# Combinational Logic Design: Part 4: Implementation of Full Adder Using Only NAND Gates

*Author Adil R.*

*May 2025*

## 1 Introduction

This project implements a full adder using only NAND gates through structural Verilog modeling. The design strictly uses gate-level instantiation without any data flow constructs, demonstrating how to build complex logic purely from NAND gate interconnections.

## 2 Design Description

### 2.1 Specifications

- Inputs: A, B, $\mathrm{C_{in}}$ (three 1-bit inputs)
- Outputs: S, $\mathrm{C_{out}}$ (sum and carry-out)
- Implementation constraint:
  - Only NAND gate primitives
  - No data flow modeling (no `assign` statements)
  - Pure structural implementation

Truth Table:

| A | B | $\mathrm{C_{in}}$ | S | $\mathrm{C_{out}}$ |
|---|---|---|---|---|
| 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 |

### 2.2 Gate-Level Implementation

The implementation uses 11 NAND gates arranged in three stages:

1. First XOR Stage (4 gates): Implements $A \oplus B$
2. Second XOR Stage (4 gates): Implements $(A \oplus B) \oplus C_{\text{in}}$ for Sum
3. Carry Stage (3 gates): Implements $C_{\text{out}} = (A \cdot B) + (C_{\text{in}} \cdot (A \oplus B))$

## 3 Verilog Implementation (Structural)

```verilog
`timescale 1ns / 1ps
module full_adder_nand(
    input A, B, Cin,
    output S, Cout
);
    // Internal wires for gate connections
    wire w1, w2, w3, w4;
    wire w5, w6, w7, w8;
    wire w9, w10;

    // First XOR stage:
    nand G1(w1, A, B);
    nand G2(w2, A, w1);
    nand G3(w3, B, w1);
    nand G4(w4, w2, w3);

    // Second XOR stage:
    nand G5(w5, w4, Cin);
    nand G6(w6, w4, w5);
    nand G7(w7, Cin, w5);
    nand G8(S, w6, w7);

    // Carry generation using 3 NAND gates
    nand G9(w9, A, B);
    nand G10(w10, Cin, w4);
    nand G11(Cout, w9, w10);
endmodule
```

Listing 1: Structural NAND-Only Full Adder

## 4 Test Bench

```verilog
`timescale 1ns / 1ps
module tb_full_adder_nand;
    // Inputs
    reg A, B, Cin;
    // Outputs
    wire S, Cout;

    // Instantiate unit under test
    full_adder_nand uut (.A(A), .B(B), .Cin(Cin), .S(S), .Cout(Cout));

    initial begin
        // Test all combinations
        $display("Time\tA B Cin\tS Cout");
        $monitor("%4t\t%b %b  %b\t%b  %b", $time, A, B, Cin, S, Cout);

        A=0; B=0; Cin=0; #10;
        A=0; B=0; Cin=1; #10;
        A=0; B=1; Cin=0; #10;
        A=0; B=1; Cin=1; #10;
        A=1; B=0; Cin=0; #10;
        A=1; B=0; Cin=1; #10;
        A=1; B=1; Cin=0; #10;
        A=1; B=1; Cin=1; #10;

        #10 $finish;
    end
endmodule
```

Listing 2: Structural Implementation Test Bench

## 5 Conclusion

- Successfully implemented full adder using only structural NAND gates
- Demonstrated the universality of NAND gates
- Pure structural modeling shows actual gate-level implementation
- Total gate count: 11 NAND gates
- Can be extended to multi-bit adders by cascading

