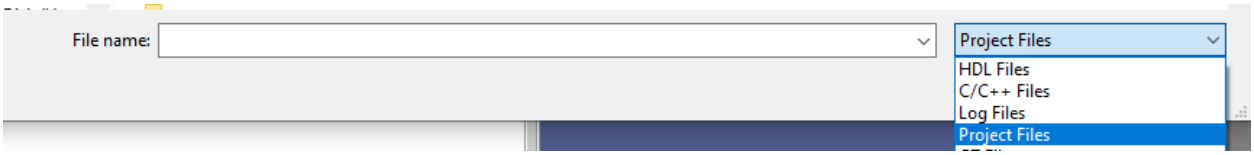
بنام خدا

علیرضا سعیدنیا

شماره دانشجویی ۴۰۰۱۰۸۳۳

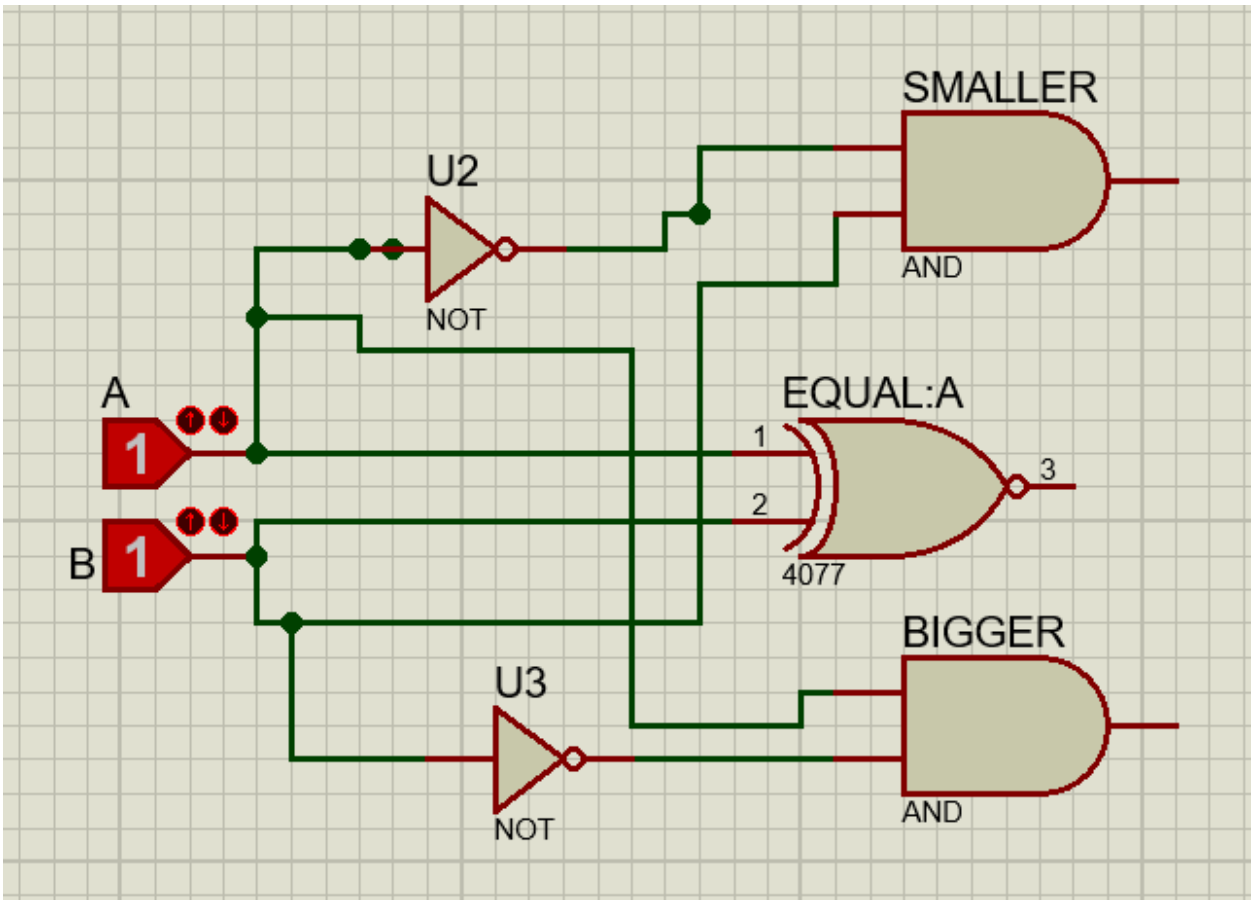
برای دسترسی به فایل ها هم فایل mpf **که شامل فایل پروژه هست را گذاشتم هم فایل های جداگانه. که برای دسترسی کامل به پروژه از طریق فایل** mpf **، این گزینه را انتخاب کنید .**



برای تولید مقایسه گر ۸ بیتی نیاز به این داریم که مقایسه گر تک بیتی بسازیم. جدول حالت مقایسه گر تک بیتی به صورت زیر خواهد بود .

INPUT		OUTPUT		
A	B	A < B	A = B	A > B
0	0	0	**1**	0
1	0	0	0	**1**
0	1	**1**	0	0
1	1	0	**1**	0

که اگر جدول کارنوی آن را بکشیم (هرچند نیازی ندارد) عمل مقایسه تک بیتی با گیتهای زیر صورت میگیرد که شکل آن را در پروتئوس کشیدم.

