# UNIVERSITY OF TEHRAN

## Report for Computer Assignment 3

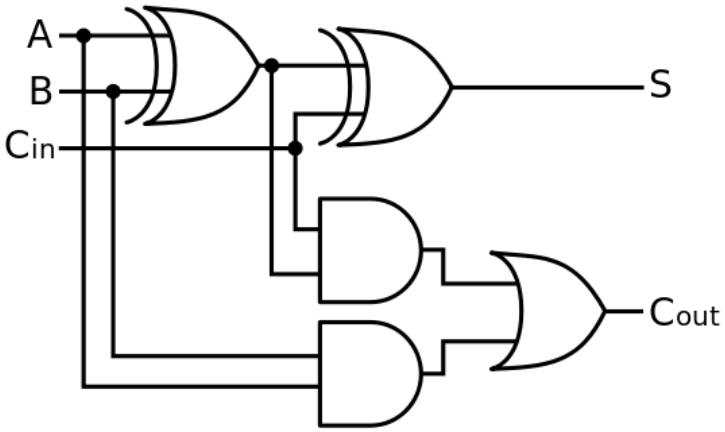
### RT Level Components, Iterative Logic, Synthesis

## Instructor : Dr. Navabi


## Danial Saeedi

# Problem 1

## Circuit Diagram




## Calculating Delay Values

### AND Gate Delay

In computer Assignment 1, we calculated the worst-case delay of NAND gate.( To 1 = 10ns ,To 0 = 8ns, Avg = 9ns)

Delay values for NOT gate are #(5,6). (Avg = 5ns)

If we invert a NAND gate, we get AND get. So the AND gate delay is 5 + 9 = 14ns.

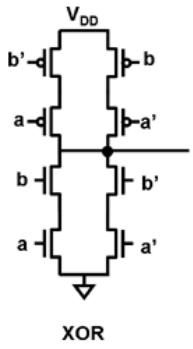
### OR Gate Delay

Delay values for NOR gate is #(10,14). (Avg = 12ns)

If we invert a NOR gate, we get OR get. So the OR gate delay is 12+5 = 17ns

### XOR Gate Delay

The XOR gate delay is 9 + 9 + 5 = 23ns



## Worst-case delay of FA

The sum output comes from two consecutive XOR gates. So worst-case delay for S is 23 + 23 = 46

The longest path from input to carry-out(co) includes AND, OR gate. So worst-case delay for CO is 14 + 17 = 31

So the worst-case delay for FA is **46ns**.

## Verilog Code

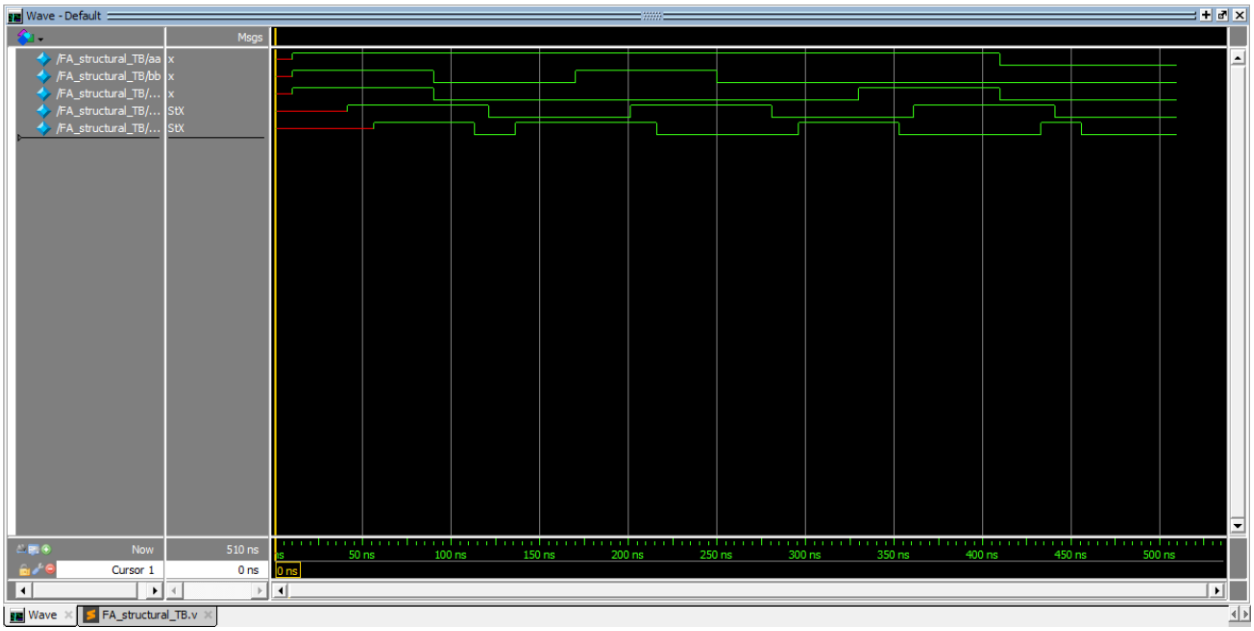
Structural implementation of FA :

```
`timescale 1ns/1ns
module FA_structural(input a,b,cin,output s,co);
    wire w1,w2,w3;
    xor #23 g1(w1,a,b);
    xor #23 g2(s,w1,cin);
    and #14 g3(w2,w1,cin);
    and #14 g4(w3,a,b);
    or #17 g5(co,w2,w3);
endmodule
```

