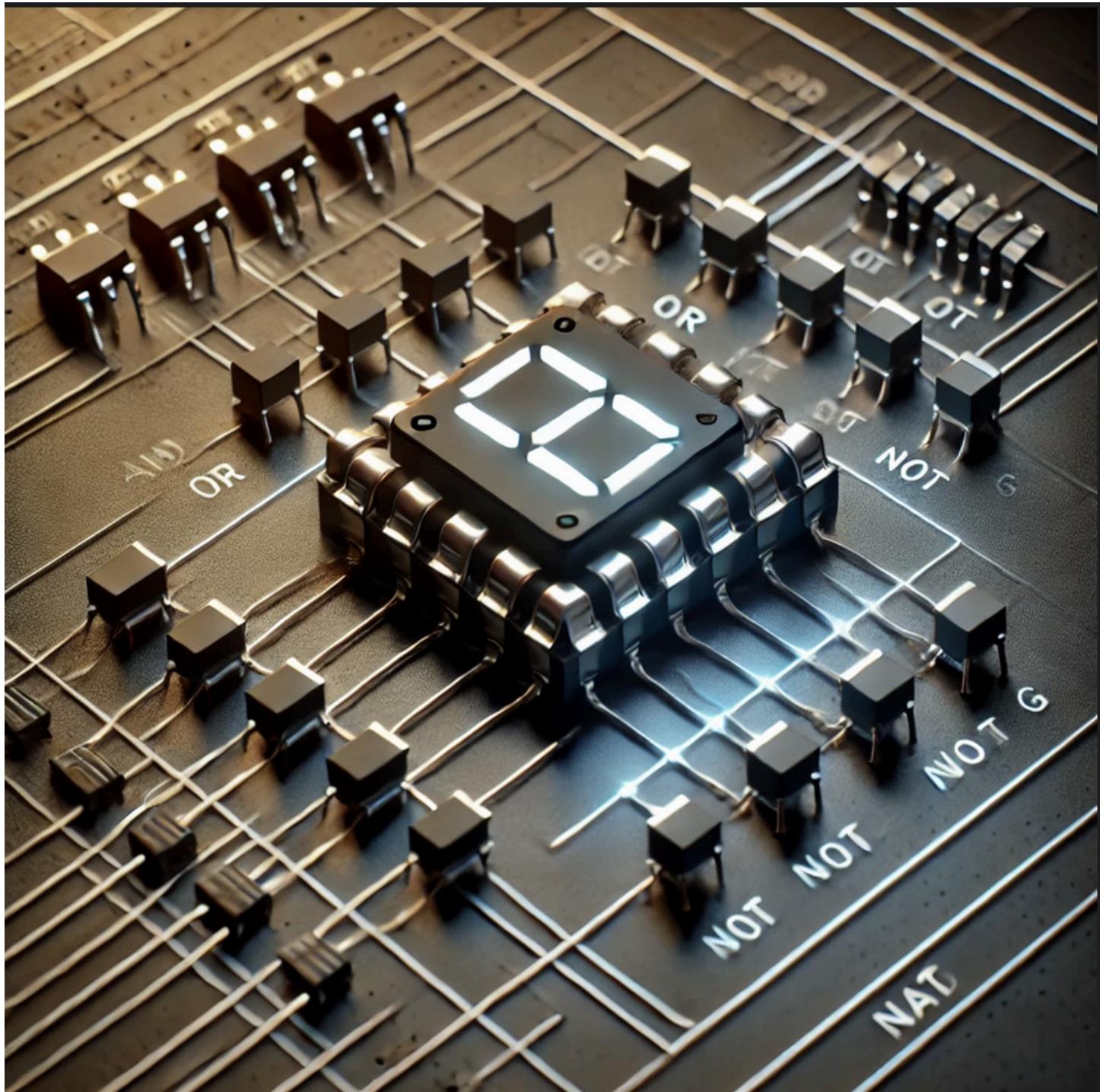


پروژه کامپیووتری دوم درس مدار منطقی

محمد امین رشید 810102454



الف) ابتدا برای اینکه ورودی و خروجی مدار به صورت one-hot باشد از یک دیکودر و یک انکودر استفاده میکنیم تا در top module از آن ها instance بگیریم و استفاده کنیم.

ابتدا در encoder یک عدد 16 بیتی میگیریم و ان را به یک عدد 4 بیتی در بازه بین منفی 8 تا 7 انکود میکنیم.

پس از انجام محاسبات لازم روی عدد 4 بیتی باید دوباره آن را 16 بیتی کنیم که برای اینکار از استفاده decoder میکنیم.

کد این دو ماژول به صورت زیر است:

```
one_hot_decoder.sv
```

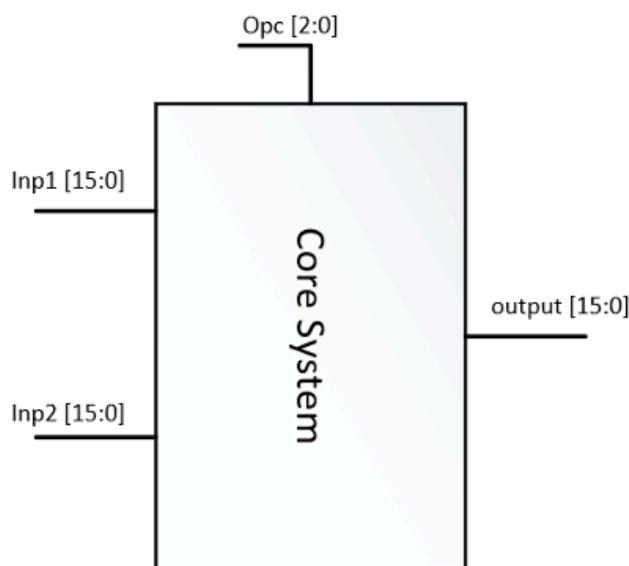
```
module one_hot_decoder(input [3:0] w, output reg [15:0] y);

    always @(*)
    begin
        case (w)
            4'b1000: y = 16'b1000000000000000;
            4'b1001: y = 16'b0100000000000000;
            4'b1010: y = 16'b0010000000000000;
            4'b1011: y = 16'b0001000000000000;
            4'b1100: y = 16'b0000100000000000;
            4'b1101: y = 16'b0000010000000000;
            4'b1110: y = 16'b0000001000000000;
            4'b1111: y = 16'b0000000100000000;
            4'b0000: y = 16'b0000000010000000;
            4'b0001: y = 16'b0000000001000000;
            4'b0010: y = 16'b0000000000100000;
            4'b0011: y = 16'b0000000000010000;
            4'b0100: y = 16'b0000000000001000;
            4'b0101: y = 16'b0000000000000100;
            4'b0110: y = 16'b0000000000000010;
            4'b0111: y = 16'b0000000000000001;
        endcase
    end
endmodule
```

```
one_hot_encoder.sv
```

```
1 module one_hot_encoder(input [15:0] w, output reg signed [3:0] y);
2
3     always @(*)
4         begin
5             case (w)
6                 16'b1000000000000000: y = -8;
7                 16'b0100000000000000: y = -7;
8                 16'b0010000000000000: y = -6;
9                 16'b0001000000000000: y = -5;
0                 16'b0000100000000000: y = -4;
1                 16'b0000010000000000: y = -3;
2                 16'b0000001000000000: y = -2;
3                 16'b0000000100000000: y = -1;
4                 16'b0000000010000000: y = 0;
5                 16'b0000000001000000: y = 1;
6                 16'b0000000000100000: y = 2;
7                 16'b0000000000010000: y = 3;
8                 16'b0000000000001000: y = 4;
9                 16'b0000000000000100: y = 5;
0                 16'b0000000000000010: y = 6;
1                 16'b0000000000000001: y = 7;
2
3             endcase
4         end
5     endmodule
6
```