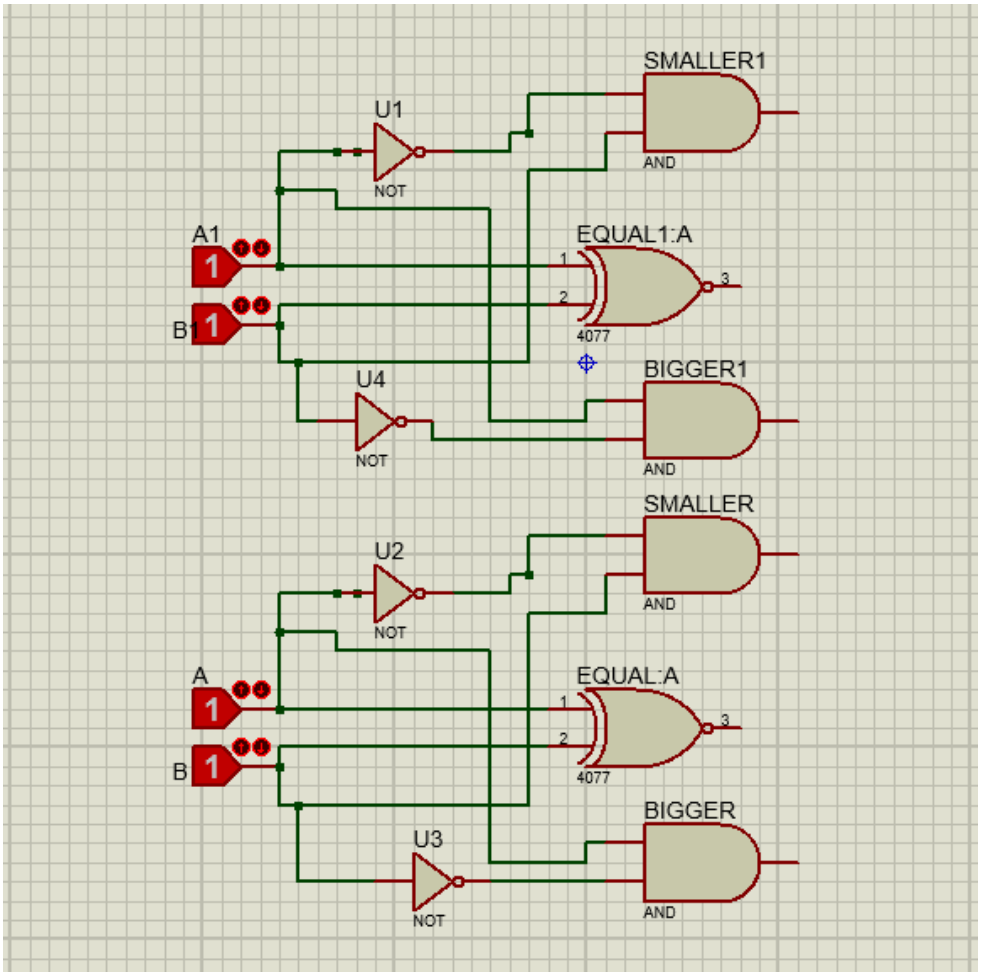


ماژول این مقایسه گر را در پروتئوس پیاده سازی کردم که طبق عکس مزبور است :

```
module comparator(s,e,bi,a,b);
input a,b;
output s,e,bi;
wire w1,w2;
xnor(e,a,b);
not(w1,a);
and (s,w1,b);
not(w2,b);
and (bi,w2,a);
endmodule;
```

سه خروجی کوچکتر بزرگتر مساوی و دو ورودی آ و بی داریم که نمایش داده شده است.