## Testbench

```
`timescale 1ns/1ns
module FA_structural_TB();
    reg aa,bb,ccin;
    wire cout,sum;
    FA_structural CUT1(aa,bb,ccin,sum,cout);

    initial begin
        #10 aa = 1; bb = 1; ccin = 1;
        #80 aa = 1; bb = 0; ccin = 0;
        #80 aa = 1; bb = 1; ccin = 0;
        #80 aa = 1; bb = 0; ccin = 0;
        #80 aa = 1; bb = 0; ccin = 1;
        #80 aa = 0; bb = 0; ccin = 0;
        #80 aa = 0; bb = 1; ccin = 1;
        #80 aa = 0; bb = 1; ccin = 0;
        #100 $stop;
    end
endmodule
```

## Simulation Result




As you can see, the worst-case delay for s and co output are **46ns** and **31ns**.

## N-bit Ripple Carry Adder (Structural)

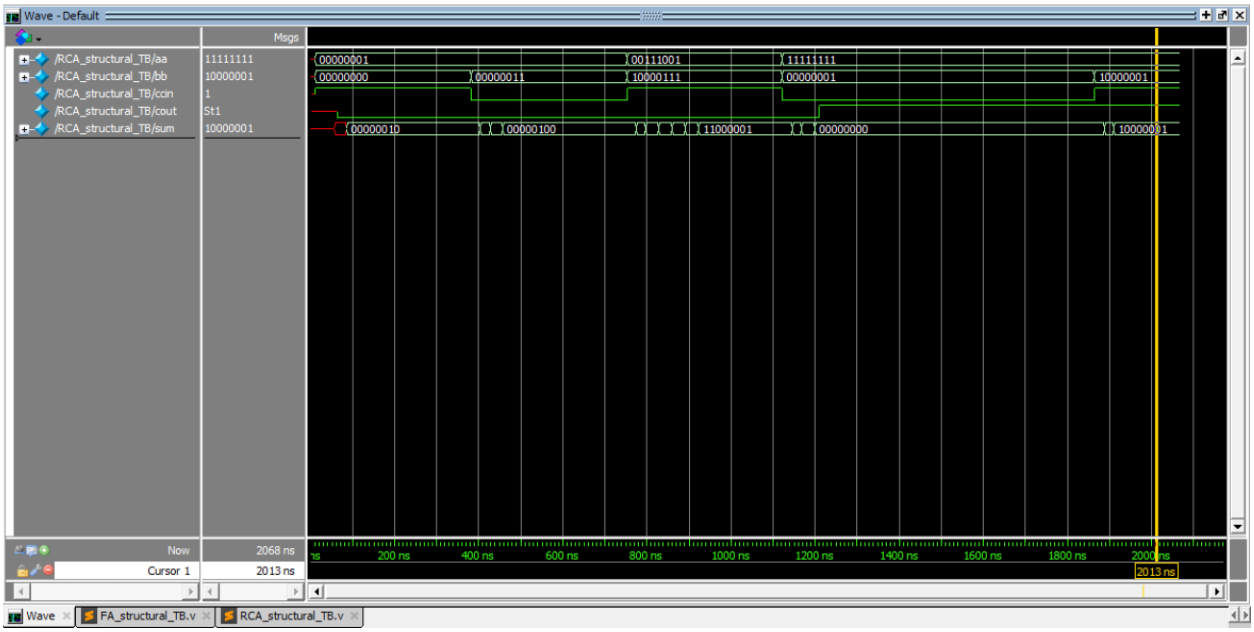
N-bit Ripple Carry Adder has been implemented using generate statement.

```
`timescale 1ns/1ns
module RCA_structural #( parameter SIZE = 8 )
                       (input [SIZE-1:0] A,B,input cin, output [SIZE-1:0] sum,output co);
    wire [0:SIZE] carry_in;
    assign carry_in[0] = cin;
    genvar i;
    generate
        for(i = 0; i < SIZE; i = i + 1) begin
            FA_structural XX(A[i],B[i],carry_in[i],sum[i],carry_in[i+1]);
        end
    endgenerate
    assign co = carry_in[SIZE];
endmodule
```

