يوسف أحمد محمد ابراهيم

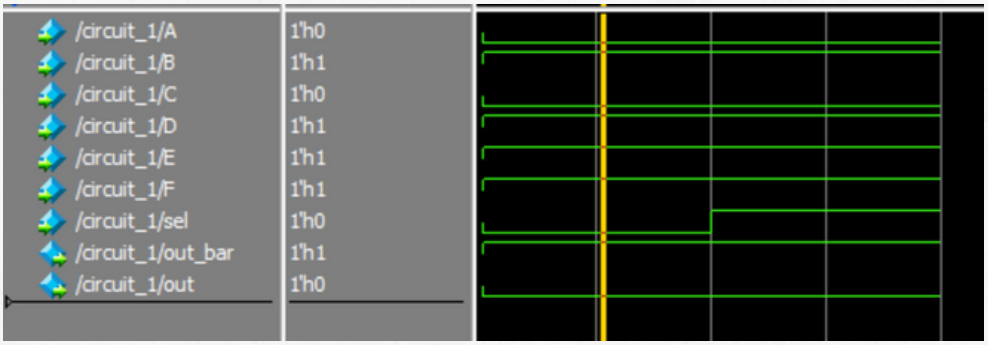
Assignment 1

Circuit_1 :

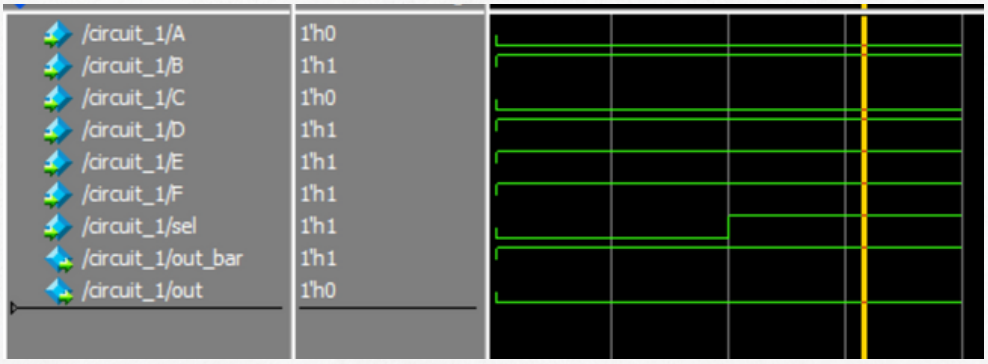
- code

```
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:

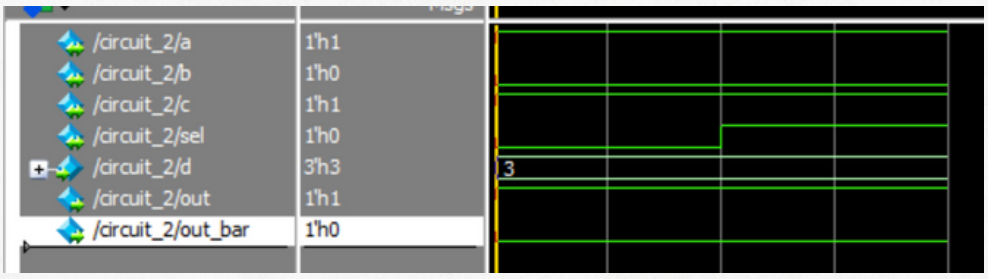


Circuit_2:

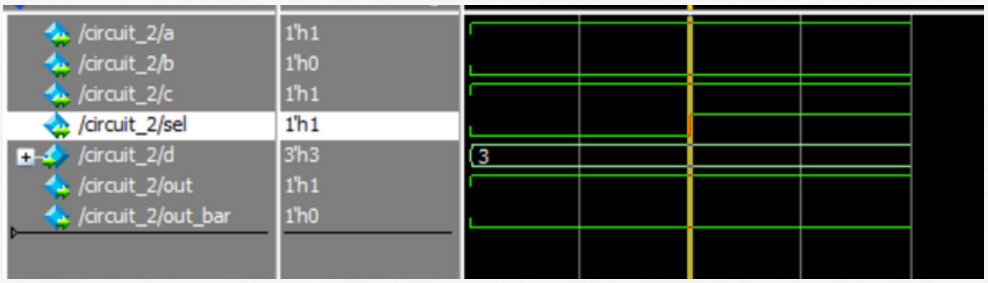
- code

```
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:

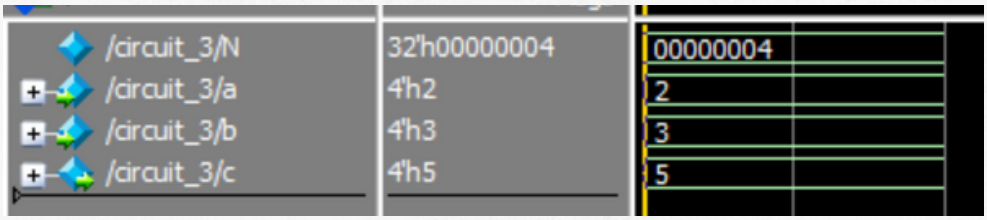


Circuit_3:

- 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:

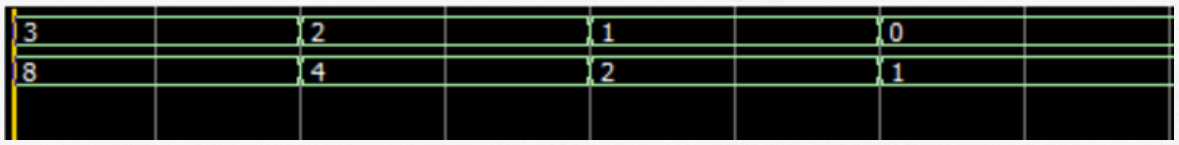


Circuit_4:

- 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:

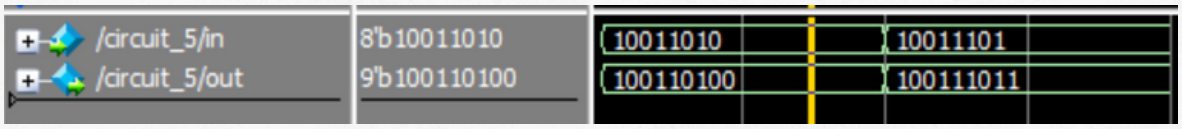


Circuit_5:

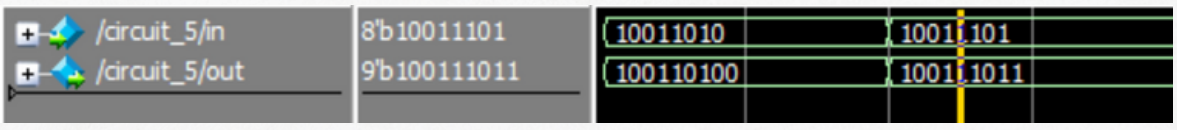
- code

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

- output_with even 1's :



- output_with odd 1's :

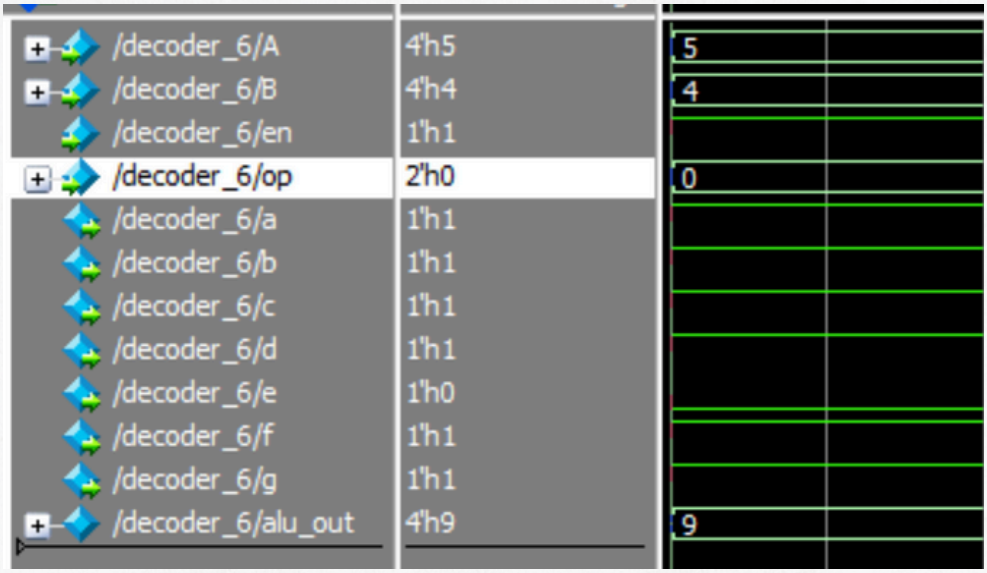


Circuit_6:

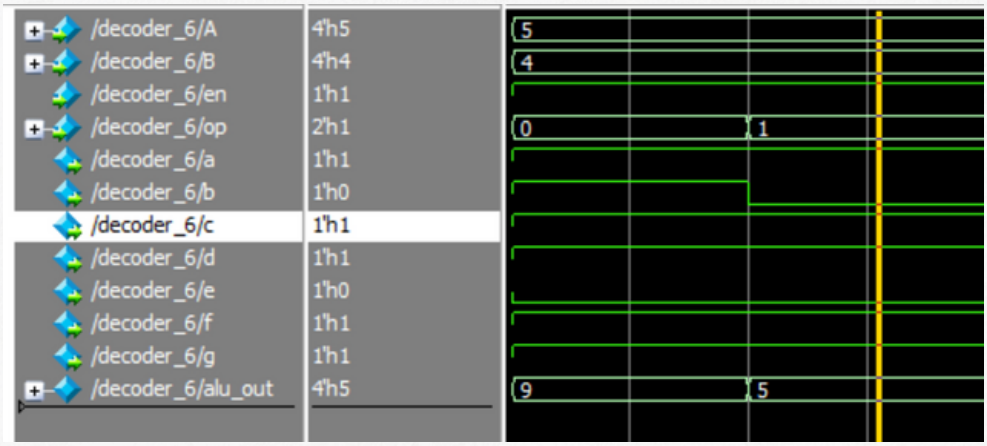
- code

```
module alu_6(input [3 : 0]A, B, [1:0]op, output reg [3 : 0]out);

    always @(*) begin
        case (op)
            00 : out = A + B;
            01 : out = A | B;
            10 : out = A - B;
            11 : out = A ^ B;
        endcase
    end
endmodule

module decoder_6(input [3:0]A, B, input en, [1:0]op,
                 output reg a, b, c, d, e, f, g);
    wire [3:0]alu_out;
    alu_6 alu(.A(A), .B(B), .op(op), .out(alu_out));

    always @(*) begin
        if(en)
            case(alu_out)
                4'h0 : {a, b, c, d, e, f, g} = 7'b1111110;
                4'h1 : {a, b, c, d, e, f, g} = 7'b0110000;
                4'h2 : {a, b, c, d, e, f, g} = 7'b1101101;
                4'h3 : {a, b, c, d, e, f, g} = 7'b1111001;
                4'h4 : {a, b, c, d, e, f, g} = 7'b0110011;
                4'h5 : {a, b, c, d, e, f, g} = 7'b1011011;
                4'h6 : {a, b, c, d, e, f, g} = 7'b1011111;
                4'h7 : {a, b, c, d, e, f, g} = 7'b1110000;
                4'h8 : {a, b, c, d, e, f, g} = 7'b1111111;
                4'h9 : {a, b, c, d, e, f, g} = 7'b1111011;
                4'ha : {a, b, c, d, e, f, g} = 7'b1110111;
                4'hb : {a, b, c, d, e, f, g} = 7'b0011111;
                4'hc : {a, b, c, d, e, f, g} = 7'b1001110;
                4'hd : {a, b, c, d, e, f, g} = 7'b0111101;
                4'he : {a, b, c, d, e, f, g} = 7'b1001111;
                4'hf : {a, b, c, d, e, f, g} = 7'b1000111;
            endcase

    end


endmodule
```

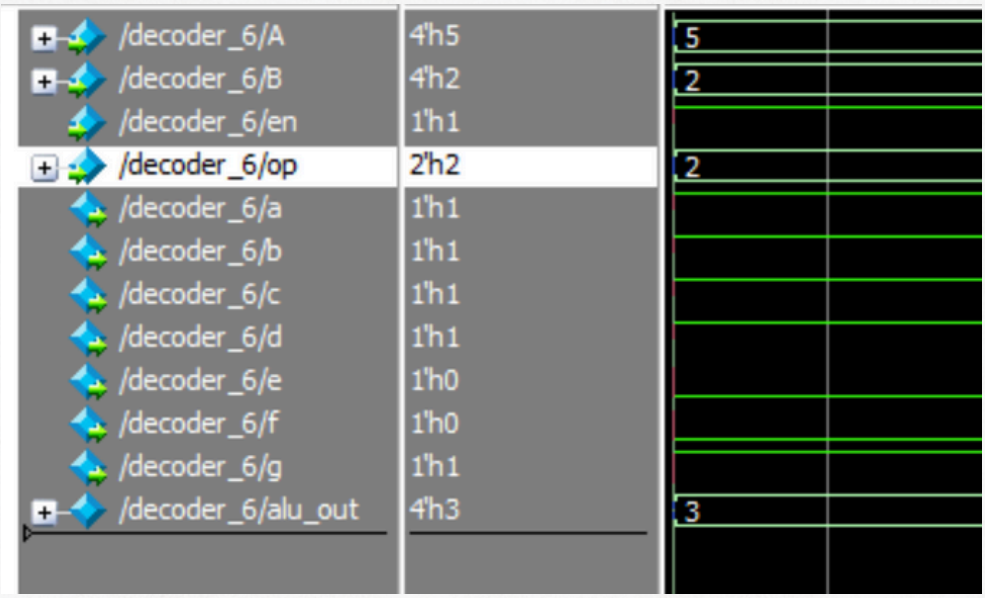
- **output_with op = 0 (addition) :**



- **output_with op = 1 (bitwise or) :**



- **output_with op = 2 (substraction) :**



- **output_with op = 3 (bitwise xor) :**