

**يوسف أحمد محمد ابراهيم**

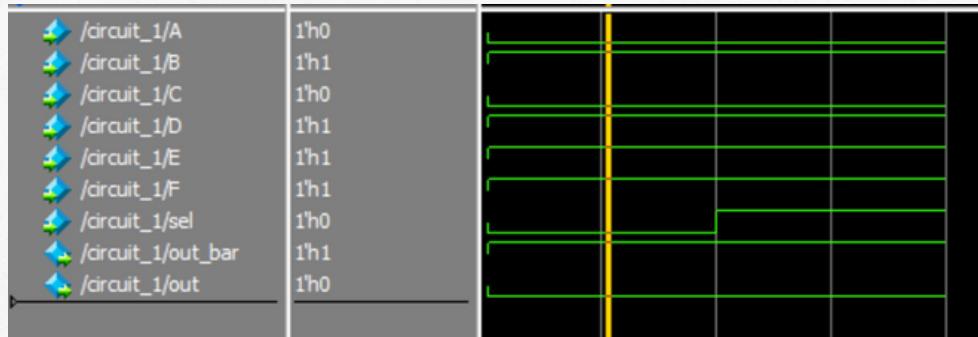
# **Assignment 1**

# Circuit\_1 :

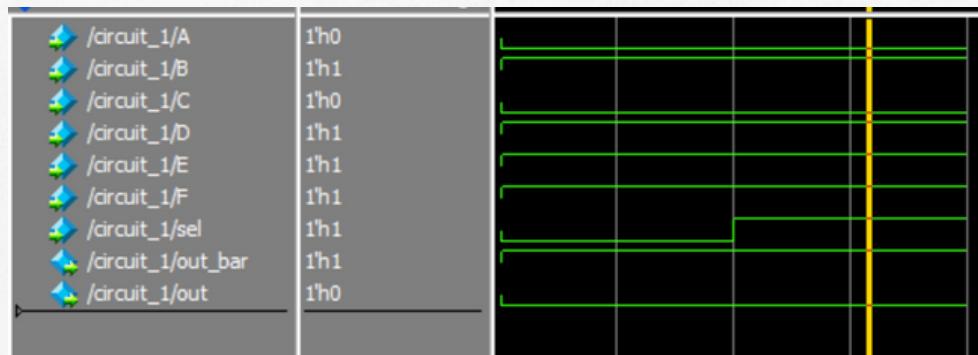
- code

```
1 module circuit_1(input A, B, C, D, E, F, sel,
2                   output out_bar, output reg out);
3
4     always @(*) begin
5         case(sel)
6             0 : out = A & B & C;
7             1 : out = ~(D ^ E ^ F);
8         endcase
9
10    end
11    assign out_bar = ~out;
12 endmodule
```

- output\_with sel = 0:



- output\_with sel = 1:

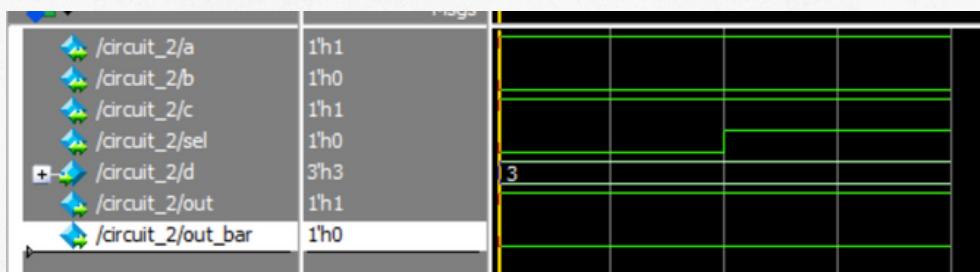


# Circuit\_2:

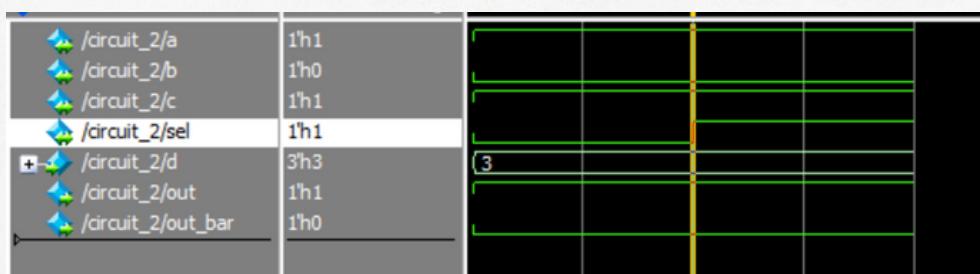
- code

```
1 module circuit_2(input a, b, c, sel, input [2:0]d,
2                   output reg out, output out_bar);
3
4   always @(*) begin
5     if (sel == 0)
6       out = (d[0] & d[1]) | d[2];
7     else
8       out = ~(a ^ b ^ c);
9   end
10
11  assign out_bar = ~ out;
12 endmodule
```

- output\_with sel = 0:



- output\_with sel = 1:



# Circuit\_3:

- code

```
1 module circuit_3(a, b, c);
2     parameter N = 4;
3     input [N - 1 : 0] a, b;
4     output [N - 1 : 0] c;
5     assign c = a + b;
6 endmodule
```

- output:

/circuit_3/N	32'h00000004	00000004
/circuit_3/a	4'h2	2
/circuit_3/b	4'h3	3
/circuit_3/c	4'h5	5

# Circuit\_4:

- code

```
1 module circuit_4(input [1:0]a, output [3:0]d);
2
3     assign d = (a == 2'b00) ? (4'b0001):
4             (a == 2'b01) ? (4'b0010):
5             (a == 2'b10) ? (4'b0100):
6             (4'b1000);
7
8 endmodule
```

- output:

3	2	1	0
8	4	2	1

# Circuit\_5:

- code

```
1 module circuit_5(input [7:0]in, output [8:0] out);
2 |   assign out = {in, ^in};
3 endmodule
```

- output\_with even 1's :

/circuit_5/in	8'b10011010	10011010	10011101
/circuit_5/out	9'b100110100	100110100	100111011

- output\_with odd 1's :

/circuit_5/in	8'b10011101	10011010	10011101
/circuit_5/out	9'b100111011	100110100	100111011

# Circuit\_6:

- code

```
1  module alu_6(input [3 : 0]A, B, [1:0]op, output reg [3 : 0]out);
2
3      always @(*) begin
4          case (op)
5              00 : out = A + B;
6              01 : out = A | B;
7              10 : out = A - B;
8              11 : out = A ^ B;
9          endcase
10     end
11 endmodule
12
13 module decoder_6(input [3:0]A, B, input en, [1:0]op,
14                     output reg a, b, c, d, e, f, g];
15     wire [3:0]alu_out;
16     alu_6 alu(.A(A), .B(B), .op(op), .out(alu_out));
17
18     always @(*) begin
19         if(en)
20             case(alu_out)
21                 4'h0 : {a, b, c, d, e, f, g} = 7'b1111110;
22                 4'h1 : {a, b, c, d, e, f, g} = 7'b0110000;
23                 4'h2 : {a, b, c, d, e, f, g} = 7'b1101101;
24                 4'h3 : {a, b, c, d, e, f, g} = 7'b1111001;
25                 4'h4 : {a, b, c, d, e, f, g} = 7'b0110011;
26                 4'h5 : {a, b, c, d, e, f, g} = 7'b1011011;
27                 4'h6 : {a, b, c, d, e, f, g} = 7'b1011111;
28                 4'h7 : {a, b, c, d, e, f, g} = 7'b1110000;
29                 4'h8 : {a, b, c, d, e, f, g} = 7'b1111111;
30                 4'h9 : {a, b, c, d, e, f, g} = 7'b1111011;
31                 4'ha : {a, b, c, d, e, f, g} = 7'b1110111;
32                 4'hb : {a, b, c, d, e, f, g} = 7'b0011111;
33                 4'hc : {a, b, c, d, e, f, g} = 7'b1001110;
34                 4'hd : {a, b, c, d, e, f, g} = 7'b0111101;
35                 4'he : {a, b, c, d, e, f, g} = 7'b1001111;
36                 4'hf : {a, b, c, d, e, f, g} = 7'b1000111;
37             endcase
38
39     end
40
41 endmodule
```