## Testbench

```
1
2  `timescale 1ns/1ns
3  module RCA_structural_TB();
4      reg [7:0] aa,bb;
5      reg ccin;
6      wire cout;
7      wire [7:0] sum;
8      RCA_structural #(8) CUT1 (aa,bb,ccin,sum,cout);
9
10     initial begin
11         #10 aa = 8'b00000001; bb = 8'b00000000; ccin = 1;
12         #371 aa = 8'b00000001; bb = 8'b00000011; ccin = 0;
13         #371 aa = 8'b00111001; bb = 8'b10000111; ccin = 1;
14         #371 aa = 8'b11111111; bb = 8'b00000001; ccin = 0;
15         #371 aa = 8'b11111111; bb = 8'b00000001; ccin = 0;
16         #371 aa = 8'b11111111; bb = 8'b10000001; ccin = 1;
17         #203 $stop;
18     end
19 endmodule
```

## Simulation Result




In a 8-bit RCA, 8 Full-Adder are cascaded together. We know the worst-case delay of FA is 46ns.

So at most, it should take **46*8 = 368ns** in the worst-case.

# Problem 2

## Verilog Code

In problem 1, we calculated worst-case delay of FA which is 46ns. So n-bit Ripple Carry Adder has **46*n** delay at most.

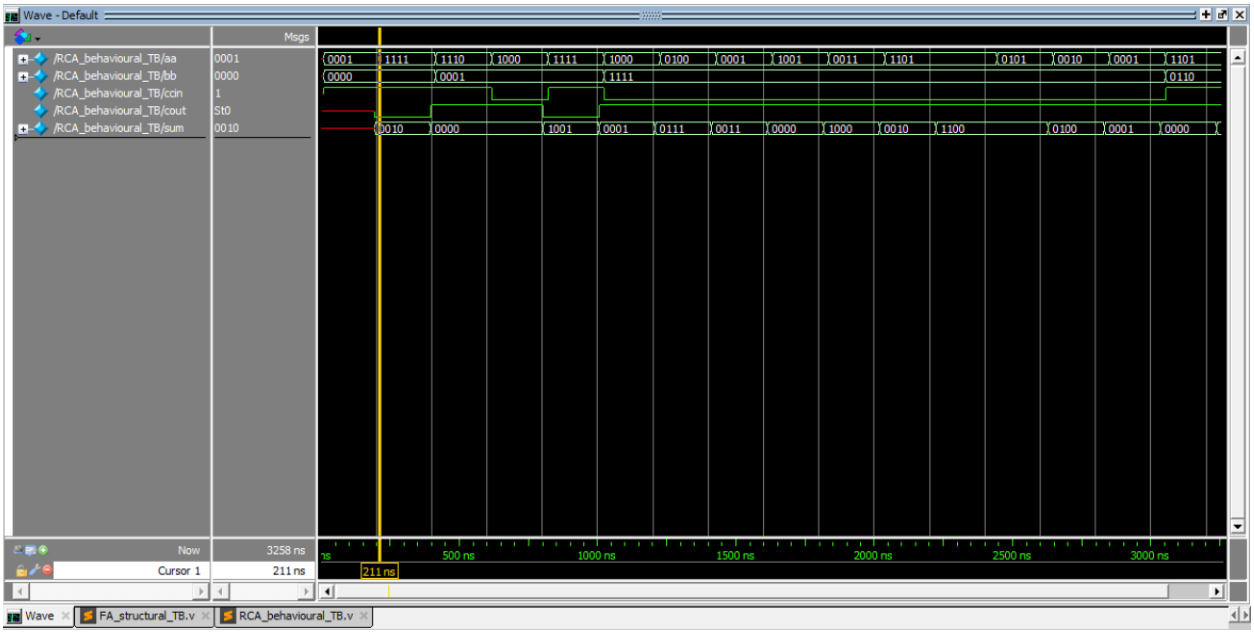
```
1  `timescale 1ns/1ns
2  module RCA_behavioural #(parameter n = 8)
3      (input [n-1:0] A,B,input cin,output [n-1:0] s, output co);
4      assign #(46*n) {co,s} = A + B + cin;
5  endmodule
```

