In His name

University of Tehran

Faculty of electrical and computer engineering

Digital Systems 1

Computer Assignment 2

Amirhosein Yavarikhoo	Name
810199514	ID Number
1401/1/23	Date

فهرست گزارش سوالات (لطفاً پس از تکمیل گزارش، این فهرست را به‌روز کنید.)

Q1

To design this RTL component, we use NOR and NOT gate from CA1. To find the logic behind the comparator we use Karnaugh maps to minimize each of the outputs. Logics are shown below:

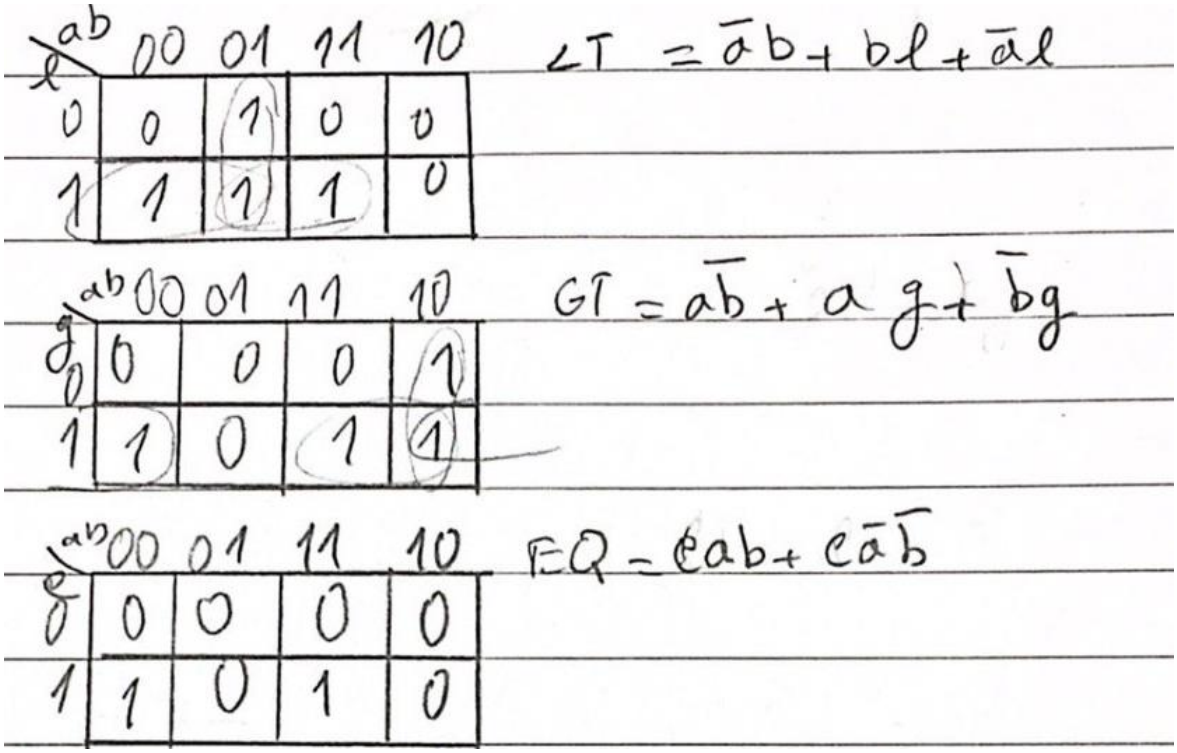


Figure 1 Logic values

So now that we have logic values of each output, we design the gate with NOR and NOT gates.

