Project 2: Ripple Carry Adder

A Comprehensive Study of Fundamental Digital Circuits

By: Abhishek Sharma

Contents

1 Project Overview

A ripple carry adder (RCA) is a simple yet fundamental digital circuit used to add two binary numbers. It consists of a series of full adders, where each full adder handles the addition of a pair of corresponding bits from the two input numbers along with a carry bit from the previous stage. The carry output of each full adder is "rippled" to the next full adder in the sequence, hence the name "ripple carry adder."

2 Ripple Carry Adder

2.1 Description

A ripple carry adder (RCA) is a digital circuit that adds two binary numbers. It consists of a series of full adders, where the carry output of each full adder is connected to the carry input of the next full adder in the sequence. The RCA is simple but has a delay that increases linearly with the number of bits. “‘latex

3 Why Choose a Ripple Carry Adder?

- **Simplicity and Clarity**: The ripple carry adder is straightforward to understand and implement, making it an excellent choice for learning the basics of binary addition.
- **Modular Design**: Each bit addition is handled by an identical full adder module, demonstrating modular design principles in digital circuits.
- **Foundation for Complex Adders**: Understanding the RCA lays the groundwork for learning more complex and faster adders, such as carry-lookahead adders or carry-save adders.
- **Educational Value**: It illustrates the concept of carry propagation, an essential aspect of binary arithmetic in digital electronics.

“‘

3.1 RTL Code

Listing 1: Ripple Carry Adder RTL Code

```
// Parameterized Ripple Carry Adder Module
module ripple_carry_adder #(parameter N = 4) (
    input logic [N-1:0] A,  // N-bit input A
    input logic [N-1:0] B,  // N-bit input B
    output logic [N-1:0] Sum, // N-bit Sum output
    output logic Cout // Carry Out
);
    logic [N-1:0] Carry; // Internal Carry bits

    // Generate Full Adders for each bit
    genvar i;
    generate
        for (i = 0; i < N; i = i + 1) begin : full_adder_gen
            if (i == 0) begin
                full_adder fa (
                    .A(A[i]),
                    .B(B[i]),
                    .Cin(0),
                    .Sum(Sum[i]),
                    .Cout(Carry[i])
                );
            end else begin
                full_adder fa (
                    .A(A[i]),
                    .B(B[i]),
                    .Cin(Carry[i-1]),
                    .Sum(Sum[i]),
                    .Cout(Carry[i])
                );
            end
        end
    endgenerate

    assign Cout = Carry[N-1];

endmodule

// Full Adder Module
module full_adder (
    input logic A,
    input logic B,
    input logic Cin,
    output logic Sum,
    output logic Cout
);
    assign Sum = A ^ B ^ Cin;  // XOR for Sum
    assign Cout = (A & B)  (Cin & (A ^ B)); // AND-OR for Carry
endmodule
```

3.2 Testbench

Listing 2: Ripple Carry Adder Testbench

```
// Testbench for Parameterized Ripple Carry Adder
module tb_ripple_carry_adder;

    // Parameter for the bit-width
    parameter N = 8;

    // Testbench signals
    logic [N-1:0] A, B;
    logic [N-1:0] Sum;
    logic Cout;

    // Instantiate the Ripple Carry Adder
    ripple_carry_adder #(N) uut (
        .A(A),
        .B(B),
        .Sum(Sum),
        .Cout(Cout)
    );

    // Stimulus block
    initial begin
        // Initialize inputs
        A = 0;
        B = 0;

        // Apply test vectors
        #10 A = 8'b00001111; B = 8'b00000001;  // Test case 1
        #10 A = 8'b11110000; B = 8'b00001111;  // Test case 2
        #10 A = 8'b10101010; B = 8'b01010101;  // Test case 3
        #10 A = 8'b11111111; B = 8'b11111111;  // Test case 4
        #10 A = 8'b10000001; B = 8'b10000001;  // Test case 5

        // End simulation
        #10 $stop;
    end

    // Monitor block to display results
    initial begin
        $monitor("Time=%0t : A=%b B=%b => Sum=%b Cout=%b", $time, A,
           B, Sum, Cout);
    end

endmodule
```

