Report for CDA 3201L Lab #4
Combinational Logic Design and Verilog HDL

Name: Tuan Khang Phan **UID:** U30382645
Name: Huu Phat Nguyen **UID:** U46082380

I. Introduction

In this experiment, a full adder circuit was designed and tested using both physical hardware and Verilog HDL. The full adder adds three 1-bit inputs (a, b, c_{in}) and produces two outputs (sum and c_{out}). First, the truth table and Boolean expressions were derived using Karnaugh maps, then optimized using XOR identities. The optimized circuit was built on a breadboard using XOR and NAND gates, and the same design was implemented in structural Verilog and simulated on EDAPlayground to verify that both implementations produce identical results.

II. Methods and Materials

1. Materials

- **Breadboard:** A prototyping board used to construct circuits without soldering. Components and wires are inserted into the holes, which are internally connected in rows, allowing quick assembly and testing.
- **Integrated Circuits (ICs):** The SN74LS86N IC (Quad 2-input XOR gate) was used to implement the XOR gates required for the *sum* output. The SN74LS00N IC (Quad 2-input NAND gate) was used to implement the NAND gates required for the carry output. The datasheets for both ICs were consulted to determine pin configurations and power supply connections.
- **Wiring:** A wire jumper kit with wires of different lengths was used to connect components on the breadboard. Alligator clips were used to connect the power supply to the breadboard rails.
- **LEDs and Resistors:** LEDs were used to indicate the logic state of the circuit outputs. When the output is logic 1, the LED emits light; when the output is logic 0, the LED remains off. A 470 Ω resistor was connected in series with each LED to limit current and prevent damage.
- **Power Supply:** A 5 V DC power supply was used to power the ICs and provide logic HIGH and LOW levels for the inputs.
- **EDA Playground:** An online Verilog simulation tool used to write, run, and test Verilog code. The Synopsys VCS simulator was selected, and the EPWave waveform viewer was used to display timing diagrams.

2. Methods

1. The truth table for the full adder was created using the three inputs (a, b, c_{in}) and two outputs (sum, c_{out}).
2. K-maps were used to simplify the Boolean expressions for sum and c_{out}.
3. The *sum* output was implemented using XOR gates, and the c_{out} output was implemented using NAND gates.
4. The circuit was optimized to reuse the intermediate $a \oplus b$ signal, reducing the total gate count

to 2 XOR gates and 3 NAND gates.

5. The circuit was constructed on the breadboard using the 74LS86 and 74LS00 ICs, with LEDs connected to the outputs to verify functionality.
6. The same optimized circuit was implemented in Verilog using structural modeling on EDA Playground.
7. A testbench was written to exercise all eight input combinations and simulate the circuit.
8. The console output and waveform were observed to confirm that the Verilog simulation matched the hardware results.

III. Lab Assignment Questions

The handwritten solutions for Questions 1, 2, and 3 are included below.

1

a	b	c_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

C_{out}: MSOP: $\Sigma m(3,5,6,7)$

$ab \backslash c_{in}$	0	1
00	0	0
01	0	1
11	1	1
10	0	1

$\Rightarrow ab + bc_{in} + ac_{in}$

Sum: MSOP: $\Sigma m(1,2,4,7)$

$ab \backslash c_{in}$	0	1
00	0	1
01	1	0
11	0	1
10	1	0

$\Rightarrow \bar a b \bar c_{in} + a \bar b \bar c_{in} + \bar a \bar b c_{in} + abc_{in}$

2 Sum using XOR

Use 2 XOR: $x = a \oplus b = \bar a b + a \bar b$

$\text{sum} = x + c_{in} = a \oplus b \oplus c_{in}$

$$\text{Sum} = \bar a \bar b c_{in} + \bar a b \bar c_{in} + abc_{in} + a\bar b \bar c_{in}$$
$$= c_{in}(\bar a \bar b + ab) + \bar c_{in}(\bar a b + a \bar b)$$
$$= c_{in}(\overline{a \oplus b}) + \bar c_{in}(a \oplus b)$$
$$= a \oplus b \oplus c_{in}$$

