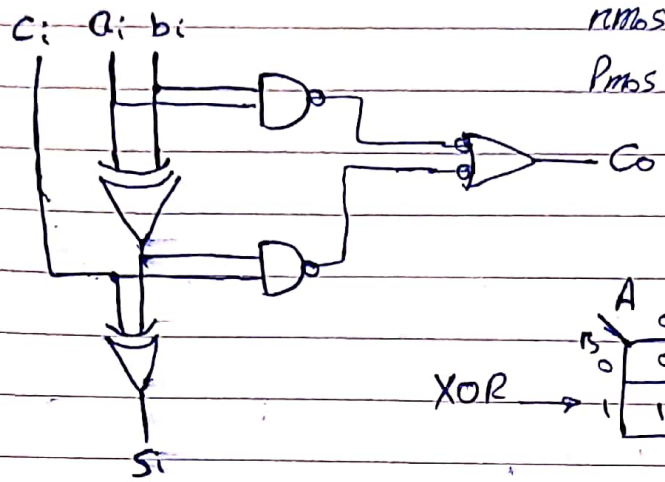
Digital Systems CA3		Mahdi Bohlool 87198365



nmos #(3, 4, 5)
Pmos #(5, 6, 7)

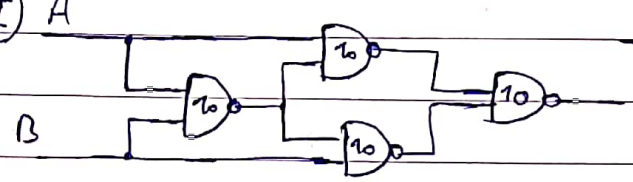
: ①

$W = (A \cdot \bar{B}) + (\bar{A} \cdot B)$

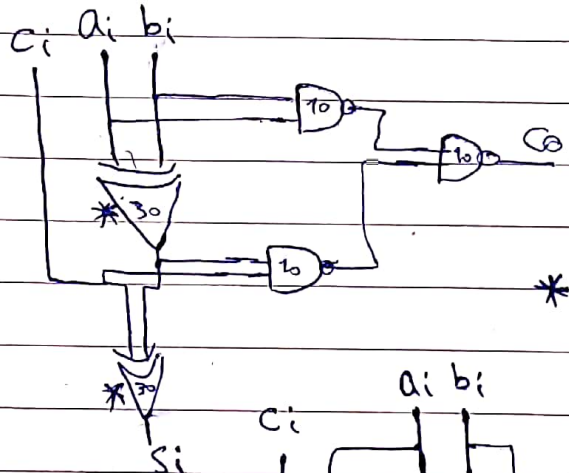
$= (A + B) \cdot (\bar{A} + \bar{B})$

$= (A + B) \cdot \overline{(A \cdot B)}$

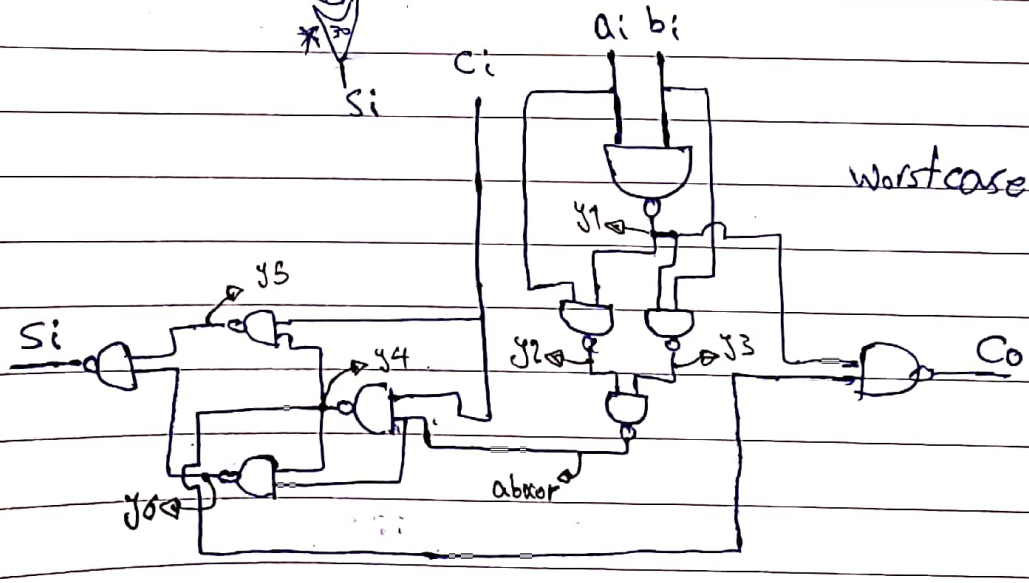
XOR → ① A



→ XOR gate delay: 30ns



*: Replace with ① diagram

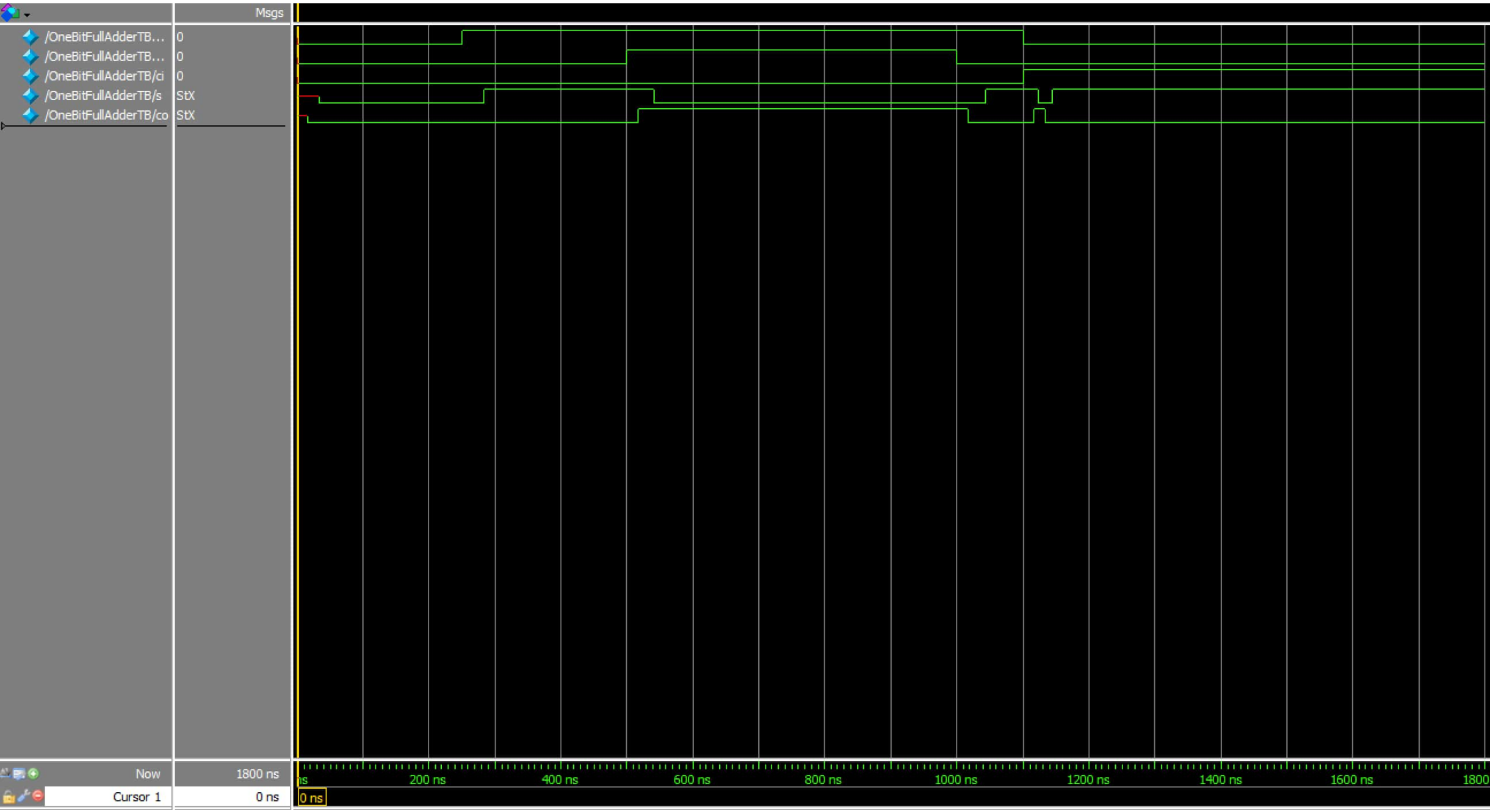


worst case delay: 33ns, 17ns

```
`include "OneBitFullAdder.v"
`timescale 1ns/1ns

module NBitAdder #(parameter N=32) (input [N-1:0] A,B, output [N-1:0] S,
Co);wire [N:0] co;
    assign co[0] = 0;
    genvar k;
    generate
        for (k = 0; k < N ; k = k + 1) begin
            OneBitFullAdder xx(A[k], B[k], co[k], S[k], co[k + 1]);
        end
    endgenerate
    assign Co = co[N];
endmodule
```

```
`include "OneBitFullAdder.v"
`timescale 1ns/1ns


module OneBitFullAdderTB();
    reg aa=0,bb=0, ci=0;
    wire s, co;
    OneBitFullAdder obfa(aa, bb, ci, s, co);
    initial begin
        #100
        #150 aa=1;
        #250 bb=1;
        #400
        repeat(3) #100 {aa, bb, ci} = $random;
        #600 $stop;
    end
endmodule
```

```
`timescale 1ns/1ns


module OneBitFullAdder (input a, b, ci, output s, co);
    wire y1, y2, y3, abXor, y4, y5, y6;
    nand #(10, 7) ab_nand(y1, a, b);
    nand #(10, 7) aab_nand(y2, a, y1), bab_nand(y3, b, y1);
    nand #(10, 7) ab_xor(abXor, y3, y2);
    nand #(10, 7) abxorci_nand(y4, abXor, ci), ciabxorci_nand(y5, y4, ci),
abxorabxorci_nand(y6, y4, abXor);
    nand #(10, 7) sum(s, y6, y5);
    nand #(10, 7) carry_out(co, y1, y4);
endmodule
```

