CA2-Report

Digital Systems I

Spring 1402-03
Digital Systems I – ECE894

Amirmortaza Rezaee
810101429

Problem 1

Let's have a look on a truth table of a simple 1-bit adder: (suppose: $w=A_0+B_0+C_0=C_1S_0$)

A_0	B_0	C_0	w	C_1	S_0
0	0	0	00	0	0
0	0	1	01	0	1
0	1	0	01	0	1
0	1	1	10	1	0
1	0	0	01	0	1
1	0	1	10	1	0
1	1	0	10	1	0
1	1	1	11	1	1

As we can see, we can replace S_0 with the logic statement below:

$$S_0 = A_0 \oplus B_0 \oplus C_0$$

And also, the Karnaugh map for C_1 will look like:

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

Writing minterms for this map, will cause in this logic statement:

$$C_1 = (A_0.B_0) + (A_0.C_0) + (B_0.C_0)$$

$$C_1=(A_0.B_0)+C_0.(A_0+B_0) = (A_0.B_0)+C_0.(A_0 \oplus B_0)$$

Actually there wasn't any problems for replacing OR with XOR because of using the AND term in the statement.

Figure 1.1 illustrates this 1-bit adder:

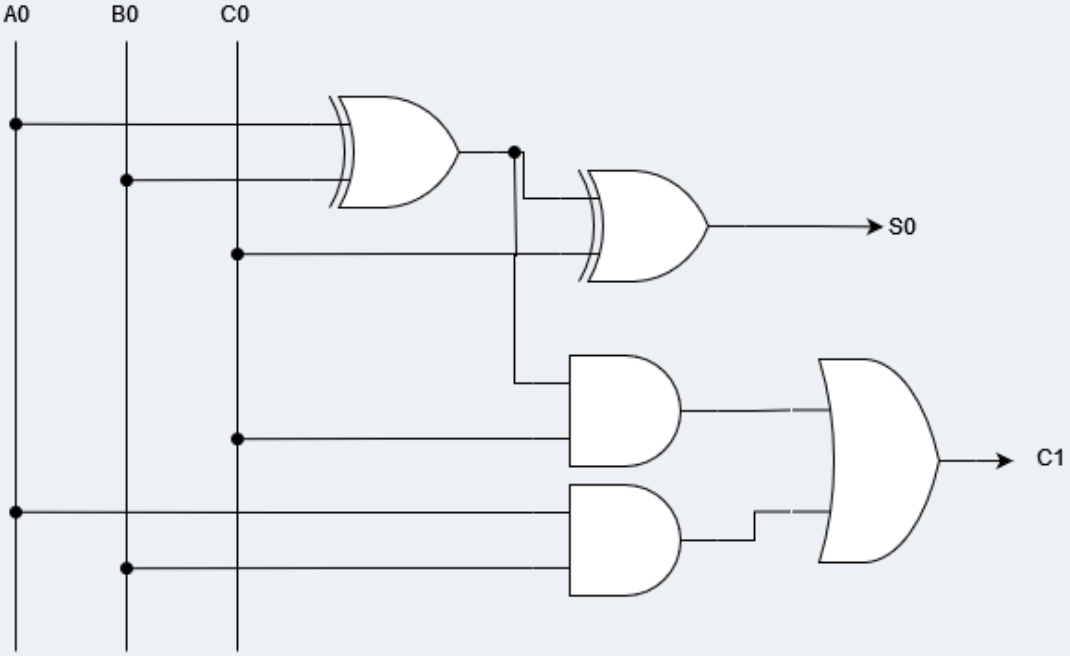

Figure 1.1

So, we can describe the problem's 2-bit adder as the figure below:

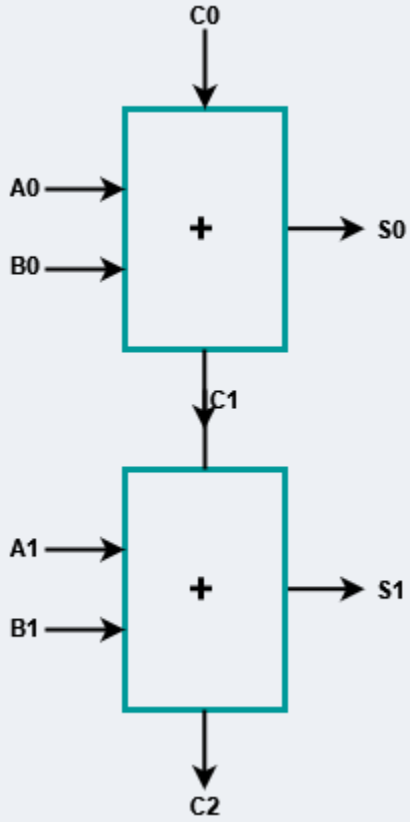

Figure 1.2

Also, we can create AND, XOR and OR gates using NOR gates:

1) $\overline{A} = \overline{A + A} \Rightarrow$

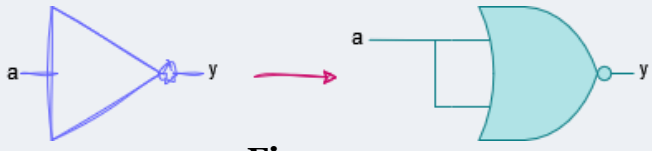

Figure 1.3

2) $A.B = \overline{\overline{A} + \overline{B}} \Rightarrow$

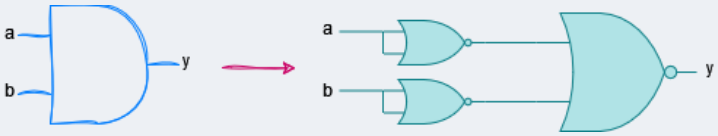

Figure 1.4

3) $A + B + C = \overline{\overline{A + B + C}} \Rightarrow$

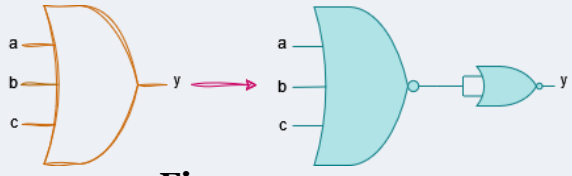

Figure 1.5

4) $A \oplus B = A.\overline{B} + \overline{A}.B = \overline{A.B + \overline{A}.\overline{B}} \Rightarrow \begin{cases} A.B = \overline{\overline{A} + \overline{B}} \\ \overline{A}.\overline{B} = \overline{A + B} \end{cases} \Rightarrow$