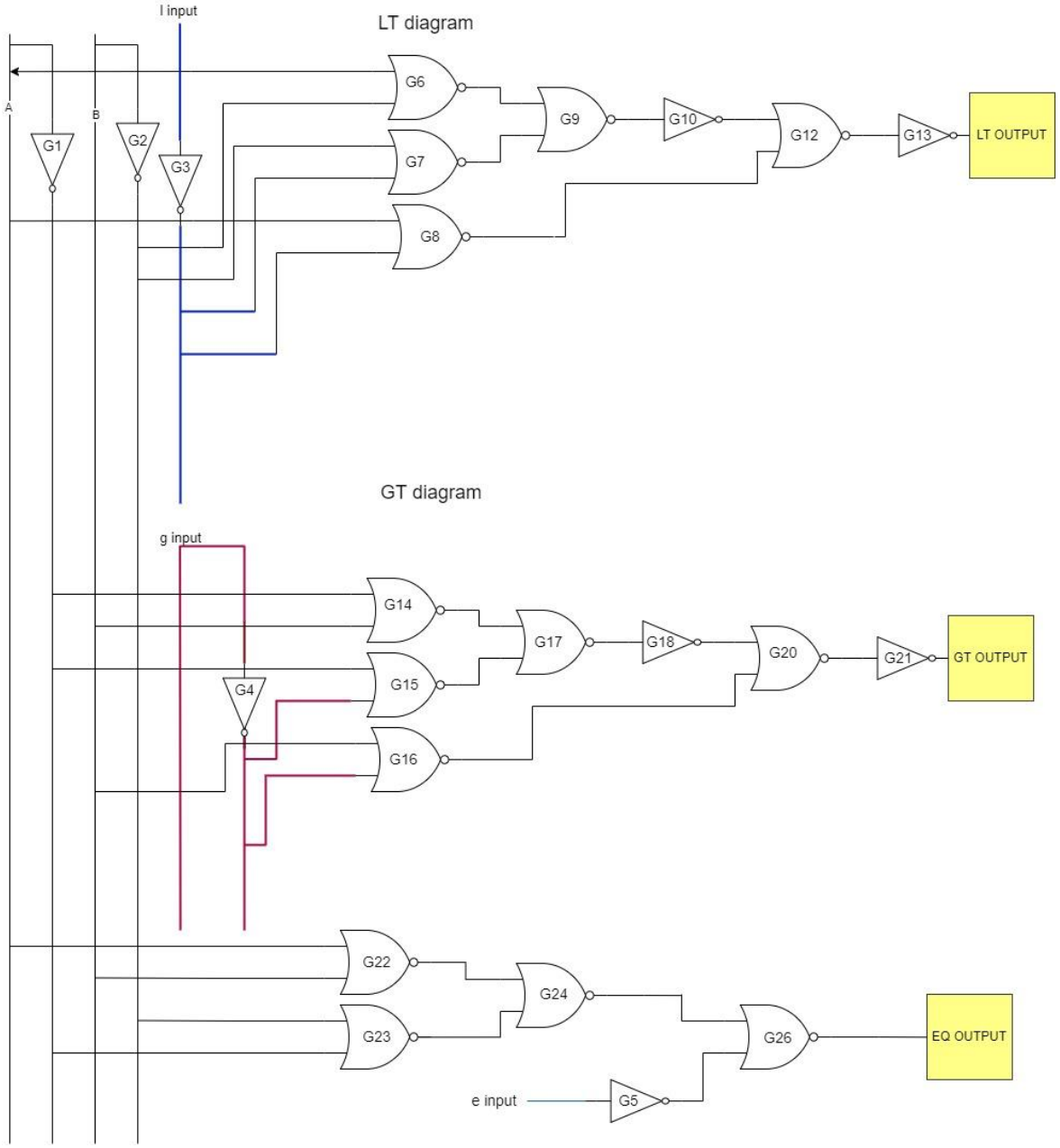


Figure 2 bit_comparator design

Verilog code for this comparator is shown below:

```
`timescale 1ns/1ns
module bit_comparator(input a,b,l,e,g,output LT,GT,EQ);
        wire [26:0] j;
        Myinverter G1(.a(a),.w(j[1]));
        Myinverter G2(.a(b),.w(j[2]));
        Myinverter G3(.a(l),.w(j[3]));
        Myinverter G4(.a(g),.w(j[4]));
        Myinverter G5(.a(e),.w(j[5]));
        MyNor G6(.a(a),.b(j[2]),.w(j[6])),G7(.a(j[2]),.b(j[3]),.w(j[7])),G8(.a(a),.b(j[3]),.w(j[8])),G9(.a(j[6]),.b(j[7]),.w(j[9])),G12(.a(j[10]),.b(j[8]),.w(j[12]));
        Myinverter G10(.a(j[9]),.w(j[10])),G13(.a(j[12]),.w(LT));
        MyNor G14(.a(j[1]),.b(b),.w(j[14])),G15(.a(j[1]),.b(j[4]),.w(j[15])),G16(.a(b),.b(j[4]),.w(j[16])),G17(.a(j[14]),.b(j[15]),.w(j[17])),G20(.a(j[18]),.b(j[16]),.w(j[20]));
        Myinverter G18(.a(j[17]),.w(j[18])),G21(.a(j[20]),.w(GT));
        MyNor G22(.a(a),.b(b),.w(j[22])),G23(.a(j[1]),.b(j[2]),.w(j[23])),G24(.a(j[22]),.b(j[23]),.w(j[24])),G26(.a(j[24]),.b(j[5]),.w(EQ));

endmodule
```

Note that gate and wire numbers correspond with the graphical representation of figure 2

b) To find out the worst case delays, we find the longest path for each output.

