Laboratory Report # 3

Name: Christian Jay. Gallardo **Date Completed:** 9/5/2025

Laboratory Exercise Title: Structural Modeling of Combinational Circuits

Target Course Outcomes:

CO1: Create descriptions of digital hardware components such as in combinational and sequential circuits using synthesizable Verilog HDL constructs.

CO2: Verify the functionality of HDL-based components through design verification tools.

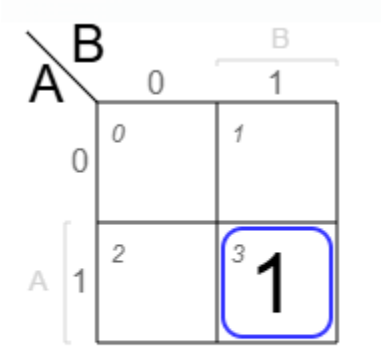
Exercise 3A:

In Exercise 3A, the main task was to design and implement a 2-to-4 Decoder using Verilog HDL. The activity involved writing the Verilog code for the decoder, compiling it to check for errors, and then running simulations to verify its functionality. The design was tested through waveform outputs and logical validation to ensure that the decoder produced the correct outputs based on the given inputs and enable signal. This allowed for both the design and behavior of the circuit to be thoroughly demonstrated and confirmed.

INPUTS			OUTPUTS			
E	I1	I0	O3	O2	O1	O0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Table 1: 2-to-4 Decoder Truth Table

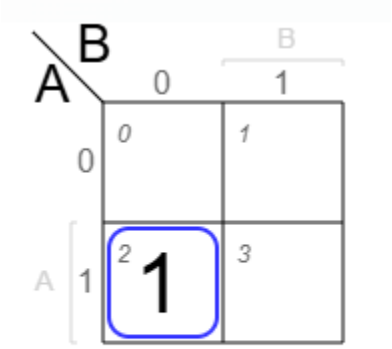
D3 KMAP



F = AB

D3 = E · A1 · A0

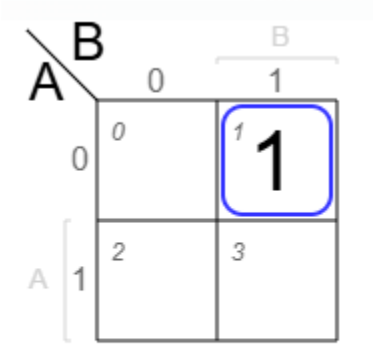
D2 KMAP



F = AB'

D2 = E · A1 · A0’

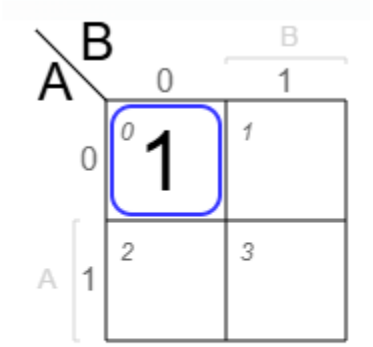
D1 KMAP



F = A'B

D1 = E · A1’ · A0

D0 KMAP



F = A'B'

D0 = E · A1’ · A0’