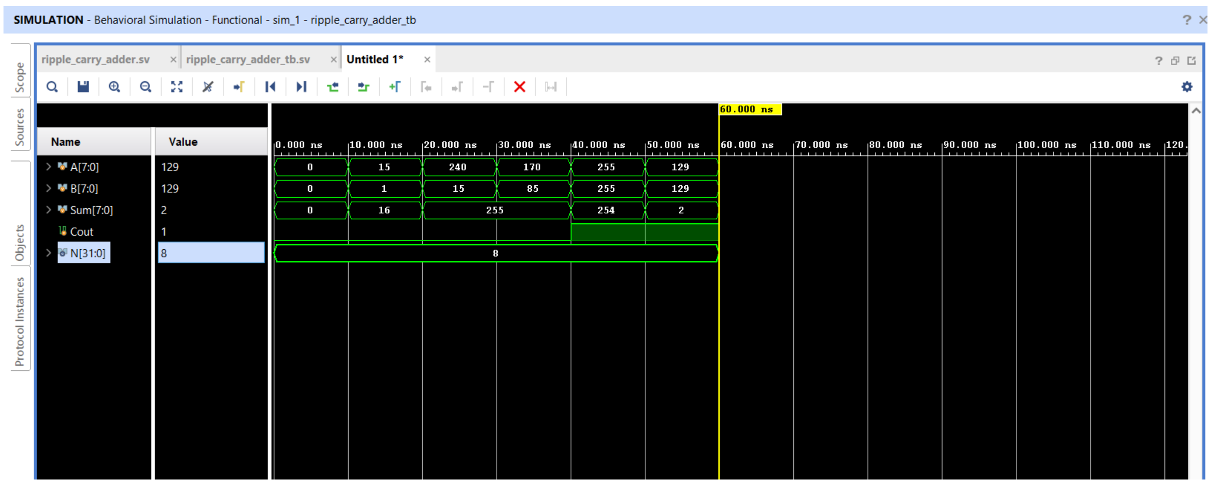
4 Explanation

- The `tb_ripple_carry_adder` testbench instantiates the `ripple_carry_adder` module with a parameter `N`, which defines the bit-width (e.g., 8 bits in this case).
- The `initial` block applies different test vectors to the inputs `A` and `B` to test the adder.
- The `monitor` statement is used to display the values of `A`, `B`, `Sum`, and `Cout` at each simulation time step.

5 Usage

- The testbench initializes the inputs and applies a series of test vectors to verify the functionality of the ripple carry adder.
- You can change the parameter `N` to test the adder with different bit-widths.

5.1 Simulation Results


SIMULATION - Behavioral Simulation - Functional - sim_1 - ripple_carry_adder_tb

ripple_carry_adder.sv | ripple_carry_adder_tb.sv | Untitled 1*

60.000 ns

Name	Value	0.000 ns	10.000 ns	20.000 ns	30.000 ns	40.000 ns	50.000 ns
A[7:0]	129	0	15	240	170	255	129
B[7:0]	129	0	1	15	85	255	129
Sum[7:0]	2	0	16	255		254	2
Cout	1						
N[31:0]	8	8					