سپس یک TOP_Module ایجاد کرده و به عنوان input دو تا عدد 16 بیتی و به عنوان خروجی یک عدد 16 بیتی تولید میکند.



شکل ۱ - شماتیک سیستم کلی شامل ورودی‌ها و خروجی

:top module کد

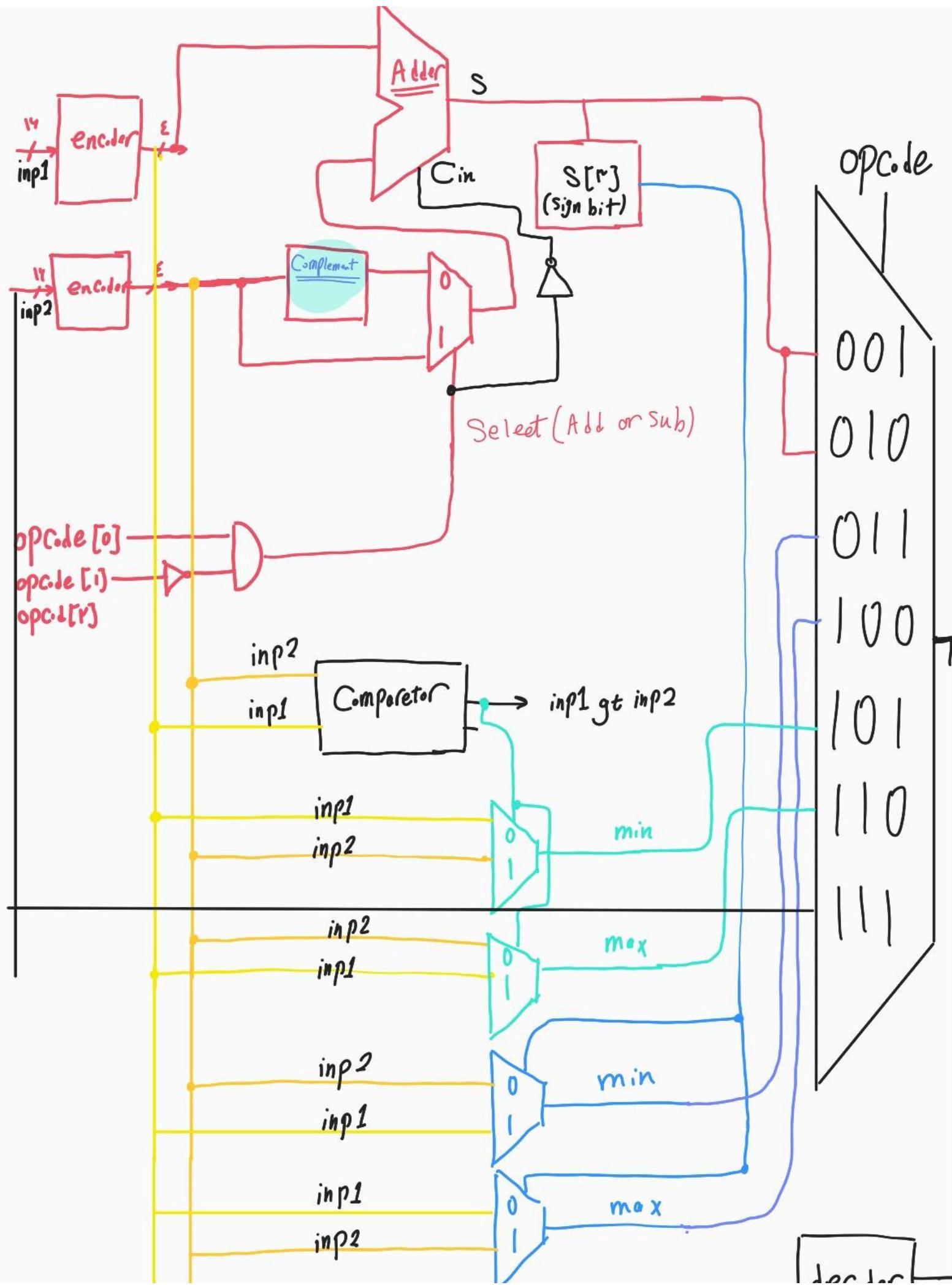
```

top_module.sv
1 module Top_Module(input [15:0] inp1_16,inp2_16,input [2:0] opcode , output [15:0] out_put_one_hot,output overflow, output [3:0] out_put);
2
3     wire [3:0] inp1_4,inp2_4;
4     one_hot_encoder T1(inp1_16,inp1_4);
5     one_hot_encoder T2(inp2_16,inp2_4);
6
7     ALU alu(inp1_4,inp2_4,opcode, out_put,overflow);
8
9     one_hot_decoder T3(out_put, out_put_one_hot);
0 endmodule

```

میرسیم به اصلی ترین بخش یعنی ALU 😊 که از آن در همین top module ، اینستنس گرفتیم.

ابتدا قبل از اینکه سراغ alu برویم یک شماتیک کلی از مدار را باهم بینیم:



Output

۹۲۰۵۴۱

در شماتیک کلی مداری که طراحی کردم یک Carry Lookahead adder و یک Comparetor استفاده کرده ام.