C_{out} implemented NAND - NAND

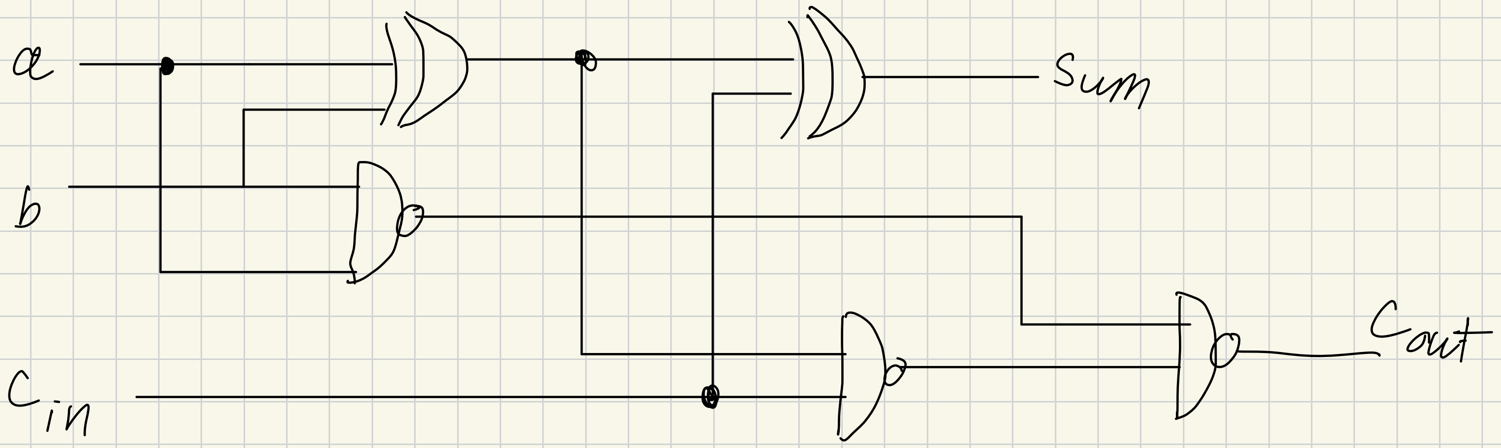
$$C_{out} = ab + bc_{in} + ac_{in}$$
$$= \overline{(\overline{ab})(\overline{bc_{in}})(\overline{ac_{in}})}$$

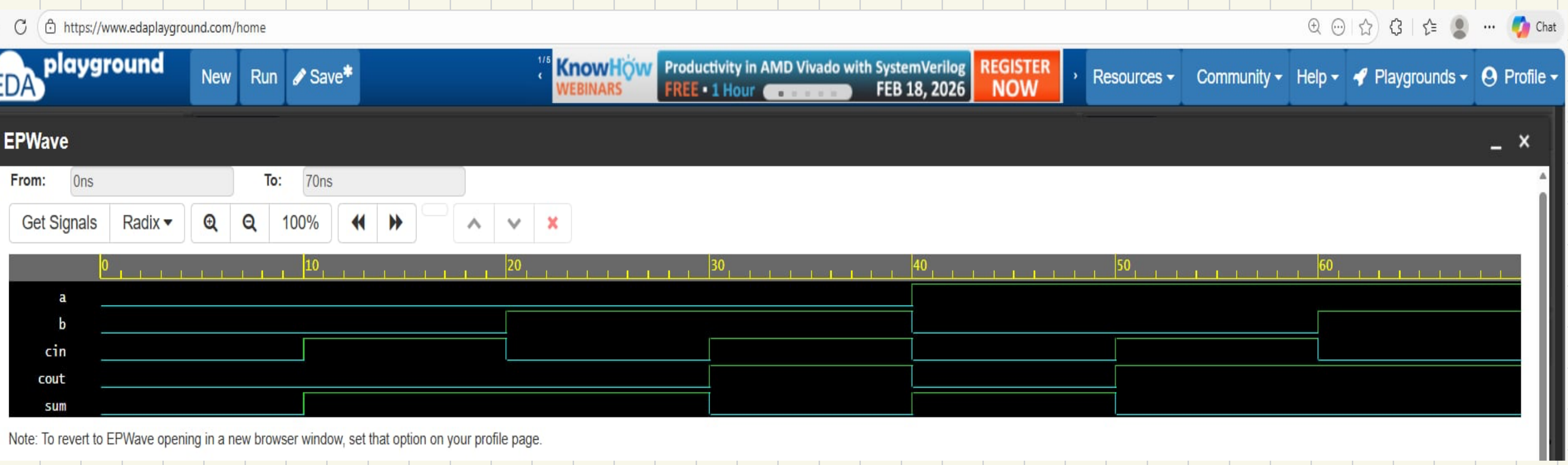
③ for a full adder, $C_{out} = 1$ when at least 2 of 3 inputs (a, b, c_{in}) are 1