Delays for 2 input NOR gates are: To 1:10ns TO 0:14ns

Delays for 2 input NOT gates are: TO 1:7ns TO 0: 5ns

- LT output: Longest path has 3 NOR gates and two inverters. When b is 0 and it changes to b=1 we have the worst case To1 delay which is: 3*10+2*7=44ns. When b is 1 and it changes to b=0 we have the worst case To0 delay which is :3*14+2*5=52ns.
- GT output: Since GT and LT diagrams are the same in logic and if we swap a with b in descriptions above, we find out worst case delays which are the same with LT.
- EQ output: Worst case To1 and to 0 occurs when one of the following inputs (a, b) changes. Worst case To1 delay is: 3*10=30 ns. Worst case To0 delay is 3*14=42ns.

c)

```
1  `timescale 1ns/1ns
2  module bitcompTB_partC();
3          logic aa=0,bb=0;
4          logic eqout,ltout,gtout;
5          logic ll=0,gg=0,ee=1;
6          bit_comparator CUT(.a(aa),.b(bb),.LT(ltout),.GT(gtout),.EQ(eqout),.l(ll),.g(gg),.e(ee));
7          initial begin
8          #200;
9          #200 bb=1;
10         #200 bb=0;
11         #200 aa=1;
12         #200 aa=0;
13         #200 bb=1;
14         #200 aa=1;
15         #200 bb=0;
16         end
17 endmodule
```

Figure 3 Testbench code

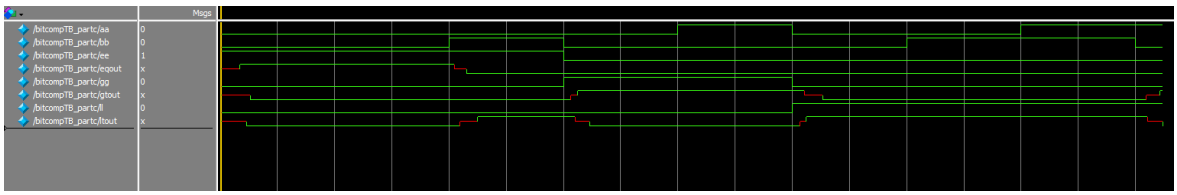


Figure 4 Testbench Result

As we can see the comparator works correctly.

d) delay values extracted from waveform are shown below:

Output	1 to 0	0 to 1
LT	45ns	50ns
GT	53ns	55ns
EQ	42 ns	30 ns

```
1   `timescale 1ns/1ns
2   module bit_comparator_assign(input a,b,l,e,g,output LT,GT,EQ);
3           assign #(50,45) LT=(~a&b)|(b&l)|(~a&l);
4           assign #(55,53) GT=(a&~b)|(a&g)|(~b&g);
5           assign #(30,42) EQ=(e&a&b)|(e&~a&~b);
6   endmodule
```

Figure 5 comparator using assign

e) This is the code written for comparing comparators.

```
1   `timescale 1ns/1ns
2   module bitcompTB_parte();
3           logic aa=0,bb=0;
4           logic eqout,ltout,gtout,ltout1,eqout1,gtout1;
5           logic ll=0,gg=0,ee=1;
6           bit_comparator CUT(.a(aa),.b(bb),.LT(ltout),.GT(gtout),.EQ(eqout),.l(ll),.g(gg),.e(ee));
7           bit_comparator_assign CUT1(.a(aa),.b(bb),.LT(ltout1),.GT(gtout1),.EQ(eqout1),.l(ll),.g(gg),.e(ee));
8           initial begin
9           #200;
10          #200 bb=1;
11          #200 bb=0;ee=0;gg=1;
12          #200 aa=1;
13          #200 aa=0;ee=0;gg=0;ll=1;
14          #200 bb=1;
15          #200 aa=1;
16          #200 bb=0;
17          end
18  endmodule
```

Figure 6 testbench code

Waveform result is shown below.