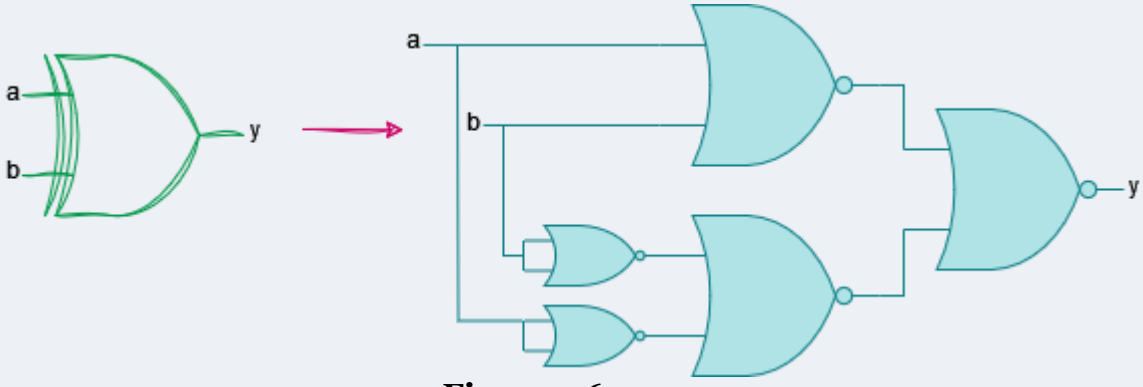

Figure 1.6

*So, **the first layer of** our adder looks like:*

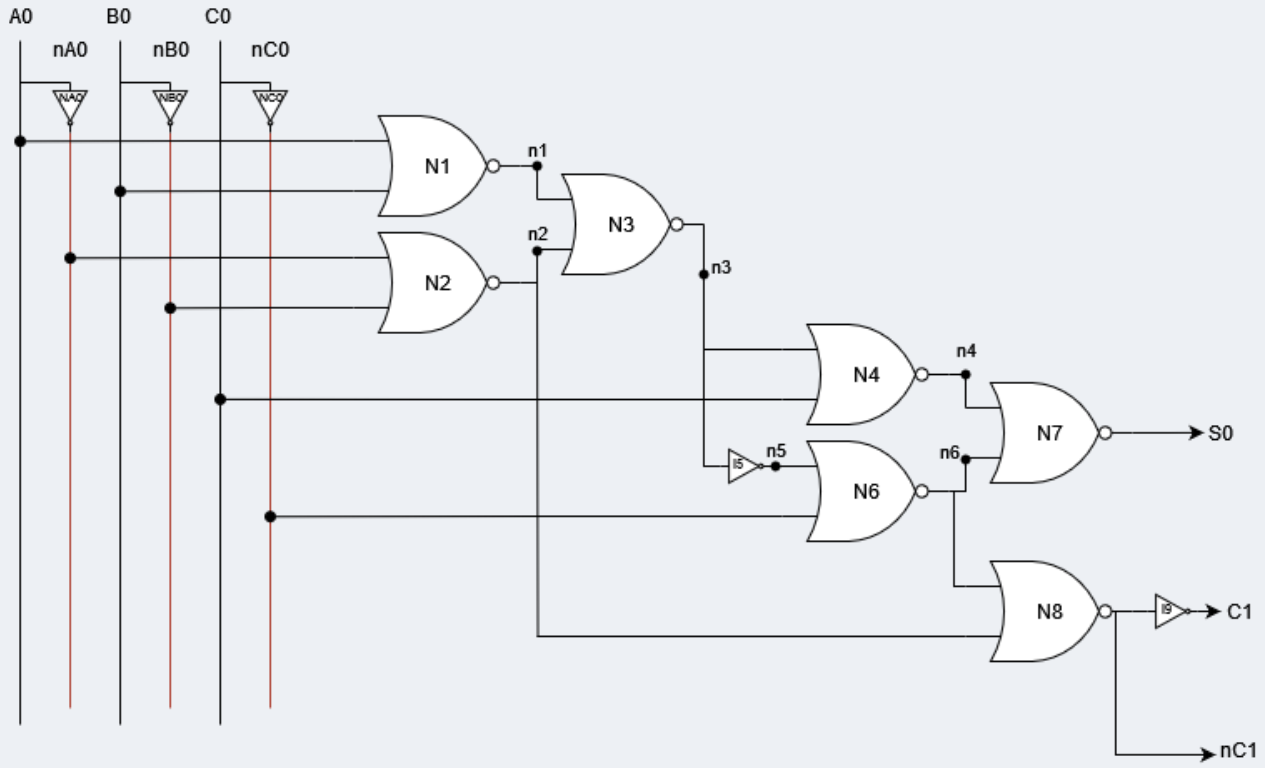

Figure 1.7

And also, the second layer looks like the diagram below:

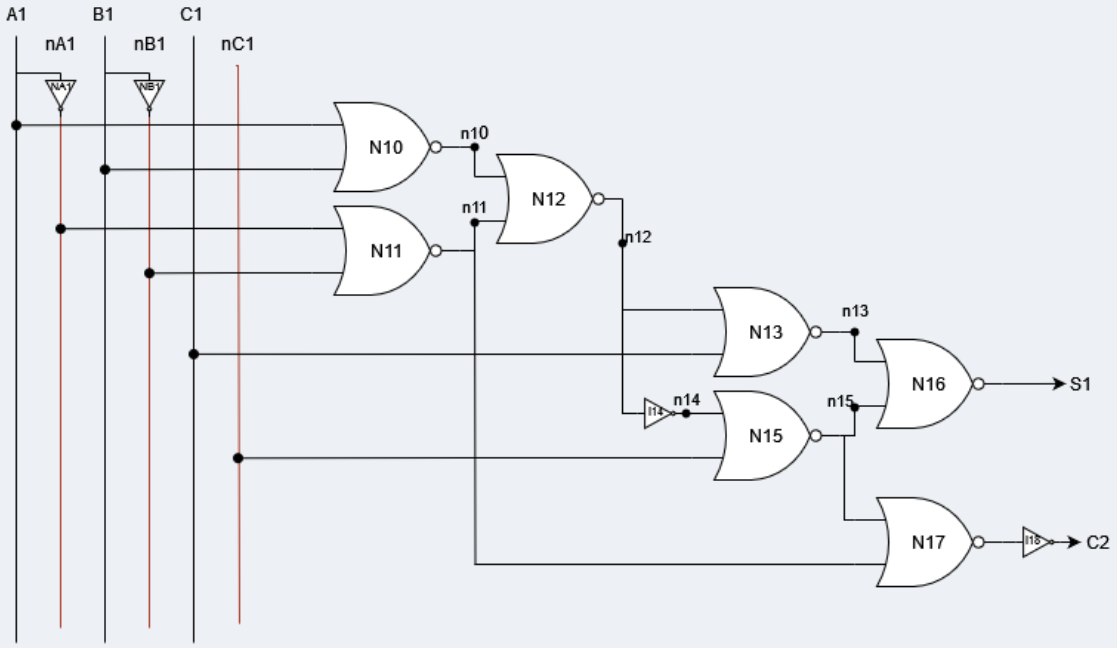

Figure 1.8

Now let's find the worst-case delays. Based on the diagrams, each path in the circuit contains at most 9 gates (including NORs and inverters). But there are 2 possible conditions containing 9 gates:

I) Changing one of a_0 or b_0 which results in the change of c_2. This case happens as the output changes to 100 from 011. e.g. $\{a_1a_0b_1b_0c_0\}$ = 00101 → 01101 , which passes through this path:

$a_0 \rightarrow N1 \rightarrow N3 \rightarrow i5 \rightarrow N6 \rightarrow N8 \rightarrow N15 \rightarrow N17 \rightarrow i18 \rightarrow c2$

figure 1.9 illustrates this transitions path:
(red numbers are the values before the change and green one's are those after we change the value of a_0)