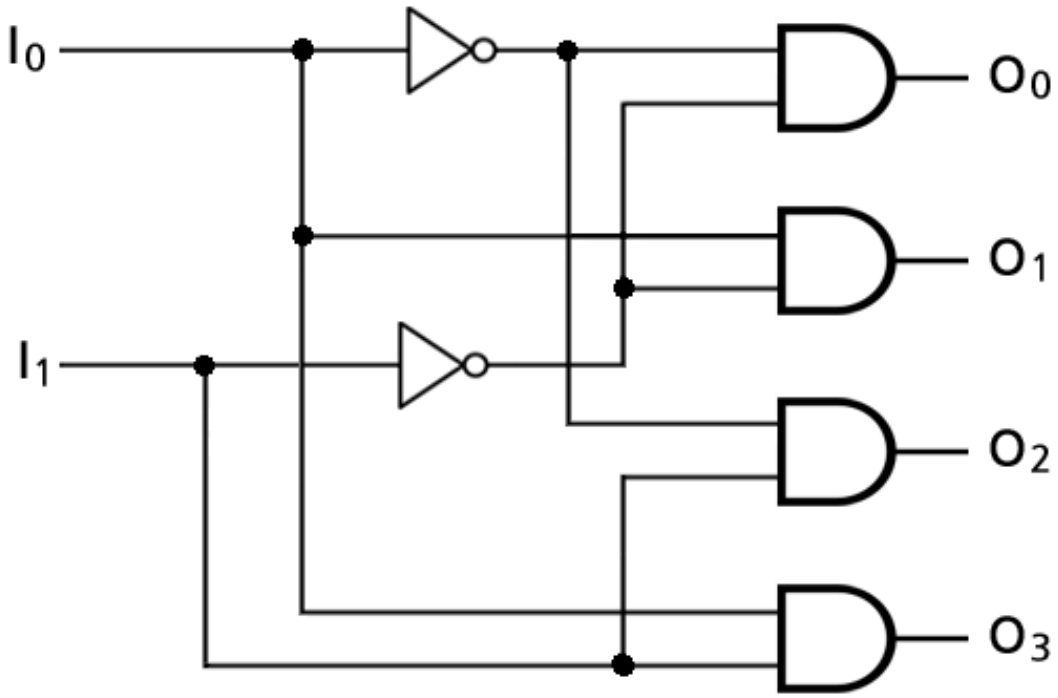


Figure 1. Logic Diagram for the 2-to-4 decoder

Minterm Equations

$$D_0 = \overline{A_1} \cdot \overline{A_0}$$
$$D_1 = \overline{A_1} \cdot A_0$$
$$D_2 = A_1 \cdot \overline{A_0}$$
$$D_3 = A_1 \cdot A_0$$

Figure 2. Minterm Equations for 2-to-4 decoder

```
// Filename:          Decoder_2x4.v
// Author             Christian Jay Y. Gallardo
// Class:             CPE3101L
// Grp. / Schudele:   GRP 4. Friday 10:30AM - 1:30PM
// Description:       Verilog design entry/


// 2-to-4 Decoder with Enable


module Decoder_2x4 (
    input wire E,          // Enable
    input wire [1:0] I,    // Inputs I1 I0
    output wire [3:0] O    // Outputs O0-O3
);

    wire I0_not, I1_not;

    // inverters or not gate
    not (I0_not, I[0]);
    not (I1_not, I[1]);

    // and gates with ENABLE
    and (O[0], E, I1_not, I0_not);
    and (O[1], E, I1_not, I[0]);
    and (O[2], E, I[1], I0_not);
    and (O[3], E, I[1], I[0]);

endmodule
```

Figure 3. Verilog Design entry

```
// Filename:          tb_Decoder_2x4.v
// Author             Christian Jay Y. Gallardo
// Class:             CPE3101L
// Grp. / Schudele:   GRP 4. Friday 10:30AM - 1:30PM
// Description:       Verilog design entry for testbench.


`timescale 1ns / 1ps

module tb_Decoder_2x4;

   reg E;
   reg [1:0] I;
   wire [3:0] O;

   // instantiate decoder
   Decoder_2x4 uut (
      .E(E),
      .I(I),
      .O(O)
   );

   //display during simulation
   initial begin
      $monitor("Time = %0t | E = %b | I=%b | O=%b", $time, E, I , O);

      //TESTING!!

      E = 0; I = 2'b00; #10;
      E = 0; I = 2'b01; #10;
      E = 0; I = 2'b10; #10;
      E = 0; I = 2'b11; #10;

      E = 1; I = 2'b00; #10;
      E = 1; I = 2'b01; #10;
      E = 1; I = 2'b10; #10;
      E = 1; I = 2'b11; #10;

      $finish;
   end
endmodule
```

Figure 4. Verilog Design entry for testbench.