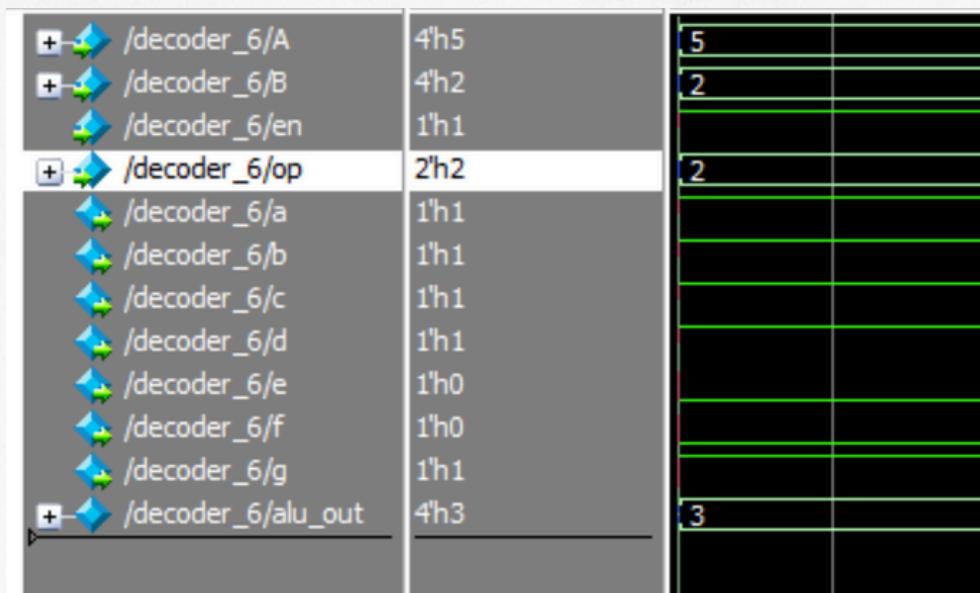
- **output\_with op = 0 (addition) :**

<code>+-- /decoder_6/A</code>	4'h5	5
<code>+-- /decoder_6/B</code>	4'h4	4
<code>+-- /decoder_6/en</code>	1'h1	
<code>+-- /decoder_6/op</code>	2'h0	0
<code>+-- /decoder_6/a</code>	1'h1	
<code>+-- /decoder_6/b</code>	1'h1	
<code>+-- /decoder_6/c</code>	1'h1	
<code>+-- /decoder_6/d</code>	1'h1	
<code>+-- /decoder_6/e</code>	1'h0	
<code>+-- /decoder_6/f</code>	1'h1	
<code>+-- /decoder_6/g</code>	1'h1	
<code>+-- /decoder_6/alu_out</code>	4'h9	9

- **output\_with op = 1 (bitwise or) :**

<code>+-- /decoder_6/A</code>	4'h5	5	
<code>+-- /decoder_6/B</code>	4'h4	4	
<code>+-- /decoder_6/en</code>	1'h1		
<code>+-- /decoder_6/op</code>	2'h1	0	1
<code>+-- /decoder_6/a</code>	1'h1		
<code>+-- /decoder_6/b</code>	1'h0		
<code>+-- /decoder_6/c</code>	1'h1		
<code>+-- /decoder_6/d</code>	1'h1		
<code>+-- /decoder_6/e</code>	1'h0		
<code>+-- /decoder_6/f</code>	1'h1		
<code>+-- /decoder_6/g</code>	1'h1		
<code>+-- /decoder_6/alu_out</code>	4'h5	9	5

- **output\_with op = 2 (subtraction) :**



- **output\_with op = 3 (bitwise xor) :**