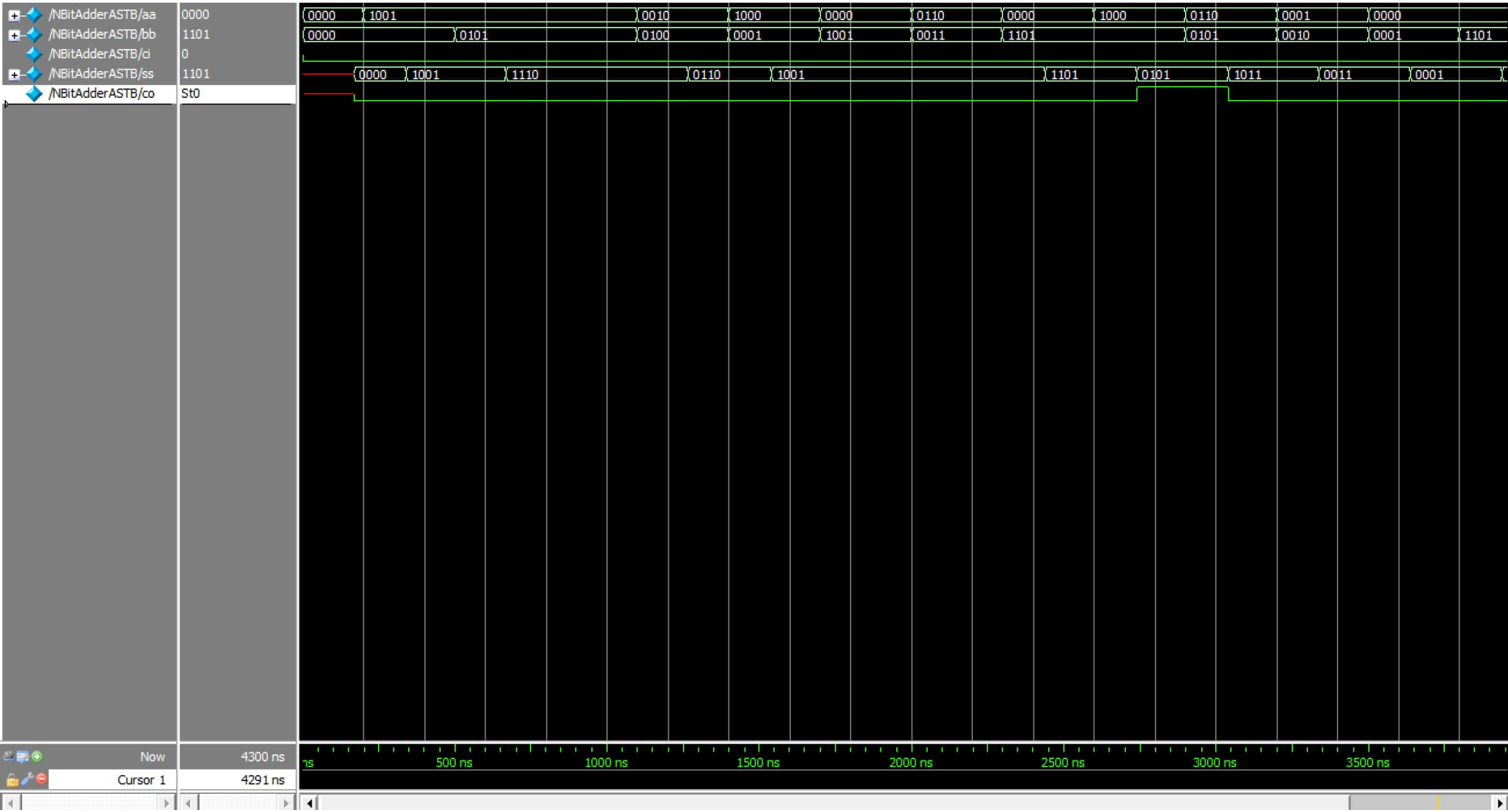


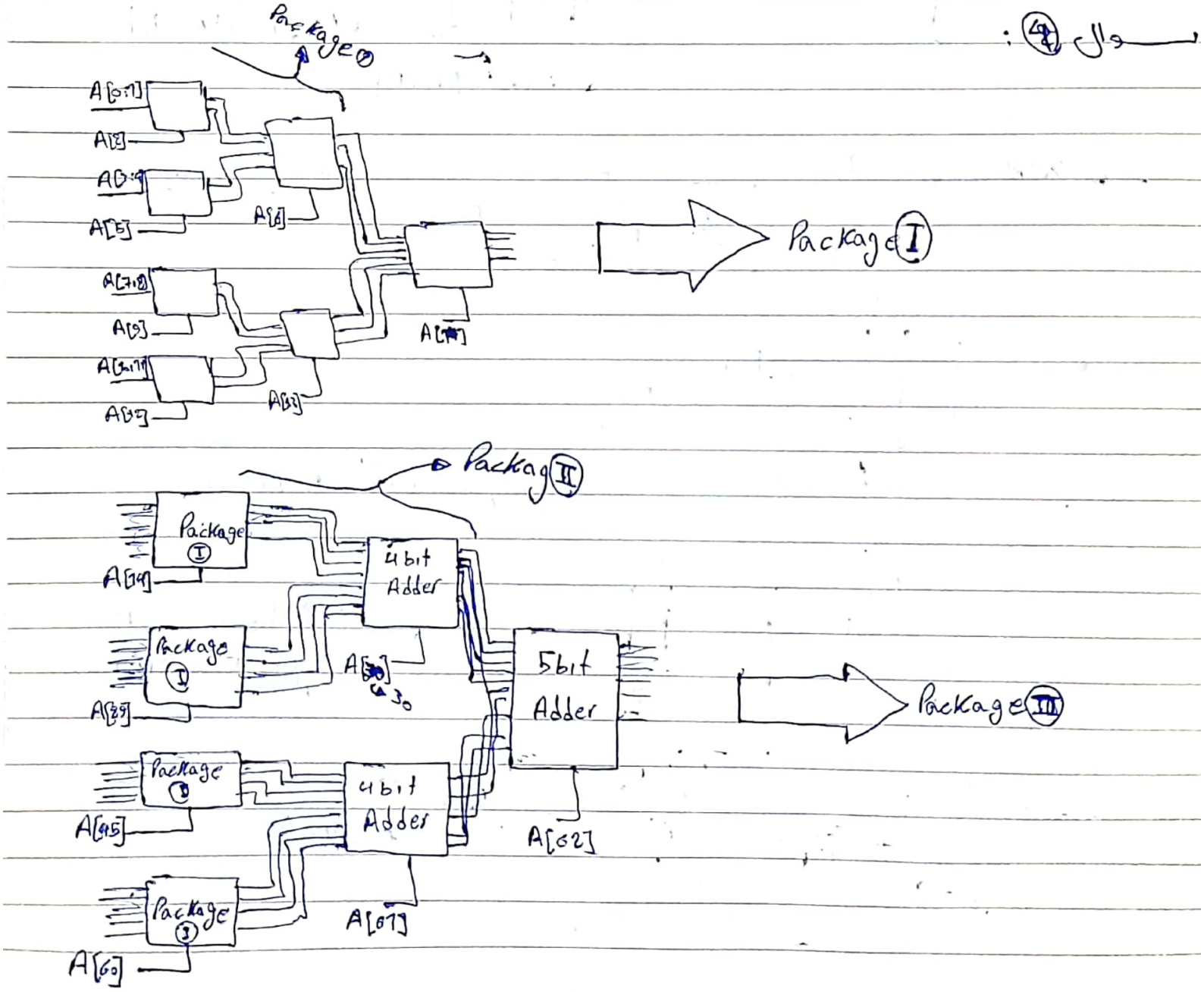
N Bit Adder Assign Statement

```
`timescale 1ns/1ns


module NBitAdderAS #(parameter N = 32) (input [N - 1: 0] A, B, input Ci, output [N - 1:  0] S, output
Co);
    assign #(N * 35, N * 42) {Co, S} = A + B + Ci;
endmodule
```