# Problem 3

## Testbench

```
1
2  `timescale 1ns/1ns
3  module RCA_behavioural_TB();
4      reg [3:0] aa,bb;
5      reg ccin;
6      wire cout;
7      wire [3:0] sum;
8      RCA_behavioural #(4) CUT1 (aa,bb,ccin,sum,cout);
9
10     initial begin
11         #10 aa = 4'b0001; bb = 4'b0000; ccin = 1;
12         #203 aa = 4'b1111; bb = 4'b0000; ccin = 1;
13         #203 aa = 4'b1110; bb = 4'b0001; ccin = 1;
14         #203 aa = 4'b1000; bb = 4'b0001; ccin = 0;
15         #203 aa = 4'b1111; bb = 4'b0001; ccin = 1;
16         #203 aa = 4'b1000; bb = 4'b1111; ccin = 0;
17         repeat(10) #203 aa = $random(); bb = $random(); ccin = $random();
18         #203 $stop;
19     end
20 endmodule
```

## Simulation Result



# Problem 4

## Circuit Diagram

## ????????

## Verilog Code

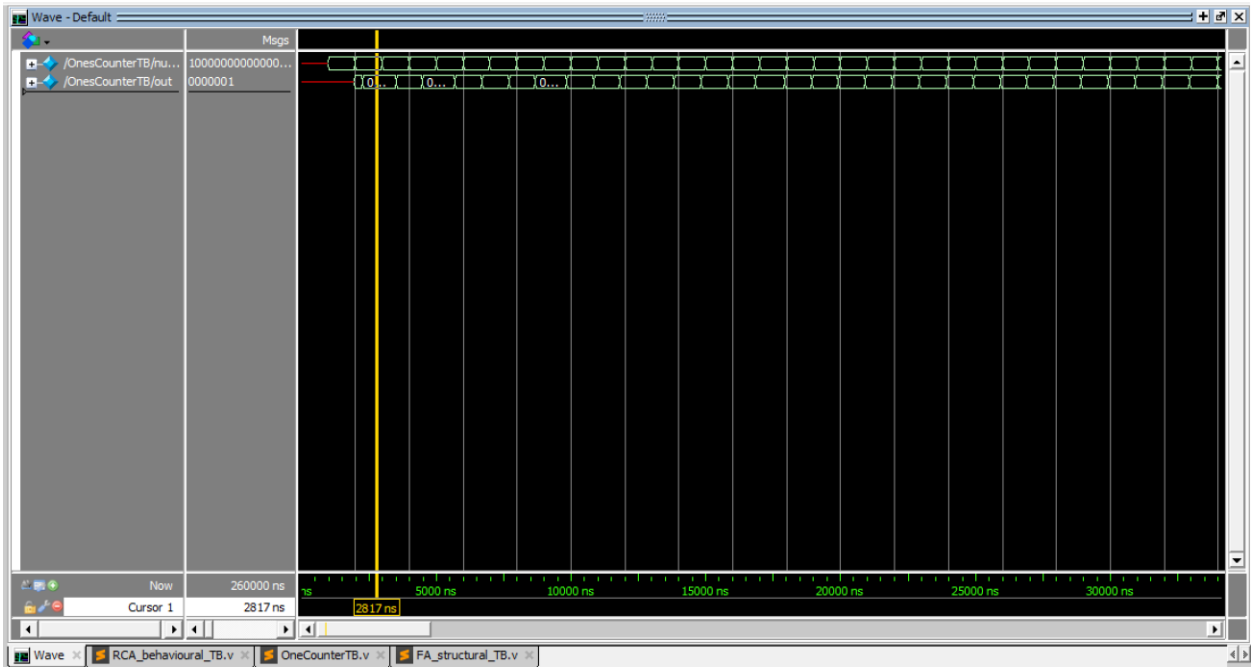
```
`timescale 1ns/1ns
module OnesCounter #(parameter levels = 6) (input [0 : 2 ** (levels + 1) - 2] in, output [levels : 0] out);
    wire c[1:2**levels];
    wire [levels-1:0] w[1:2**levels];
    assign out = {c[1], w[1]};
    genvar i, j;
    generate
        for(i = 0; i < levels; i = i + 1) begin
            for(j = 2 ** i; j < 2 ** (i + 1); j = j + 1) begin
                if(i < levels - 1)
                    RCA_behavioural #(levels - i) XX({c[2 * j], w[2 * j][levels - i - 2 : 0]}, {c[2 * j + 1],
                        w[2 * j + 1][levels - i - 2 : 0]}, in[j - 1], w[j][levels - i - 1 : 0], c[j]);
