=1,Cin=1--->Sum=0,Cout=1
#
# A=0,B=1,Cin=0--->Sum=1,Cout=0
#
```

3- Instantiate the Full adder four times and connect the circuit as shown.

Code (Ripple Adder):

```
17 //Structural Model : 4 Bit Ripple Carry Adder
18 module ripple_adder_4bit(Cout,Sum,A,B,Cin);
19  output [3:0] Sum;
20  output Cout;
21  input [3:0] A,B;
22  input Cin;
23 wire c1,c2,c3;
24  //full_adder fa(S,Cout,A,B,Cin);
25  full_adder FA1(Sum[0],c1,A[0],B[0],Cin),
26     FA2(Sum[1],c2,A[1],B[1],c1),
27     FA3(Sum[2],c3,A[2],B[2],c2),
28     FA4(Sum[3],Cout,A[3],B[3],c3);
29 endmodule
```

4- Now again write a test bench and simulate the 4 bit RCA.

Code (Test Bench For Ripple Adder):

```
31 //Test Bench : 4 Bit Ripple Carry Adder
32 module test_ripple_adder_4bit();
33  // Inputs
34  reg [3:0] A;
35  reg [3:0] B;
36  reg Cin;
37  // Outputs
38  wire [3:0] Sum;
39  wire Cout;
40  // Instantiate the Unit Under Test (UUT)
41  ripple_adder_4bit rpa(.Sum(Sum), .Cout(Cout), .A(A), .B(B),.Cin(Cin));
42  initial
43         begin
44
45         $monitor("A=%d   B=%d   Cin=%d  Sum=%d          Cout=%d",A,B,Cin,Sum,Cout);
46         A=4'b0000;B=4'b0000;Cin=1'b0;
47         #10
48         A=4'b0010;B=4'b0010;Cin=1'b0;
49         #10
50         A=4'b0010;B=4'b0100;Cin=1'b0;
51         #10
52         A=4'b0110;B=4'b1010; Cin=1'b0;
53         #10
54         A=4'b1111;B=4'b1111;Cin=1'b0;
55          end
56 endmodule
```

Output (Ripple Adder):

```
# Compile of example.v was successful.
vsim work.test_ripple_adder_4bit
# vsim work.test_ripple_adder_4bit
# Loading work.test_ripple_adder_4bit
# Loading work.ripple_adder_4bit
# Loading work.full_adder
# Loading work.half_adder
run
# A= 0  B= 0 Cin=0 Sum= 0  Cout=0
# A= 2  B= 2 Cin=0 Sum= 4  Cout=0
# A= 2  B= 4 Cin=0 Sum= 6  Cout=0
# A= 6  B=10 Cin=0 Sum= 0  Cout=1
# A=15  B=15 Cin=0 Sum=14  Cout=1
```

Xilinx:

1. Make new project in Xilinx and add the files that you simulated in ModelSim.
2. Add User Constraint File inputs should be locked with the switches, C0 should be permanently “0” while S0-S4 and C4 with LEDS

Task 1b:

Design the 4 bit full adder using data flow level modeling.

Data Flow Level Modeling:

Code (Ripple Adder):

```
 1 module SUM (S, A, B, Cin);
 2         output S;
 3         input A, B, Cin;
 4         // Behavioral Code
 5         assign S = A^B^Cin;
 6
 7 endmodule
 8
 9 module CARRY (Cout, A, B, Cin);
10         output Cout;
11         input A, B, Cin;
12         // Behavioral Code
13         assign Cout = (A&B) | (B&Cin) | (A&Cin);
14 endmodule
15
16 module FA1 (Sum, Cout, A, B, Cin);
17         output Sum, Cout;
18         input A, B, Cin;
19         // Structural Code
20         SUM s1 (.S(Sum), .A(A), .B(B), .Cin(Cin));
21         CARRY c1 (.Cout(Cout), .A(A), .B(B), .Cin(Cin));
22 endmodule
24 module RCA (Cout, S, A, B,Cin);
25         output Cout;
26         output [3:0] S;
27         input [3:0] A, B;
28         input Cin;
29         wire [2:0] C;   //Intermediate/Internal Carries
30         // FA1 ;   //FA1's Interface (I/O Pins)
31         FA1 fa0 (S[0], C[0], A[0], B[0], Cin);
32         FA1 fa1 (S[1], C[1], A[1], B[1], C[0]);
33         FA1 fa2 (S[2], C[2], A[2], B[2], C[1]);
34         FA1 fa3 (S[3], Cout, A[3], B[3], C[2]);
35 endmodule
36
```

Code (Test Bench):