برای comparetor ، دو تاوردی به آن میدهم و سیگنال خروجی که دریافت میکنم اگر 1 باشد یعنی از inp2 بزرگ تر است و اگر 0 باشد بالعکس.

اما برای CLA ورودی اولم همان inp1 است ولی ورودی دوم میتواند inp2 و یا two's complement آن باشد . برای اینکار از یک مولتی پلکسر استفاده کردم تا مشخص کنم کدام را به عنوان ورودی به CLA دهم. بدین منظور نیاز دارم تا سیگنال مناسبی به عنوان select به مولتی پلکسر دهم تا بتوانم مشخص کنم که کدام ورودی انتخاب شود.

حالت های مختلف opcode براساس زیر است:

Instruction	Opcode	Operation
ADD	001	Result = $inp1 + inp2$
SUB	010	Result = $inp1 - inp2$
Min (by adder)	011	Result = min ($inp1, inp2$)
Max (by adder)	100	Result = max ($inp1, inp2$)
Min (by comparator)	101	Result = min ($inp1, inp2$)
Max (by comparator)	110	Result = max ($inp1, inp2$)
Move	111	Result = $inp2$

پس ما در زمان هایی که میخواهیم تفریق کنیم و min و max adder براساس حساب کنیم باید از تفریق کننده استفاده کنیم. و در حالت جمع باید از جمع کننده استفاده کنیم. و در سایر بخش ها تفاوتی ندارد(don't care).

برای طراحی این بخش از کارنو مپ کمک میگیریم. و خروجی را بر حسب بیت های opcode حساب میکنیم.

که معادله بھینه منطقی آن به شکل b.-c میشود. یعنی اند بیت صفر opcode با بیت اول opcode.

خروجی این اند را به مولتی پلکسر میدهیم تا نقیض ورودی دوم را به CLA بدهیم. همچنین Cin آن را هم 1 میکنیم.

opCode (سیستم حاسوب)

a	b	c	d
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

ab	00	01	11	10
c	0	0	0	0
1	1	0	0	0

\overline{bc}

برای پیدا کردن Min و Max در alu دو تا روش داریم که هر دو را پیاده سازی می‌کنیم.

ابتدا by adder است که ما در این 2 حالت CLA (Min and Max) با تفیریق انجام میدهیم و خروجی ما 2 می‌شود. بیت اول آن را بررسی می‌کنیم (بیت علامت)

اگر 1 بود:

$\min = \text{inp1}$

$\max = \text{inp2}$

اگر 0 بود:

$\min = \text{inp2}$

max=inp1

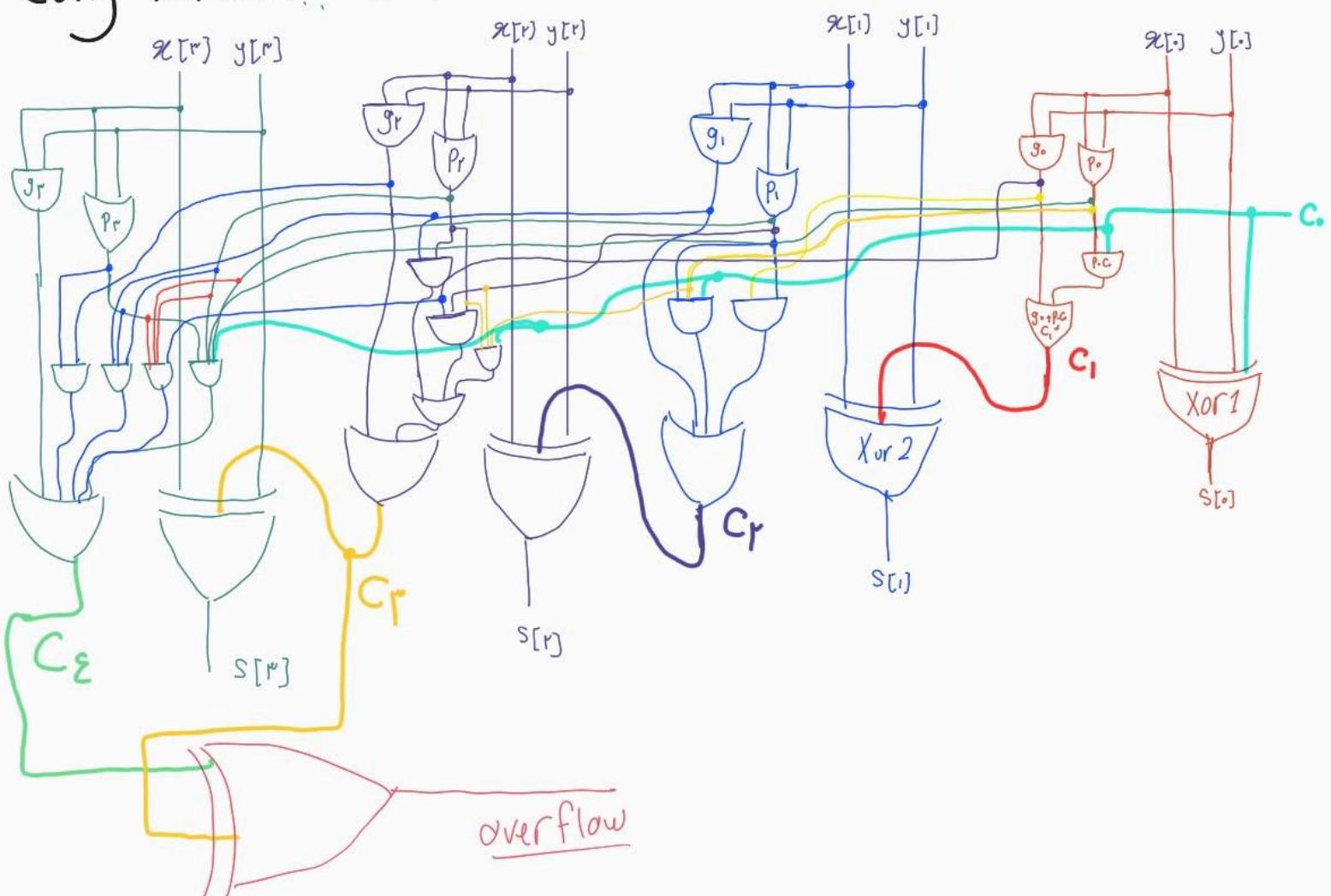
انتخاب این ها را به وسیله مولتی پلکسر انجام میدهیم.

در روش دوم که by comparetor است باز هم محاسبه میکنیم کدام ورودی بزرگ تر است و براساس مولتی پلکسر min و max را تعیین میکنیم.