Figure 1: Simulation results of RCA

5.2 Schematic

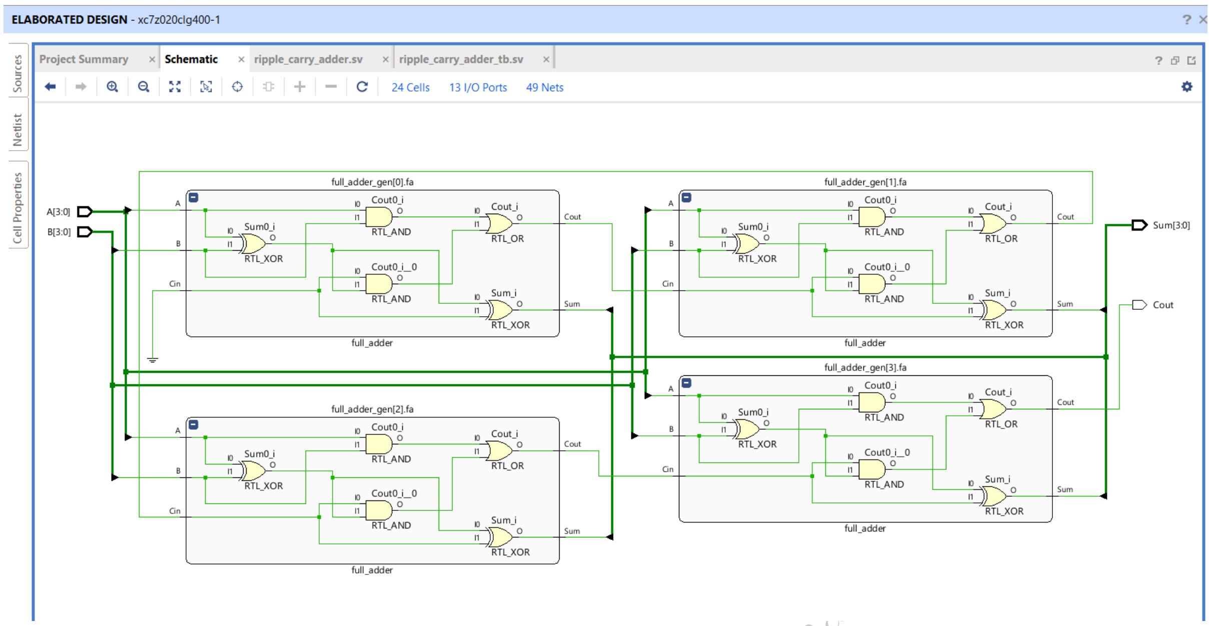


Figure 2: Schematic of RCA

5.3 Synthesis Design

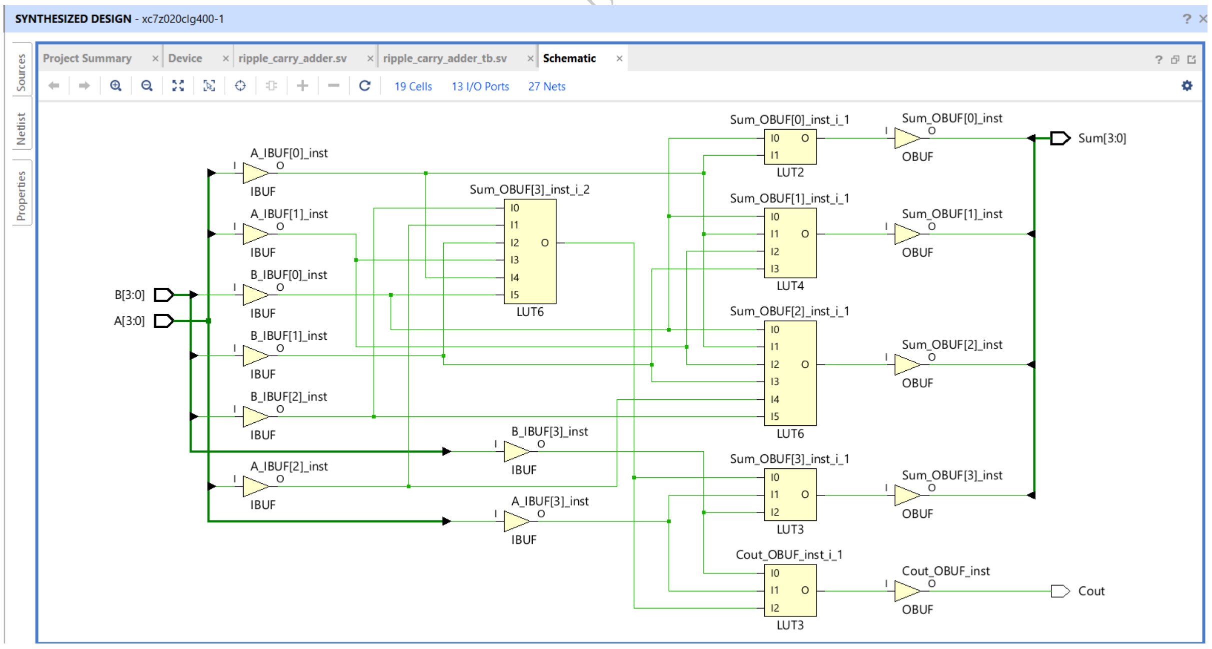


Figure 3: Synthesis Design of RCA

5.4 Advantages

- **Simplicity**: Ripple carry adders are straightforward to understand and implement, making them an excellent choice for learning and teaching basic digital design concepts.

- **Modular Design**: The design is modular, with each bit addition handled by an identical full adder, which simplifies design and verification.
- **Ease of Implementation**: Because of their simplicity, RCAs can be easily implemented using basic logic gates and are well-suited for use in simple digital circuits.

5.5 Disadvantages

- **High Propagation Delay**: The ripple carry adder suffers from high propagation delay due to the sequential carry propagation through each full adder. This limits its performance in high-speed applications.
- **Scalability Issues**: As the number of bits increases, the delay increases linearly, which can lead to significant performance issues for wide adders.
- **Inefficiency in Large-Scale Systems**: For large-scale systems requiring high-speed arithmetic operations, ripple carry adders are less efficient compared to more advanced adder designs like carry-lookahead or carry-save adders.

5.6 Applications

- **Basic Arithmetic Units**: Ripple carry adders are used in simple arithmetic units for binary addition, such as in basic microprocessors and digital systems.
- **Simple Digital Circuits**: They are used in applications where simplicity is more critical than performance, such as in small embedded systems or educational tools.
- **Error Detection and Correction Systems**: In some error detection and correction systems, RCAs are employed to perform addition operations on error-checking codes.

6 Delay in Ripple Carry Adder

Delay in digital circuits refers to the time it takes for a change in input to propagate through the circuit and produce a corresponding change at the output. For adders, this is typically measured as the time from the application of the inputs until the output stabilizes.