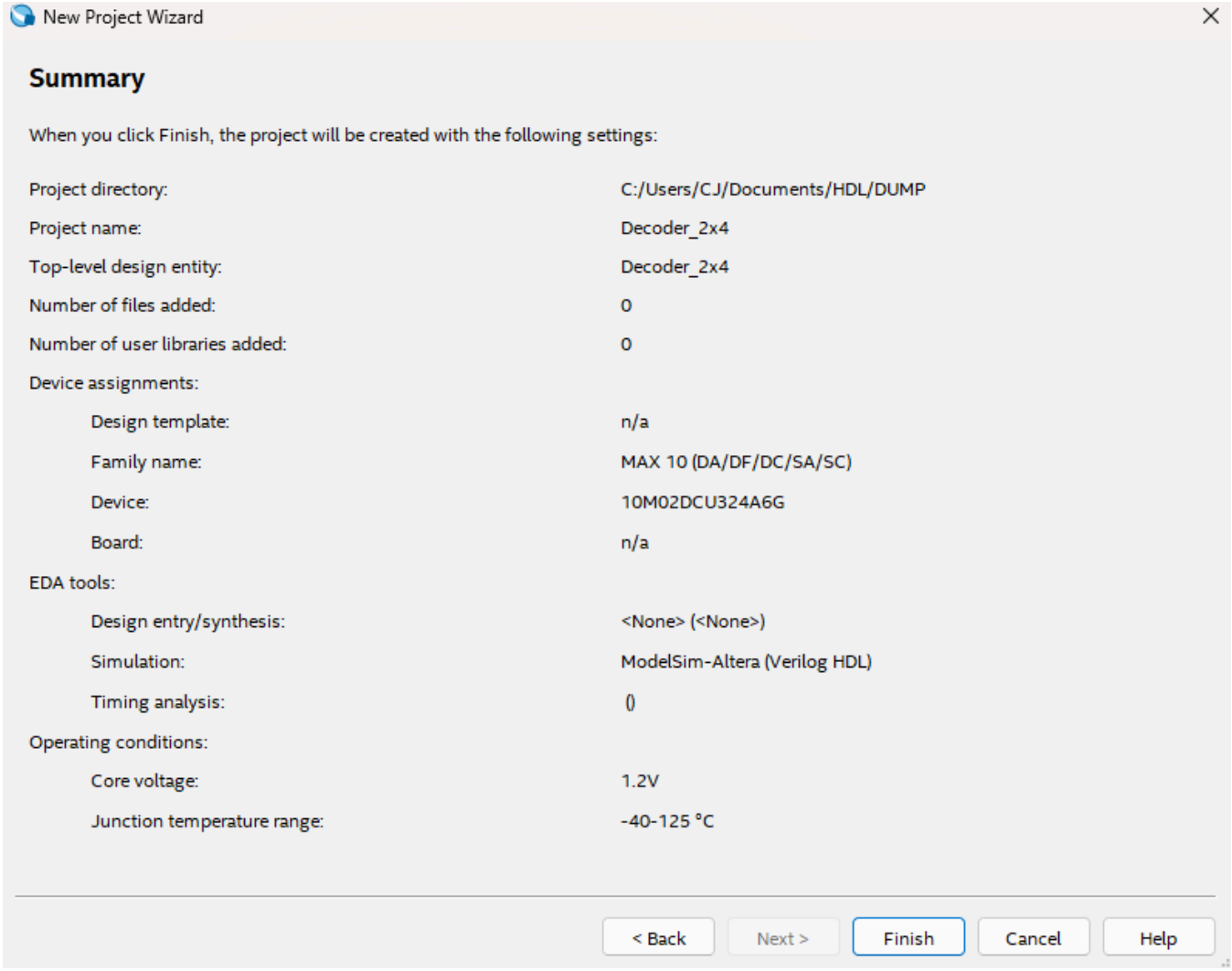


Figure 5. Compilation report for the flow of summary

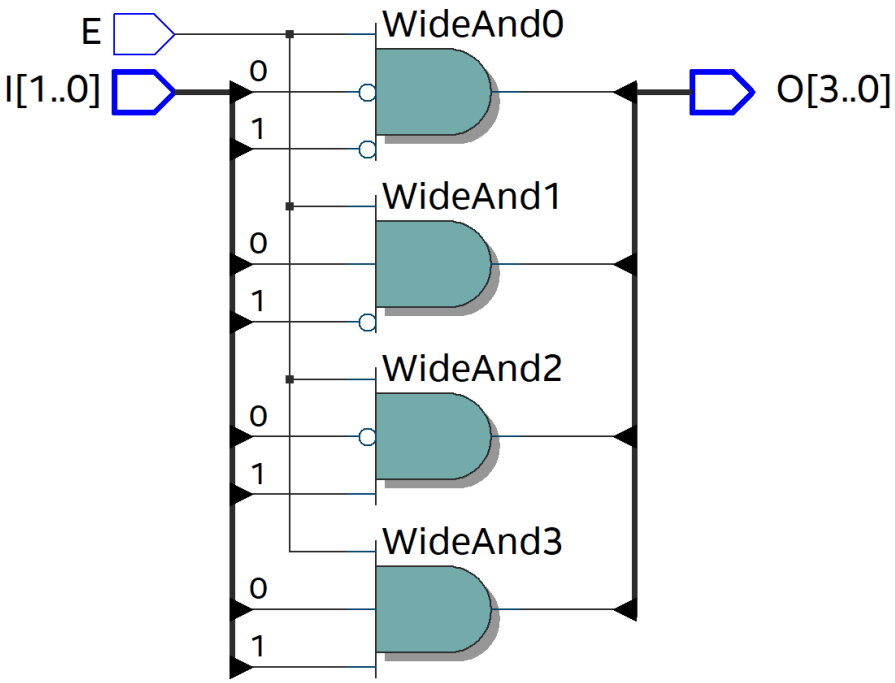


Figure 6. Schematic Diagram for the synthesized circuit using RTL Viewer

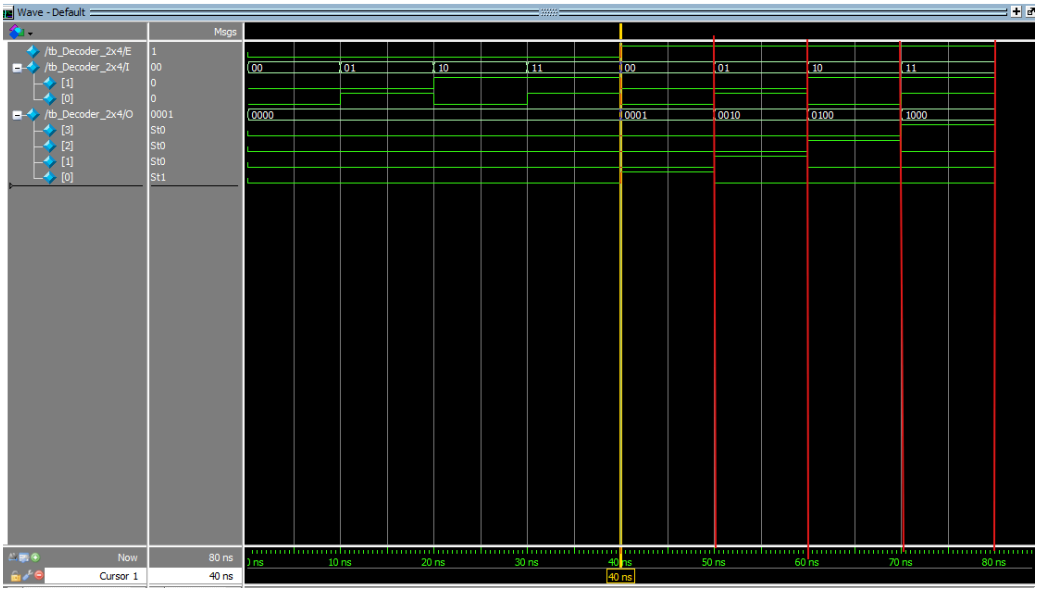


Figure 7. Proof of successful simulation result

The simulation is successful because the outputs match the expected truth table of a 2-to-4 decoder. For each input combination (`00`, `01`, `10`, `11`) with enable active, only one output line is high (`0001`, `0010`, `0100`, `1000`), confirming the design works correctly.

Exercise 3B

In Exercise 3B, a 4-bit adder was designed in Verilog HDL using the full adder code from Exercise 2B. The circuit was built through gate-level primitives and structural modeling, where four full adders were connected in series to form a ripple-carry adder. A separate testbench was also created to verify its operation using eight test cases, demonstrating correct functionality in performing binary addition.

```
1  // Filename:         Adder4Bit.v
