# UNIVERSITY OF TEHRAN

College of Engineering

Electrical and Computer Engineering Department

Digital Systems I, ECE 894

Computer Assignment 1


Shahaboddin Sheybani

810101454


Spring 1402-03

# Q1:

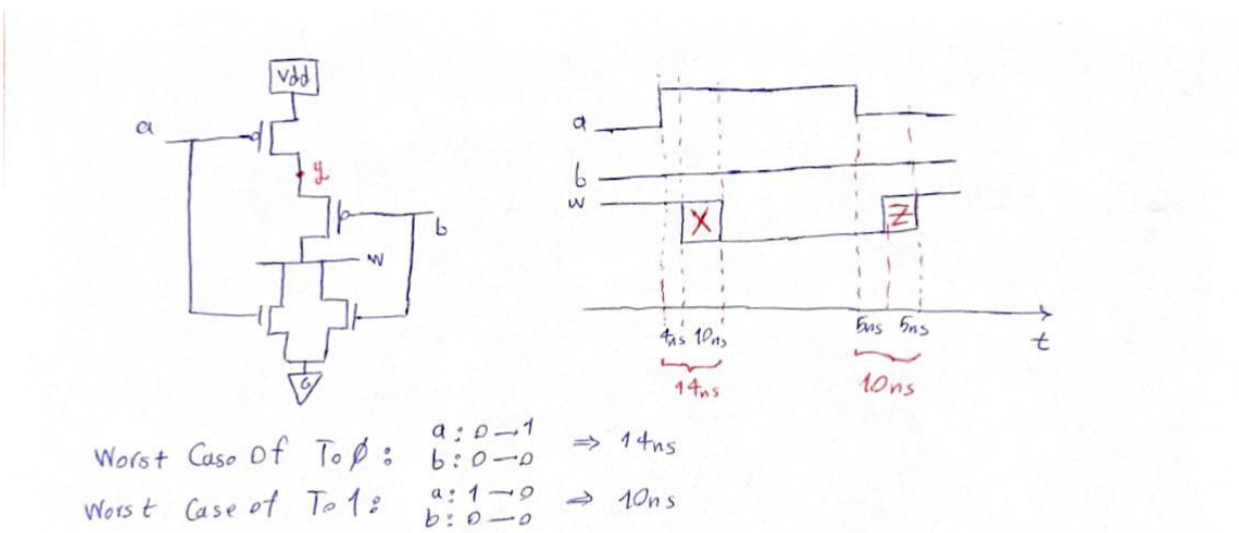
We are going to design and simulate  NOR gate using hand-simulation to determine the worst-case delay. This process allows us to observe how transistors adapt to each other's behavior.

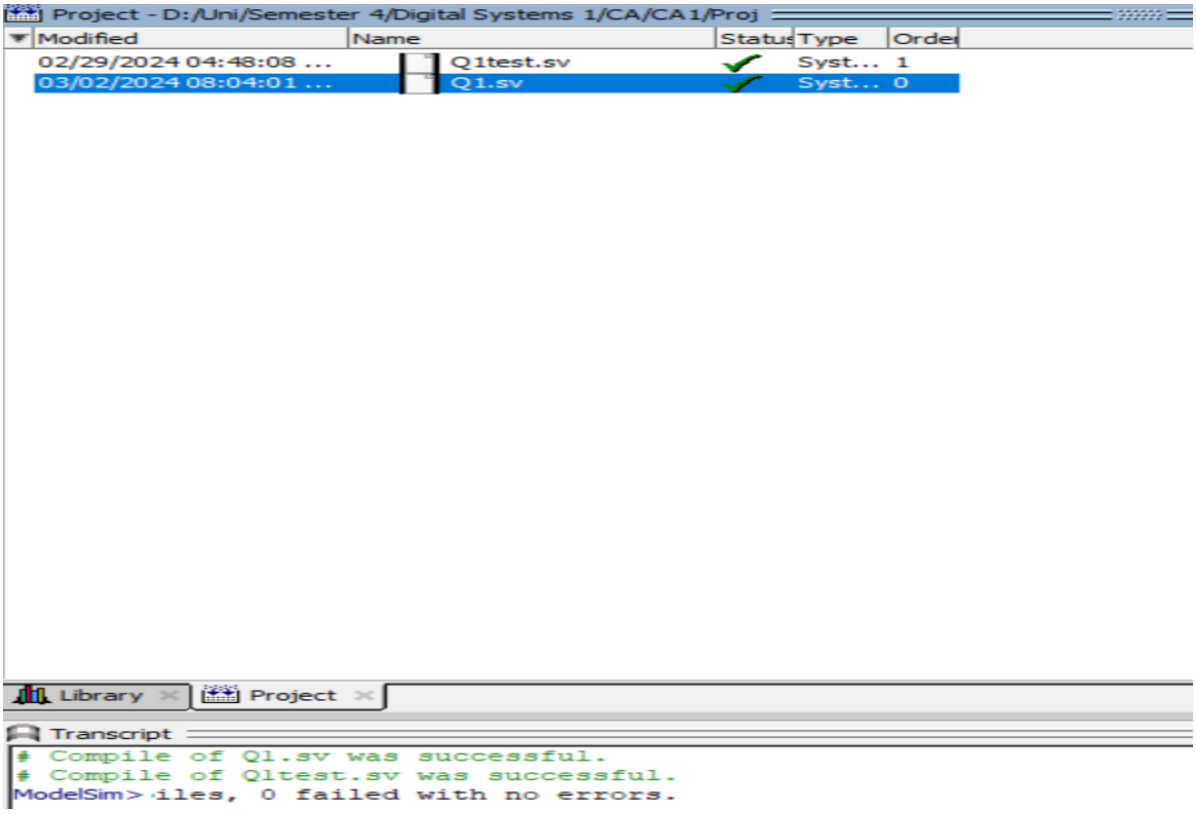
I have created a testbench that incorporates various input transitions, as shown in both the code and the waveform. The resulting waveform closely resembles the one obtained from hand-simulation.

For more explanation you should know delays how achieved.

To1 : both of pull-up transistor have to carry 1 and we have (5+5 = 10ns) delay

To0 ; both of pull-up transistors have to insulate and delay will be ( 7 + 7 = 14ns) which they are toZ delays of pmos transistor.





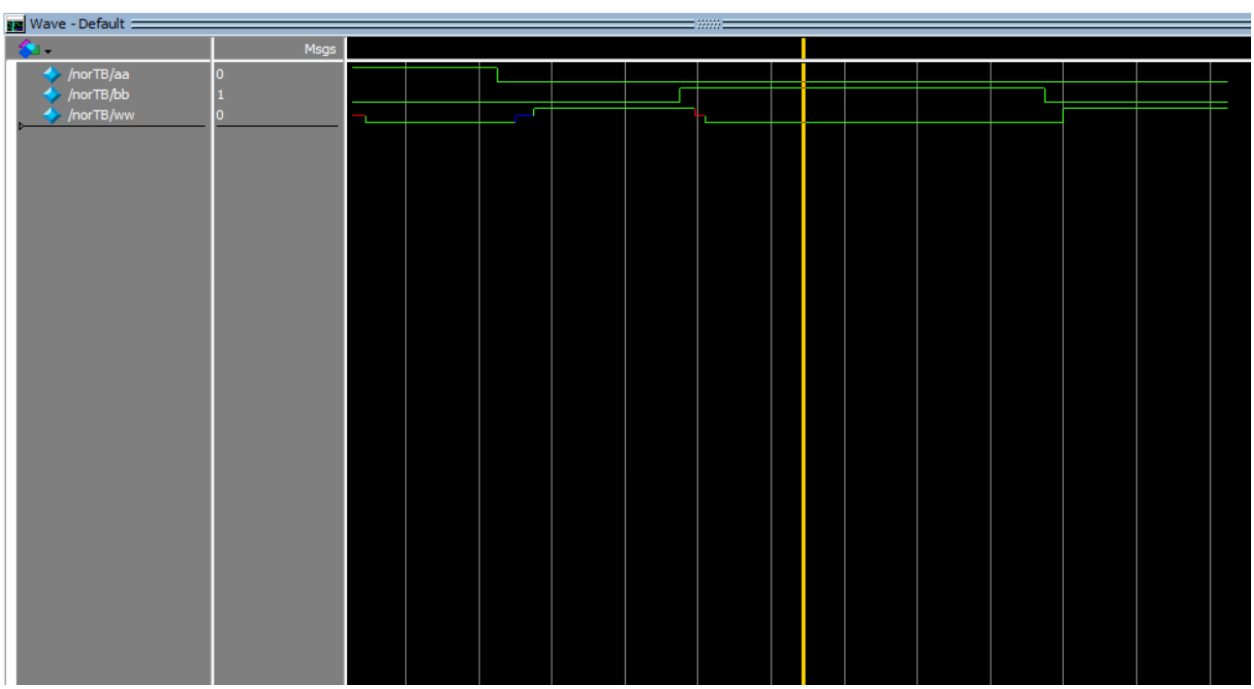

```
`timescale 1ns/1ns
module mynor(input a ,b , output w);
        supply1 vdd;
        supply0 Gnd;
        wire y;
        nmos #(3,4,5) G1(w,Gnd,a);
        nmos #(3,4,5) G2(w,Gnd,b);
        pmos #(5,6,7) G3(y,vdd,a);
        pmos #(5,6,7) G4(w,y,b);

endmodule
```