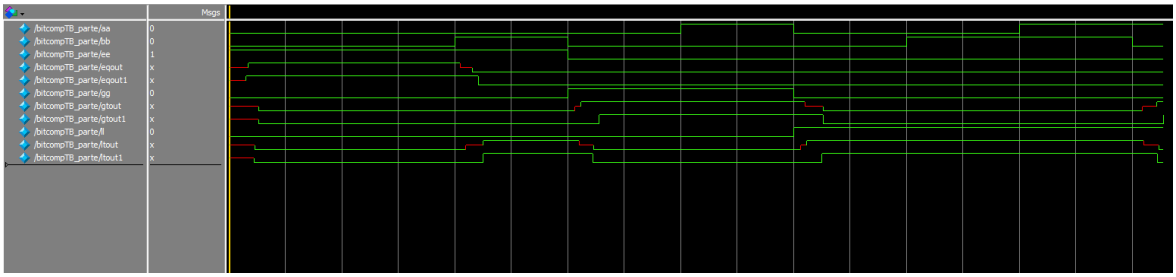
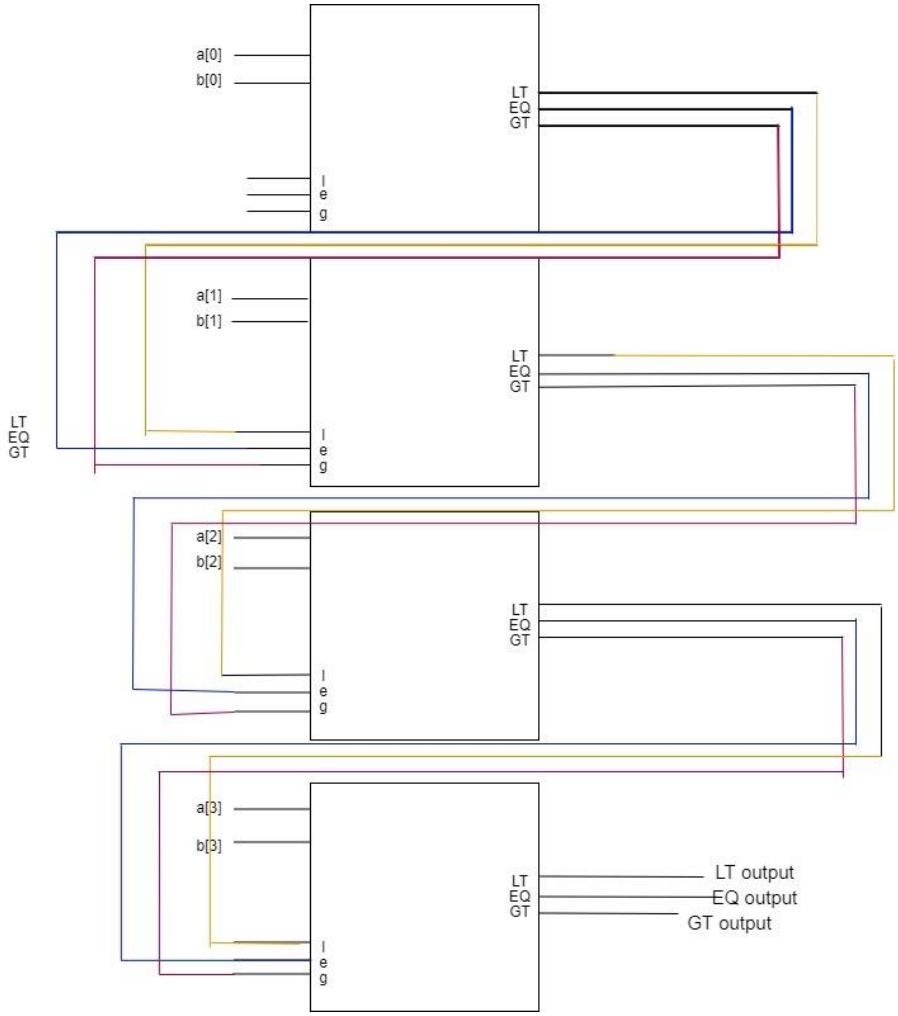


Figure 7 waveform of 1-bit comparators

Because we used assign statement, we no longer have X value in the output. We designed first comparator using gates that are based on transistors so we have delay values based on transistor delays and it's not always worst case so delay values differ.