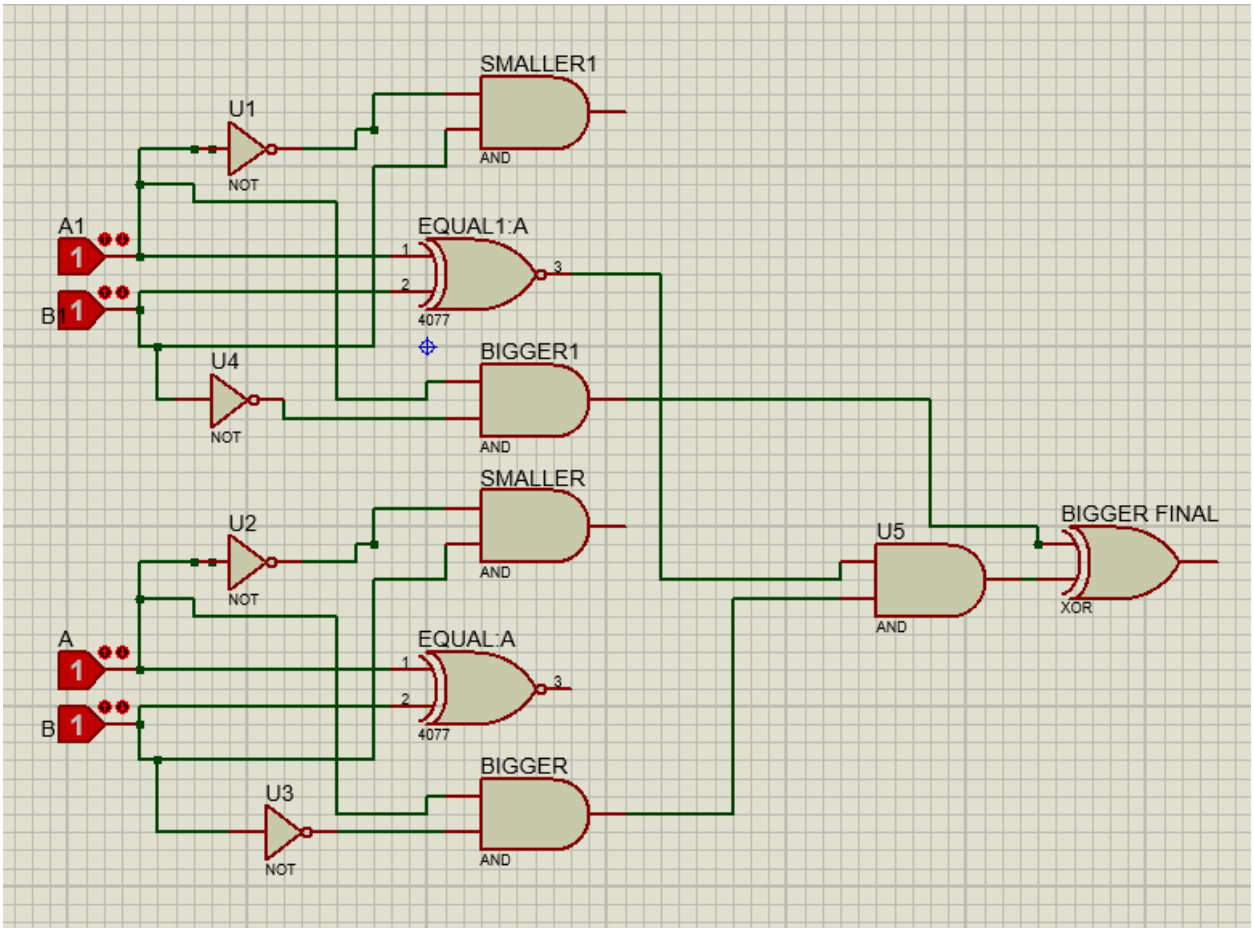
حال باید این مقایسه گر تک بیتی را دو بیتی کنم ، پس به دوتا از این مقایسه گر ها نیاز دارم که چیزی است که در عکس پایین میخواهم.



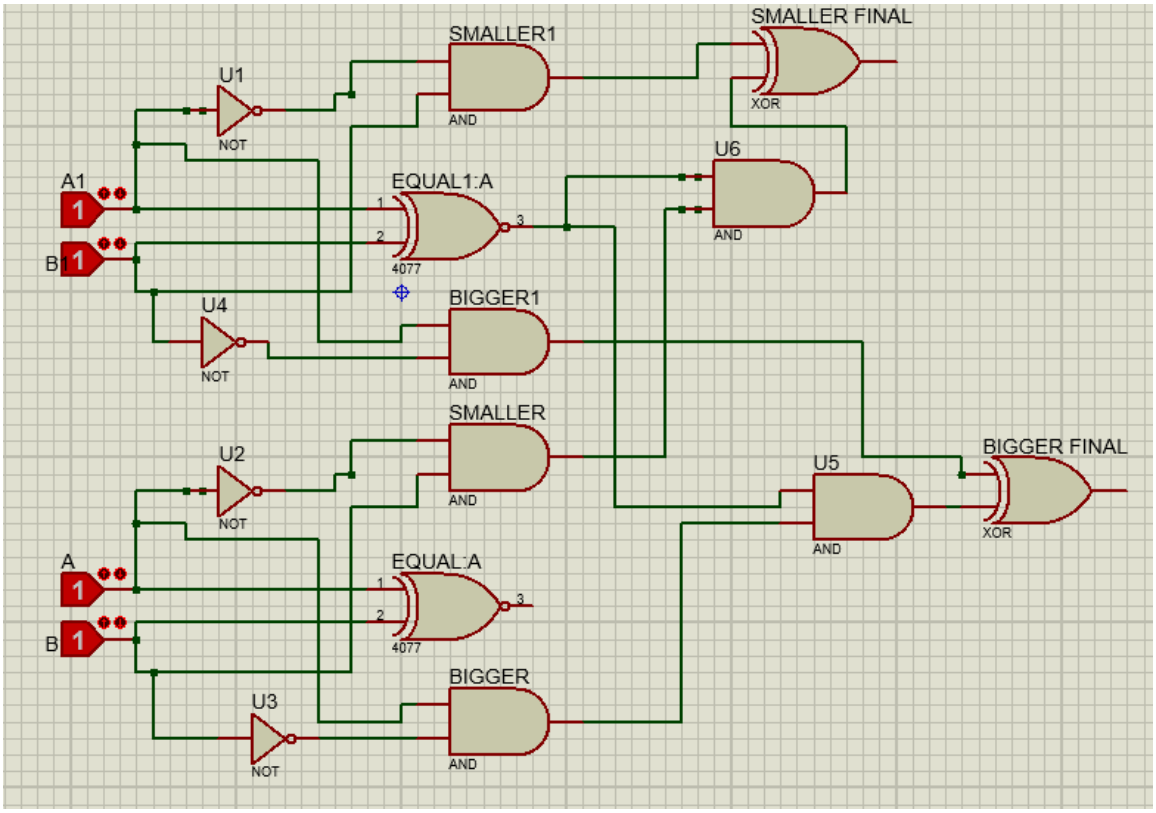
مشاهده میشود که ۶ خروجی شد ولی من سه خروجی میخواهم پس باید یک ارتباطی برقرار کنم که خروجی هایم سه تا شوند.