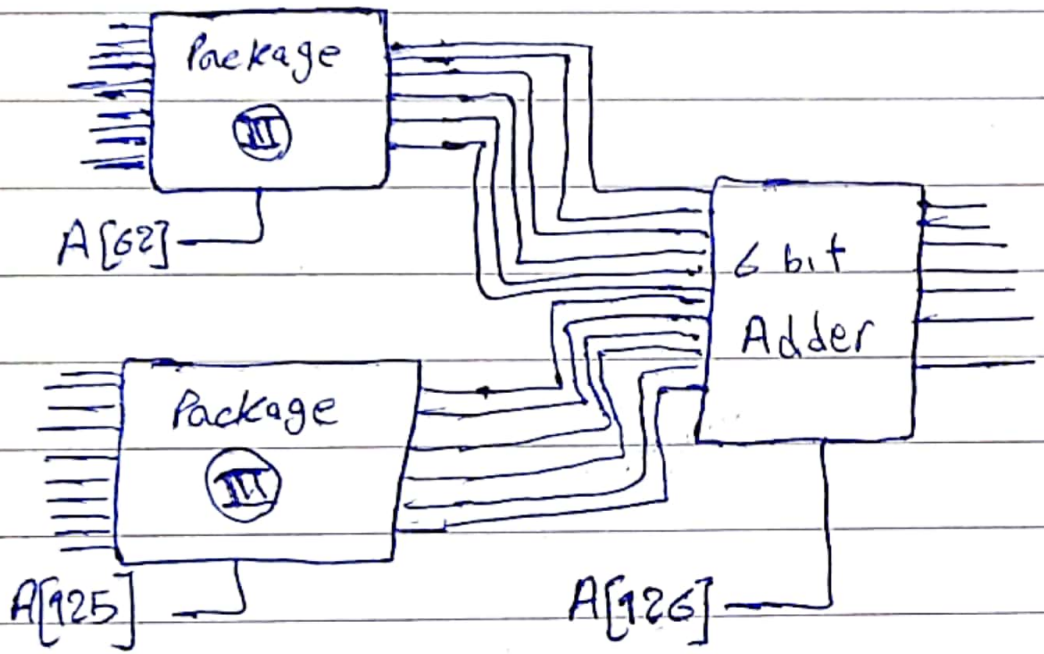
N Bit Adder Assign Statement TB

```
`include "NBitAdderAS.v"

`timescale 1ns/1ns


module NBitAdderASTB( );
    reg [3:0] aa, bb;
    assign aa = 4'b0;
    assign bb = 4'b0;
    reg ci = 0;
    wire [3:0] ss;
    wire co;
    NBitAdderAS #(4) n_bit_adder(aa, bb, ci, ss, co);
    initial begin
      #100
      #100 aa=4'b1001;
      #300 bb=4'b0101;
      #300
      repeat(10) #300 {aa, bb}  = $random;
      #500 $stop;
    end
endmodule
```