2  // Author           Christian Jay Y. Gallardo
3  // Class:           CPE3101L
4  // Grp. / Schudele: GRP 4. Friday 10:30AM - 1:30PM
5  // Description:     Verilog design entry
6
7
8  `timescale 1ns / 1ps
9  `default_nettype none
10
11 module Adder4Bit(
12     input  wire [3:0] A,
13     input  wire [3:0] B,
14     input  wire       Cin,
15     output wire [3:0] S,
16     output wire       Cout
17 );
18     wire c1, c2, c3; // internal ripple carries
19
20     // Bit 0 (LSB)
21     FullAdder FA0(
22         .a(A[0]), .b(B[0]), .cin(Cin),
23         .s(S[0]), .cout(c1)
24     );
25
26     // Bit 1
27     FullAdder FA1(
28         .a(A[1]), .b(B[1]), .cin(c1),
29         .s(S[1]), .cout(c2)
30     );
31
32     // Bit 2
33     FullAdder FA2(
34         .a(A[2]), .b(B[2]), .cin(c2),
35         .s(S[2]), .cout(c3)
36     );
37
38     // Bit 3 (MSB)
39     FullAdder FA3(
40         .a(A[3]), .b(B[3]), .cin(c3),
41         .s(S[3]), .cout(Cout)
42     );
43 endmodule
44
45 `default_nettype wire
46
```