$\Rightarrow (a, b, c_{in}) = (0, 1, 1)$
$(a, b, c_{in}) = (1, 0, 1)$
$(a, b, c_{in}) = (1, 1, 0)$
$(a, b, c_{in}) = (1, 1, 1)$

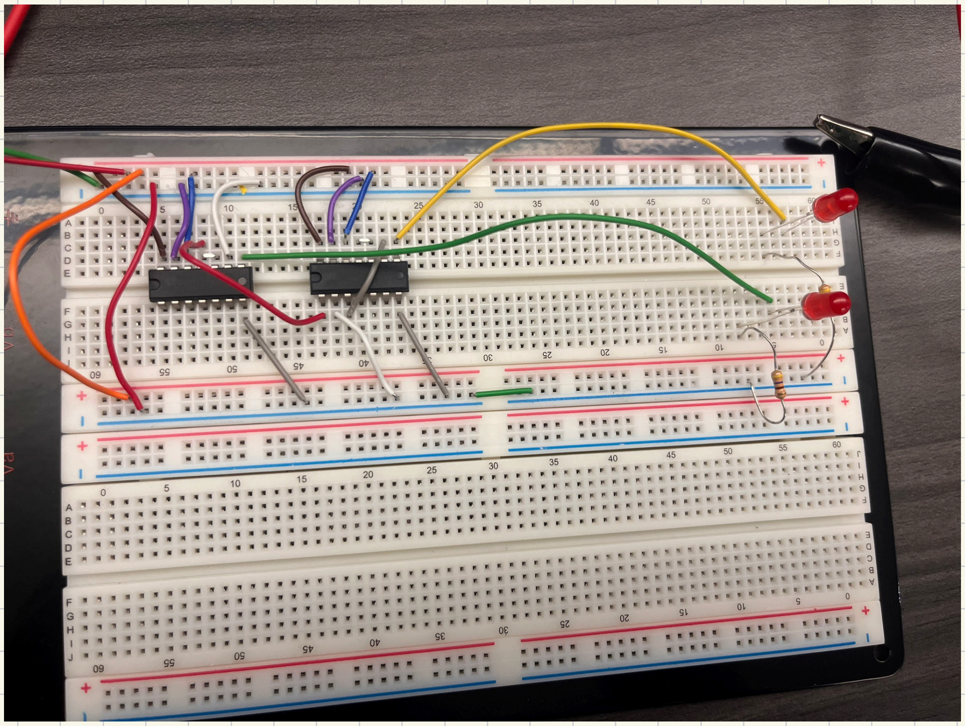
$$\Rightarrow C_{out} = ab + C_{in}(a \oplus b)$$
$$= ab + C_{in}(\bar{a}b + a\bar{b})$$
$$= \overline{(\overline{ab})(\overline{a \oplus b\, C_{in}})}$$