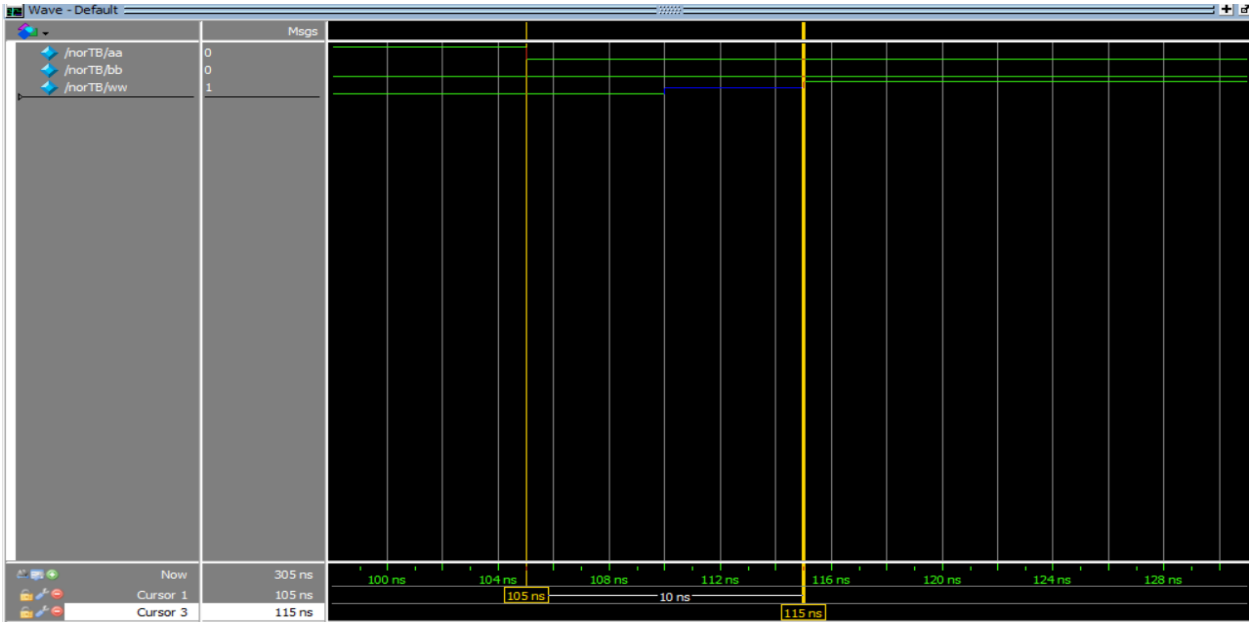
```
`timescale 1ns/1ns
module norTB();
        logic aa;
        logic bb;
        logic ww;
        mynor CUT1(aa,bb,ww);
        initial begin
        #5 aa = 0; bb = 0;
        #50 aa = 1;
        #50 aa = 0;
        #50 bb = 1;
        #50 aa = 0;
        #50 bb = 0;
        #50 $stop;
        end

endmodule
```

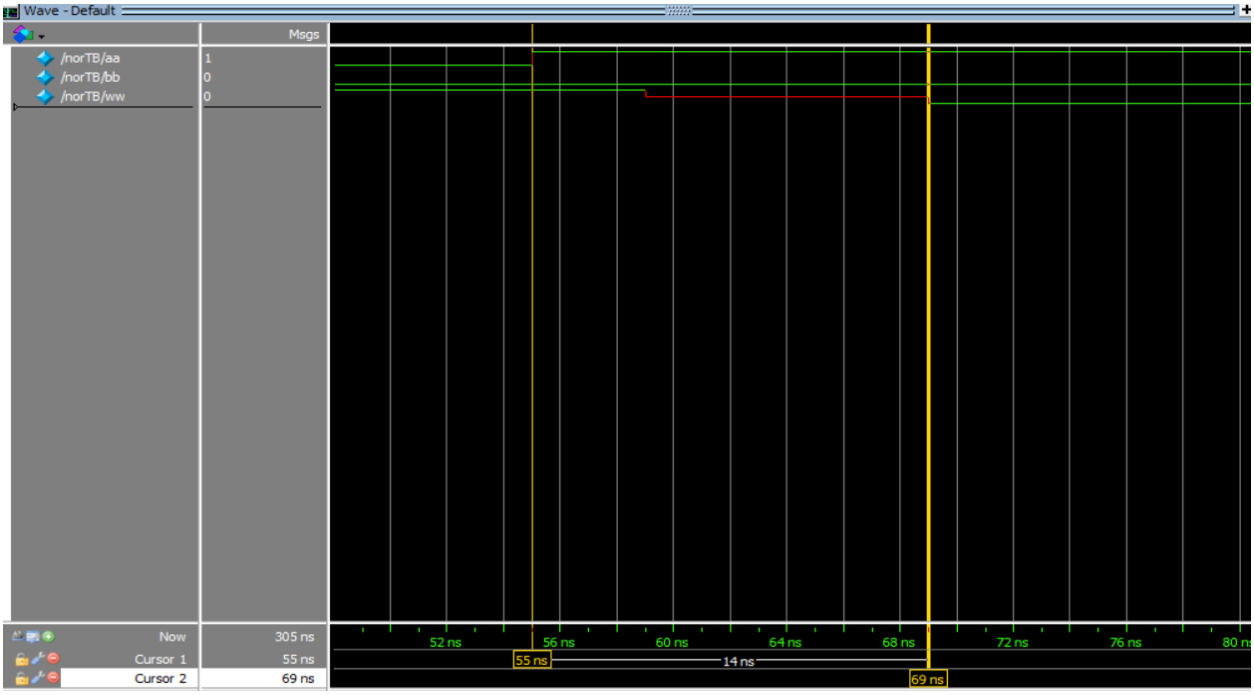
General picture of waveform:



Worst case of To1 delay:




Worst case of To0 delay:

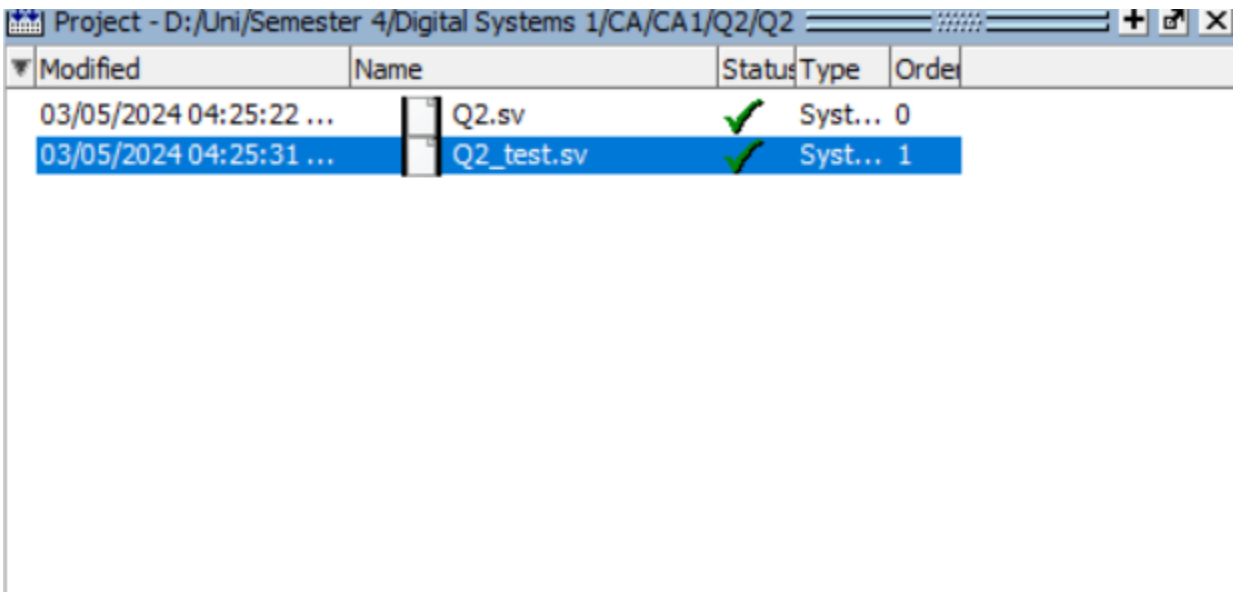


# Q2:

This section describes the simulation of a multiplexer using two NMOS transistors. The provided logic design showcases the circuit, where the power supply originates from the initial inverter.

I performed hand-simulation to determine the worst-case delays for to1 and to0, as well as created additional testbenches. The results, including waveforms, demonstrate the successful identification of both valid output states.

A point which I have to add is that in multiplexer we wont have z on th ouptput cause its impossible that both of the nmos (passing) transistor conduct so we only have worst case of to1 & to0.