یک عدد زمانی از یک عدد دیگر بزرگتر است که سمت چپ ترین رقم ان از عدد دیگر بزرگتر باشد یـــا هم اگر رقم ها از سمت چپ باز هم مساوی شدند اخرین رقمی که بزرگتر شد به ما نشان بدهد که عدد ما در کل بزرگتر است.

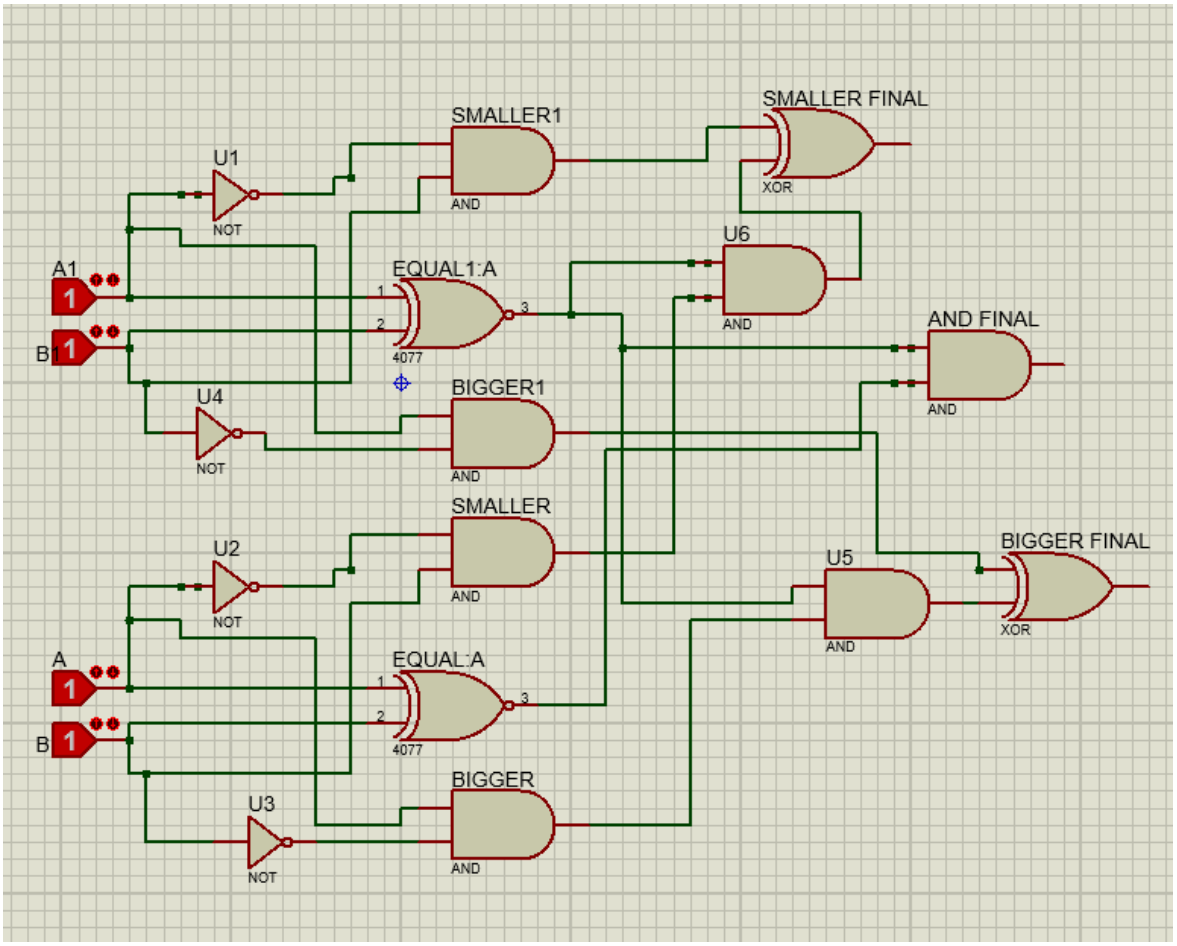
همین حرف را با گیتها پیاده سازی کردم و خروجی بزرگتر را ساختم



همین کار را برای کوچکتر بودن انجام میدهیم :



برای مساوی بودن هم واضح است تمام ارقام باید مساوی باشند پس آن هم انجام میدهیم:



بسیار خب حال همین کاری که انجام دادم را در یک ماژول جداگانه نوشتم. ماژول عکس زیر وظیفه دارد شش تا خروجی کوچکتر برابر و بزرگتر را دریافت کند و فقط سه تا خروجی به ما بدهد.

```
module connector(b2,e2,s2,b1,e1,s1,st,bit,et);
input b2,e2,s2,b1,e1,s1;
output st,bit,et;
wire w1,w2;

and(et,e2,e1);
and(w1,e2,b1);
xor(bit,w1,b2);
and(w2,e2,s1);
xor(st,w2,s2);
endmodule;
```