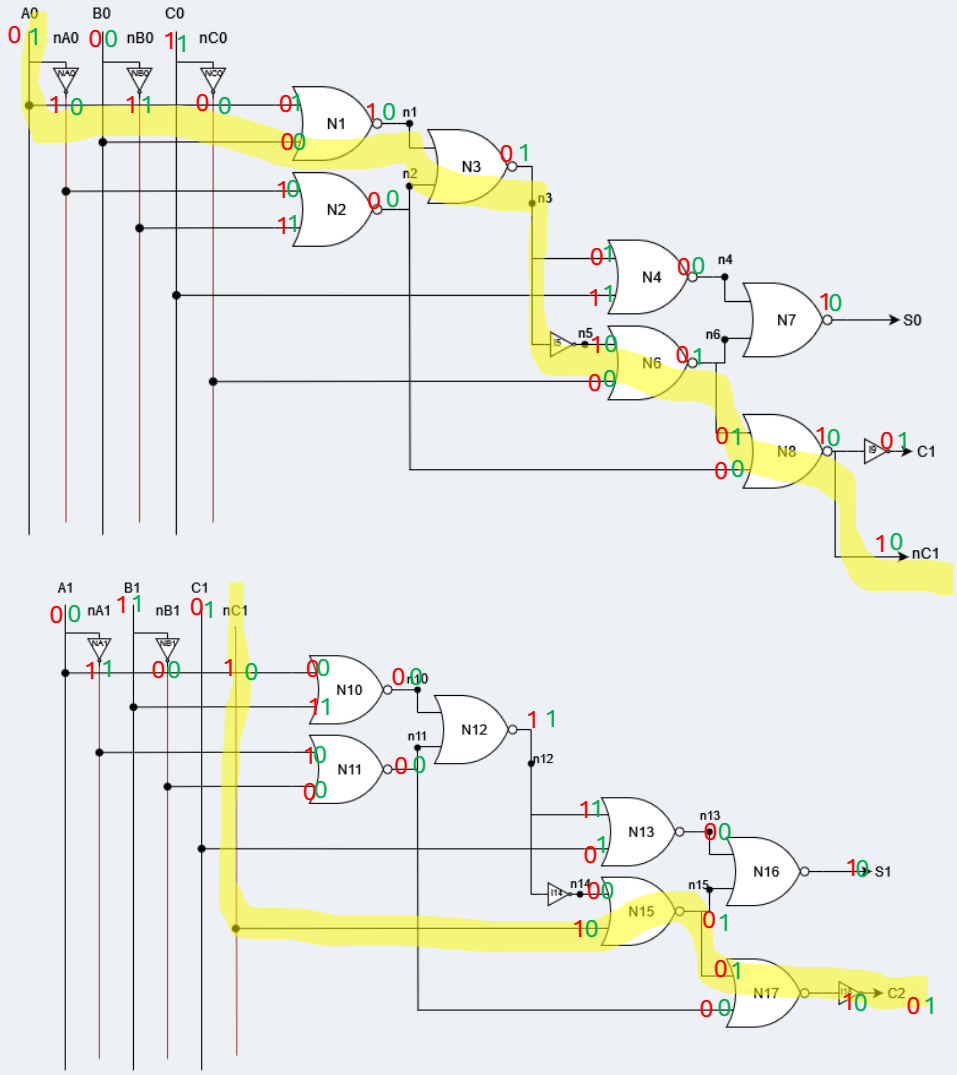

Figure 1.9

So, the delay for this path will be calculated like this:

*14+10+7+10+14+10+14+5=**84 ns***

(14: NORs to-0 , 10:NORs to-1 , 7: NOTs to-0 , 5:NOTs to-1)

II) *Changing one of a_0 or b_0 which results in some changes in s_1. This case happens as the output changes to 111 from 110. e.g. $\{a_1a_0b_1b_0c_0\}$ = 11111 → 11101 , which passes through this path:*

b_0 →NBO →N2 →N3 →i5 →N6 →N8 →i9 →N13 →N16 →s1

figure 1.10 illustrates this transitions path:

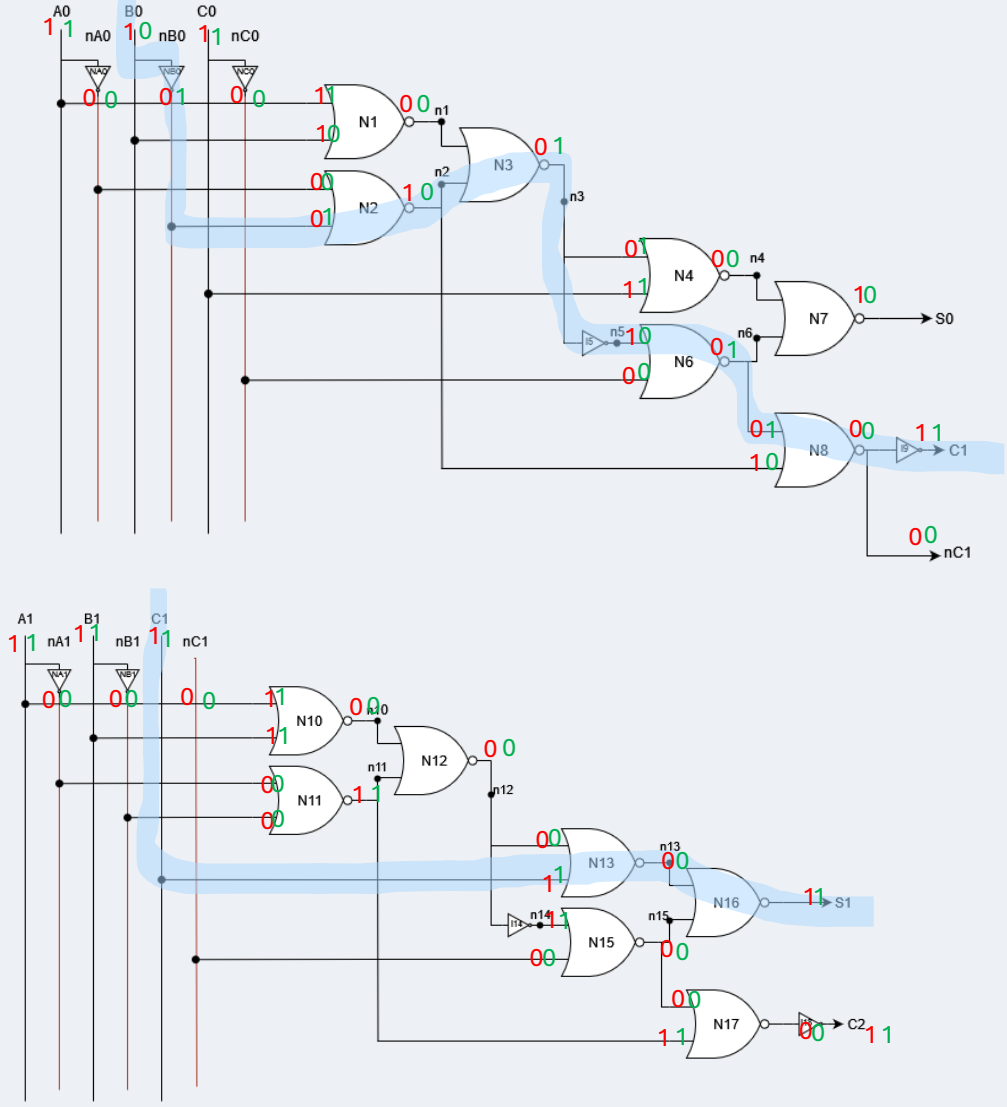

Figure 1.10

As we can see, s1 didn't changed. But the output of N8 will be changed for a while before N6's output changed to 1. So there'll be a hazard!

The short path:

NB0-T0-1 + N2-TO-0 + N8-T0-1 + i9-T0-0 + N13-TO-1 + N16-TO-0 =5+14+10+7+10+14 = ***60 ns***

The short path:

*NB0-T0-1 + N2-TO-0 + N3-T0-1 + i5-T0-0 + N6-TO-1 + N8-TO-0 + i9-TO-1 + N13-TO-0 + N16-TO-1 = 5+14+10+7+10+14+5+14+10 =****89ns***

So,

Time that hazard occurs: ***60 ns***

Hazard's length: 89 -60 = ***29 ns***

Thus, we can report ***89 ns*** *as the worst-case delay of our adder.*

Figures 1.11, 1.12 and 1.13 illustrate the Verilog description and the testbench of this adder:

```
`timescale 1ns/1ns
module inv (input a, output w);
    not #(5,7) (w,a);
endmodule

module thenor (input a,b, output w);
    nor #(10,14) (w,a,b);
endmodule

module noradder (input a1,a0,b1,b0,c0, output c2,s1,s0);
    wire na0,nb0,nc0,n1,n2,n3,n4,n5,n6,c1,nc1,na1,nb1,n10,n11,n12,n13,n14,n15,n17;
    supply1 Vdd;
    supply0 Gnd;
    //first_layer
    inv NA0 (a0,na0);
    inv NB0 (b0,nb0);
    inv NC0 (c0,nc0);
    thenor N1 (a0,b0,n1);
    thenor N2 (na0,nb0,n2);
    thenor N3 (n1,n2,n3);
    thenor N4 (n3,c0,n4);
    inv i5 (n3,n5);
    thenor N6 (n5,nc0,n6);
    thenor N7 (n4,n6,s0);
    thenor N8 (n6,n2,nc1);
    inv i9 (nc1,c1);
    //second_layer
    inv NA1 (a1,na1);
    inv NB1 (b1,nb1);
    thenor N10 (a1,b1,n10);
    thenor N11 (na1,nb1,n11);
    thenor N12 (n10,n11,n12);
    thenor N13 (n12,c1,n13);
    inv i14 (n12,n14);
    thenor N15 (n14,nc1,n15);
    thenor N16 (n13,n15,s1);
    thenor N17 (n15,n11,n17);
    inv i18 (n17,c2);
endmodule
```