```
37 module stim_RCA(); //test bench
38 reg [3:0]A,B; //4 bits inputs
39 reg Cin;
40 wire Cout; //1 bits output
41 wire [3:0]S; //4 bits outputs
42 RCA r(Cout, S, A, B,Cin);
43 initial
44         begin
45         $monitor("A=%d   B=%d   Cin=%d  Sum=%d          Cout=%d",A,B,Cin,S,Cout);
46         A=4'b0000;B=4'b0000;Cin=1'b0;
47         #10
48         A=4'b0010;B=4'b0010;Cin=1'b0;
49         #10
50         A=4'b0010;B=4'b0100;Cin=1'b0;
51         #10
52         A=4'b0110;B=4'b1010; Cin=1'b0;
53         #10
54         A=4'b1111;B=4'b1111;Cin=1'b0;
55         end
56 endmodule
```

Output:

```
# Compile of rippleadder.v was successful.
vsim work.stim_RCA
# vsim work.stim_RCA
# Loading work.stim_RCA
# Loading work.RCA
# Loading work.FA1
# Loading work.SUM
# Loading work.CARRY
run
# A= 0   B= 0 Cin=0 Sum= 0  Cout=0
# A= 2   B= 2 Cin=0 Sum= 4  Cout=0
# A= 2   B= 4 Cin=0 Sum= 6  Cout=0
# A= 6   B=10 Cin=0 Sum= 0  Cout=1
# A=15   B=15 Cin=0 Sum=14  Cout=1
```

2nd Code:

```
module rippleadder(cout,sum,a,b,cin);
input cin; //1 bits input
input [3:0]a,b;//4 bits inputs
output cout; ////1 bits output
output [3:0]sum; //4 bits outputs
assign {cout,sum}=a+b+cin; //behavioral modeling
endmodule

module stim_ripple(); //test bench
reg cin; //1 bits input
reg [3:0]a,b; //4 bits inputs
wire cout; //1 bits output
wire [3:0]sum; //4 bits outputs
rippleadder ra(cout,sum,a,b,cin);
initial
        begin
        $display("A     B       Cin             Sum             Cout");
        a=4;b=6;cin=1'b0;
        #10
        $display("%d %d  %d                     %d                      %d",a,b,cin,sum,cout);
        a=11;b=5;cin=1'b0;
        #10
        $display("%d %d  %d                     %d                      %d",a,b,cin,sum,cout);
        a=9;b=3;cin=1'b0;
        #10
        $display("%d %d %d                      %d                      %d",a,b,cin,sum,cout);
        a=2;b=1;cin=1'b0;
        #10
        $display("%d %d  %d                     %d                      %d",a,b,cin,sum,cout);
        a=5;b=3;cin=1'b0;
        #10
        $display("%d %d  %d                     %d                      %d",a,b,cin,sum,cout);
        end

//initial
        //begin
        //$monitor("A=%b  B=%b          Cin=%b          Sum=%b          Cout=%b",a,b,cin,sum,cout);
        //end
endmodule
```

Output:

```
# Compile of 4bitrippleadder.v was successful.
vsim work.stim_ripple
# vsim work.stim_ripple
# Loading work.stim_ripple
# Loading work.rippleadder
run
# A  B Cin  Sum  Cout
#  4  6  0   10   0
# 11  5  0    0   1
#  9  3  0   12   0
#  2  1  0    3   0
#  5  3  0    8   0
```