Q2

a) we use outputs of the components as inputs for other components. RTL1 compares the least significant bit and RTL4 gives us the output of the whole comparing.




```
`timescale 1ns/1ns
module quad_comparator(input [3:0] a,b,input e,l,g,output GT,LT,EQ);
        wire [2:0] eqout,ltout,gtout;
        bit_comparator_assign RTL1(.a(a[0]),.b(b[0]),.l(l),.e(e),.g(g),.LT(ltout[0]),.GT(gtout[0]),.EQ(eqout[0]));
        bit_comparator_assign RTL2(.a(a[1]),.b(b[1]),.l(ltout[0]),.e(eqout[0]),.g(gtout[0]),.LT(ltout[1]),.GT(gtout[1]),.EQ(eqout[1]));
        bit_comparator_assign RTL3(.a(a[2]),.b(b[2]),.l(ltout[1]),.e(eqout[1]),.g(gtout[1]),.LT(ltout[2]),.GT(gtout[2]),.EQ(eqout[2]));
        bit_comparator_assign RTL4(.a(a[3]),.b(b[3]),.l(ltout[2]),.e(eqout[2]),.g(gtout[2]),.LT(LT),.GT(GT),.EQ(EQ));
endmodule
```

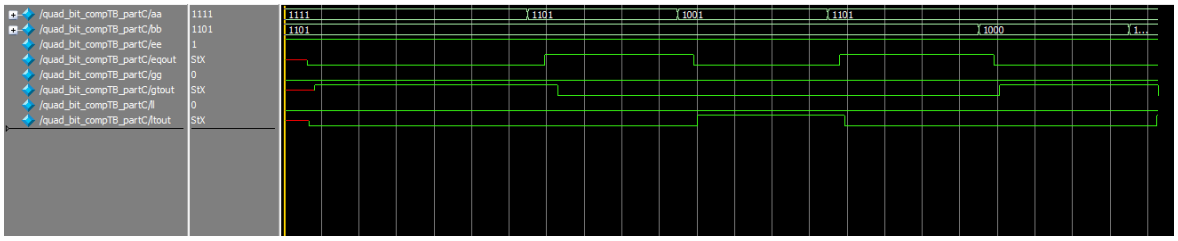
b) because of cascading, all of our delays are multiplied by 4.

Output	1 to 0	0 to 1
LT	180ns	200ns
GT	212ns	220ns
EQ	168 ns	120 ns

c) Testbench code is shown below:

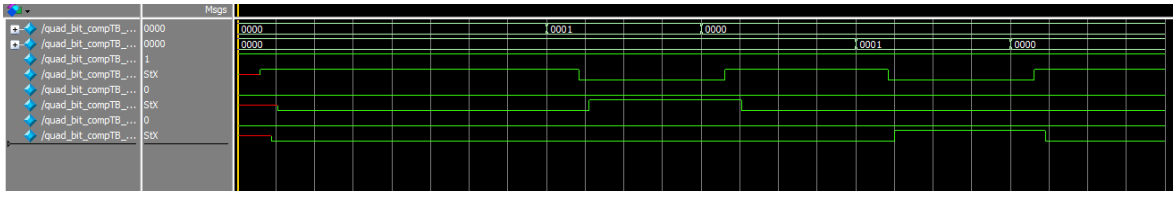
```
1   `timescale 1ns/1ns
2   module quad_bit_compTB_partC();
3           logic [3:0] aa=4'b1111,bb=4'b1101;
4           logic ll=0,ee=1,gg=0;
5           wire ltout,eqout,gtout;
6           quad_comparator CUT1(aa,bb,ee,ll,gg,gtout,ltout,eqout);
7           initial begin
8           #500;
9           #800 aa=4'b1101;
10          #800 aa=4'b1001;
11          #800 aa=4'b1101;
12          #800 bb=4'b1000;
13          #800 bb=4'b1111;
14          end
15  endmodule
16
```

Waveform of this testbench is:



Another testbench to show worst case delays:

```
1   `timescale 1ns/1ns
2   module quad_bit_compTB_worstdelay();
3           logic [3:0] aa=4'b0000,bb=4'b0000;
4           logic ll=0,ee=1,gg=0;
5           wire ltout,eqout,gtout;
6           quad_comparator CUT1(aa,bb,ee,ll,gg,gtout,ltout,eqout);
7           initial begin
8           #800;
9           #800 aa=4'b0001;
10          #800 aa=4'b0000;
11          #800 bb=4'b0001;
12          #800 bb=4'b0000;
13          #800;
14          end
15  endmodule
```



d) Screenshot below shows how to write module with assign.

```
1   `timescale 1ns/1ns
2   module quad_comparator_assign(input [3:0] a,b,input e,l,g,output GT,LT,EQ);
3           assign #(200,180) LT=(a<b)|((a==b)&l);
4           assign #(220,212) GT=(a>b)|((a==b)&g);
5           assign #(120,168) EQ=(a==b)&(e==1'b1);
6   endmodule
7
```

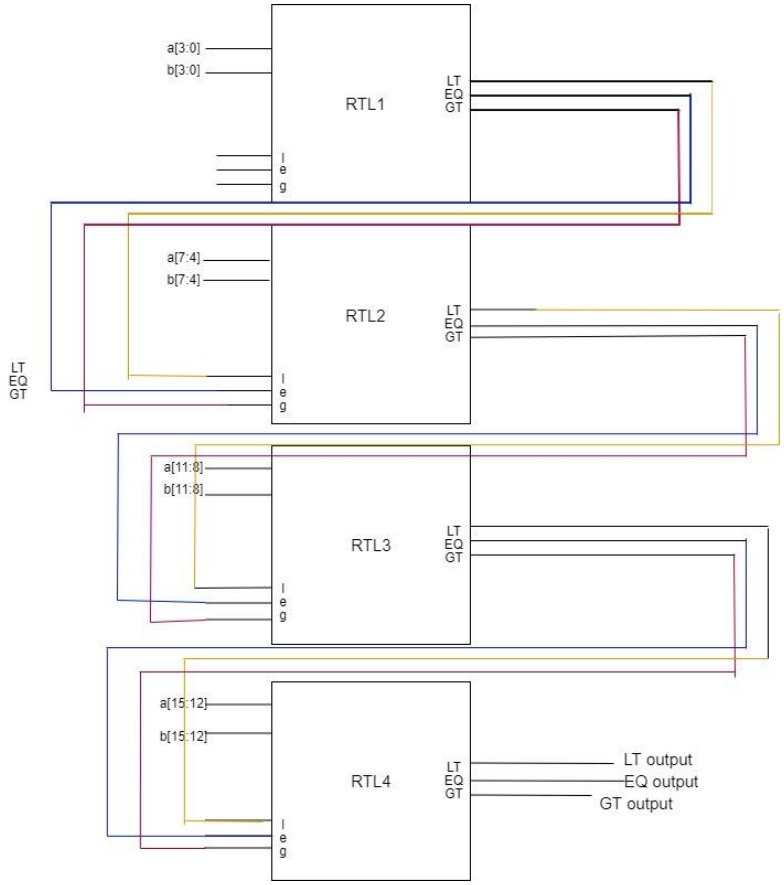
e)

```
1   `timescale 1ns/1ns
2   module quad_bit_compTB_parte();
3           logic [3:0] aa=4'b1111,bb=4'b1101;
4           logic ll=0,ee=1,gg=0;
5           wire ltout,eqout,gtout,ltoutl,gtoutl,eqoutl;
6           quad_comparator CUT1(aa,bb,ee,ll,gg,gtout,ltout,eqout);
7           quad_comparator_assign CUT2(aa,bb,ee,ll,gg,gtoutl,ltoutl,eqoutl);
8           initial begin
9           #500;
10          #800 aa=4'b1101;
11          #800 aa=4'b1001;
12          #800 aa=4'b1101;
13          #800 bb=4'b1000;
14          #800 bb=4'b1111;
15          end
16  endmodule
17
18
```

The second quad comparator with assign has more delay values but as we can see. This happens because we defined assign with worst case delay but the transition doesn't necessarily occur in least significant beat. because the basis of both comparators is assign, we don't have X values in transitions.

Q3

a)Same as Q2 part a, we use 4 comparators and cascade it.