در آخر هم از یک مولتی پلکسر بزرگ استفاده میکنیم تا از بین سیگنال های تولیدی ، سیگنال مطابق opcode را خروجی دهیم.

اکنون نوبت توضیح CLA است. ابتدا مدار آن را میبینیم و سپس کدش را (کد گیت لول)

Carry lookahead adder 8



$$C_1 = g_0 + p_0 C_0$$

$$C_2 = g_1 + p_1 g_0 + p_1 p_0 C_0$$

$$C_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 C_0$$

$$C_\epsilon = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 C_0$$

با توجه به ساختار بالا که به صورت گیت لول است میتوانیم کدش را بزنیم. ساختار آن با این شکل است که قرار است دو عدد 4 بیتی با هم جمع زده شوند و اینکار بیت به بیت انجام میشود و carry به قسمت بعد منقل میشود. در این ساختار هر carry فقط وابسته به C_0 است و منتظر تولید کری قبل از خود نمیشود و بدین منظور بهینه است.

برای زدن کدش کافیست طبق فرمول منطقی $C1, C2, C3, C4$ آن هارا پیدا کنیم و سپس هر بار از XOR سه ورودی استفاده کنیم تا Si را پیدا کنیم.

همچنین برای تشخیص OVERFLOW میتوانیم $C4$ با $C3$ را با هم XOR کنیم.

:CLA کد

adder.sv

```
1  module ADD(input [3:0] x,y, input cin,output [3:0] s,output overflow);
3      wire [4:0] carry;
4      wire [3:0] g;
5      wire [3:0] p;
6      wire[9:0] wires;
7      assign carry[0]=cin;
8
9      xor (s[0],x[0],y[0],carry[0]);
10     xor (s[1],x[1],y[1],carry[1]);
11     xor (s[2],x[2],y[2],carry[2]);
12     xor (s[3],x[3],y[3],carry[3]);
13
14    and(g[0],x[0],y[0]);
15    and(g[1],x[1],y[1]);
16    and(g[2],x[2],y[2]);
17    and(g[3],x[3],y[3]);
18
19
20    or(p[0],x[0],y[0]);
21    or(p[1],x[1],y[1]);
22    or(p[2],x[2],y[2]);
23    or(p[3],x[3],y[3]);
24
25    and(wires[0],p[0],carry[0]);
26    or(carry[1],g[0],wires[0]);
27
28
29    and(wires[1],p[0],p[1],carry[0]);
30    and(wires[2],p[1],g[0]);
31    or(carry[2],g[1],wires[1],wires[2]);
32
33    and(wires[3],p[0],p[1],p[2],carry[0]);
34    and(wires[4],p[1],p[2],g[0]);
35    and(wires[5],p[2],g[1]);
36    or(carry[3],g[2],wires[3],wires[4],wires[5]);
37
38    and(wires[6],p[0],p[1],p[2],p[3],carry[0]);
39    and(wires[7],p[1],p[2],p[3],g[0]);
40    and(wires[8],p[2],p[3],g[1]);
41    and(wires[9],p[3],g[2]);
42    or(carry[4],g[3],wires[6],wires[7],wires[8],wires[9]);
```

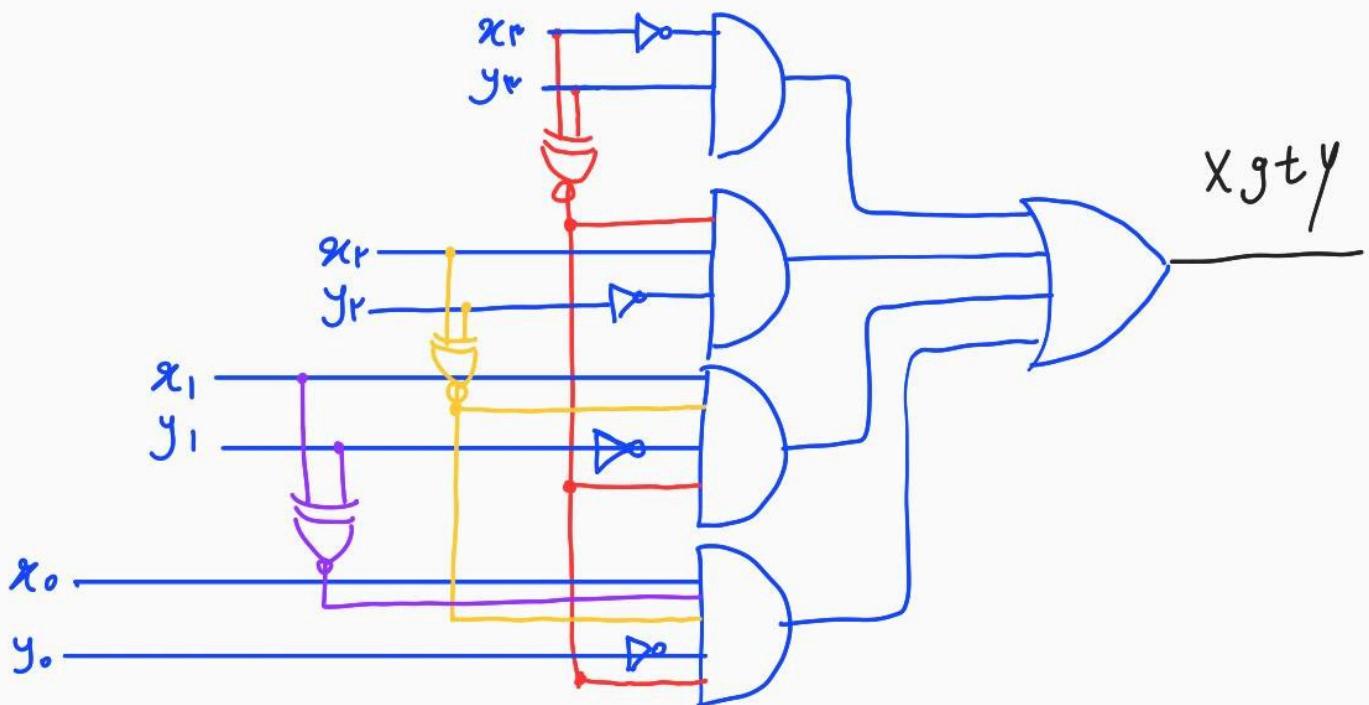
```

43
44     xor(overflow, carry[3], carry[4]);
45
46 endmodule

```

سپس میرسیم به comparator، تنها نکته قابل توجه این است که برخی اعداد منفی اند. برای این مدار را به صورت زیر طراحی میکنیم:

Comparator



سپس برای کد گیت لول داریم(کاملا مشابه طراحی بالا):

comparator.sv

```
module COMPARETOR(input [3:0] x,y ,output xgt);  
  
    wire[3:0] wires;  
    wire [3:1] i;  
    wire not_x3 , not_y2 , not_y1 , not_y0;  
    xnor (i[1],x[1],y[1]);  
    xnor (i[2],x[2],y[2]);  
    xnor (i[3],x[3],y[3]);  
  
    not(not_x3,x[3]);  
    not(not_y2,y[2]);  
    not(not_y1,y[1]);  
    not(not_y0,y[0]);  
  
    and(wires[0],x[0],not_y0,i[3], i[2],i[1]);  
    and(wires[1],x[1],not_y1,i[3],i[2]);  
    and(wires[2],x[2],not_y2,i[3]);  
    and(wires[3],not_x3,y[3]);  
  
    or(xgt,wires[0],wires[1],wires[2],wires[3]);  
  
endmodule
```

اکنون میتوانیم کد ALU را به راحتی بزنیم و در آن یکبار از مقایسه کننده و جمع کننده بگیریم.

برای مولتی پلکسر ها از سینتکس `case a : ?` و برای مولتی پلکسر بزرگ تر (سباه در شکل) از `always` statement درون `instance` استفاده میکنیم.

```

1 module ALU(input [3:0] inp1, input [2:0] opcode, output logic [3:0] out_put, wire overflow);
2
3     wire [3:0] f;
4     wire cin;
5     wire [3:0] s;
6     assign f=(opcode[0] & ~opcode[1]) ? inp2 : ~inp2;
7     assign cin=~(opcode[0] & ~opcode[1]);
8     wire xgty;
9     ADD adder(inp1,f,cin,overflow);
10    COMPARATOR comparetor(inp1,inp2,xgty);
11    wire [3:0] min_by_add,max_by_add,min_by_comp,max_by_comp;
12
13    assign min_by_add=s[3]?inp1 : inp2;
14    assign max_by_add=s[3]?inp2 : inp1;
15
16
17    assign min_by_comp= xgty?inp2 : inp1;
18    assign max_by_comp= xgty?inp1 : inp2;
19
20
21    always @(*)
22    begin
23        case(opcode)
24
25            3'b001 : out_put=s;
26            3'b010 : out_put=s;
27            3'b011 : out_put=min_by_add;
28            3'b100 : out_put=max_by_add;
29            3'b101 : out_put=min_by_comp;
30            3'b110 : out_put=max_by_comp;
31            3'b111 : out_put=inp2;
32
33        endcase
34    end
35
36 endmodule
37

```

اکنون باید تست بنچ بنویسیم و مدارمان را تست بکنیم:

در تست بنچ 4تا ورودی مختلف میدهیم و همه opcode های مختلف را چک میکنیم. و شکل موج های آن ها در Model Sim شبیه سازی میکنیم.

```
3 module Top_Module_tb;
5 logic [15:0] inp1_16, inp2_16;
6 logic [2:0] opcode;
7 wire [15:0] out_put_one_hot;
8 wire [3:0] out_put;
9 wire overflow;
10 Top_Module uut (inp1_16, inp2_16, opcode, out_put_one_hot, overflow , out_put);
11 integer i;
12
13 initial begin
14     inp1_16 = 16'b0000010000000000;
15     inp2_16 = 16'b000000000000100;
16
17     for (i = 1; i < 8; i = i + 1) begin
18         opcode = i % 8;
19         #10;
20     end
21
22
23     inp1_16 = 16'b000000000000100;
24     inp2_16 = 16'b0000000000001000;
25
26     for (i = 1; i < 8; i = i + 1) begin
27         opcode = i % 8;
28         #10;
29     end
30
31     inp1_16 = 16'b000000010000000;
32     inp2_16 = 16'b0010000000000000;
33
34     for (i = 1; i < 8; i = i + 1) begin
35         opcode = i % 8;
36         #10;
37     end
38
39     inp1_16 = 16'b0000100000000000;
40     inp2_16 = 16'b0000100000000000;
41
42     for (i = 1; i < 8; i = i + 1) begin
43         opcode = i % 8;
44         #10;
45     end
46 end
47
48 endmodule
```

شبیه سازی Wave ها و سیگنال ها:

حالت اول 😊

صحت سنجی:

تمامی اعداد مکمل 2 هستند.(two's complement)

inp1=-3 (decimal)

inp2=5 (decimal)

opcode:

001: sum → -3+5=2 -----> (decimal) 0010

010:sub → -3-5=-8 -----> (decimal) 1000

011:min → -3 -----> (decimal)1101

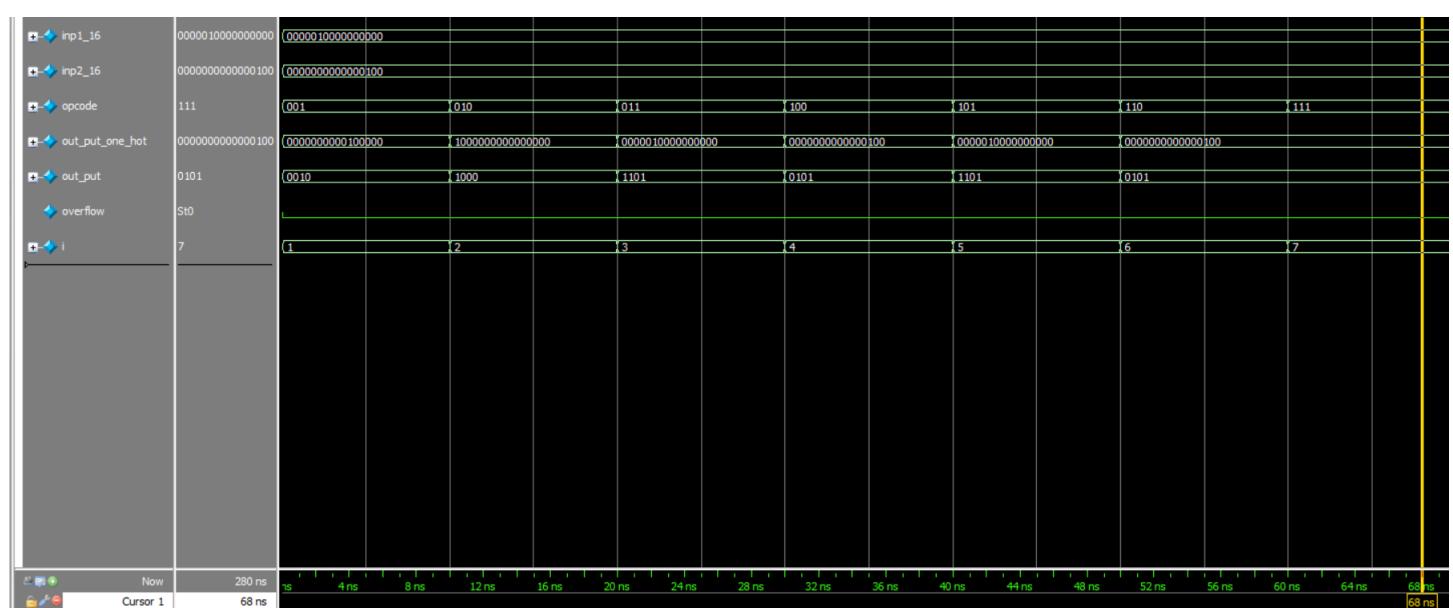
100:max → 5 -----> (decimal) 0101

101:min → -3 -----> (decimal)1101----->(one_hot)0000010000000000

110:max → 5 -----> (decimal)0101 ----->(one_hot)00000000000000100

111:inp2 → 5 -----> (decimal)0101

کاملا با سیگنال های خروجی مطابقت دارد.+ (ها one_hot)



inp1=5 (decimal)

inp2=4 (decimal)

opcode:

001: sum → 4+5=9 -----> باعث overflow و میشود!!

و سیگنال overflow در این حالت 1 میشود.