                else
                    RCA_behavioural #(levels - i) XX(in[2 * j - 1], in[2 * j + 1 - 1], in[j - 1],
                        w[j][levels - i - 1 : 0], c[j]);
            end
        end
    endgenerate
endmodule
```

# Problem 5

## Testbench

```
`timescale 1ns / 1ns
module OnesCounterTB();
    reg [126 : 0] numb;
    wire [6 : 0] out;
    OnesCounter #6 onecounter(numb,out);
    initial begin
        //Marching-1
        #1000 numb = 127'b0;
        repeat(127*2) begin
            #1000 numb = {~numb[0], numb[126 : 1]};
        end

        //Other tests
        #1000 numb = 127'b101111100011111111111111000000;
        #1000 numb = 127'b101111100011111111111111100001100110011;
        #1000 numb = 127'b101111100011111111111111100000011111111111;
        #1000 numb = 127'b10111110001111111111111100000011111111110000000000000111111111;
        #1000 $stop;
    end
endmodule
```

## Simulation Result



# Problem 6

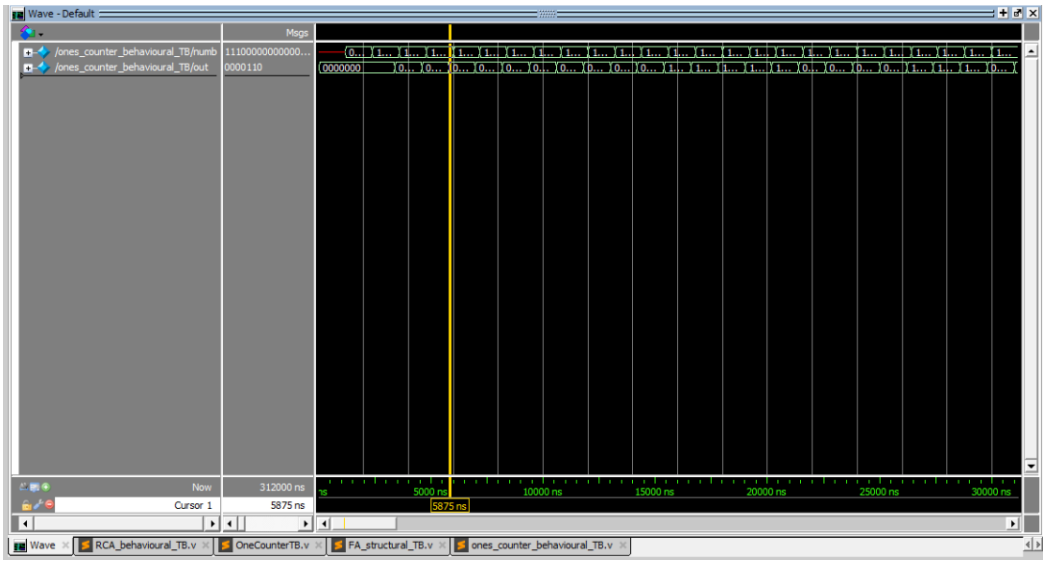
## Verilog Code

```
`timescale 1ns/1ns
module ones_counter_behavioural #(parameter n = 127)
    (input [n-1:0] A,output reg [n-1:0] count);

    integer i;
    always @ (A) begin
        count = 127'b0;
        #1000
        for (i = 0; i < n; i = i + 1) begin
            if (A[i] == 1'b1)
                count = count + 1;
        end
    end
endmodule
```