Figure 1.11 Verilog description

```
`timescale 1ns/1ns
module noradderTB ();
    logic aa1,aa0,bb1,bb0,cc0;
    wire cc2,ss1,ss0;
    noradder CUT(aa1,aa0,bb1,bb0,cc0,cc2,ss1,ss0);
    initial begin
    aa1=0; aa0=0; bb1=1; bb0=0; cc0=1;
    #100 aa0=1;
    #100 $stop;
    end
endmodule
```

Figure 1.12 Verilog Testbench for the first path

```
`timescale 1ns/1ns
module noradderTB2 ();
    logic aa1,aa0,bb1,bb0,cc0;
    wire cc2,ss1,ss0;
    noradder CUT22(aa1,aa0,bb1,bb0,cc0,cc2,ss1,ss0);
    initial begin
    aa1=0; aa0=0; bb1=0; bb0=0; cc0=0;
    #100 aa0=1;
    #100 bb0=1;
    #100 aa1=1;
    #100 cc0=1;
    #100 bb1=1;
    #100 bb0=0; //hazard on s1 = 89ns
    #100 $stop;
    end
endmodule
```

Figure 1.13 Verilog Testbench for the second path

Figures 1.14 and 1.15 show the Verilog simulation results.

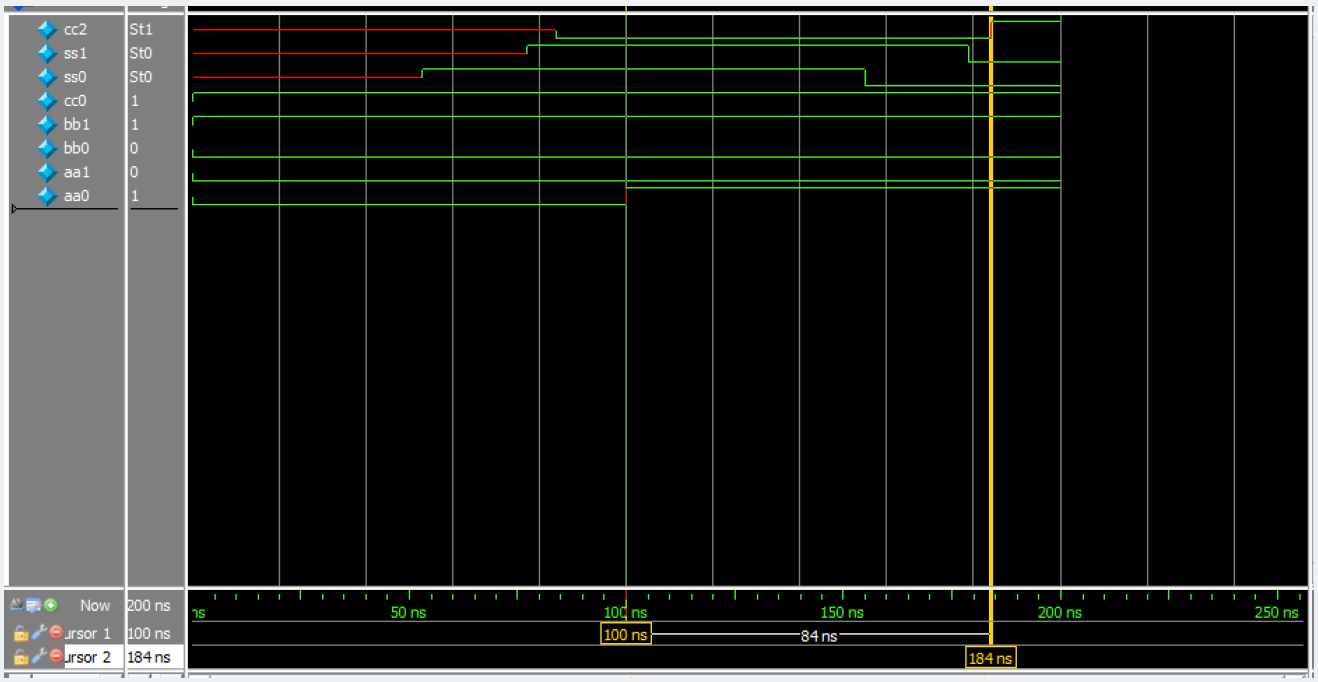

Figure 1.14 Simulation for the first path

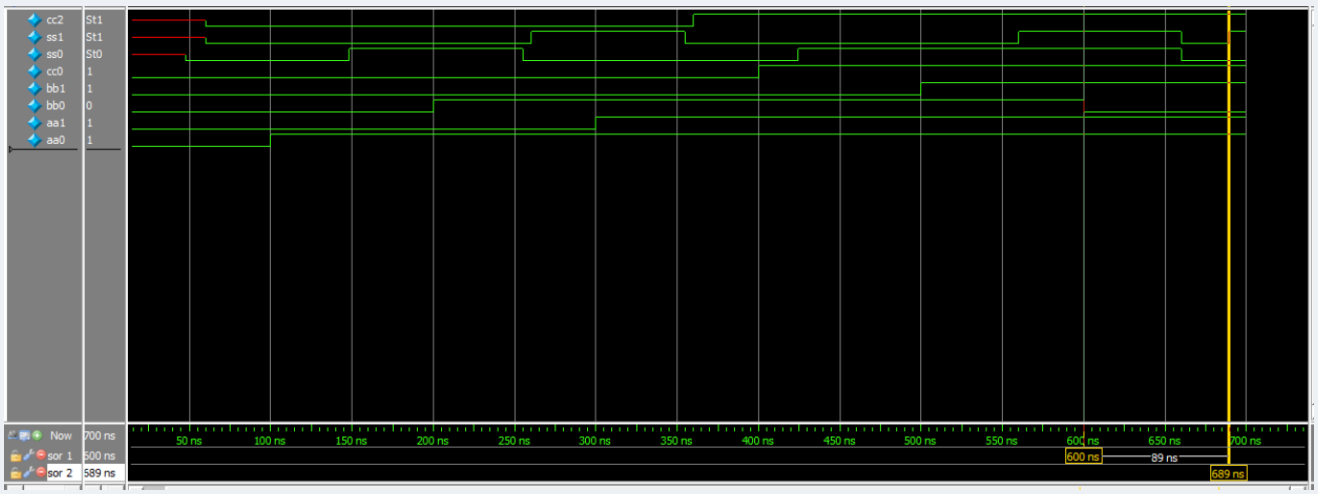

Figure 1.15 Simulation for the second path

Problem 2

Figures 2.1 and 2.2 illustrate the Verilog description and the testbench of this adder (with an assign statement):

```
`timescale 1ns/1ns
module q2adder (input a1,a0,b1,b0,c0, output c2,s1,s0);
    assign #89 {c2,s1,s0}={a1,a0}+{b1,b0}+{c0};
endmodule
```

Figure 2.1 Verilog description

```
`timescale 1ns/1ns
module adderq2TB ();
    logic aa1,aa0,bb1,bb0,cc0;
    wire cc2,ss1,ss0;
    q2adder CUTq2(aa1,aa0,bb1,bb0,cc0,cc2,ss1,ss0);
    initial begin
        aa1=0; aa0=0; bb1=0; bb0=0; cc0=0;
    #100 aa0=1;
    #100 bb0=1;
    #100 aa1=1;
    #100 cc0=1;
    #100 bb1=1;
    #100 bb0=0;
    #100 aa1=0;
    #100 aa0=0;
    #100 cc0=1;
    #100 aa0=1;
    #100 $stop;
    end
endmodule
```

Figure 2.2 Verilog Testbench

Figure 2.3 shows the Verilog simulation result.