010:sub → 5-4=1 -----> (decimal) 0001

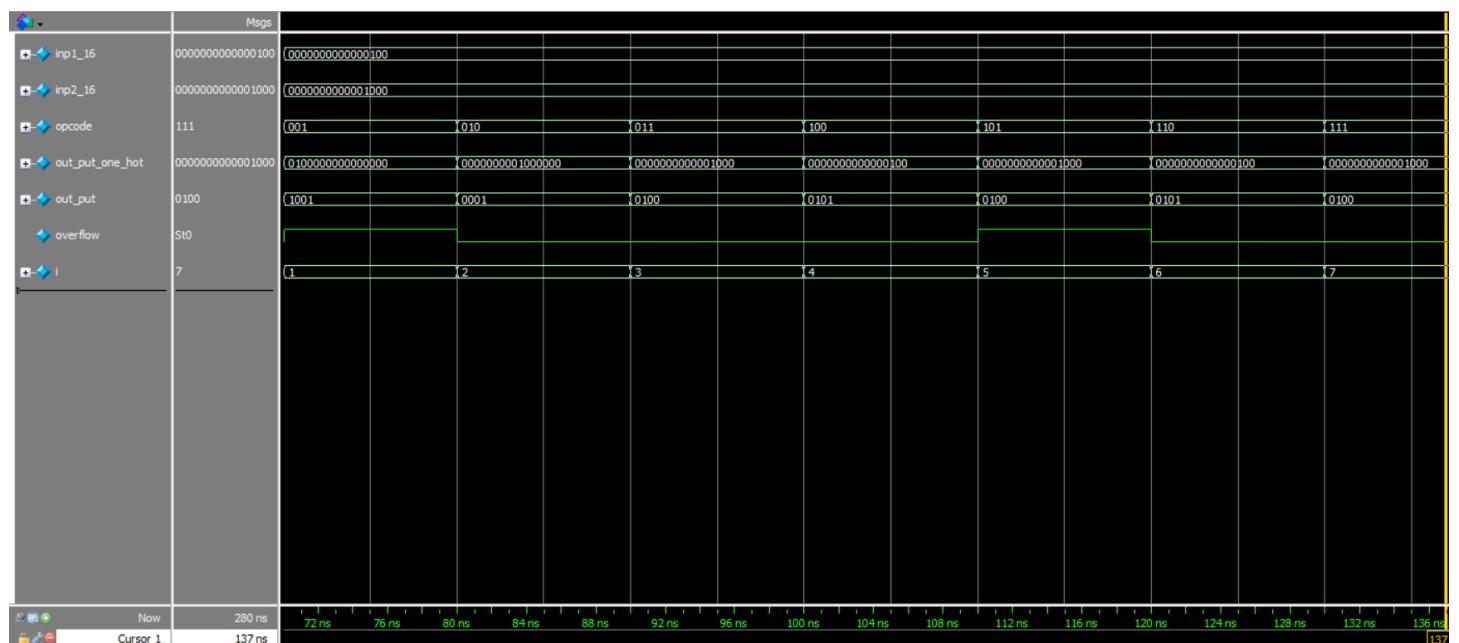
011:min → 4 -----> (decimal)1101

100:max → 5 -----> (decimal) 0101

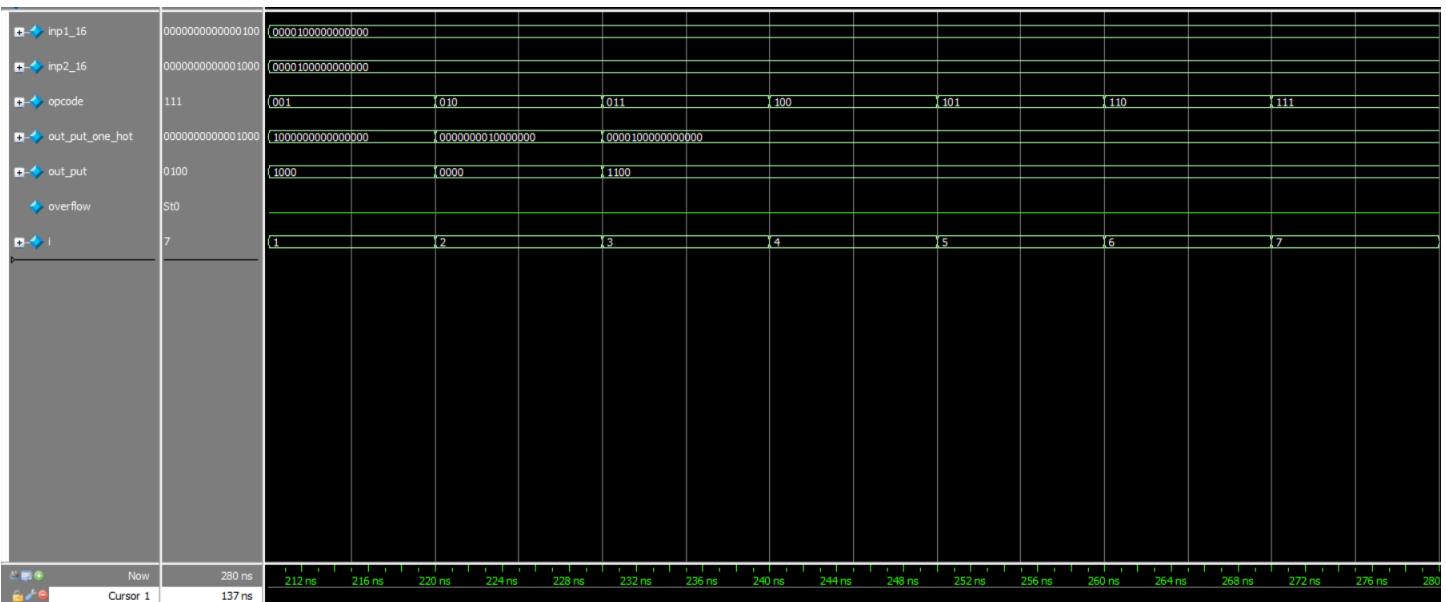
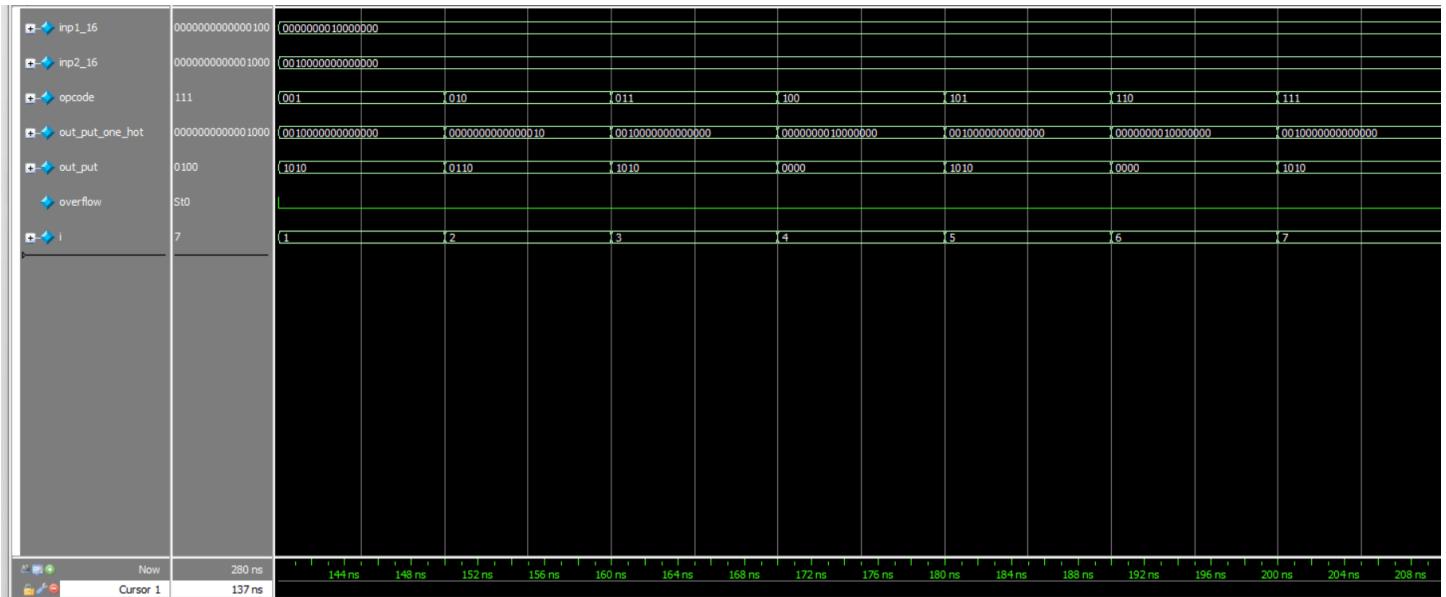
101:min → 4 -----> (decimal)1101----->(one_hot)00000000000001000

110:max → 5 -----> (decimal)0101 ----->(one_hot)0000000000000000100

111:inp2 → 4 -----> (decimal)0101



دو حالت دیگر را مانند حالات بالا میتوان به راحتی صحت سنجی کرد.



بخش امتیازی:

در این قسمت با تغییرات جزئی در مازول اصلی پروژه اول، یک مازول sevensegment decoder ساختیم که در واقع segment 8 است زیرا بیت آخر آن را که همان MSB است را به عنوان بیت علامت در نظر گرفتیم که اگر عدد منفی بود آن سگمنت اضافی روشن شود.

هم چنین تاخیر هارا هم برداشتیم و وردی را به صورت آرایه تعریف کردیم.

از این مازول در همان test bench یک اینستنس میگیریم و سیگنال خروجی 4 بیتی را به عنوان ورودی و یک سیگنال 8 بیتی را از آن خروجی میگیریم.

باید توجه داشته باشیم که اگر عدد منفی بود(-1 تا -8) باید ابتدا بیت MSB را یک کرده و سپس سایر سگمنت هارا براساس عدد معادل مکل 2 این اعداد حساب کنیم.

seven_segment_decoder.sv

```
module sevenSegmentDecoder(input logic [3:0] inp_s, output logic [7:0] out);

wire [3:0] inp;
assign inp = inp_s[3] ? (~inp_s + 1) : inp_s;
assign out[7] = inp[3];

wire not_inp3, not_inp2, not_inp1, not_inp0;
not N1 (not_inp3, inp[3]);
not N2 (not_inp2, inp[2]);
not N3 (not_inp1, inp[1]);
not N4 (not_inp0, inp[0]);

wire [8:0] terms;
and (terms[0], inp[2], inp[0]);
and (terms[1], not_inp0, not_inp2);
and (terms[2], inp[1], inp[0]);
and (terms[3], not_inp1, not_inp0);
and (terms[4], not_inp2, inp[1]);
and (terms[5], inp[1], not_inp0);
and (terms[6], inp[2], not_inp1, inp[0]);
and (terms[7], inp[2], not_inp0);
and (terms[8], inp[2], not_inp1);

or (out[0], inp[3], inp[1], terms[0], terms[1]); // a
or (out[1], not_inp2, terms[2], terms[3]); // b
or (out[2], not_inp1, inp[0], inp[2]); // c
or (out[3], inp[3], terms[4], terms[1], terms[5], terms[6]); // d
or (out[4], terms[5], terms[1]); // e
or (out[5], inp[3], terms[3], terms[7], terms[8]); // f
or (out[6], inp[3], terms[8], terms[5], terms[4]); // g

endmodule
```

module Top_Module_tb;

```
logic [15:0] inp1_16, inp2_16;
logic [2:0] opcode;
wire [15:0] out_put_one_hot;
wire [3:0] out_put;
wire overflow;
wire [7:0] seven_segment;
Top_Module uut (inp1_16, inp2_16, opcode, out_put_one_hot, overflow, out_put);
sevenSegmentDecoder ssd(out_put, seven_segment);
integer i;
```