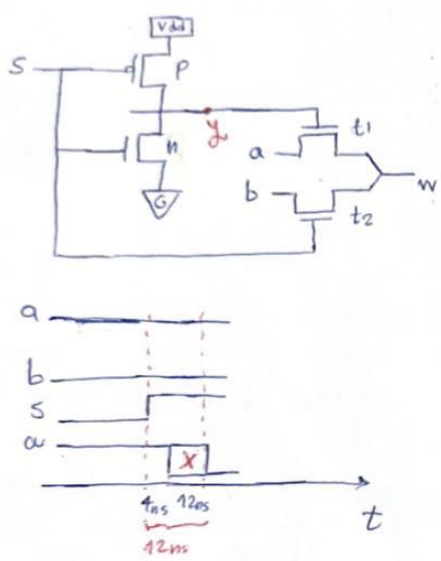


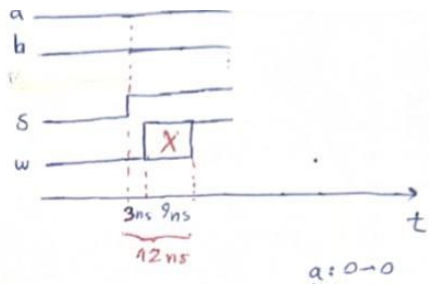
D:/Uni/Semester 4/Digital Systems 1/CA/CA1/Q2/Q2.sv (/passMUXtest/cut2) - Default

```
`timescale 1ns/1ns
module passMUX(input a,b,s , output w);
  wire y;
  supply1 vdd;
  supply0 gnd;
  nmos #(3,4,5) n(y,gnd,s);
  pmos #(5,6,7) p(y,vdd,s);
  nmos #(3,4,5) t1(w,a,y);
  nmos #(3,4,5) t2(w,b,s);
endmodule
```

D:/Uni/Semester 4/Digital Systems 1/CA/CA1/Q2/Q2_test.sv (/passMUXtest) - Default

```
`timescale 1ns/1ns
module  passMUXtest();
logic aa;
logic bb;
logic ss;
wire ww;
passMUX cut2(aa,bb,ss,ww);
initial begin
  aa = 0;
  bb = 0;
  ss = 0;
  #50
  #50 aa = 0;
  #50 bb = 1;
  #50 ss = 1;
  #50 ss = 0;
  #50 aa = 1;
  #50 bb = 0;
  #50 ss = 1;
  #50 $stop;
end
endmodule
```

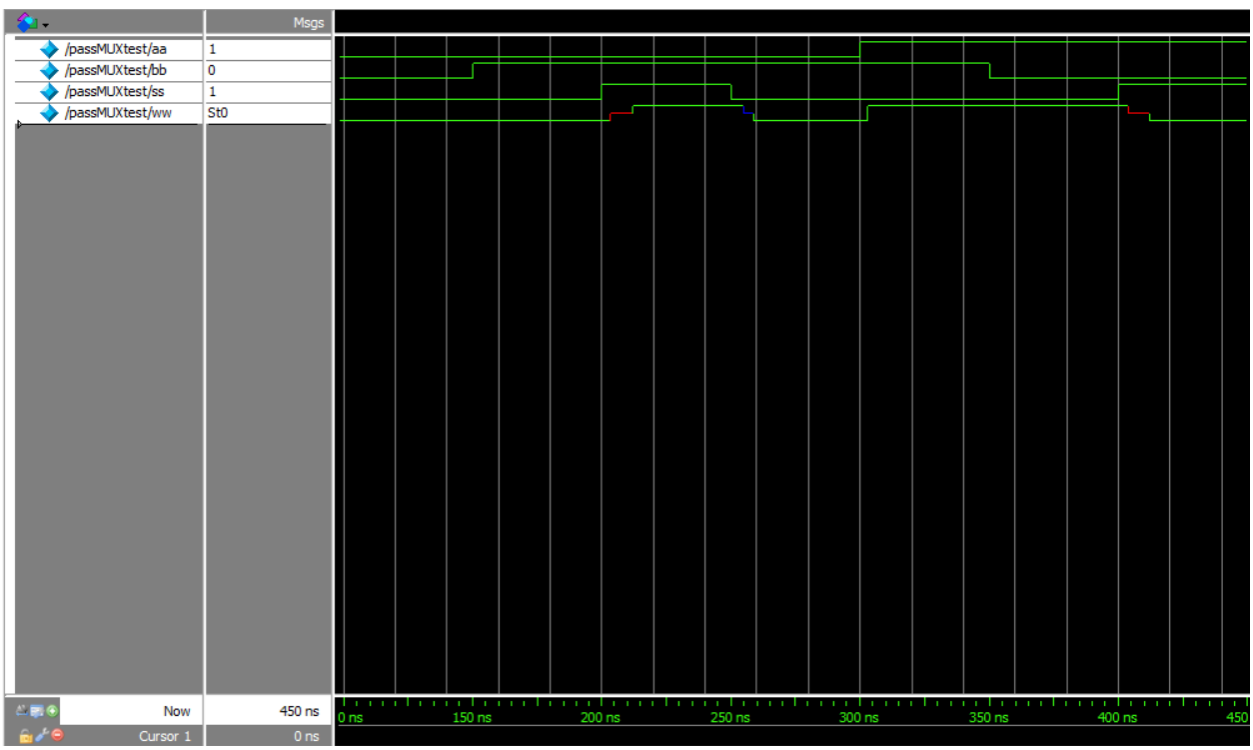


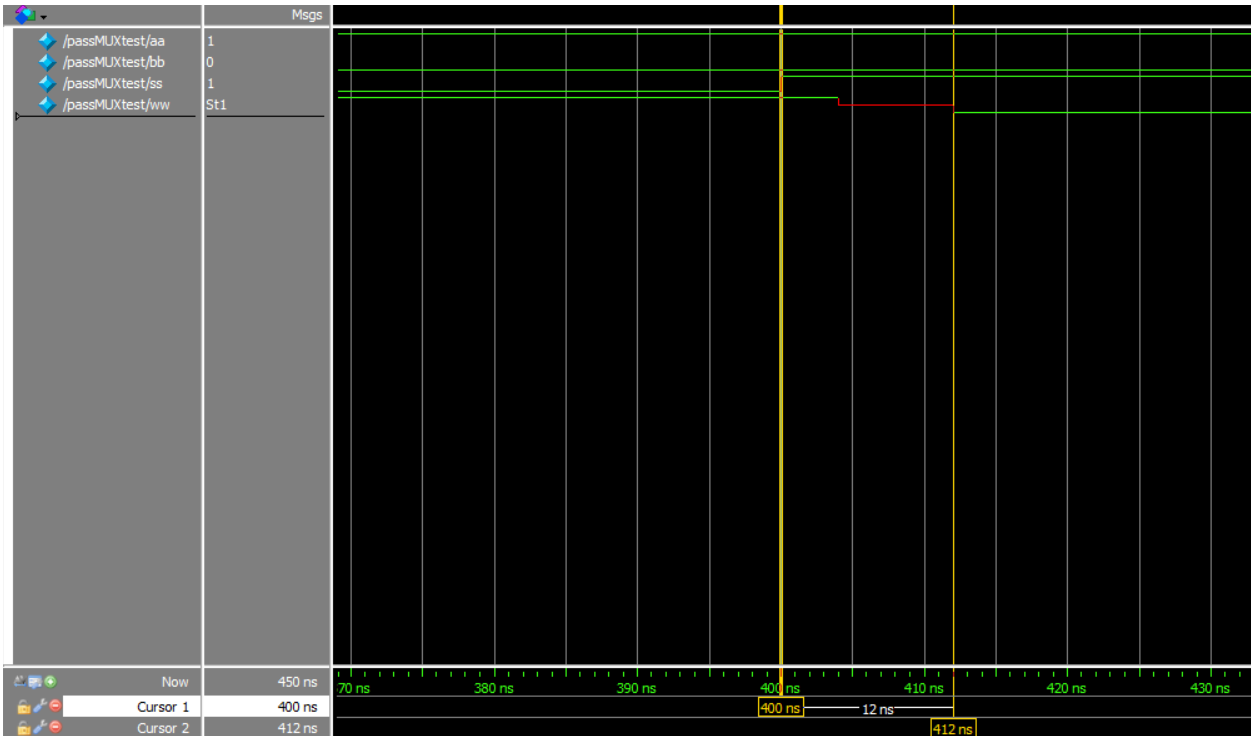

Worst Case of To 1: a: 0→0, b: 0→0, S: 0→1 ⇒ 12ns

Worst Case of To ϕ: a: 1→1, b: 0→0, S: 0→1 ⇒ 12ns

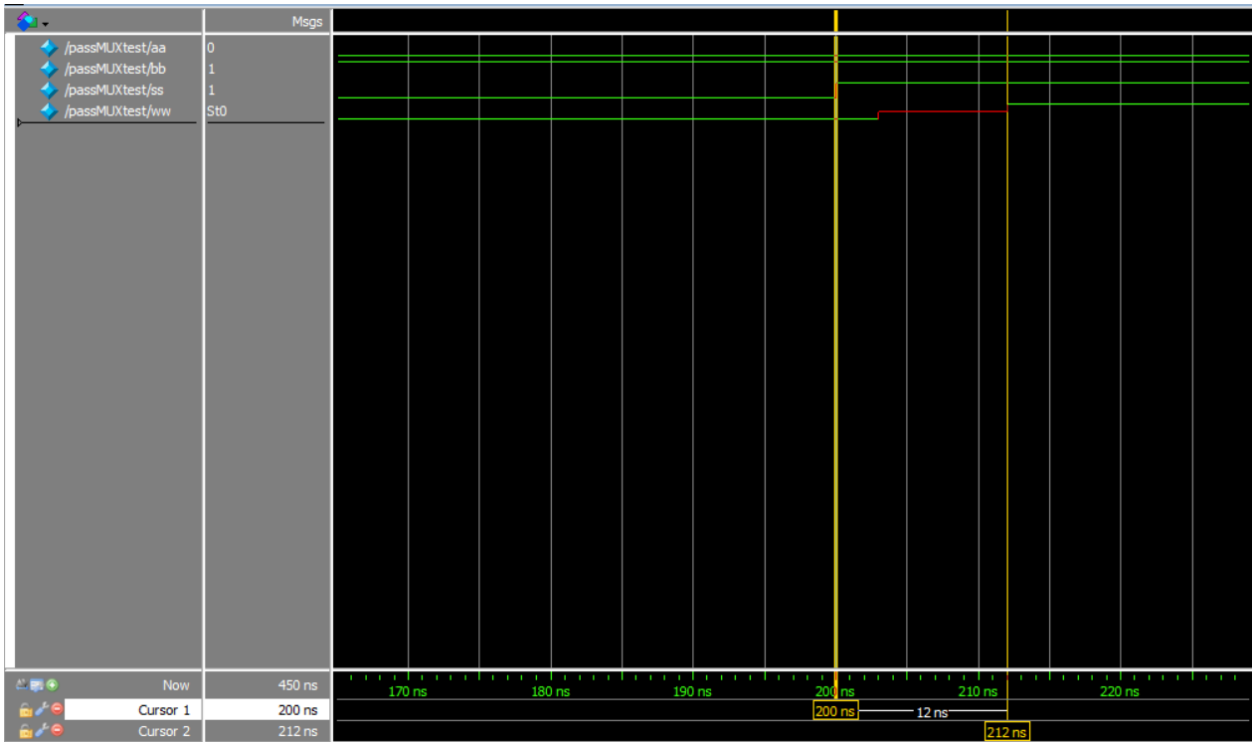
General picture of waveform :



