**Amrita Vishwa Vidyapeetham**

**Amrita School of Engineering, Coimbatore**

**Department of Computer Science and Engineering**

**Practice Question**

**--------------------------------------------------------------------------------------------------------------------------**

**Sub Code: 19CS211 Sub Title: COA**

**Roll No: CB.EN.U4CSE19453 Name: R.ABHINAV**

**Practice Lab Date: 08-03-2021**

1. Design an Up-Counter using the half-adder circuit

* **Verilog Code:**

module half\_adder

(

i\_bit1,

i\_bit2,

o\_sum,

o\_carry

);

input i\_bit1;

input i\_bit2;

output o\_sum;

output o\_carry;

assign o\_sum = i\_bit1 ^ i\_bit2; // bitwise xor

assign o\_carry = i\_bit1 & i\_bit2; // bitwise and

endmodule // half\_adder

* **Testbench :**

module half\_adder\_tb;

reg r\_BIT1 = 0;

reg r\_BIT2 = 0;

wire w\_SUM;

wire w\_CARRY;

half\_adder half\_adder\_inst

(

.i\_bit1(r\_BIT1),

.i\_bit2(r\_BIT2),

.o\_sum(w\_SUM),

.o\_carry(w\_CARRY)

);

initial

begin

$monitor(r\_BIT1, r\_BIT2, w\_SUM, w\_CARRY);

r\_BIT1 = 1'b0;

r\_BIT2 = 1'b0;

#10;

r\_BIT1 = 1'b0;

r\_BIT2 = 1'b1;

#10;

r\_BIT1 = 1'b1;

r\_BIT2 = 1'b0;

#10;

r\_BIT1 = 1'b1;

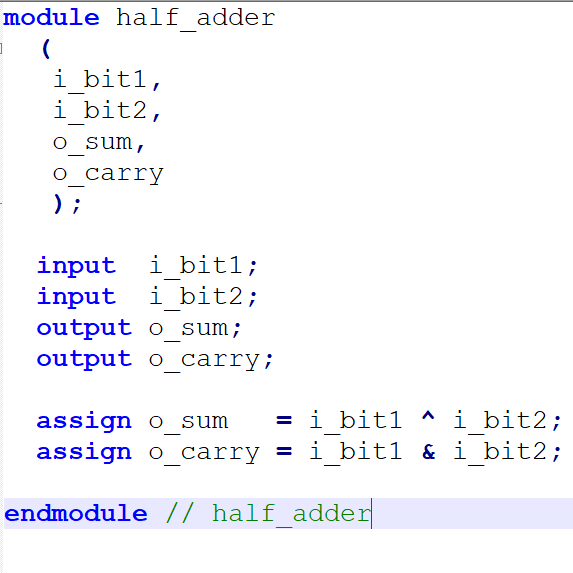
r\_BIT2 = 1'b1;

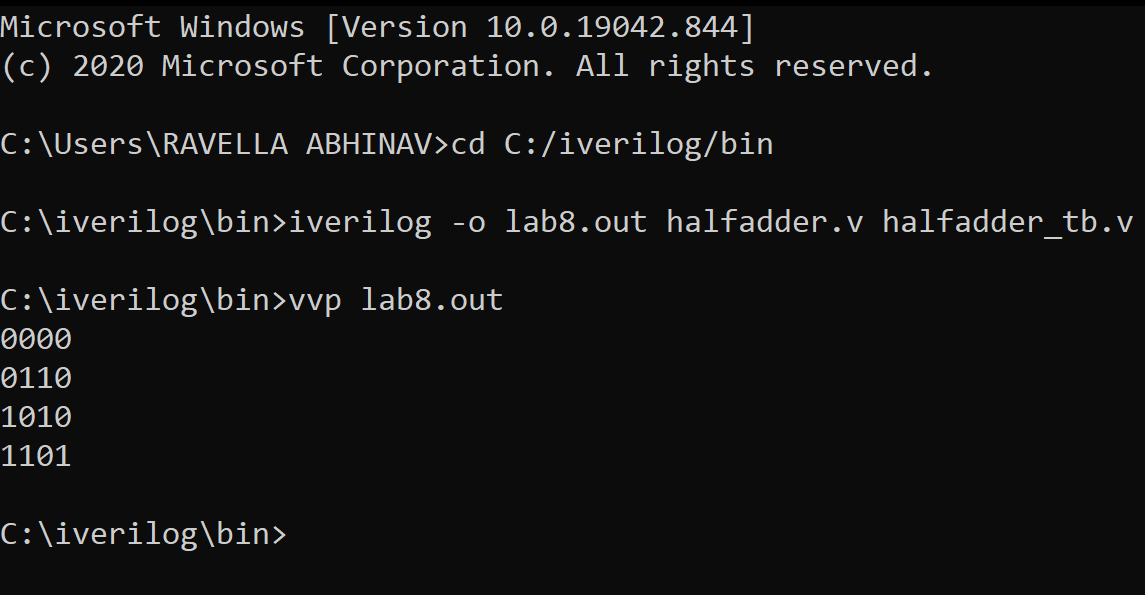
#10;