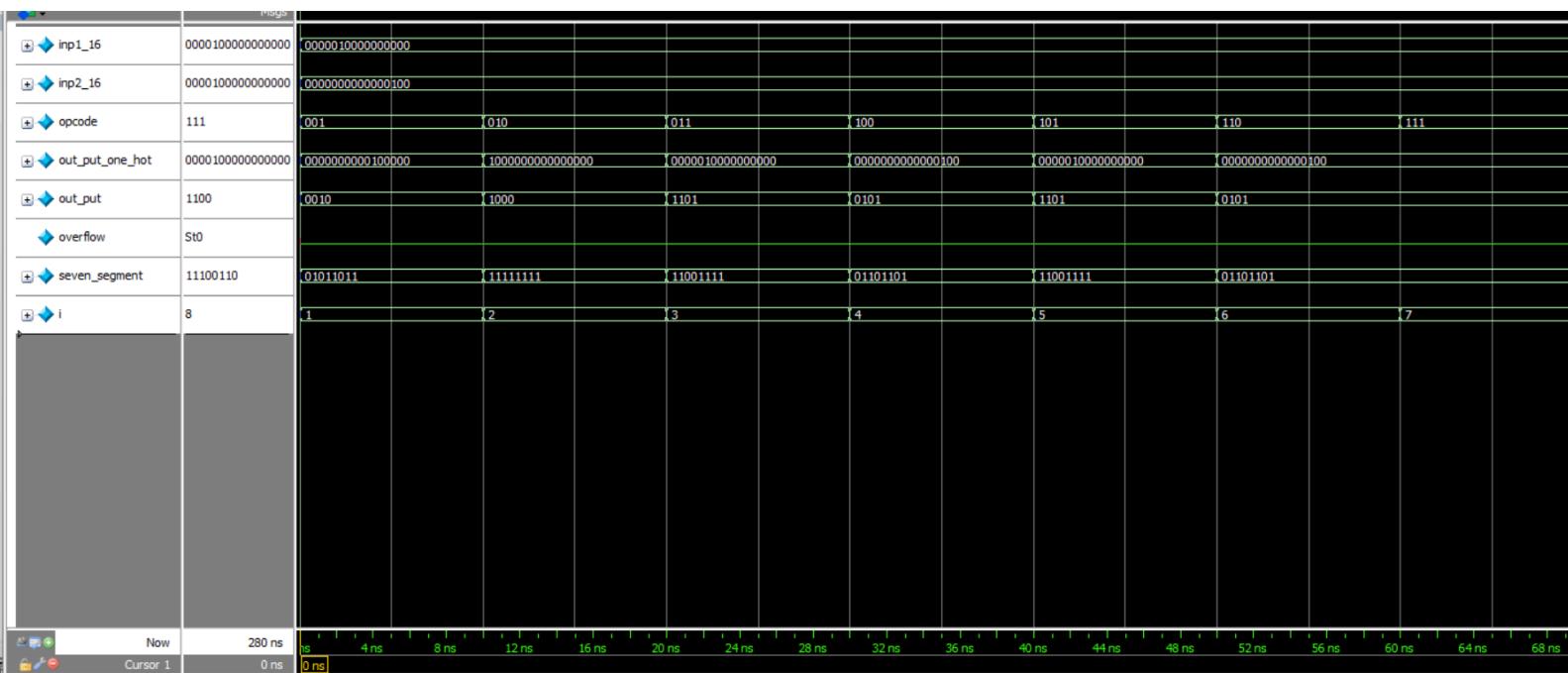
اکنون دوباره باید تست کنیم تا ببینیم خروجی صحیح است یا نه.

به عنوان مثال برای ورودی های

inp1=-3 (decimal)

inp2=5 (decimal)

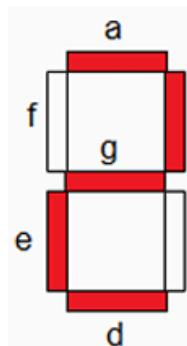
داریم:



در حالت $i=1$ خروجی برابر 2 شده است. 2 مثبت است پس بیت 8 خاموش باید باشد و سایر بیت ها هم با شکل

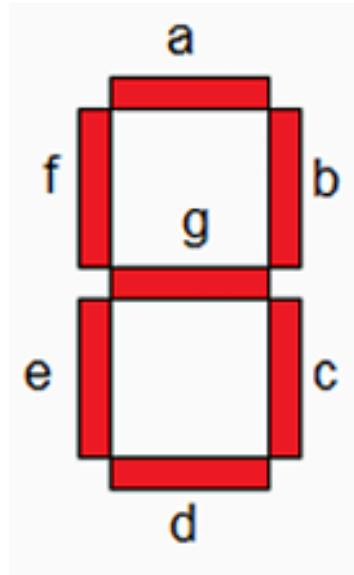
01011011

زیر مطابقت دارد:

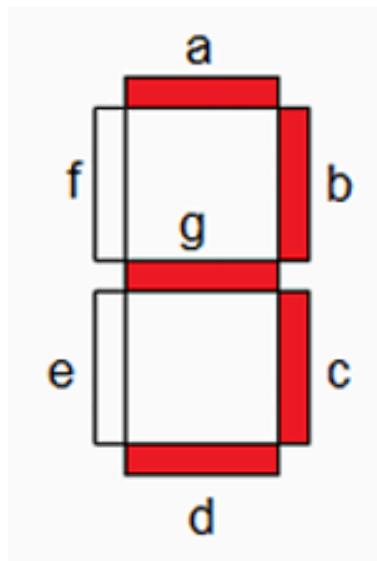


و یا در حالت $i=2$ خروجی منفی هشت شده است. پس MSB باید 1 باشد و سایر سگمنت ها همگی روشن (مانند

11111111 شکل زیر)



و یا در $i=3$ خروجی ما برابر منفی ۳ است پس باز هم بیت MSB یک است. و سایر بیت‌ها هم همانند شکل زیر:



The end.