Worst case of To0 :




Worst case of to1:



# Q3:

Designing a 4-to-1 Multiplexer using NOR Gates

Gate Delays:

It's important to consider the propagation delays associated with different logic gates. While 2-input NOR gates might have delays of 14 and 10 nanoseconds (ns) and 3-input NOR gates with their additional transistors(totally 6 transistors) exhibit increased delays, typically 1.5x those of 2-input gates. This translates to delays of approximately 21ns(3*7(toZ of pmos)) and 15ns(3*5(toZ of nmos)). Similarly, 4-input NOR gates can have delays around 28ns(4*7) and 20ns(4*5)

Inverting with NOR Gates:

To create a NOT gate using NOR gates, connect both inputs of a NOR gate together. Since a NOR gate outputs a logic 1 only when both inputs are logic 0, connecting both inputs essentially forces the output to be the logical complement of the single input.

Equivalent Circuit with NOR Gates:

By applying Boolean algebra, we can design an equivalent circuit for the desired function using only NOR gates. This circuit is typically presented in a diagram for better understanding.

Identifying Waveforms and Worst-Case Delays:

By analyzing the designed circuit and considering the individual gate delays, we can identify the critical path and determine the worst-case delay of the entire 4-to-1 multiplexer. This information is crucial for understanding the timing behavior of the circuit in real-world applications.

Following these steps allows you to design and analyze a 4-to-1 multiplexer using only NOR gates, taking into account the propagation delays of different gate types and utilizing the logic behavior of NOR gates to achieve desired functionalities.

Project - D:/Uni/Semester 4/Digital Systems 1/CA/CA1/Q3/Q3

| Modified | Name | Status | Type | Order |
|---|---|---|---|---|
| 03/05/2024 05:27:30 ... | Q3_test.sv | ✓ | Syst... | 1 |
| 03/05/2024 05:27:16 ... | Q3.sv | ✓ | Syst... | 0 |

D:/Uni/Semester 4/Digital Systems 1/CA/CA1/Q3/Q3.sv (/mux4_nor_test/cut3) - Default

```
`timescale 1ns/1ns
module mux4_nor(input a,b,c,d,s1,s0,output w);

  nor #(10,14) (a_bar,a,a),
               (b_bar,b,b),
               (c_bar,c,c),
               (d_bar,d,d),
               (s0_bar,s0,s0),
               (s1_bar,s1,s1);

  nor #(15,21) (a1,a_bar,s0,s1),
               (c1,c_bar,s0_bar,s1),
               (b1,b_bar,s0,s1_bar),
               (d1,d_bar,s0_bar,s1_bar);

  nor #(20,28) (w_bar,a1,b1,c1,d1);
  nor #(10,14) (w,w_bar,w_bar);
endmodule
```

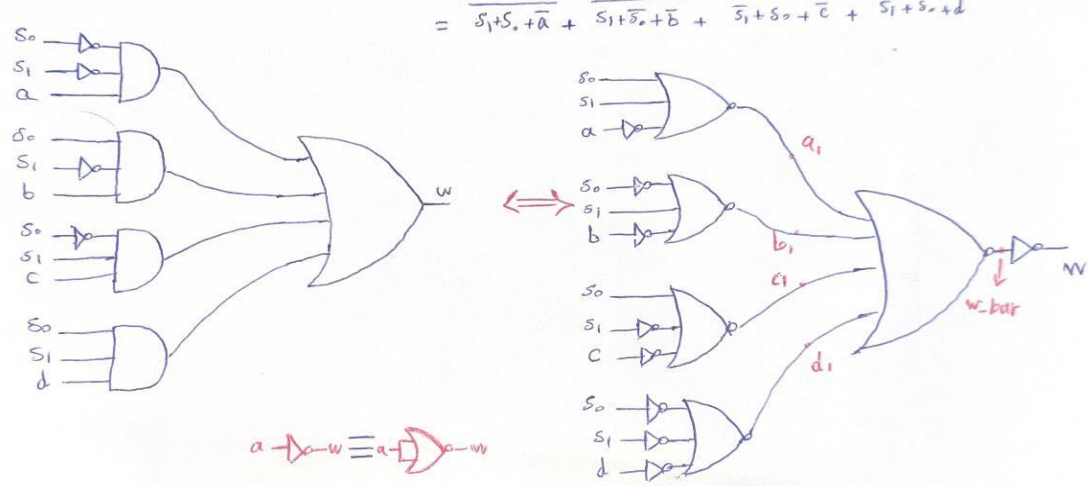
D:/Uni/Semester 4/Digital Systems 1/CA/CA1/Q3/Q3_test.sv (/mux4_nor_

```
`timescale 1ns/1ns
module mux4_nor_test();
  logic aa,bb,cc,dd,s0s,s1s,ww;
  mux4_nor cut3(aa,bb,cc,dd,s0s,s1s,ww);
  initial begin
    aa = 0;
    bb = 0;
    cc = 0;
    dd = 0;
    s0s = 0;
    s1s = 0;
    #50
    #100 aa = 1;
    #100 aa = 0;
    #100 s0s = 1;
    #100 bb = 1;
    #100 bb = 0;
    #100 s0s = 0;
    #100 s1s = 1;
    #100 cc = 1;
    #100 cc = 0;
   #100 s0s = 1;
    #100 dd = 1;
    #100 dd = 0;
    #100 $stop;
    end
endmodule
```

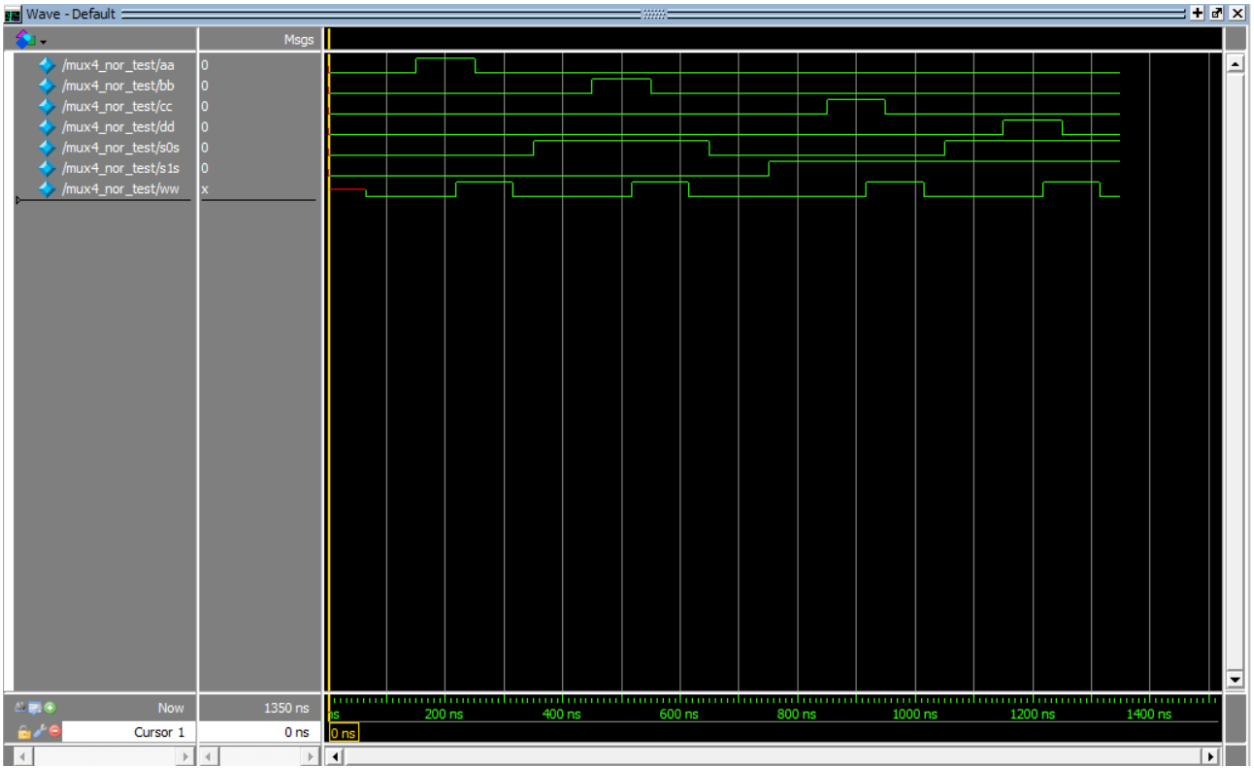
(3)