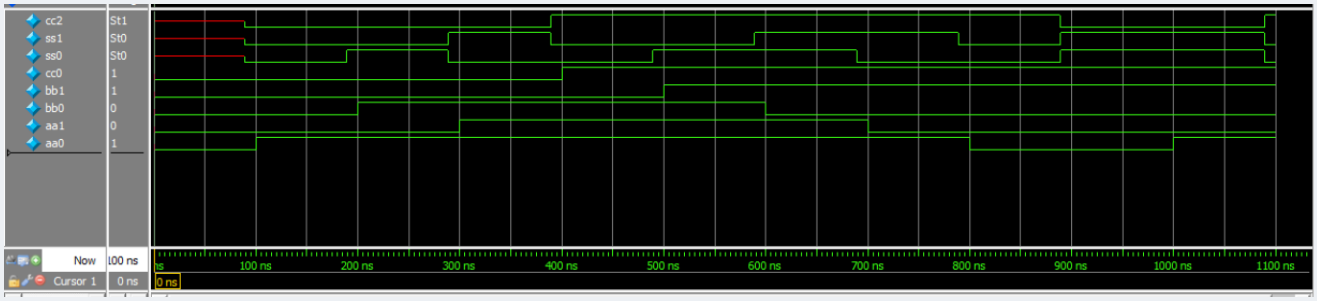

Figure 2.3

Problem 3

As there in the first problem, we can replace S_0 with the logic statement below:

$$S_0 = A_0 \oplus B_0 \oplus C_0$$

And also, the Karnaugh map for C_1 will look like:

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

Spiriting the map in to 2 parts, results in a MUX that is controlled by c and the inputs are (a.b) and (a+b).

Figure 3.1 illustrates this 1-bit adder:

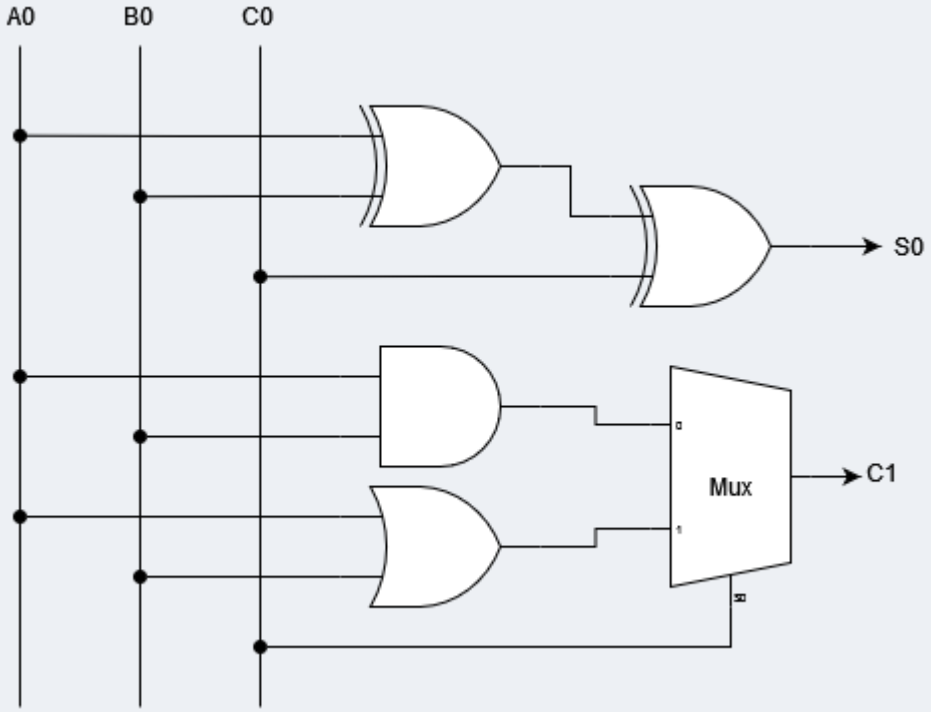

Figure 3.1

We can create AND, XOR and OR gates using multiplexers:

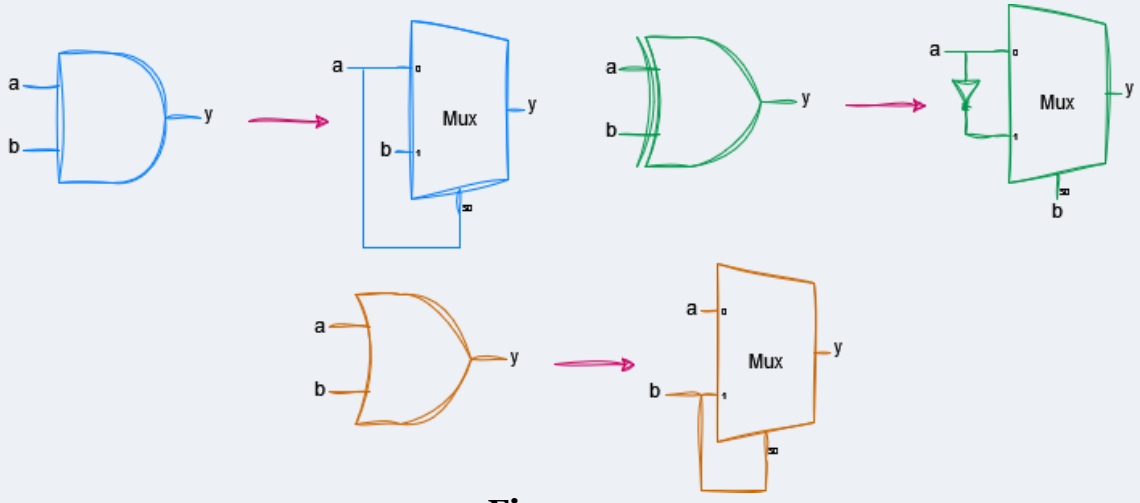

Figure 3.2

*So, **<u>the first layer of</u>** our adder with multiplexers looks like:*

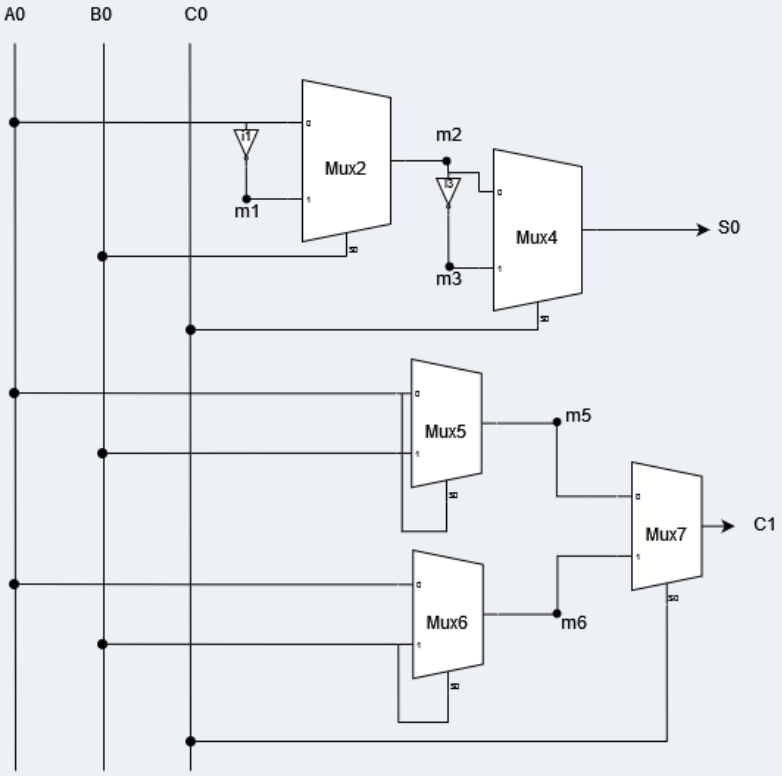

Figure 3.3

And also, the second layer looks like the diagram below:

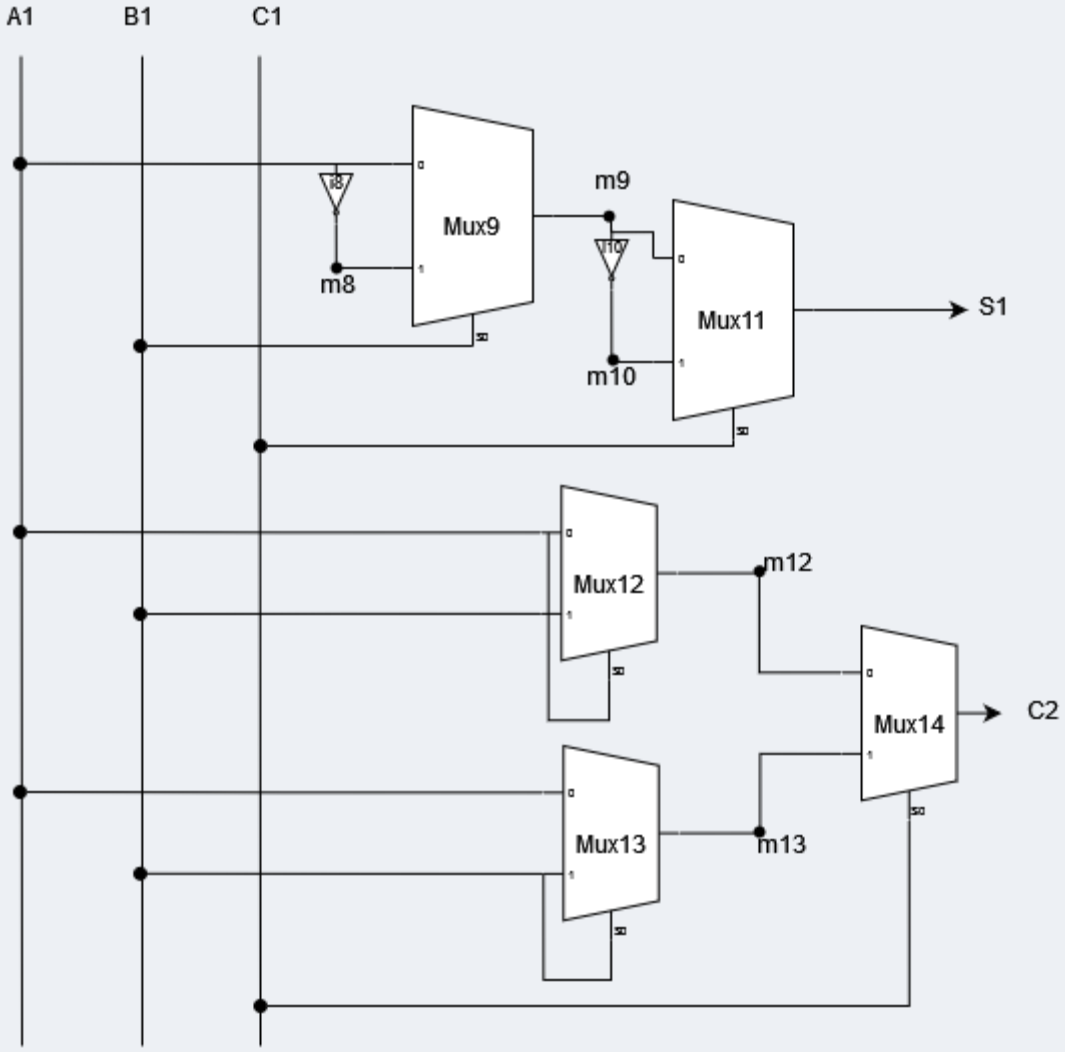

Figure 3.4

Now let's find the worst-case delays. Based on the diagrams, each path in the circuit contains at most 3 MUXs. So the worst-case delay for this circuit will be $3 \times 12 = \mathbf{36\ ns}$.

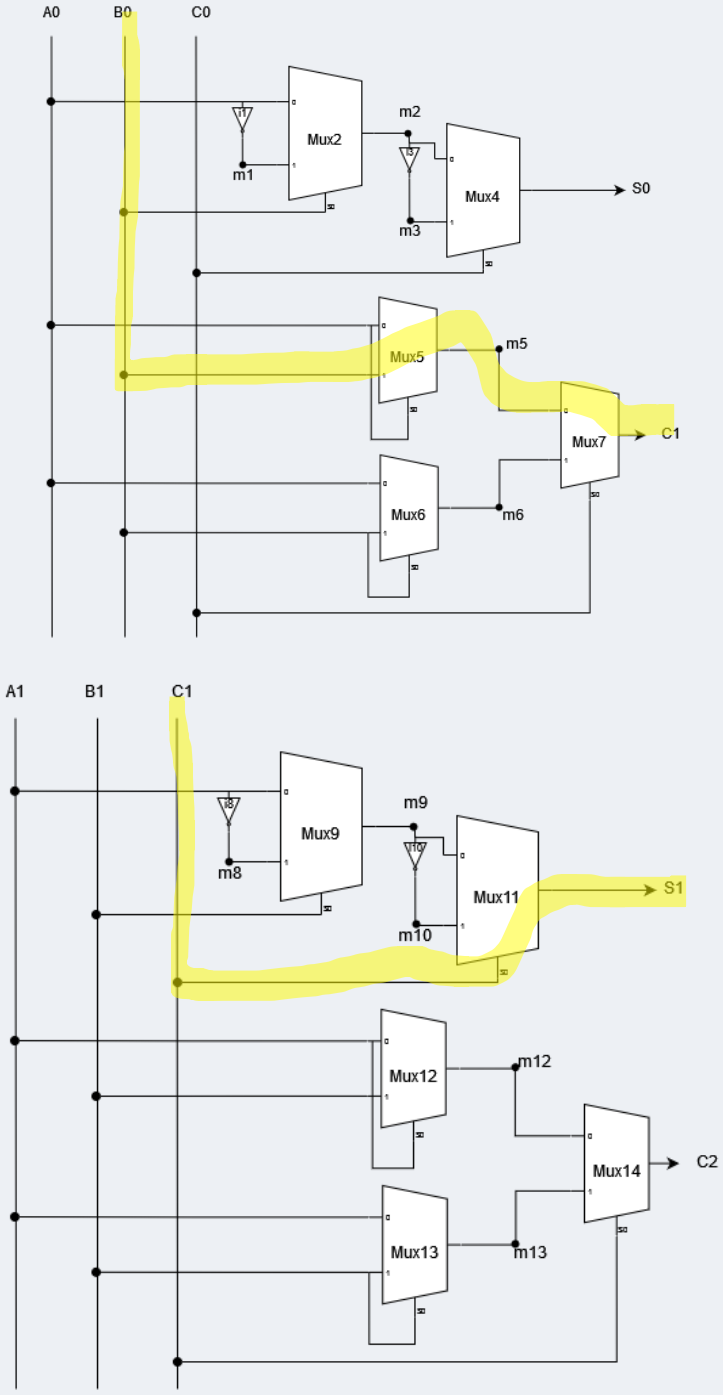

Figure 3.5

Figures 3.6 and 3.7 illustrate the Verilog description and the testbench of this adder:

```
`timescale 1ns/1ns
module inv (input a, output w);
    not #(5,7) (w,a);
endmodule
module themux (input s,a,b, output w);
    assign #12 w = ~s ? a:b;
endmodule
module muxadder (input a1,a0,b1,b0,c0, output c2,s1,s0);
    wire m1,m2,m3,m5,m6,c1,m8,m9,m10,m12,m13;
    supply1 vdd;
    supply0 Gnd;
    //first_layer
    inv i1 (a0,m1);
    themux MUX2 (b0,a0,m1,m2);
    inv i3 (m2,m3);
    themux MUX4 (c0,m2,m3,s0);
    themux MUX5 (a0,a0,bl0,m5);
    themux MUX6 (b0,a0,b0,m6);
    themux MUX7 (c0,m5,m6,c1);
    //second_layer
    inv i8 (a1,m8);
    themux MUX9 (b1,a1,m8,m9);
    inv i10 (m9,m10);
    themux MUX11 (c1,m9,m10,s1);
    themux MUX12 (a1,a1,b1,m12);
    themux MUX13 (b1,a1,b1,m13);
    themux MUX14 (c1,m12,m13,c2);
endmodule
```