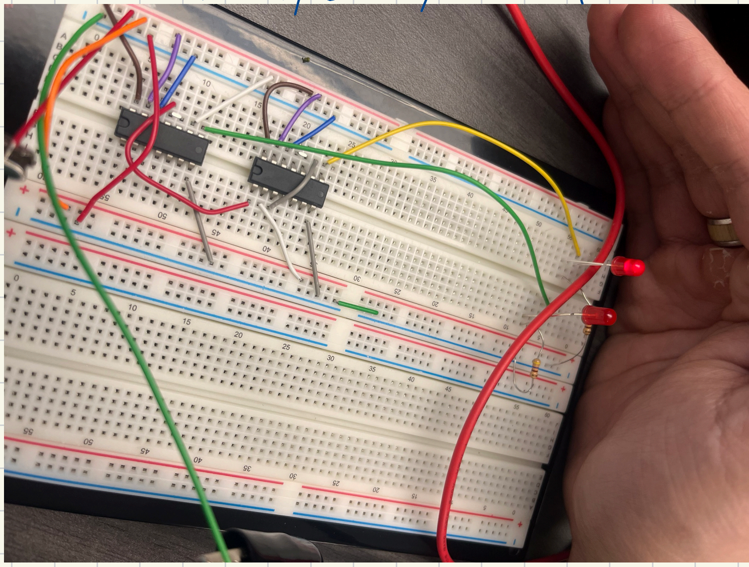
$$Sum = a \oplus b \oplus C_{in}$$

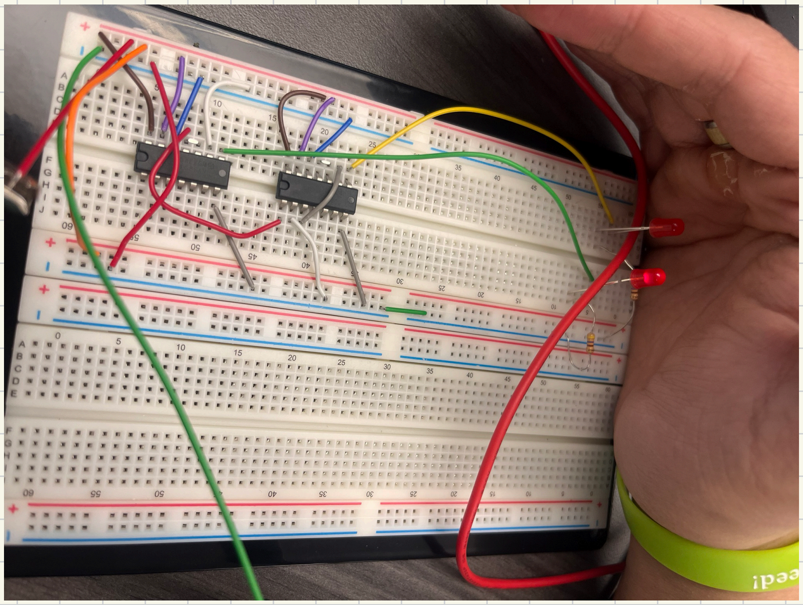




Note: To revert to EPWave opening in a new browser window, set that option on your profile page.







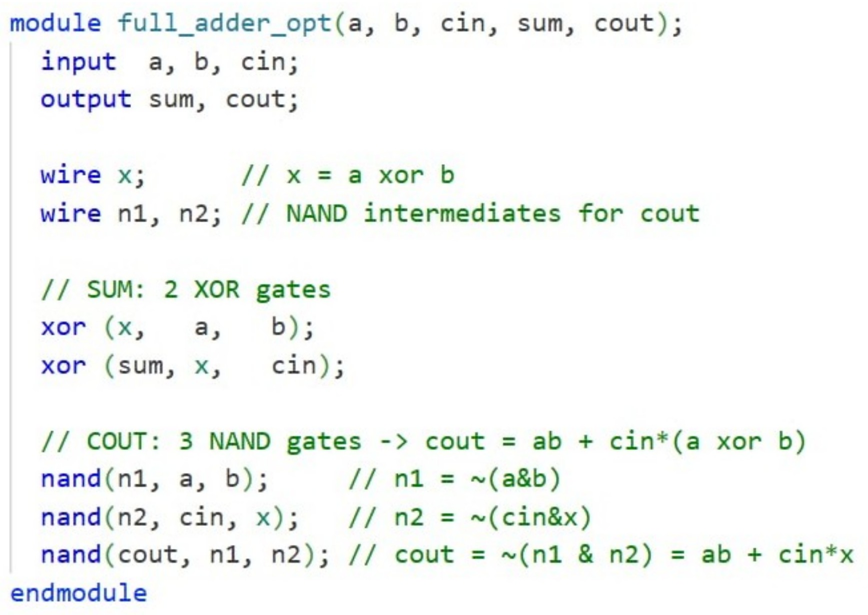
$a = 1,\ b = 0,\ c_{in} = 1$

$a = 1,\ b = 0,\ c_{in} = 0$

IV. Experimental Results

1. Verilog Module

The structural Verilog module below implements the optimized full adder using the gate-level primitives `xor` and `nand`:

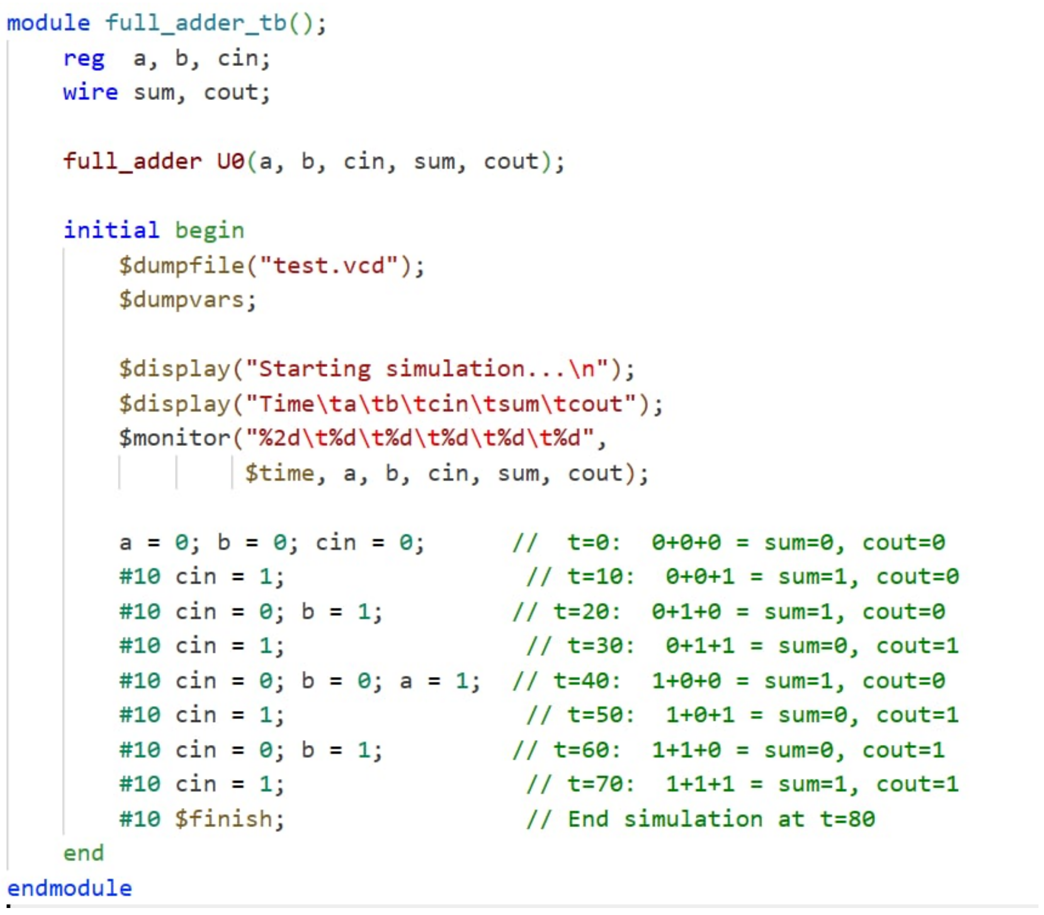


```
module full_adder_opt(a, b, cin, sum, cout);
  input  a, b, cin;
  output sum, cout;

  wire x;       // x = a xor b
  wire n1, n2; // NAND intermediates for cout

  // SUM: 2 XOR gates
  xor (x,   a,   b);
  xor (sum, x,   cin);

  // COUT: 3 NAND gates -> cout = ab + cin*(a xor b)
  nand(n1, a, b);      // n1 = ~(a&b)
  nand(n2, cin, x);    // n2 = ~(cin&x)
  nand(cout, n1, n2); // cout = ~(n1 & n2) = ab + cin*x
endmodule
```

Figure 1: *Verilog module for the optimized full adder.*

2. Verilog Testbench

The testbench exercises all eight input combinations in sequential 10-time-unit steps, cycling through the full truth table in binary order:



```verilog
module full_adder_tb();
    reg  a, b, cin;
    wire sum, cout;

    full_adder U0(a, b, cin, sum, cout);

    initial begin
        $dumpfile("test.vcd");
        $dumpvars;

        $display("Starting simulation...\n");
        $display("Time\ta\tb\tcin\tsum\tcout");
        $monitor("%2d\t%d\t%d\t%d\t%d\t%d",
                  $time, a, b, cin, sum, cout);

        a = 0; b = 0; cin = 0;      //  t=0:  0+0+0 = sum=0, cout=0
        #10 cin = 1;                 // t=10:  0+0+1 = sum=1, cout=0
        #10 cin = 0; b = 1;         // t=20:  0+1+0 = sum=1, cout=0
        #10 cin = 1;                 // t=30:  0+1+1 = sum=0, cout=1
        #10 cin = 0; b = 0; a = 1;  // t=40:  1+0+0 = sum=1, cout=0
        #10 cin = 1;                 // t=50:  1+0+1 = sum=0, cout=1
        #10 cin = 0; b = 1;         // t=60:  1+1+0 = sum=0, cout=1
        #10 cin = 1;                 // t=70:  1+1+1 = sum=1, cout=1
        #10 $finish;                 // End simulation at t=80
    end
endmodule
```