$$W = \bar{S_1}\bar{S_0}a + \bar{S_1}S_0 b + S_1 S_0 c + S_1 S_0 d$$

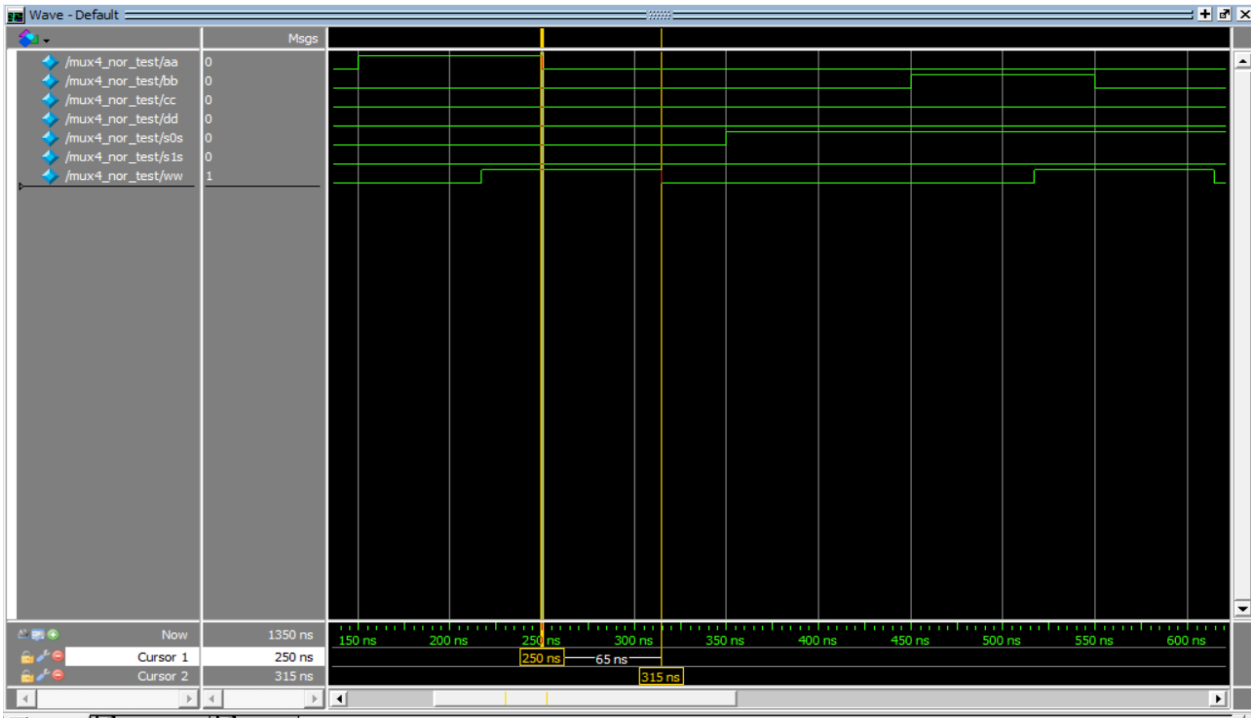
$$= \overline{\overline{S_1+S_0+\bar{a}}} + \overline{\overline{S_1+\bar{S_0}+\bar{b}}} + \overline{\overline{\bar{S_1}+S_0+\bar{c}}} + \overline{\overline{\bar{S_1}+\bar{S_0}+\bar{d}}}$$



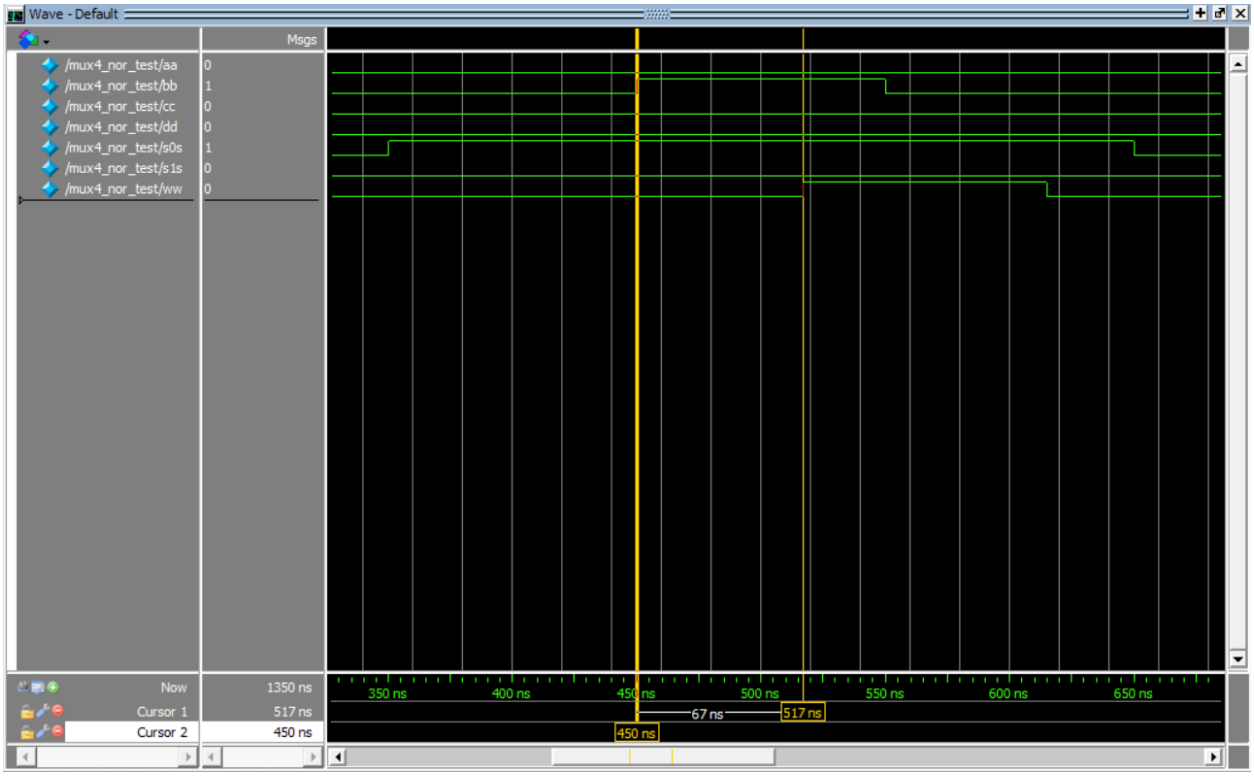
# General picture of waveform:



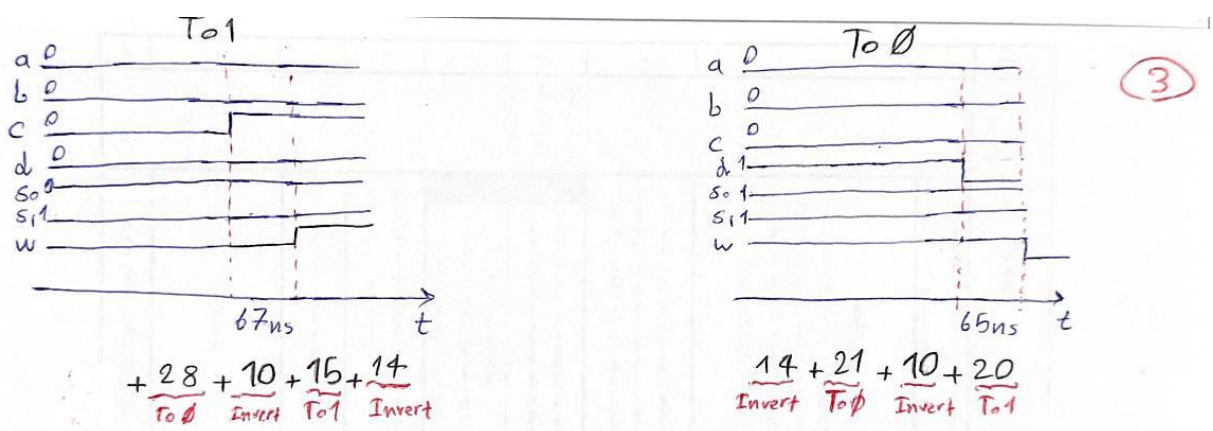

# Worst case delay of to0:



# Worst case of to1 delay:




# Hand simulation :



# Q4:

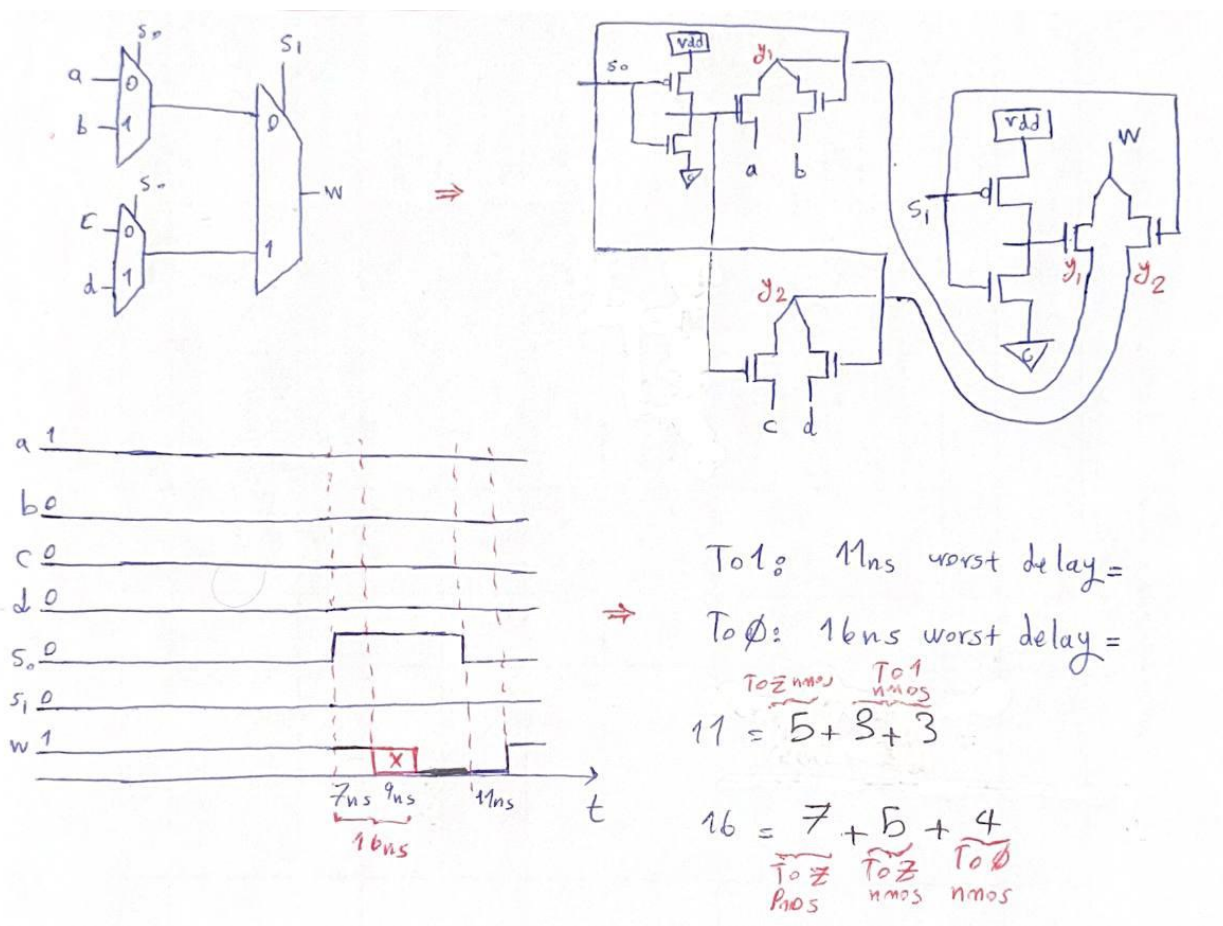
In this section, we will design a 4-to-1 multiplexer using 2-to-1 multiplexers. The logic design is shown below.

In a testbench, I aim to demonstrate the worst-case delays for inputs to0 and to1, and verify that they match the results obtained through manual simulation.

It is important to note that the delays are transistor-based, and I have implemented them from the first stage (Q1).

Also I should say that in this question I call passmux module which has used in Q2.

For calculating worst case we should consider that changing s0 or s1 can make more delay cause our inverter will be worked.

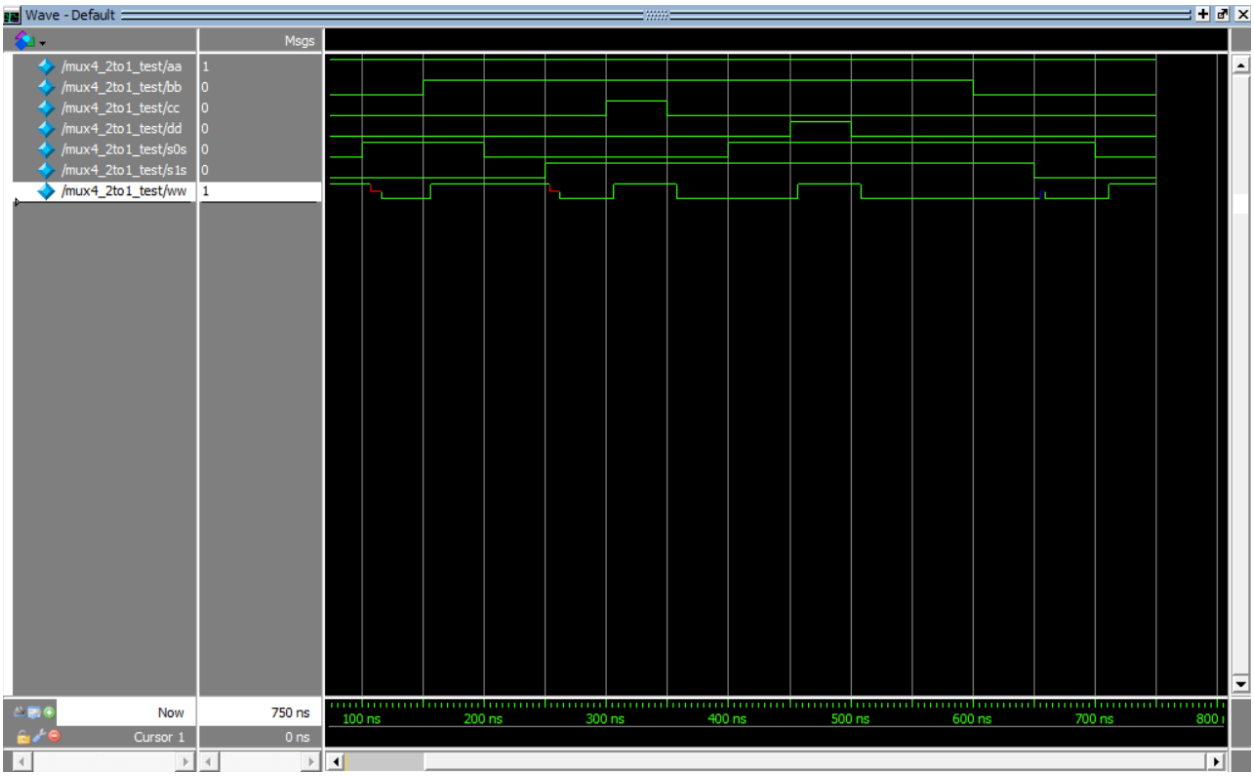


D:/Uni/Semester 4/Digital Systems 1/CA/CA1/Q4/Q4.sv (/mux4_2to1_test/cut4) - Default

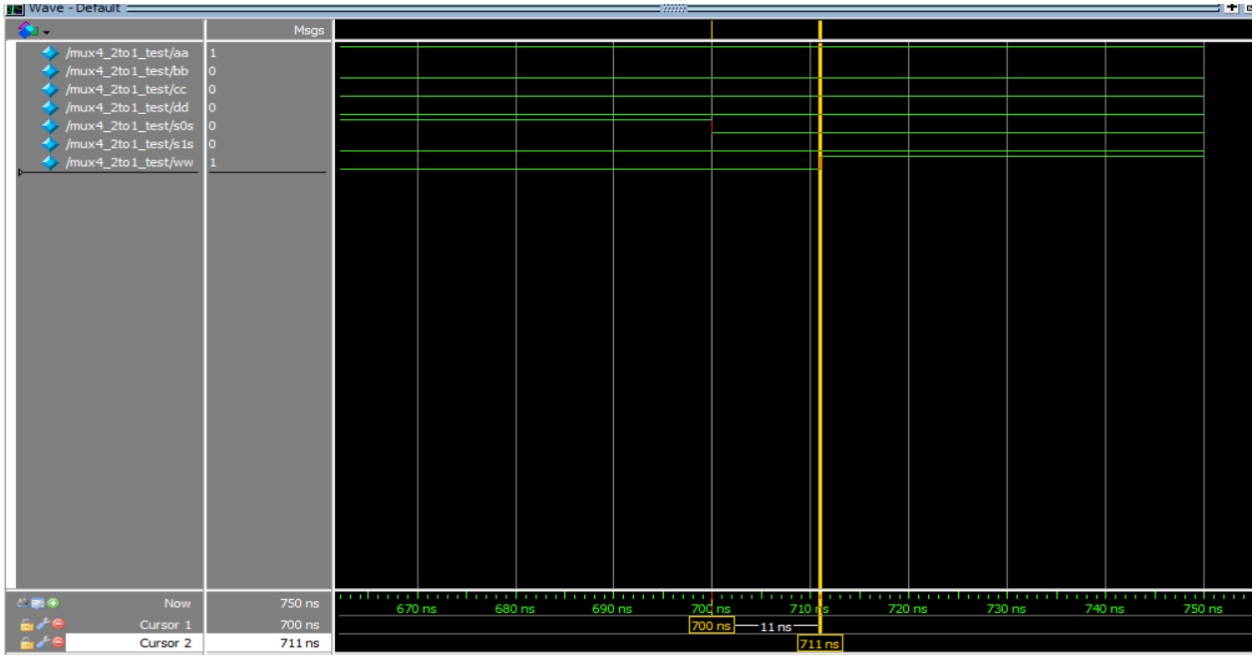
```
`timescale 1ns/1ns
module mux4_2to1 (input a,b,c,d,s0,s1,output w);
  passMUX m1(a,b,s0,y1);
  passMUX m2(c,d,s0,y2);
  passMUX m3(y1,y2,s1,w);
endmodule
```

```
`timescale 1ns/1ns
module mux4_2to1_test();
  logic aa,bb,cc,dd,s0s,s1s,ww;
  mux4_2to1 cut4(aa,bb,cc,dd,s0s,s1s,ww);
  initial begin
        aa = 1;
        bb = 0;
        cc = 0;
        dd = 0;
        s0s = 0;
        s1s = 0;
        #50
        #50 s0s = 1;
        #50 bb = 1;
        #50 s0s = 0;
        #50 s1s = 1;
        #50 cc = 1;
        #50 cc = 0;
        #50 s0s = 1;
        #50 dd = 1;
        #50 dd = 0;
        #50 cc = 0;
        #50 bb = 0;
        #50 s1s = 0;
        #50 s0s = 0;
        #50 $stop;
  end
endmodule
```