Figure 8. Verilog design for 4-bit adder module

```
1  // Filename:         tb_Adder4Bit.v
2  // Author           Christian Jay Y. Gallardo
3  // Class:           CPE3101L
4  // Grp. / Schudele: GRP 4. Friday 10:30AM - 1:30PM
5  // Description:     Verilog design entry for testbench
6
7
8
9  `timescale 1ns / 1ps
10
11 module tb_Adder4Bit;
12
13     reg  [3:0] A, B;
14     reg        Cin;
15     wire [3:0] S;
16     wire       Cout;
17
18     // Instantiate the 4-bit adder
19     Adder4 uut (
20         .A(A),
21         .B(B),
22         .Cin(Cin),
23         .S(S),
24         .Cout(Cout)
25     );
26
27     initial begin
28         // Display header
29         $monitor("Time=%0t | A=%b B=%b Cin=%b || Cout=%b S=%b",
30                  $time, A, B, Cin, Cout, S);
31
32         // Test cases
33         A = 4'b0000; B = 4'b0000; Cin = 0; #10;
34         A = 4'b0001; B = 4'b0010; Cin = 0; #10;
35         A = 4'b0101; B = 4'b0011; Cin = 0; #10;
36         A = 4'b0101; B = 4'b0011; Cin = 1; #10;
37         A = 4'b1111; B = 4'b0001; Cin = 0; #10;
38         A = 4'b1111; B = 4'b1111; Cin = 1; #10;
39
40         $finish;  // End simulation
41     end
42
43 endmodule
44
```

Figure 9. Verilog design for testbench of 4-bit adder module

New Project Wizard

Summary

When you click Finish, the project will be created with the following settings:

Project directory:	C:/Users/CJ/Documents/HDL/Adder4Bit
Project name:	Adder4Bit
Top-level design entity:	Adder4Bit
Number of files added:	0
Number of user libraries added:	0
Device assignments:	
Design template:	n/a
Family name:	MAX 10 (DA/DF/DC/SA/SC)
Device:	10M02DCU324A6G
Board:	n/a
EDA tools:	
Design entry/synthesis:	<None> (<None>)
Simulation:	ModelSim-Altera (Verilog HDL)
Timing analysis:	()
Operating conditions:	
Core voltage:	1.2V
Junction temperature range:	-40-125 °C