Figure 3.6 Verilog description

```
`timescale 1ns/1ns
module muxadderTB ();
    logic aa1,aa0,bb1,bb0,cc0;
    wire cc2,ss1,ss0;
    muxadder CUTq3(aa1,aa0,bb1,bb0,cc0,cc2,ss1,ss0);
    initial begin
    aa1=1; aa0=0; bb1=0; bb0=0; cc0=1;
    #100 bb0=1;
    #100 $stop;
    end
```

Figure 3.7 Verilog Testbench

Figure 3.8 shows the Verilog simulation results.

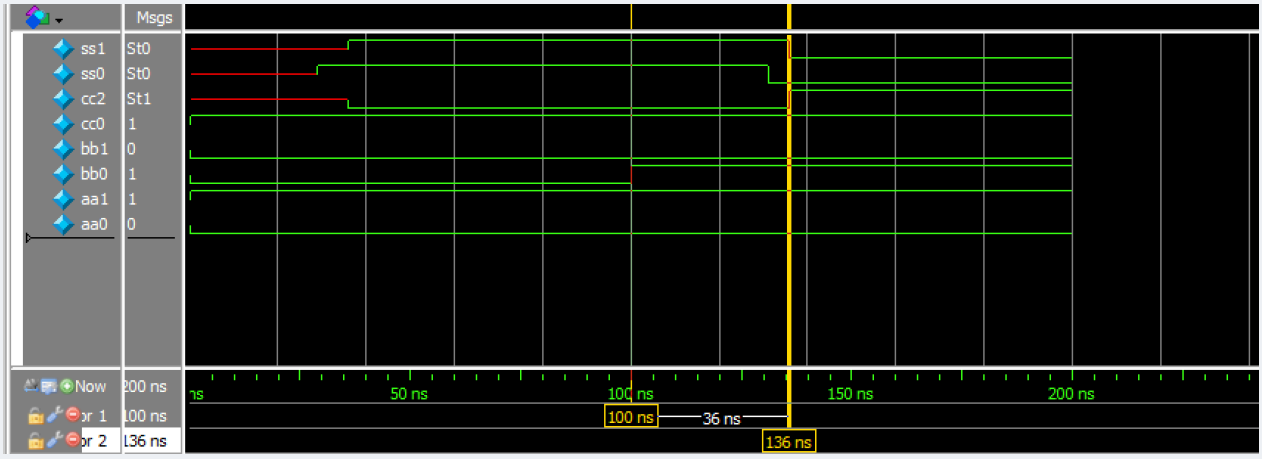

Figure 3.8

Problem 4

Figures 4.1 and 4.2 illustrate the Verilog description and the testbench of this adder (with an assign statement):

```
`timescale 1ns/1ns
module q4adder (input a1,a0,b1,b0,c0, output c2,s1,s0);
    assign #36 {c2,s1,s0}={a1,a0}+{b1,b0}+{c0};
endmodule
```

Figure 4.1 Verilog description

```
`timescale 1ns/1ns
module adderq4TB ();
    logic aa1,aa0,bb1,bb0,cc0;
    wire cc2,ss1,ss0;
    q4adder CUTq4(aa1,aa0,bb1,bb0,cc0,cc2,ss1,ss0);
    initial begin
        aa1=0; aa0=0; bb1=0; bb0=0; cc0=0;
    #100 aa0=1;
    #100 bb0=1;
    #100 aa1=1;
    #100 cc0=1;
    #100 bb1=1;
    #100 bb0=0;
    #100 aa1=0;
    #100 aa0=0;
    #100 cc0=0;
    #100 aa0=1;
    #100 $stop;
    end
endmodule
```

Figure 4.2 Verilog Testbench

Figure 4.3 shows the Verilog simulation result.

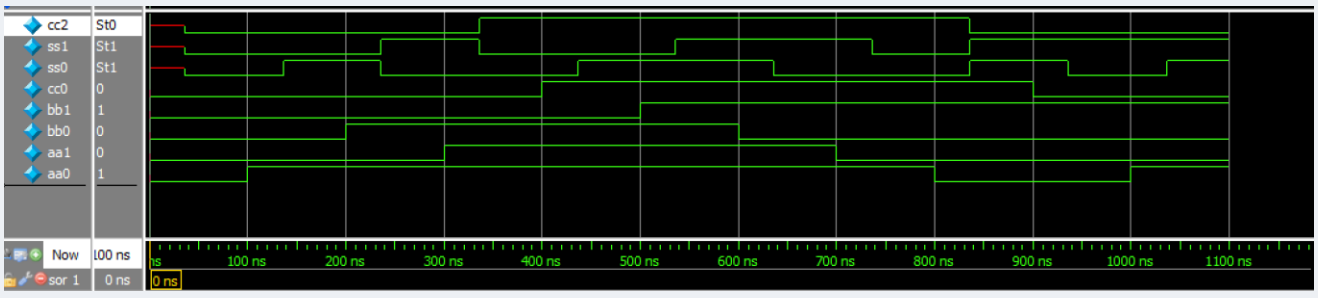

Figure 4.3

Problem 5

For this problem, a single Verilog testbench has been generated in which, there are a series of outputs for the 1st problem and a series of outputs for the 3rd one. (fig. 5.1)

```
`timescale 1ns/1ns
module Q5TB ();
    logic aa1,aa0,bb1,bb0,cc0;
    wire cc2q1,ss1q1,ss0q1,cc2q3,ss1q3,ss0q3;
    noradder CUT1(aa1,aa0,bb1,bb0,cc0,cc2q1,ss1q1,ss0q1);
    initial begin
    aa1=0; aa0=0; bb1=0; bb0=0; cc0=0;
    #100 aa0=1;
    #100 bb0=1;
    #100 aa1=1;
    #100 cc0=1;
    #100 bb1=1;
    #100 bb0=0;
    #100 $stop;
    end
    muxadder CUT2(aa1,aa0,bb1,bb0,cc0,cc2q3,ss1q3,ss0q3);
    initial begin
    aa1=0; aa0=0; bb1=0; bb0=0; cc0=0;
    #100 aa0=1;
    #100 bb0=1;
    #100 aa1=1;
    #100 cc0=1;
    #100 bb1=1;
    #100 bb0=0;
    #100 $stop;
    end
endmodule
```

Figure 5.1

Figure 5.2 shows the Verilog simulation results.