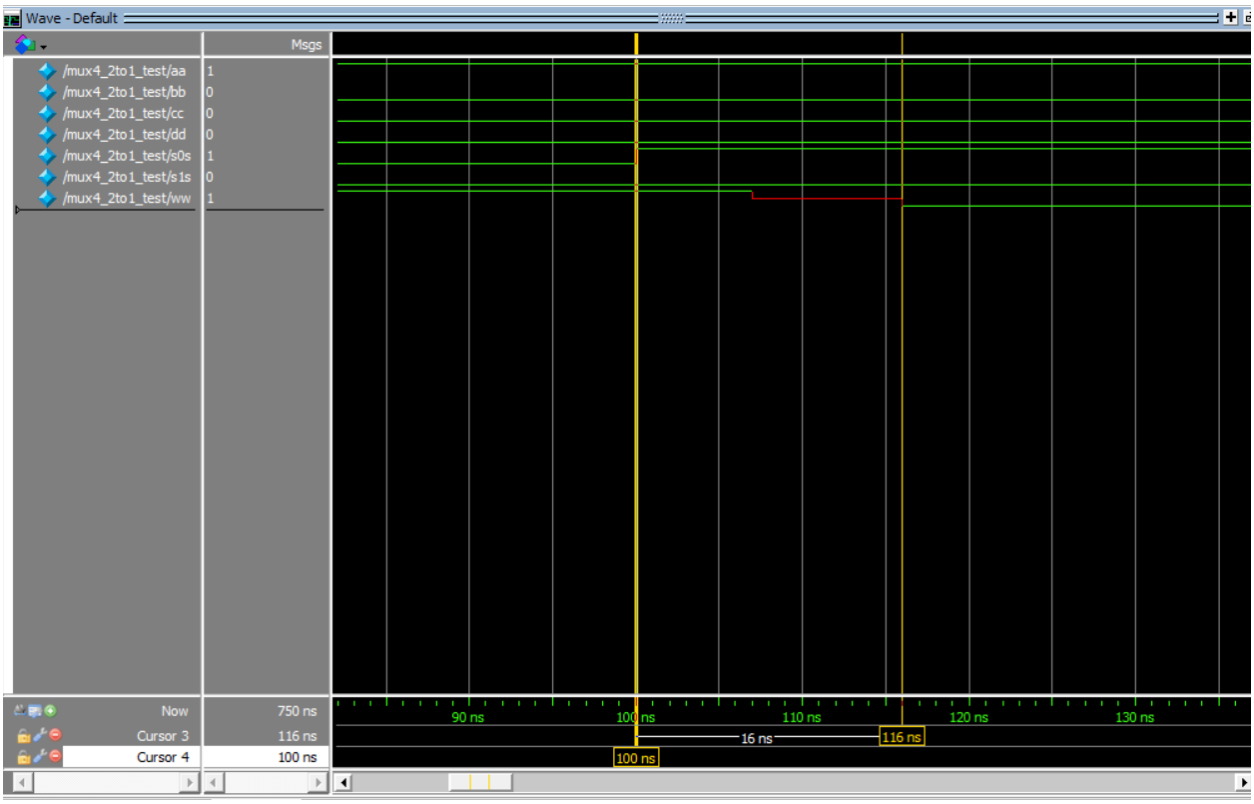
General picture of waveform :

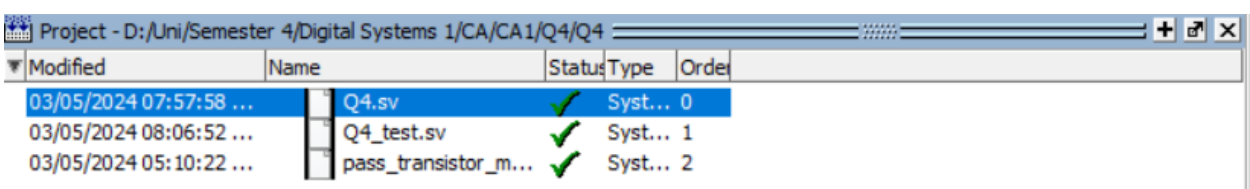



Worst case of to1 delay:



Worst case of to0 delay:




# Q5:

In this section, we will compare the modules of q4 and q5 in terms of transistor count, delay, and power consumption.

Firstly, when utilizing NOR gates to construct a 4-to-1 multiplexer, we require a minimum of 12 NOR gates, each containing 4 transistors. Thus, the total transistor count would be 12 * 4 = 48 transistors.

However, we discovered an alternative method using only 10 transistors, thereby saving 38 transistors by implementing passing transistors.

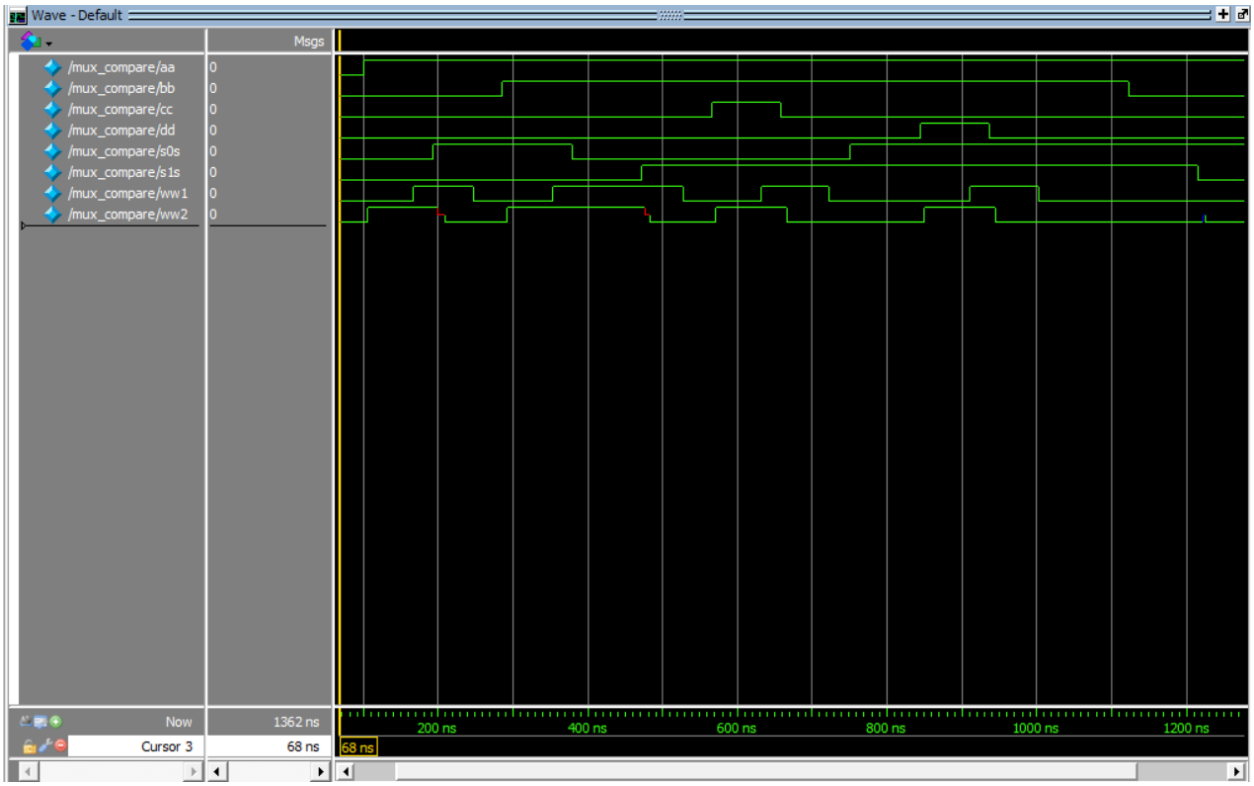
Additionally, upon analyzing the waveform results, it becomes evident that employing more transistors results in increased delay. For instance, in comparing the transition delay of the passing transistor multiplexer (around 6ns), it significantly outperforms the NOR gate version (67ns).