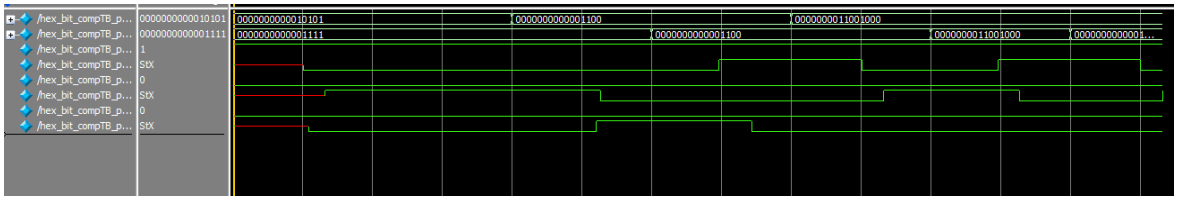

```
1   `timescale 1ns/1ns
2   module hex_comparator(input [15:0] a,b,input e,l,g,output GT,LT,EQ);
3           wire [2:0] eqout,ltout,gtout;
4           quad_comparator_assign RTL1(.a(a[3:0]),.b(b[3:0]),.l(l),.e(e),.g(g),.LT(ltout[0]),.GT(gtout[0]),.EQ(eqout[0]));
5           quad_comparator_assign RTL2(.a(a[7:4]),.b(b[7:4]),.l(ltout[0]),.e(eqout[0]),.g(gtout[0]),.LT(ltout[1]),.GT(gtout[1]),.EQ(eqout[1]));
6           quad_comparator_assign RTL3(.a(a[11:8]),.b(b[11:8]),.l(ltout[1]),.e(eqout[1]),.g(gtout[1]),.LT(ltout[2]),.GT(gtout[2]),.EQ(eqout[2]));
7           quad_comparator_assign RTL4(.a(a[15:12]),.b(b[15:12]),.l(ltout[2]),.e(eqout[2]),.g(gtout[2]),.LT(LT),.GT(GT),.EQ(EQ));
8   endmodule
9
10
```

b)Because we have 4 quad comparators connected to each other (series connection) delay values are quad comparator delay values multiplied by 4.

Output	1 to 0	0 to 1
LT	720ns	800ns
GT	848ns	880ns
EQ	672 ns	480 ns

c)

```
`timescale 1ns/1ns
module hex_bit_compTB_partC();
        logic [15:0] aa=16'd21,bb=16'd15;
        logic ll=0,ee=1,gg=0;
        wire ltout,eqout,gtout;
        hex_comparator CUT1(aa,bb,ee,ll,gg,gtout,ltout,eqout);
        initial begin
        #1000;
        #1000 aa=16'd12;
        #1000 bb=16'd12;
        #1000 aa=16'd200;
        #1000 bb=16'd200;
        #1000 bb=4'd300;
        end
endmodule
```

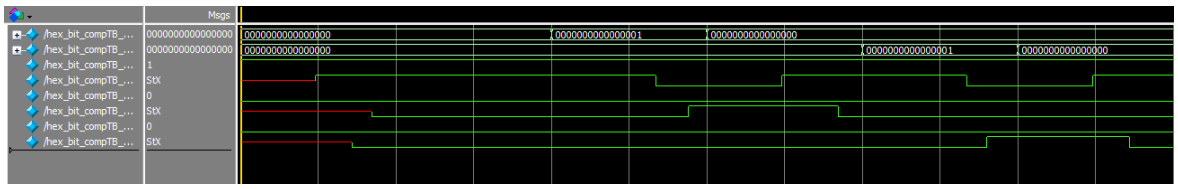


As we can see from the waveforms above, the comparator works correctly.

Another testbench to show worst case delays.

```
`timescale 1ns/1ns
module hex_bit_compTB_worstdelay();
        logic [15:0] aa=16'd0,bb=16'd0;
        logic ll=0,ee=1,gg=0;
        wire ltout,eqout,gtout;
        hex_comparator CUT1(aa,bb,ee,ll,gg,gtout,ltout,eqout);
        initial begin
        #1000;
        #1000 aa[0]=1'b1;
        #1000 aa[0]=1'b0;
        #1000 bb[0]=1'b1;
        #1000 bb[0]=1'b0;
        #1000 ;
        end
endmodule
```