end

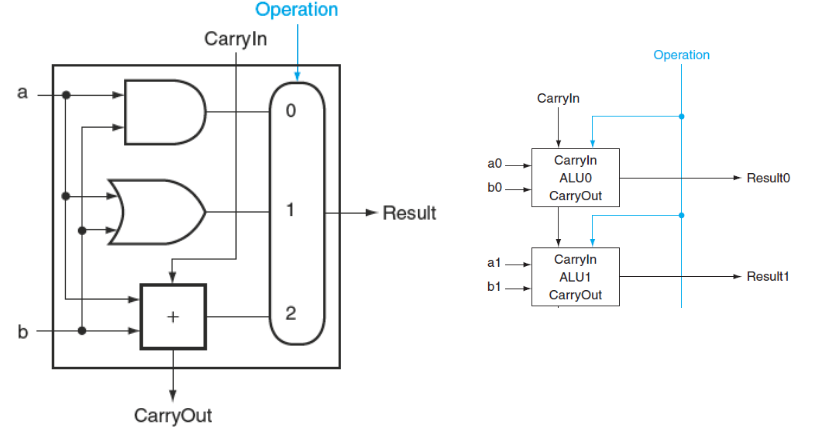
endmodule

* **Snippets :**

****

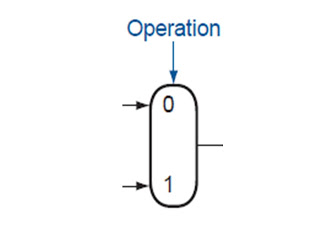


1. Design an Down-Counter using the half-subtractor circuit
2. Design an 2-bit ALU using the following circuit and logic



Based on the diagram above,

* the multiplexer on the right side select either a AND b or a OR b depending on whether the value of operation is 0 or 1.
* While the line in color control the multiplexer and the line is to distinguish from the lines containing data.



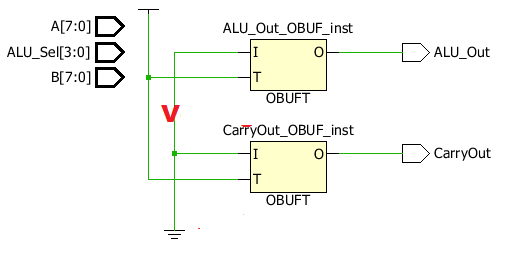
* CarryOut=(b.CarryIn)+ (a.CarryIn)+(a.b)+(a.b.CarryIn)
* Simplified :

CarryOut=(b.CarryIn)+(a.CarryIn)+(a.b)

* If (a.b.CarryIn) is true, all other three terms must be true and so the last term can be leaving out corresponding to the 4th line of the table)

|  |  |  |
| --- | --- | --- |
| INPUTS | | |
| a | b | CarryIn |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |

1. Write Verilog code for ALU. Write testbench Verilog code for the ALU for simulation and testing.



**Verilog Code :**

module alu(

input [7:0] A,B,

input [3:0] ALU\_Sel,

output [7:0] ALU\_Out,

output CarryOut

);

reg [7:0] ALU\_Result;

wire [8:0] tmp;

assign ALU\_Out = ALU\_Result;

assign tmp = {1'b0,A} + {1'b0,B};

assign CarryOut = tmp[8];

always @(\*)

begin

case(ALU\_Sel)

4'b0000:

ALU\_Result = A + B ;

4'b0001:

ALU\_Result = A - B ;

4'b0010:

ALU\_Result = A \* B;

4'b0011:

ALU\_Result = A/B;

4'b0100:

ALU\_Result = A<<1;

4'b0101:

ALU\_Result = A>>1;

4'b0110:

ALU\_Result = {A[6:0],A[7]};

4'b0111:

ALU\_Result = {A[0],A[7:1]};

4'b1000:

ALU\_Result = A & B;

4'b1001:

ALU\_Result = A | B;

4'b1010:

ALU\_Result = A ^ B;

4'b1011:

ALU\_Result = ~(A | B);

4'b1100:

ALU\_Result = ~(A & B);

4'b1101:

ALU\_Result = ~(A ^ B);

4'b1110:

ALU\_Result = (A>B)?8'd1:8'd0 ;

4'b1111:

ALU\_Result = (A==B)?8'd1:8'd0 ;

default: ALU\_Result = A + B ;

endcase

end

endmodule

**Test Bench:**

module tb\_alu;

reg[7:0] A,B;

reg[3:0] ALU\_Sel;

wire[7:0] ALU\_Out;

wire CarryOut;

integer i;

alu test\_unit(

A,B,

ALU\_Sel,

ALU\_Out,

CarryOut

);

initial begin

A = 8'h0A;

B = 4'h02;

ALU\_Sel = 4'h0;

for (i=0;i<=15;i=i+1)

begin

ALU\_Sel = ALU\_Sel + 8'h01;

#10;

end;

A = 8'hF6;

B = 8'h0A;

end

endmodule

**Snippets:**