حال که مقایسه گر دو بیتی را ساختیم میتوانیم با ۴ بار instantiation **چهار بیتی بسازیم و به همین صورت هشت بیتی را ...**

در تست بنچ هم همین کار را کردم .

```
module moghayese;
reg [7:0]a,b;
wire [7:0]s,e,bi;
comparator comp5(s[7],e[7],bi[7],a[7],b[7]);
comparator comp6(s[6],e[6],bi[6],a[6],b[6]);
comparator comp7(s[5],e[5],bi[5],a[5],b[5]);
comparator comp8(s[4],e[4],bi[4],a[4],b[4]);
comparator comp(s[3],e[3],bi[3],a[3],b[3]);
comparator comp2(s[2],e[2],bi[2],a[2],b[2]);
comparator comp3(s[1],e[1],bi[1],a[1],b[1]);
comparator comp4(s[0],e[0],bi[0],a[0],b[0]);
connector me4(bi[7],e[7],s[7],bi[6],e[6],s[6],st4,bit4,et4);
connector me3(bi[5],e[5],s[5],bi[4],e[4],s[4],st3,bit3,et3);
connector me2(bi[3],e[3],s[3],bi[2],e[2],s[2],st2,bit2,et2);
connector me(bi[1],e[1],s[1],bi[0],e[0],s[0],st1,bit1,et1);

connector me6(bit4,et4,st4,bit3,et3,st3,f11,f21,f31);
connector me5(bit2,et2,st2,bit1,et1,st1,f10,f20,f30);

connector final(f21,f31,f11,f20,f30,f10,finalsmaller,finalbigger,finalequal);
initial
begin
#1 a=8'd55;b=8'd64;
end
endmodule;
```

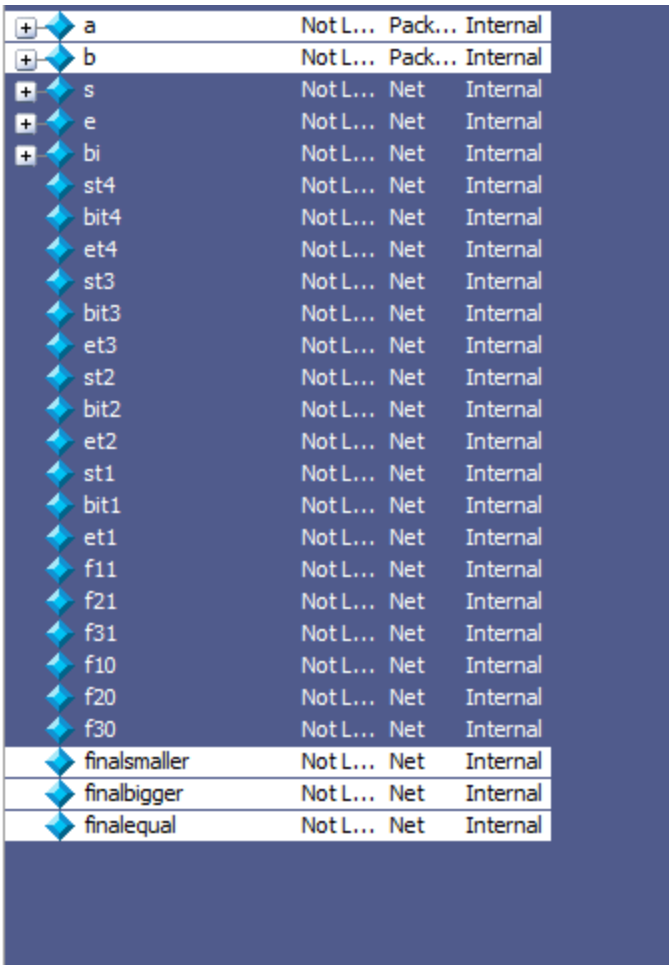
ورودی هارا رجیستر و خروجی هارا وایر میگیریم.

هشت بار ماژول مقایسه گر تک بیتی را (همان کامپراتور) صدا زده ام تا یک مقایسه گر ۸ بیتی بسازم ولی خب خروجی ها هنوز ۳ تا نیست ، ۲۴ تاست . پس باید انهارا ۳ تا کنم . با ۷ بار صدا زدن کامپراتور این کار را کرده ام که هر بار هی خروجی ها کمتر و کمتر میشود .

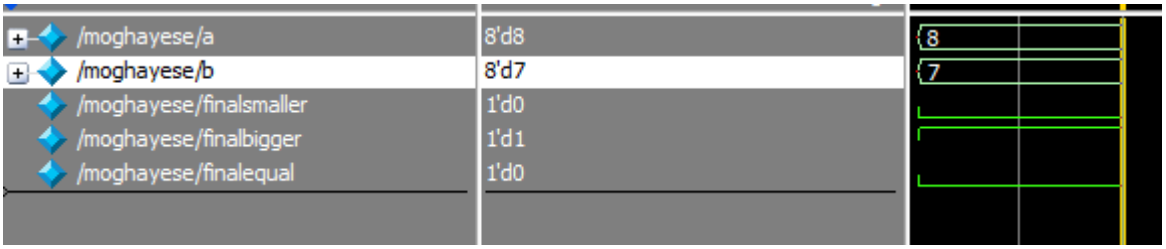
با صدا زدن کانکتور برای ۷ بار ۲۴ تا خروجی را هی تبدیل به ۳ تا خروجی و سه تا خروجی کردم که در نهایت به یک خروجی سه تایی برسم. دقت کنید ترتیب صدا زدن بزرگتر و کوچکتر و مساوی در تابع مهم است.