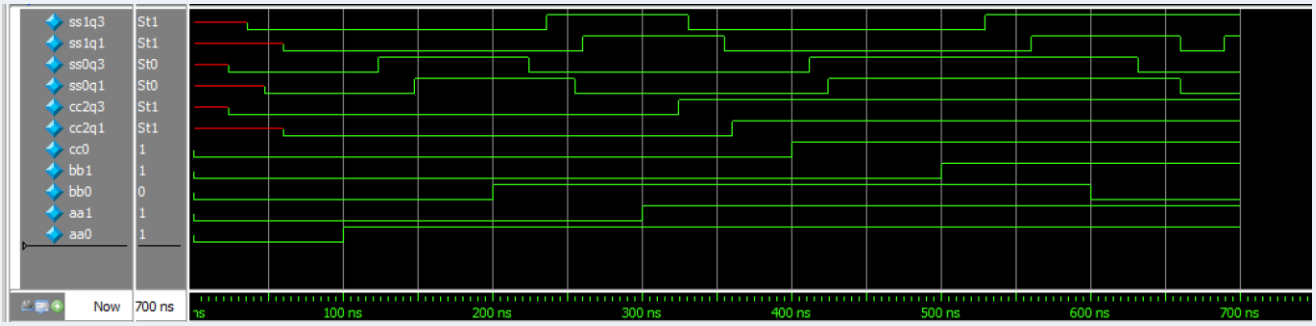

Figure 3.10 Verilog simulation

As expected, the output of the 3rd problem's structure, changes with a less delay in comparison with the 1st structure.

*In the NOR gate structure, we have to use 14 2-input NOR gates and 9 inverters; which means there are (14×4 + 9×2 =) **74** transistors at all in this struct. On the other hand, we have 10 2-to-1 MUX gates and 4 inverters. So, there are **48** transistors in total.*

So, the adder with NOR gates, consumes more power than the other one in the result of the difference in the number of transistors.

But as in CA1, pass-transistor-based MUX's will result in weak-one and because of the inverters in their initial structures, there'll be a short circuit; so, the power consumption will increase.

Problem 6

The assign statement for this problem and the Yosys libraries and the command we use, illustrated in the figures below:

```
`timescale 1ns/1ns
module Q6 (input a1,a0,b1,b0,c0, output c2,s1,s0);
    assign {c2,s1,s0}={a1,a0}+{b1,b0}+{c0};
endmodule
```

Figure 6.1

```
`timescale 1ns/1ns
C:\Users\user\AppData\Local\Temp\Rar$DIa19904.15286\synth.v
module NOT(A, Y);
input A;
output Y;
assign #(5,7) Y = ~A;
endmodule

module NOR(A, B, Y);
input A, B;
output Y;
assign #(10,14) Y = ~(A | B);
endmodule

module NOR3(A, B, C, Y);
input A, B, C;
output Y;
assign #(15,21) Y = ~(A | B | C);
endmodule
```

Figure 6.2

```
library(demo) {
  cell(BUF) {
    area: 6;
    pin(A) { direction: input; }
    pin(Y) { direction: output;
             function: "A"; }
  }
  cell(NOT) {
    area: 3;
    pin(A) { direction: input; }
    pin(Y) { direction: output;
             function: "A'"; }
  }

  cell(NOR) {
    area: 4;
    pin(A) { direction: input; }
    pin(B) { direction: input; }
    pin(Y) { direction: output;
             function: "(A+B)'"; }
  }

    cell(NOR3) {
    area: 5;
    pin(A) { direction: input; }
    pin(B) { direction: input; }
    pin(C) { direction: input; }
    pin(Y) { direction: output;
             function: "(A+B+C)'"; }
  }


}
```

Figure 6.3

```
# read design
read_verilog Q6.sv

# generic synthesis
synth -top Q6

# mapping to mycells.lib
dfflibmap -liberty mycells.lib
abc -liberty mycells.lib
clean

# write synthesized design
write_verilog synth.v
```

Figure 6.4 Commands

The synthetization's result is written to the synth.v file.
The testbench and the result of simulation are shown in the figures below.

```
`timescale 1ns/1ns
module Q6TB ();
    logic aa1,aa0,bb1,bb0,cc0;
    wire cc2,ss1,ss0;
    Q6syn CUT66(aa1,aa0,bb1,bb0,cc0,cc2,ss1,ss0);
    initial begin
    aa1=0; aa0=0; bb1=0; bb0=0; cc0=0;
    #100 aa0=1;
    #100 bb0=1;
    #100 aa1=1;
    #100 cc0=1;
    #100 bb1=1;
    #100 bb0=0; //hazard on s1 = 89ns
    #100 $stop;
    end
endmodule
```

Figure 6.5 Verilog Testbench

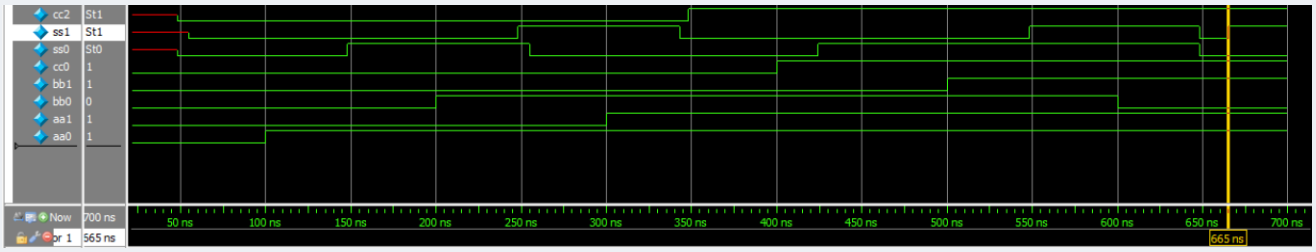

Figure 6.6 Simulation result

*As it is obvious, the worst case delay for this structure is **65ns**.*

Also, the structure of synthetizing illustrated in this diagram:

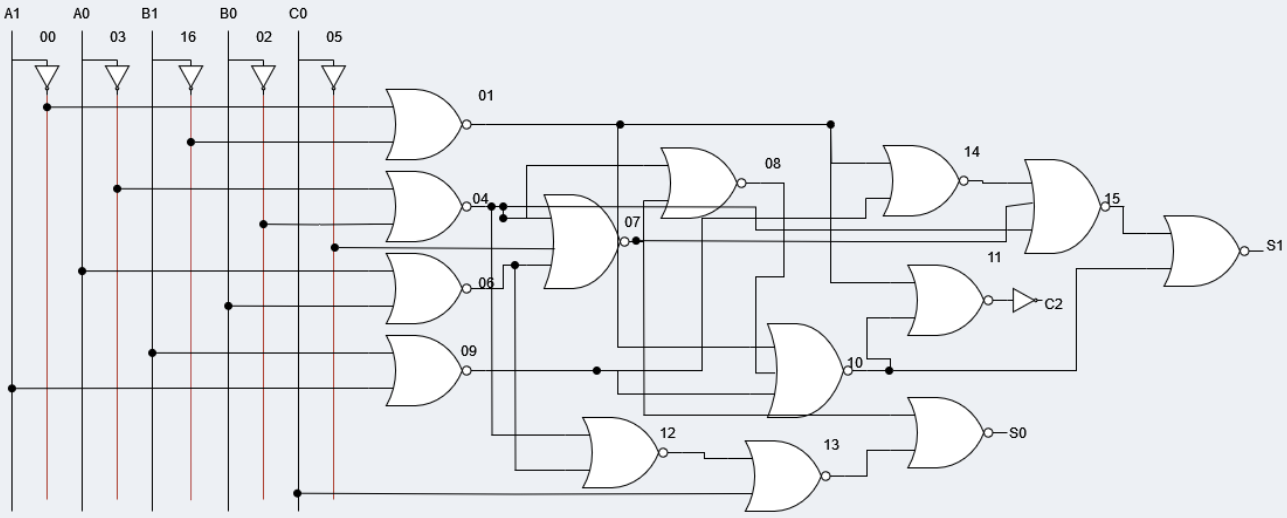

Figure 6.7

Problem 7

The number of gates and the delay value of the synthesized structure is less than our design. This is actually because of the use of 3-input NORs and also restricted available gates in Yosys's library.