Figure 2: *Testbench for the full adder module.*

3. Simulation Console Output

The console output below confirms all eight input combinations match the expected truth table:

Table 1: *Simulation console output from Synopsys VCS.*

Time	a	b	c_{in}	sum	c_{out}
0	0	0	0	0	0
10	0	0	1	1	0
20	0	1	0	1	0
30	0	1	1	0	1
40	1	0	0	1	0
50	1	0	1	0	1
60	1	1	0	0	1
70	1	1	1	1	1

4. Simulation Waveform

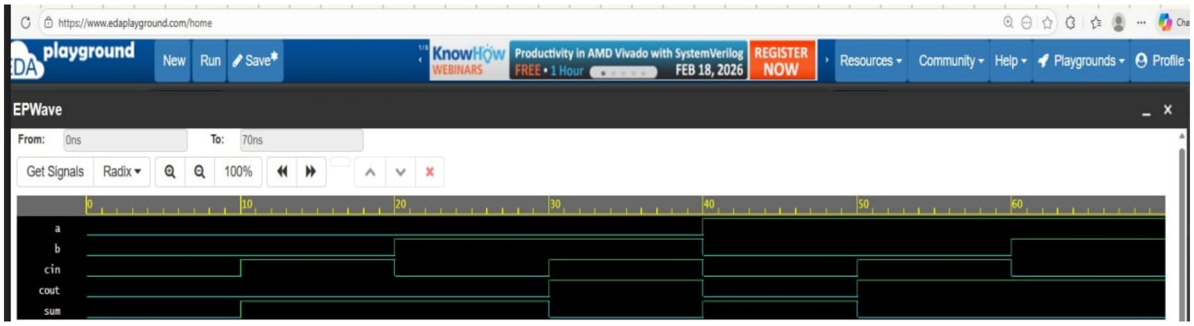


Figure 3: *EPWave waveform from EDAPlayground showing signals a, b, c_{in}, c_{out}, and sum for all eight input combinations (0 to 70 ns).*

V. Discussion

The experimental results matched the expected truth table of the full adder. The *sum* output correctly represented the XOR of the three inputs, and the carry out (c_{out}) output correctly indicated a carry when at least two inputs were 1. The breadboard implementation functioned properly, and the LEDs displayed the correct output combinations for all input cases. Additionally, the Verilog simulation produced the expected console output and waveform results. These findings confirm that both the hardware implementation and the Verilog design were correct and functionally equivalent.

The key insight of Question 3 is that $c_{out} = 1$ precisely when at least two of the three inputs are 1 (a majority function). Expressing this as $c_{out} = ab + c_{in}(a \oplus b)$ allows the intermediate wire $x = a \oplus b$, which is already required for computing *sum*, to be shared. This reduces the c_{out} path to only 3 NAND gates instead of the 4 required by the direct NAND-NAND form of the MSOP. The sharing works because c_{in} produces a carry when exactly one of a and b is 1, and that condition is exactly $a \oplus b = 1$.

A minor difficulty was encountered during the wiring process on the breadboard. Without a clear visualization of the optimized circuit, arranging the connections between the two ICs required careful attention to the pin configurations from the datasheets.

VI. Conclusion

In this lab, a full adder circuit was successfully designed, implemented, and verified using both hardware and Verilog HDL. The truth table and Boolean expressions were derived using Karnaugh maps, and the circuit was implemented using XOR and NAND gates. The design was optimized to reuse the $a \oplus b$ intermediate signal, reducing the total gate count to 2 XOR gates (74LS86) and 3 NAND gates (74LS00). The breadboard implementation and Verilog simulation both produced correct outputs for all input combinations. This lab reinforced the understanding of combinational logic circuits, Boolean algebra, and Verilog structural modeling.