در نهایت خواسته و جواب مساله برابر هست با final bigger – final smaller – final equal

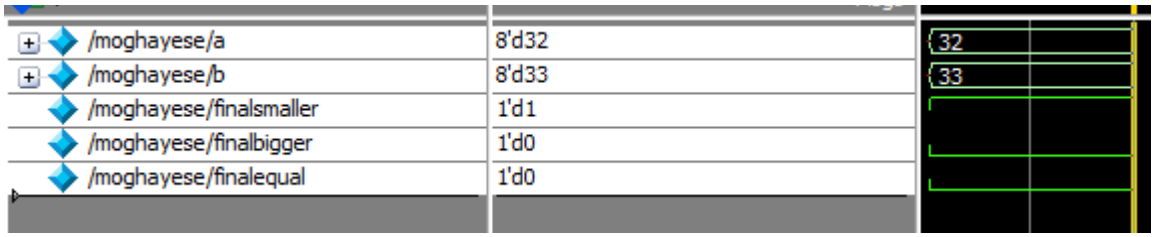
حال به سراغ wave **و نشان دادن درستی این برنامه میرویم.**



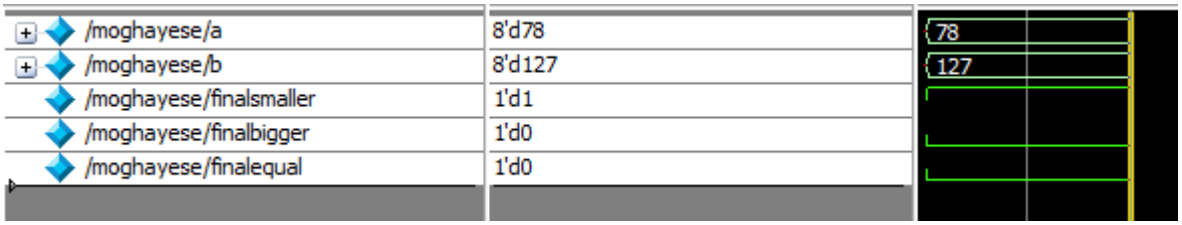
فقط ورودی ها و سه تا خروجی نهایی برای ما مهم هستند پس ان هارا سلکت میکنیم و اد ویو میکنیم و سپس ران.



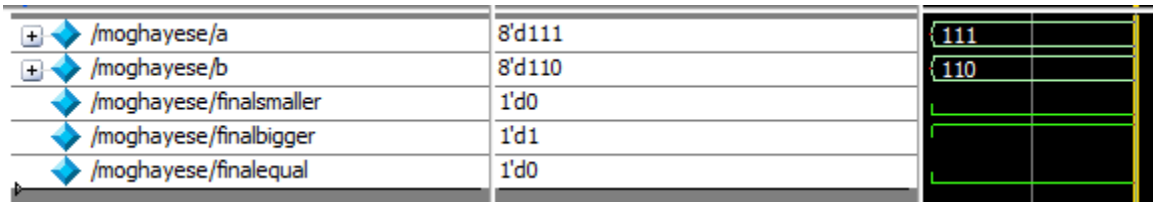
هشت از هفت بزرگتر است پس فاینال بیگر ۱ شد.



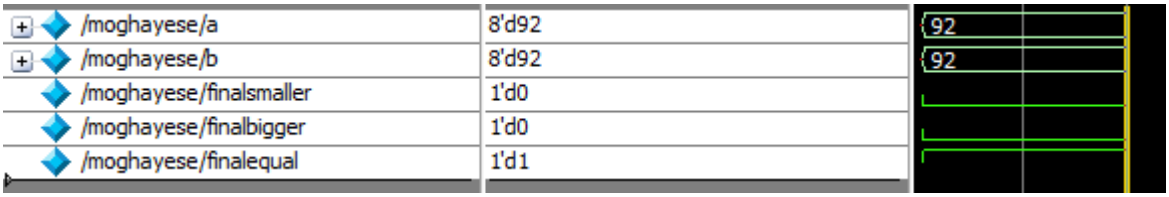
سی و دو از سی و سه کوچکتر است پس فاینال اسمالر یک شد.



هفتاد و هشت از ۱۲۷ کوچکتر است پس اسمالر یک شد.

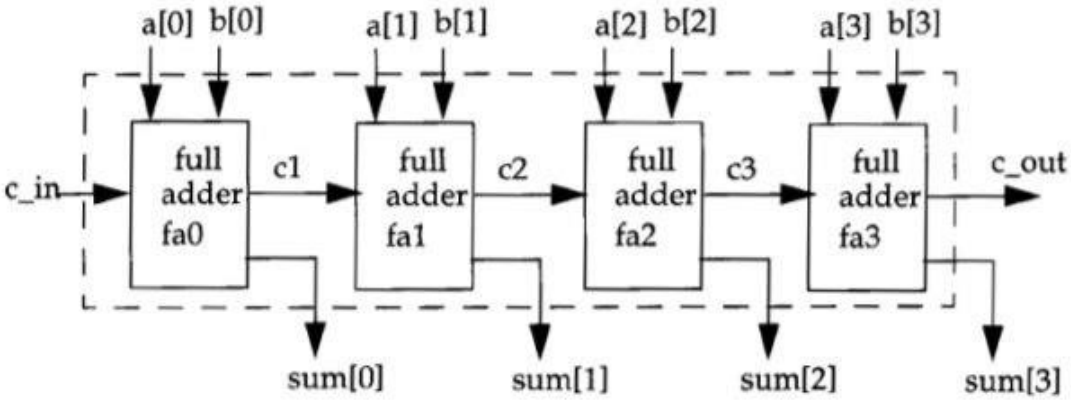


صد و یازده از صد و ده بزرگتر است پس بیگر ۱ شد.