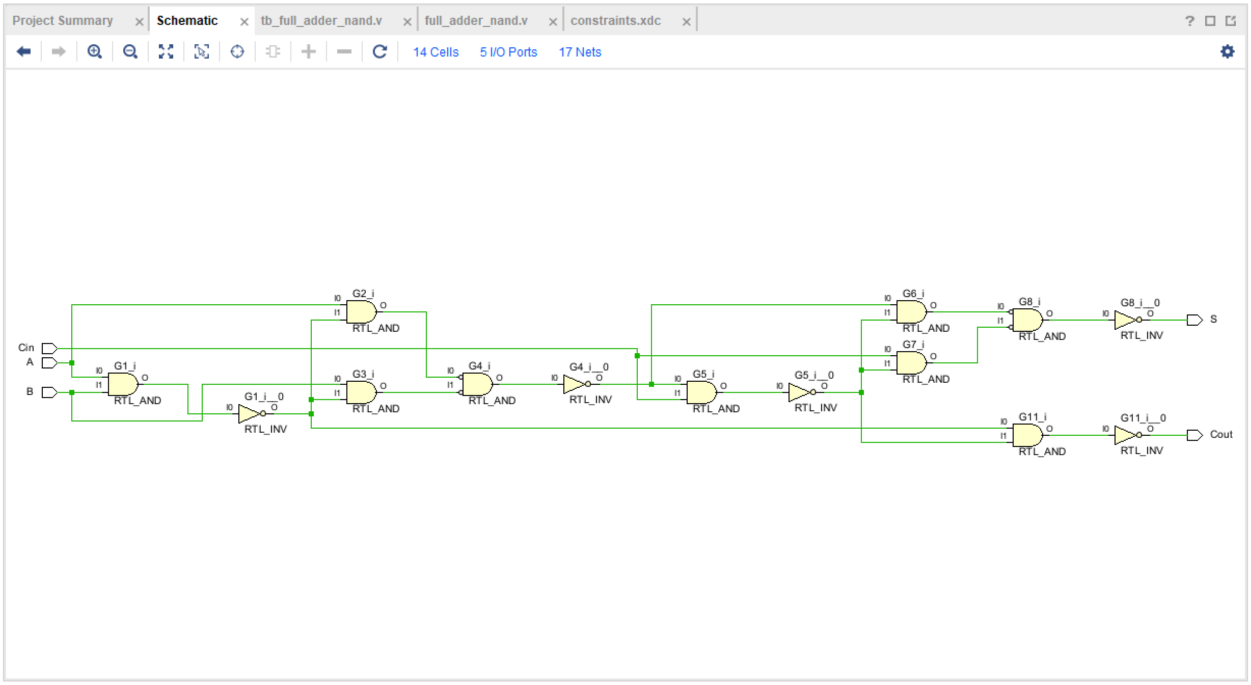

Figure 1: schematic

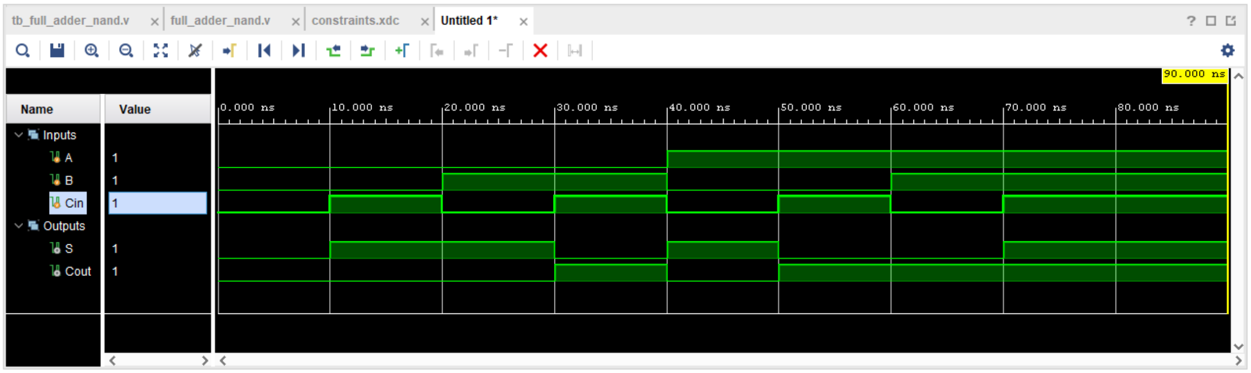

Figure 2: waveform