السؤال (4):

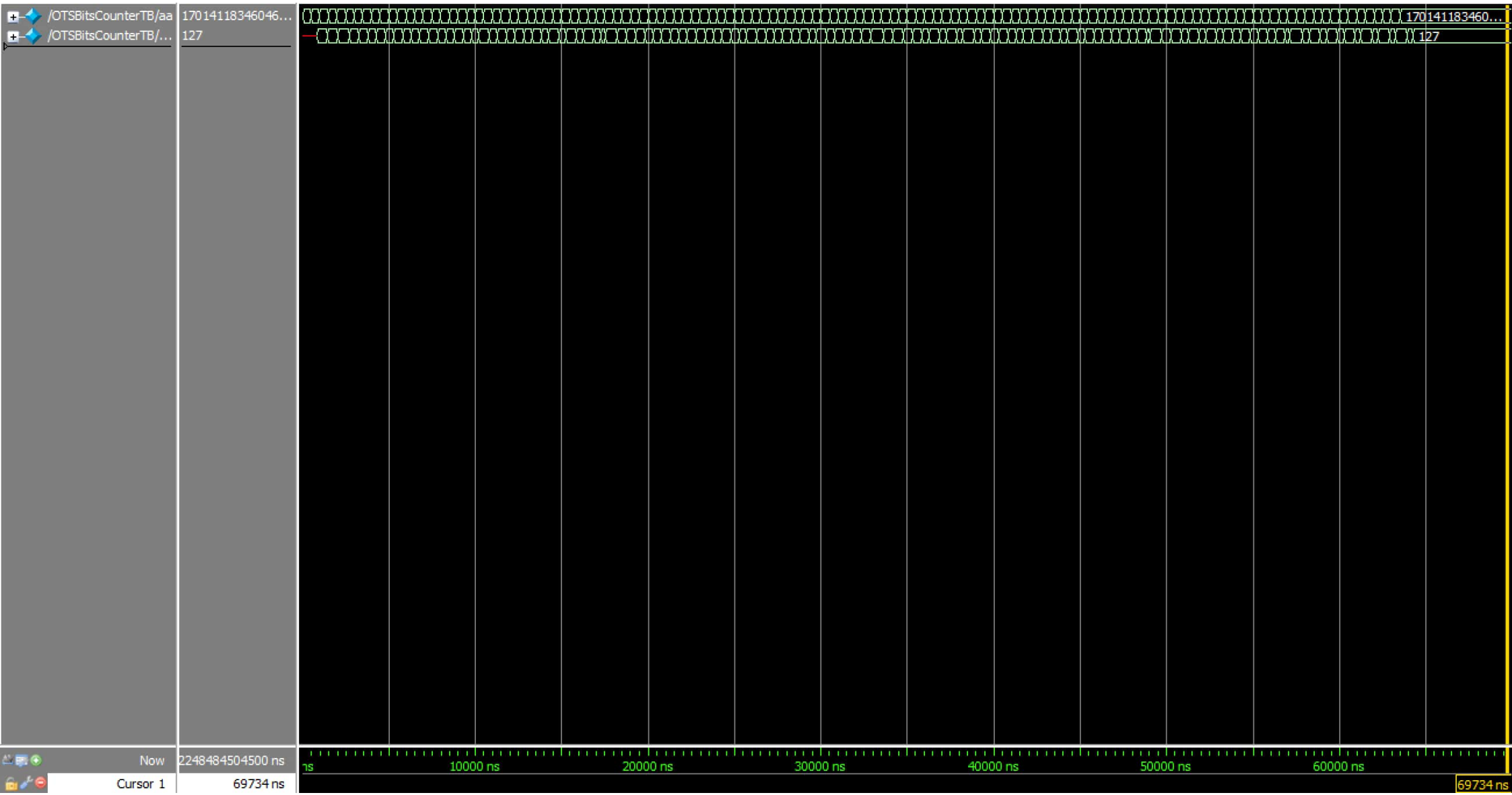


127-Bit One's Counter

```
`include "NBitAdderAS.v"
`timescale 1ns/1ns



module PackageZero (input [6: 0]A, output [2:0] C);
    wire [1:0] fa_one_res, fa_two_res;
    NBitAdderAS #(1) fa_one(A[0], A[1], A[2], fa_one_res[0], fa_one_res[1]);
    NBitAdderAS #(1) fa_two(A[3], A[4], A[5], fa_two_res[0], fa_two_res[1]);
    NBitAdderAS #(2) adder_pz(fa_one_res, fa_two_res, A[6], C[1:0], C[2]);
endmodule


module PackageOne (input [14:0]A, output [3:0]C);
    wire [5:0] adder_pi_res;
    genvar k;
    generate
        for (k = 0; k <= 1; k = k + 1) begin
            PackageZero xx(A[7 * k + 6: 7 * k], adder_pi_res[3 * k + 2: 3 * k]);
        end
    endgenerate
    NBitAdderAS #(3) adder_pi(adder_pi_res[5:3], adder_pi_res[2:0], A[14], C[2:0], C[3]);
endmodule


module PackageTwo(input [30:0]A, output [4:0]C);
    wire [7:0] adder_pt_res;
    genvar k;
    generate
        for (k = 0; k <= 1; k = k + 1) begin
            PackageOne xx(A[15 * k + 14:15 * k], adder_pt_res[4 * k + 3: 4 * k]);
        end
    endgenerate
    NBitAdderAS #(4) adder_pt(adder_pt_res[3:0], adder_pt_res[7:4], A[30], C[3:0], C[4]);
endmodule

module PackageThree (input [62:0]A, output [5:0]C);
    wire [9:0] adder_pthree_res;
    genvar k;
    generate
        for (k = 0; k <= 1; k = k + 1) begin
            PackageTwo xx(A[31 * k + 30: 31 * k], adder_pthree_res[5 * k + 4: 5 * k]);
        end
    endgenerate
  NBitAdderAS #(5) adder_pthree(adder_pthree_res[4:0], adder_pthree_res[9:5], A[62], C[4:0],
                                                                                C[5]);
endmodule

module OTSBitsCounter(input [126:0] A, output [6:0]C);
    wire [11:0] pthree_res;
    genvar k;
    generate
        for (k = 0; k <= 1; k = k + 1) begin
