ECE 310
Lab 2 Report: 4-Bit Ripple-Carry Adder

Ohm Patel

September 14, 2025

Abstract

This lab involved the design and implementation of a gate-level full adder and its extension into a 4-bit ripple-carry adder (RCA). The objective was to gain practical experience in structural modeling and to analyze the carry propagation behavior in ripple-carry systems. A gate-level model of the full adder was constructed using basic logic gates, which was then used to build a 4-bit RCA. A Verilog testbench was used to evaluate the design across several representative test cases, including edge cases such as all zeros, maximum inputs, and carry propagation scenarios. The simulation results confirmed correct operation of the RCA.

1 Introduction, Background, and Theory

A full adder is a fundamental digital circuit that adds two binary digits along with a carry input, producing a sum and carry output. At the gate-level, it can be constructed using XOR, AND, and OR gates. By combining four full adders, a 4-bit ripple-carry adder is formed, where the carry-out of each stage becomes the carry-in of the next.

The ripple-carry adder demonstrates the concept of carry propagation delay: the sum cannot be finalized until the carry from the previous stage is computed. While simple and efficient for small bit-widths, this design becomes slow for larger word sizes, motivating more advanced adder architectures such as carry-lookahead adders.

2 Design and Implementation

The implementation proceeded in two stages:

2.1 Full Adder

A gate-level full adder was implemented using XOR, AND, and OR gates as shown in Listing 1.

Listing 1: Gate-level full adder implementation

```
module full_adder_struct(
    output S, Cout,             // Specify outputs S, Cout (Sum, Carry Out)
    input  A, B, Cin            // Specify inputs A, B, Cin (4-bit A, B and Carry in)
);
    wire w1, w2, w3;            // Internal Wires
    xor(w1, A, B);              // First XOR gate
    xor(S, w1, Cin);            // Second XOR output S (Sum)
    and(w2, A, B);              // First AND gate
    and(w3, w1, Cin);           // Second AND gate
    or(Cout, w2, w3);           // First OR output Cout (Carry Out)
endmodule
```

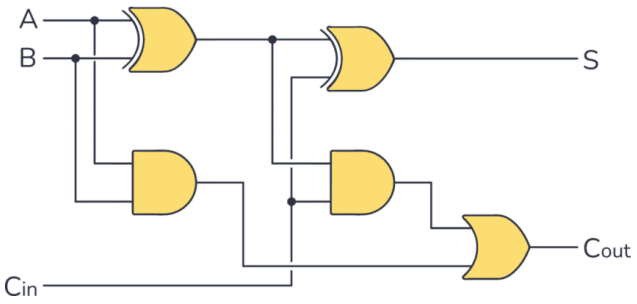

Figure 1: Gate-level diagram of the full adder (copied from lab file).

2.2 4-Bit Ripple-Carry Adder

The ripple-carry adder was constructed by cascading four instances of the full adder, as shown in Listing 2.

Listing 2: 4-bit ripple-carry adder

```
module rca_4bit(
    output [3:0] S,                                          // Specify 4-bit output
    output Cout,                                             // Specify carry out output
    input [3:0] A, B                                         // Specify 2 4-bit inputs
);
    wire c1, c2, c3;
    full_adder_struct fa0 (S[0], c1, A[0], B[0], 1'b0);  // Full adder 1
    full_adder_struct fa1 (S[1], c2, A[1], B[1], c1);    // Full adder 2
    full_adder_struct fa2 (S[2], c3, A[2], B[2], c2);    // Full adder 3
    full_adder_struct fa3 (S[3], Cout, A[3], B[3], c3);  // Full adder 4
endmodule
```

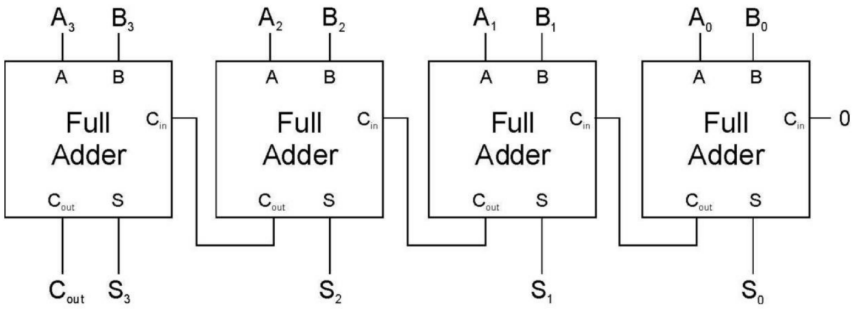

Figure 2: Structural diagram of the 4-bit ripple-carry adder (copied from lab file).