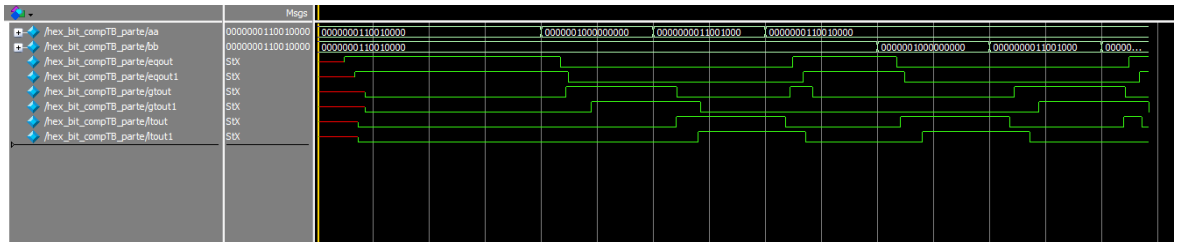
The waveform result is shown below:



d)Depending on the bit that changes the outcome, we will have many different delay values so we use worst case delay to define the assign statement.

```
`timescale 1ns/1ns
module hex_comparator_assign(input [15:0] a,b,input e,l,g,output GT,LT,EQ);
        assign #(800,720) LT=(a<b)|((a==b)&l);
        assign #(880,848) GT=(a>b)|((a==b)&g);
        assign #(672,480) EQ=(a==b)&(e==1'b1);
endmodule
```

e)



Delays of the original hex_comparator is lower than the one we defined with assign statement.

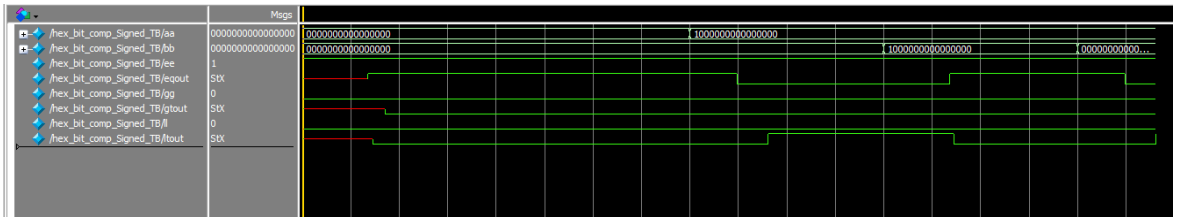
f)When the sign bits of both inputs are the same, we have no issue. However, when sign bits are different, we should complement sign bit and compare the inputs using that. We can't change value of one bit in an array in systemverilog so we create a dummy array for both a and b, we assign values of 15 bits the same and for the sign bit we use an inverter and complement the sign bit. This way, the output will be correct.

```
`timescale 1ns/1ns
module hex_comparator_signed(input [15:0] a,b,input e,l,g,output GT,LT,EQ);
        wire [15:0] aa,bb;
        assign aa[14:0]=a[14:0];
        assign bb[14:0]=b[14:0];
        assign #(7,5) aa[15]=~a[15];
        assign #(7,5) bb[15]=~b[15];
        hex_comparator_assign RTL1(aa,bb,e,l,g,GT,LT,EQ);
endmodule
```

Testbench for signed comparator:

```
`timescale 1ns/1ns
module hex_bit_comp_Signed_TB();
        logic [15:0] aa=16'b0000000000000000,bb=16'b0000000000000000;
        logic ll=0,ee=1,gg=0;
        wire ltout,eqout,gtout;
        hex_comparator_signed CUT1(aa,bb,ee,ll,gg,gtout,ltout,eqout);
        initial begin
        #2000;
        #2000 aa[15]=1'b1;
        #2000 bb[15]=1'b1;
        #2000 bb[15]=1'b0;
        end
endmodule
```

Waveform from testbench that indicates the comparator works correctly.


Msgs
/hex_bit_comp_Signed_TB/aa
/hex_bit_comp_Signed_TB/bb
/hex_bit_comp_Signed_TB/ee
/hex_bit_comp_Signed_TB/eqout
/hex_bit_comp_Signed_TB/gg
/hex_bit_comp_Signed_TB/gtout
/hex_bit_comp_Signed_TB/ll
/hex_bit_comp_Signed_TB/ltout
0000000000000000
0000000000000000
1
StX
0
StX
0
StX
0000000000000000
1000000000000000
1000000000000000
0000000000...