Figure 10. Compilation Report for the Flow Summary

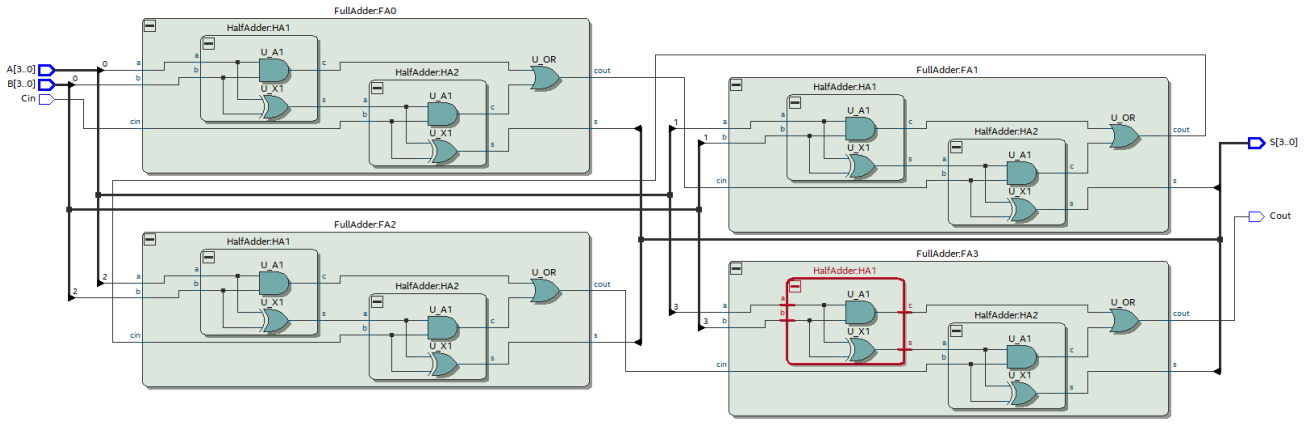


Figure 11. Schematic diagram of the synthesized circuit using RTL viewer

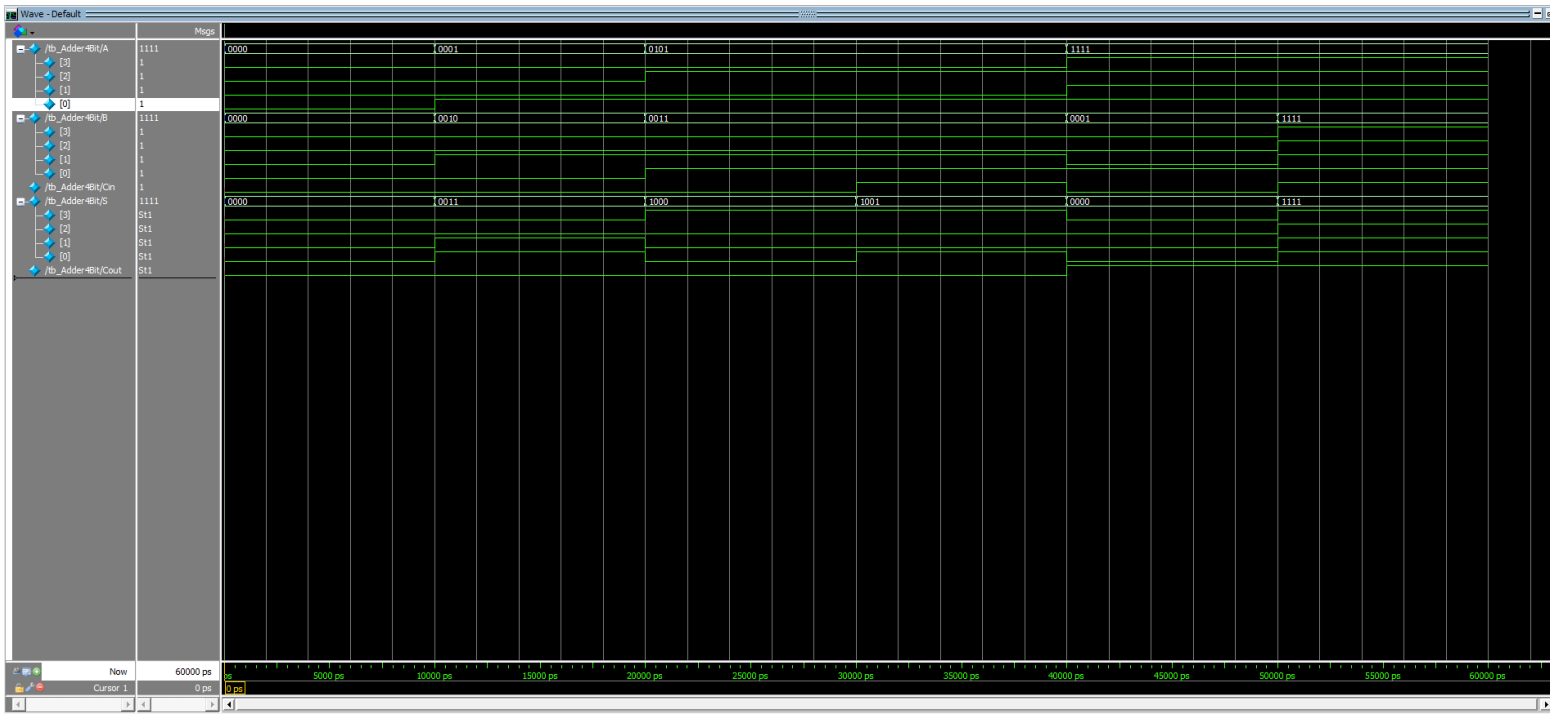


Figure 12. Proof of successful simulation result

The 4-bit adder designed using Verilog HDL works as expected, accurately performing binary addition. When the carry-in (`C_in`) is set to 0, the circuit executes standard addition of the two 4-bit inputs `A` and `B`. If `C_in` is set to 1, the adder includes the carry in the calculation, effectively adding one more to the result.

In the simulation waveform, at around **20 ns**, the inputs are `A = 1111` (15 in decimal) and `B = 0001` (1 in decimal) with `C_in = 0`. The output becomes `S = 0000` with a carry-out `C_out = 1`. This correctly represents the sum **15 + 1 = 16**, where the 4-bit sum rolls over to zero and the extra bit is carried out. This behavior confirms that the ripple-carry adder functions correctly in handling both normal sums and overflow cases.