2.3 Testbench

A testbench was created to evaluate the RCA using representative input vectors (Listing 3). These vectors were chosen to verify different behaviors:

- All zeros (baseline check).
- Maximum values (overflow and carry-out behavior).
- Propagation of carry across all 4 stages.
- Mixed values to test intermediate carries.
- Edge cases with MSB addition.

Listing 3: Testbench for 4-bit RCA

```
initial begin
    A = 4'b0000; B = 4'b0000; #10;
    A = 4'b1111; B = 4'b1111; #10;
    A = 4'b1111; B = 4'b0001; #10;
    A = 4'b1010; B = 4'b0110; #10;
    A = 4'b0111; B = 4'b0001; #10;
    A = 4'b1000; B = 4'b1000; #10;
    $finish;
end
```

3 Results and Analysis

The lab required justification for at least four representative test vectors. Six test cases were selected to ensure full coverage of RCA behavior:

- **0000 + 0000:** Ensures the baseline case produces no spurious outputs.
- **1111 + 1111:** Tests overflow, verifying that Cout is correctly asserted.
- **1111 + 0001:** Validates carry propagation across all four stages.
- **1010 + 0110:** Demonstrates both carry and non-carry additions in mixed positions.
- **0111 + 0001:** Shows carry into the most significant bit without overflow.
- **1000 + 1000:** Confirms correct handling of MSB-only addition.

These test vectors go beyond trivial cases and explicitly test propagation, overflow, and edge conditions, thereby justifying their inclusion for verification.

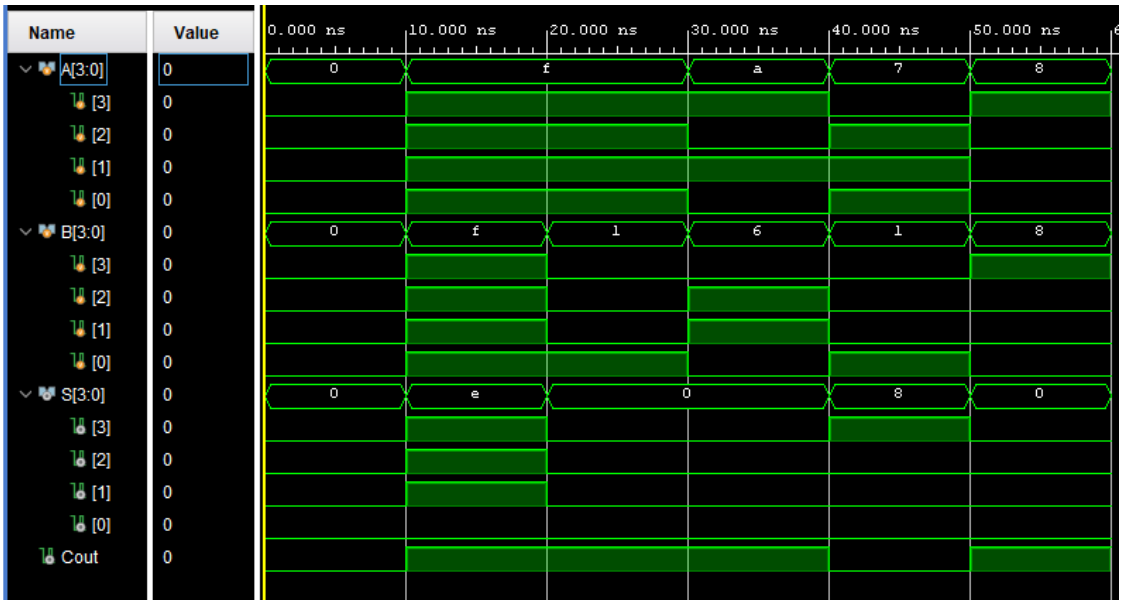

Figure 3: Simulation waveform of the 4-bit ripple-carry adder showing selected test cases.

The waveform output (Figure 3) confirms that the RCA design produced the expected outputs in each case, validating correctness.

4 Conclusion

A 4-bit ripple-carry adder was successfully implemented using gate-level full adders in Verilog. The testbench results validated the correctness of the design across multiple representative input vectors. The results highlighted the expected functionality of carry propagation and demonstrated both overflow and intermediate carry behavior. This experiment reinforced the hierarchical design methodology in digital systems and provided practical insight into the limitations of ripple-carry adders.

This report was compiled using LaTeX .