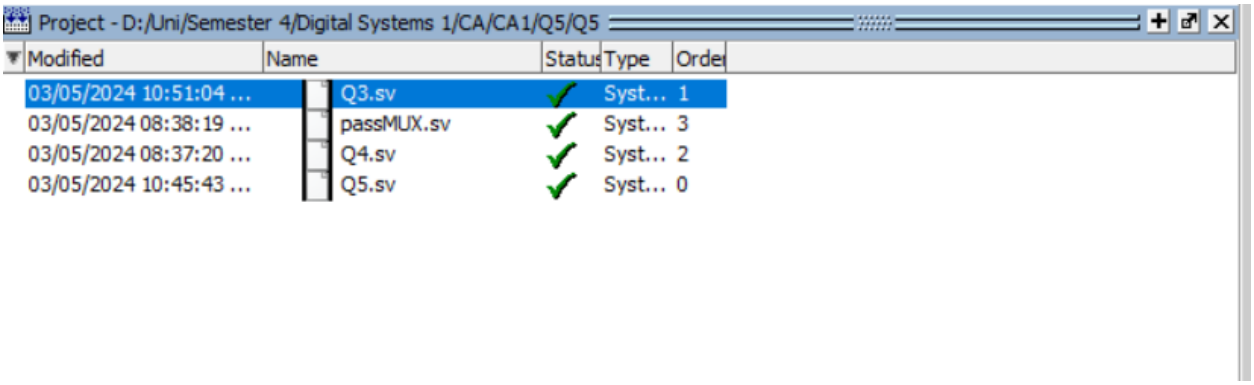
Nonetheless, it is essential to acknowledge that there's no free lunch in this scenario. While fewer transistors lead to reduced delay, there is a trade-off in power consumption.

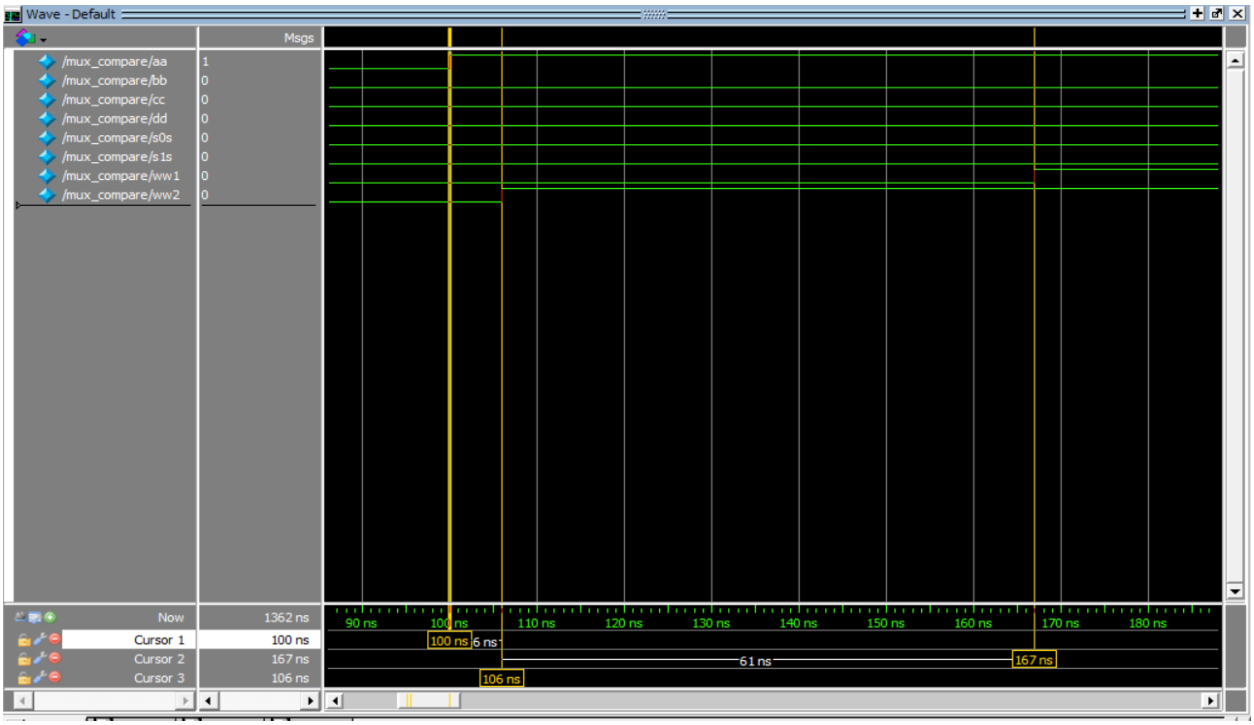
In the case of passing transistors, they draw power solely from the inverters of the previous gate, resulting in weaker 1s or 0s at the output. Conversely, NOR gates have their dedicated power supplies, thereby consuming less power despite the increased delay.

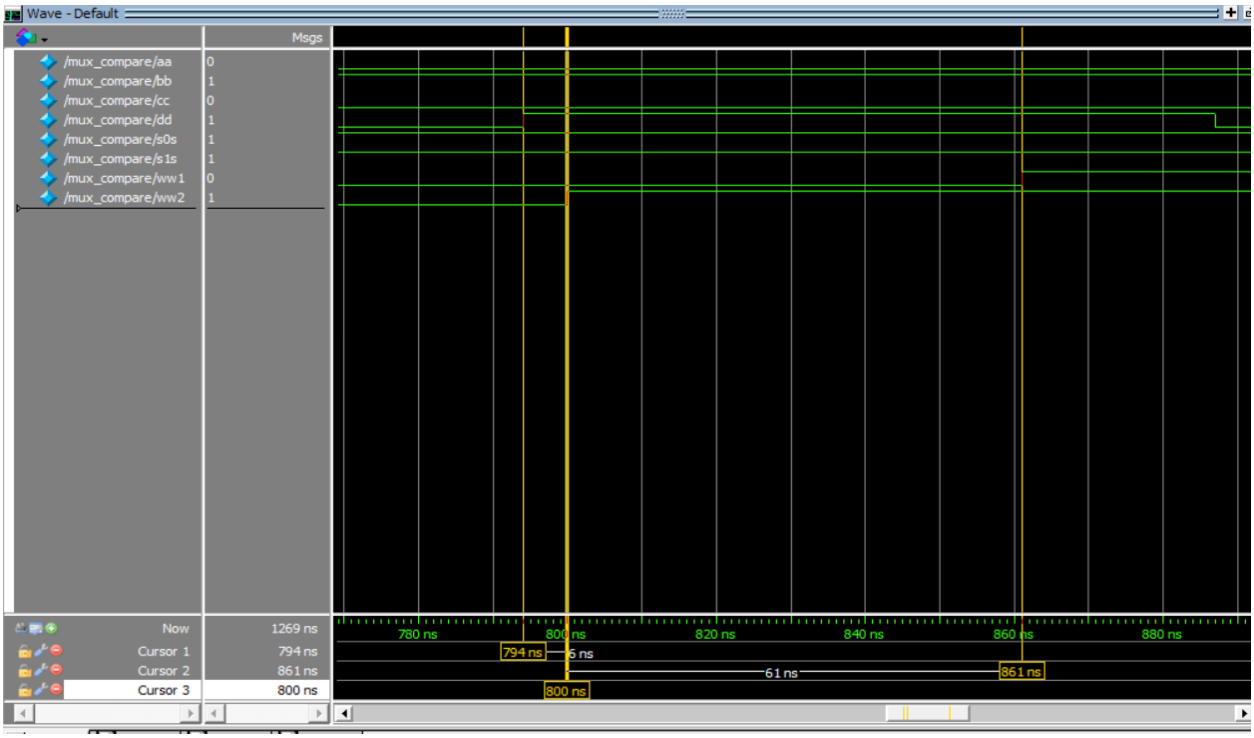
Its necessary that you know ww2 shows output using passing transistors and ww1 represent output of multiplexer using NOR gates.

General picture of waveform:




Wave - Default
Msgs
/mux_compare/aa 1
/mux_compare/bb 0
/mux_compare/cc 0
/mux_compare/dd 0
/mux_compare/s0s 0
/mux_compare/s1s 0
/mux_compare/ww1 0
/mux_compare/ww2 0
Now 1362 ns
Cursor 1 100 ns
Cursor 2 167 ns
Cursor 3 106 ns
90 ns
110 ns
120 ns
130 ns
140 ns
150 ns
160 ns
170 ns
180 ns
100 ns
6 ns
61 ns
167 ns
106 ns


Wave - Default
Msgs
/mux_compare/aa 0
/mux_compare/bb 1
/mux_compare/cc 0
/mux_compare/dd 1
/mux_compare/s0s 1
/mux_compare/s1s 1
/mux_compare/ww1 0
/mux_compare/ww2 1
Now 1269 ns
Cursor 1 794 ns
Cursor 2 861 ns
Cursor 3 800 ns
780 ns
820 ns
840 ns
880 ns
794 ns
6 ns
61 ns
861 ns
800 ns

D:/Uni/Semester 4/Digital Systems 1/CA/CA1/Q5/Q5.sv (/mux_compare) - Default

```
`timescale 1ns/1ns
module mux_compare();
  logic aa,bb,cc,dd,s0s,s1s,ww1,ww2;
  mux4_nor cut5(aa,bb,cc,dd,s0s,s1s,ww1);
  mux4_2to1 cut6(aa,bb,cc,dd,s0s,s1s,ww2);
  initial begin
        aa = 0;
        bb = 0;
        cc = 0;
        dd = 0;
        s0s = 0;
        s1s = 0;
        #50
        #50 aa = 1;
        #93 s0s = 1;
        #93 bb = 1;
        #93 s0s = 0;
        #93 s1s = 1;
        #93 cc = 1;
        #93 cc = 0;
        #93 s0s = 1;
        #93 dd = 1;
        #93 dd = 0;
        #93 cc = 0;
        #93 bb = 0;
        #93 s1s = 0;
        #93 s0s = 0;
    end
endmodule
```