## Testbench

```
`timescale 1ns / 1ns
module ones_counter_behavioural_TB();
    reg [126 : 0] numb;
    wire [6 : 0] out;
    ones_counter_behavioural onecounter(numb,out);
    initial begin
        //Marching-1
        #1200 numb = 127'b0;
        repeat(127*2) begin
            #1200 numb = {~numb[0], numb[126 : 1]};
        end

        //Other tests
        #1200 numb = 127'b101111100011111111111111000000;
        #1200 numb = 127'b101111100011111111111111100001100110011;
        #1200 numb = 127'b101111100011111111111111000001111111111;
        #1200 numb = 127'b101111100011111111111111000000111111111100000000000000111111111;
        #1200 $stop;
    end
endmodule
```

## Simulation Result

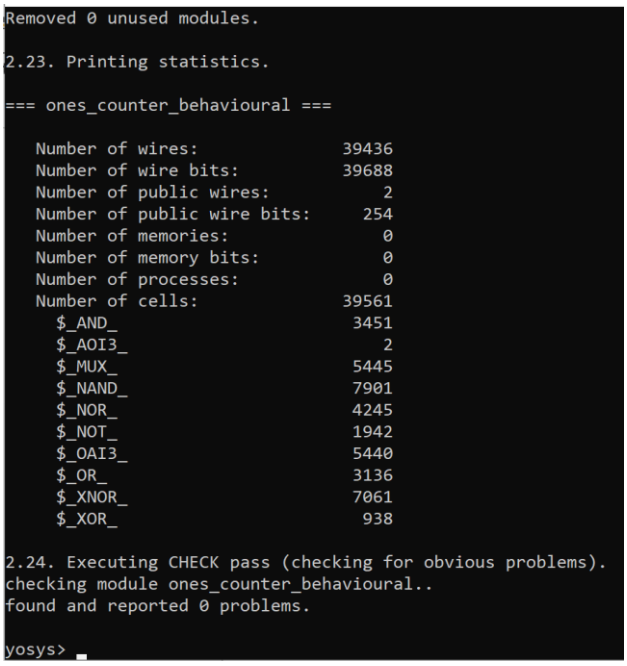



The output of problem 5 and 6 are the same.

# Problem 7

The synthesized file can be found in **OnesCounter2_Synth.v**

Here you can see the number of wires and cells needed for this synthesis :




## Problem 4 and 7 comparision

In problem 4, we have one 6-bit RCA, two 5-bit RCA, four 4-bit RCA, eight 3-bit RCA, sixteen 2-bit RCA and thirty-two 1-bit Adder.

Full-Adder circuit contains 5 gates and a n-bit RCA contains n FA pieces. So n-bit RCA contains **5n** logic gates.

**Number of gates** used in problem4 :

$$5*6 + 2*5*5 + 4*5*4 + 8 * 5 * 3 + 16 * 5 * 2 + 32 * 5 * 1 = 600$$

**We only used 600 logic gates, whereas problem 7 used 40000 logic gates!**