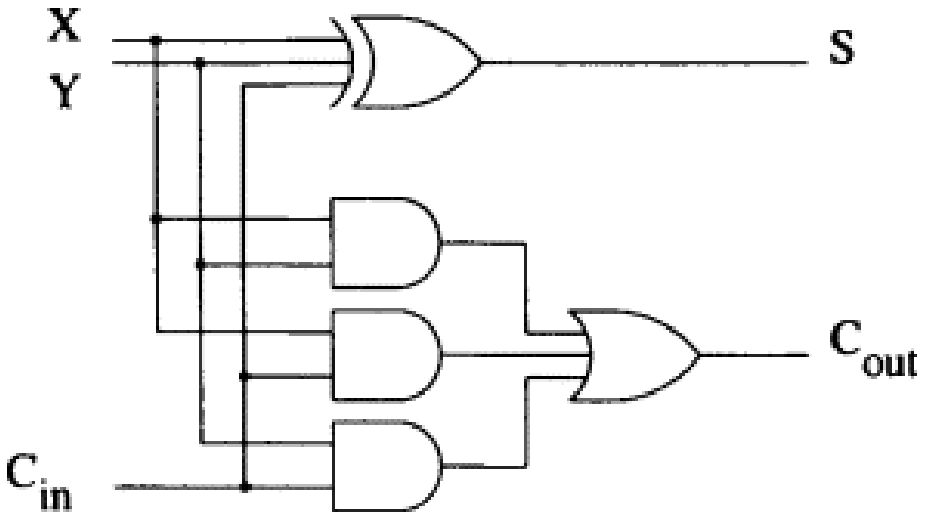


نود و دو با نود و دو برابر است پس ایکوال یک شد.

برای ساخت فول ادر سی و دو بیتی به همچین شکلی نیاز داریم



اگر با جدول حالت پیش برویم ، یکی از راه های ساختن فول ادر به صورت عکس زیر است.



همین را ماژولش را ساختم که به صورت عکس زیر است

```
module onebitfulladder (a,b,cin,cout,sum);


input  a;
input  b;
input cin;
output sum;
output cout;
wire w1,w2,w3;
xor xor1(sum,a,b,cin);
and and1(w1,a,b);
and and2(w2,a,cin);
and and3(w3,cin,b);
or or1(cout,w1,w2,w3);

endmodule;
```

که می دانیم برای ساختن یک فول ادر دو بیتی باید بیاییم و کاری کنیم که cout **قبلی برابر با** cin **خروجی باشد پس یک ماژول میانی داریم که ۳۲ بار دارد این ماژول اصلی مارا صدا میزند و سپس یک کری ۱–۳۲ تایی را هی به فول ادرهای ما پاس میدهد .**

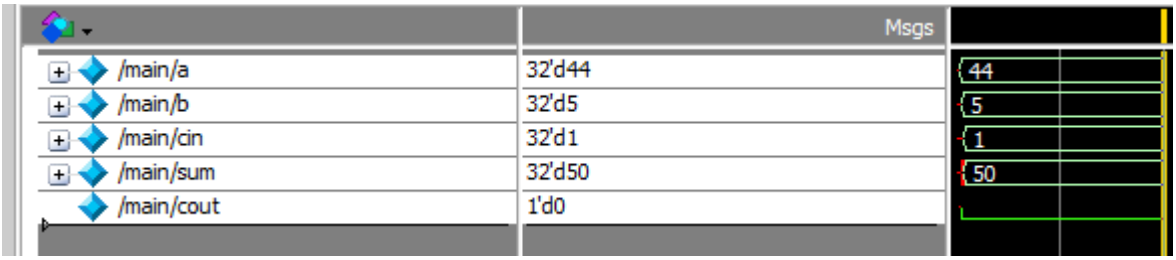
و هدف ما طبق عکس ابتدایی جمع کردن بیت به بیت هست پس سام و خروجی نهایی هم بیت به بیت بیرون می اید که از چپ به راست ان را میخوانیم. در آخر هم یک cout **خروجی داریم .**

همین را تبدیل به ماژول میکنیم. که به صورت عکس مزبور است. ورودی ها طبیعتا ۳۲ بیتی و خروجی کری ۱ بیتی است . سام هم سی و دو بیتی است. ولی کری ای که دارد رد و بدل میشود همان ۳۱ بیتی است که در بالا دلیلش را گفتم چرا.