            PackageThree xx(A[63 * k + 62: 63 * k], pthree_res[6 * k + 5: 6 * k]);
        end
    endgenerate
    NBitAdderAS #(6) adder_pf(pthree_res[5:0], pthree_res[11:6], A[126], C[5:0], C[6]);
endmodule
```

127-Bit One's Counter TB

```
`include "OTSBitsCounter.v"
`timescale 1ns/1ns


module OTSBitsCounterTB();
    reg [126:0] aa = 127'b0;
    wire [6:0] ww;
    OTSBitsCounter bits_counter(aa, ww);
    always #500 aa = {1'b1, aa[126:1]};
endmodule
```



127-Bit One's Counter Alway Statement

```
`timescale 1ns/1ns


module OTSBitsCounterAlwaysST (input [126:0]A, output reg [6:0]
C); integer i;
    always @(*) begin
        C = 7'b0000000;
        for (i = 0; i < 127; i = i+1) begin
            if(A[i] == 1'b1) C = #350 C + 1;
        end
    end
endmodule
```

سوال (7):

The circuit drawn in question 4 was very different from the circuit formed in question six by Always statement. because we had synthesized all parts of it in a top_down design.

Top_down design: NAND: 21 NOR: 40 NOT: 16

Always statement. NAND: 1679 NOR: 1622 NOT: 364

```
4.6.2. Re-integrating ABC results.
ABC RESULTS:                NAND cells:       21
ABC RESULTS:                 NOR cells:       40
ABC RESULTS:                 NOT cells:       19
ABC RESULTS:            internal signals:     23
ABC RESULTS:               input signals:     13
ABC RESULTS:              output signals:      7
Removing temp directory.
```

```
4.1.2. Re-integrating ABC results.
ABC RESULTS:                NAND cells:     1679
ABC RESULTS:                 NOR cells:     1622
ABC RESULTS:                 NOT cells:      364
ABC RESULTS:            internal signals:    2465
ABC RESULTS:              input signals:      127
ABC RESULTS:             output signals:        7
Removing temp directory.
```