6.1 Technical Details of Ripple Carry Adder

A ripple carry adder (RCA) is a straightforward implementation of an adder for binary numbers. It consists of a series of full adders, each of which adds corresponding bits of the input numbers along with the carry from the previous bit. The output of each full adder is a sum bit and a carry bit, which is propagated to the next full adder in the series.

6.2 Propagation Delay in Ripple Carry Adder

The propagation delay in a ripple carry adder is relatively high compared to more advanced adder designs. This is primarily due to the nature of carry propagation in the adder. The delay in an RCA can be understood as follows:

- **Carry Propagation Path**: In an RCA, each full adder must wait for the carry bit from the previous full adder before it can produce its own sum and carry outputs. This creates a dependency chain that propagates from the least significant bit (LSB) to the most significant bit (MSB).
- **Delay Accumulation**: The total delay of the RCA is the sum of the delays of all the full adders. Since each full adder introduces a delay for the carry to propagate through, the overall delay increases linearly with the number of bits N.

- **Critical Path**: The critical path in an RCA is the longest path that a signal must travel through, which in this case is from the LSB's input through all subsequent full adders to the MSB's carry output. The delay of this path determines the overall speed of the adder.

Mathematically, the total delay T_{RCA} of an N-bit ripple carry adder can be approximated as:

$$T_{RCA} \approx N \cdot T_{FA}$$

where T_{FA} is the delay of a single full adder.

6.3 Improving Delay

To reduce the propagation delay, more advanced adder designs such as carry-lookahead adders (CLA) or carry-save adders (CSA) are used. These designs use parallel processing and other techniques to shorten the carry propagation path, thereby reducing the overall delay.