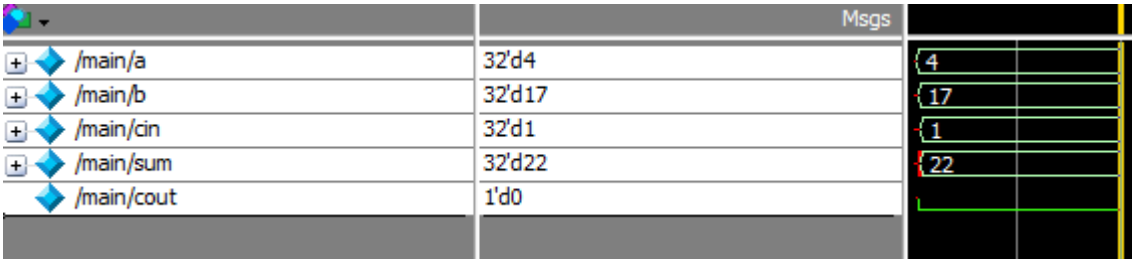
```
module add32bits(a,b,cin,cout,sum);
input [31:0] a,b;
input cin;
output [31:0] sum;
output cout;
wire [30:0]c;

onebitfulladder f1(a[0],b[0],cin,c[0],sum[0]);
onebitfulladder f2(a[1],b[1],c[0],c[1],sum[1]);
onebitfulladder f3(a[2],b[2],c[1],c[2],sum[2]);
onebitfulladder f4(a[3],b[3],c[2],c[3],sum[3]);
onebitfulladder f5(a[4],b[4],c[3],c[4],sum[4]);
onebitfulladder f6(a[5],b[5],c[4],c[5],sum[5]);
onebitfulladder f7(a[6],b[6],c[5],c[6],sum[6]);
onebitfulladder f8(a[7],b[7],c[6],c[7],sum[7]);
onebitfulladder f9(a[8],b[8],c[7],c[8],sum[8]);
onebitfulladder f10(a[9],b[9],c[8],c[9],sum[9]);
onebitfulladder f11(a[10],b[10],c[9],c[10],sum[10]);
onebitfulladder f12(a[11],b[11],c[10],c[11],sum[11]);
onebitfulladder f13(a[12],b[12],c[11],c[12],sum[12]);
onebitfulladder f14(a[13],b[13],c[12],c[13],sum[13]);
onebitfulladder f15(a[14],b[14],c[13],c[14],sum[14]);
onebitfulladder f16(a[15],b[15],c[14],c[15],sum[15]);
onebitfulladder f17(a[16],b[16],c[15],c[16],sum[16]);
onebitfulladder f18(a[17],b[17],c[16],c[17],sum[17]);
onebitfulladder f19(a[18],b[18],c[17],c[18],sum[18]);
onebitfulladder f20(a[19],b[19],c[18],c[19],sum[19]);
onebitfulladder f21(a[20],b[20],c[19],c[20],sum[20]);
onebitfulladder f22(a[21],b[21],c[20],c[21],sum[21]);
onebitfulladder f23(a[22],b[22],c[21],c[22],sum[22]);
onebitfulladder f24(a[23],b[23],c[22],c[23],sum[23]);
onebitfulladder f25(a[24],b[24],c[23],c[24],sum[24]);
onebitfulladder f26(a[25],b[25],c[24],c[25],sum[25]);
onebitfulladder f27(a[26],b[26],c[25],c[26],sum[26]);
onebitfulladder f28(a[27],b[27],c[26],c[27],sum[27]);
onebitfulladder f29(a[28],b[28],c[27],c[28],sum[28]);
onebitfulladder f30(a[29],b[29],c[28],c[29],sum[29]);
onebitfulladder f31(a[30],b[30],c[29],c[30],sum[30]);
onebitfulladder f32(a[31],b[31],c[30],cout,sum[31]);



endmodule;
```

حال نوبت به تست ورودی ها میرسد ورودی هارا رجیستر و خروجی هارا وایر میگیریم.

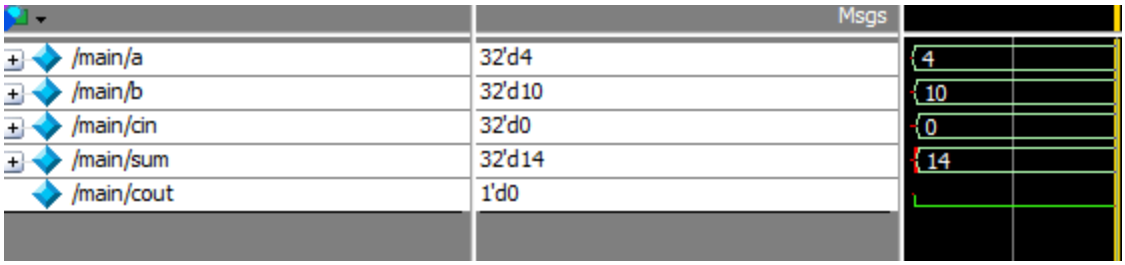
```
module main;
reg [31:0] a;
reg [31:0] b;
reg [31:0] cin;

wire [31:0]sum;
wire cout;
add32bits me (a,b,cin,cout,sum);

initial
begin

#1 a=32'd44;
#1 b=32'd5;
#1cin=32'd1;

end

endmodule;
```



جمع عدد ۴۴ با ۵ با یک کری این برابر ۵۰ است.



جمع ۴ با ۱۷ با یک کری این برابر ۲۲ است.



جمع ۴ با ده بدون کری این برابر ۱۴ است.