Project 15: Parallel Adder

A Comprehensive Study of Advanced Digital Circuits

By: Abhishek Sharma, Gati Goyal , Nikunj Agrawal , Ayush Jain

Contents

1 Introduction

A parallel adder is a digital circuit designed to add binary numbers quickly by processing multiple bits simultaneously. Unlike serial adders that handle bits one at a time, parallel adders use multiple full adders to compute all bit positions at once, reducing the overall addition time. This makes them ideal for high-speed arithmetic operations in processors and digital systems.

2 Key Concepts

1. **Bitwise Addition**: A parallel adder performs addition on all bit positions of the input numbers simultaneously.
2. **Full Adders**: It uses multiple full adder circuits, each handling a single bit of the inputs and its corresponding carry.
3. **Carry Propagation**: Each full adder generates a carry-out, which is used by the next full adder to compute the final result.
4. **Simultaneous Processing**: Unlike serial adders, which process bits sequentially, parallel adders compute all bits in parallel, speeding up the addition process.
5. **Speed Advantage**: The primary advantage is the reduction in addition time by eliminating the carry propagation delay of serial adders.
6. **Hardware Complexity**: While faster, parallel adders require more hardware (multiple full adders) compared to serial adders.
7. **Application**: Ideal for high-speed arithmetic operations in digital systems such as processors and digital signal processors.

3 Steps in Parallel Adder

1. **Input Initialization:**
   - Load the binary numbers to be added into the parallel adder.
   - Each input number is split into individual bits corresponding to the bit positions.
2. **Bitwise Addition:**
   - For each bit position, a full adder circuit adds the corresponding bits from the input numbers, along with any carry from the previous bit position.
3. **Full Adder Operation:**
   - **Sum Calculation:** Computes the sum of the two input bits and the carry-in from the previous stage.
   - **Carry Generation:** Determines the carry-out to be sent to the next full adder.
4. **Carry Propagation:**
   - The carry-out from each full adder is used as the carry-in for the next full adder in the chain, continuing until the most significant bit is processed.
5. **Result Compilation:**
   - Collect the sum outputs from all full adders to form the final sum.
   - The carry-out from the last full adder becomes the final carry-out of the addition.
6. **Output Generation:**
   - Output the final sum and carry-out results.
   - The sum is a binary number representing the result of the addition, and the carry-out indicates if there was an overflow.

4 Why to Choose It

- **Speed:** Computes all bits simultaneously, reducing addition time and carry propagation delays.
- **Efficiency:** Offers high throughput, making it ideal for high-performance systems.
- **Simplicity:** Modular design with straightforward carry handling.
- **Versatility:** Easily scalable for different bit-widths and adaptable to various applications.

5 Diagrams

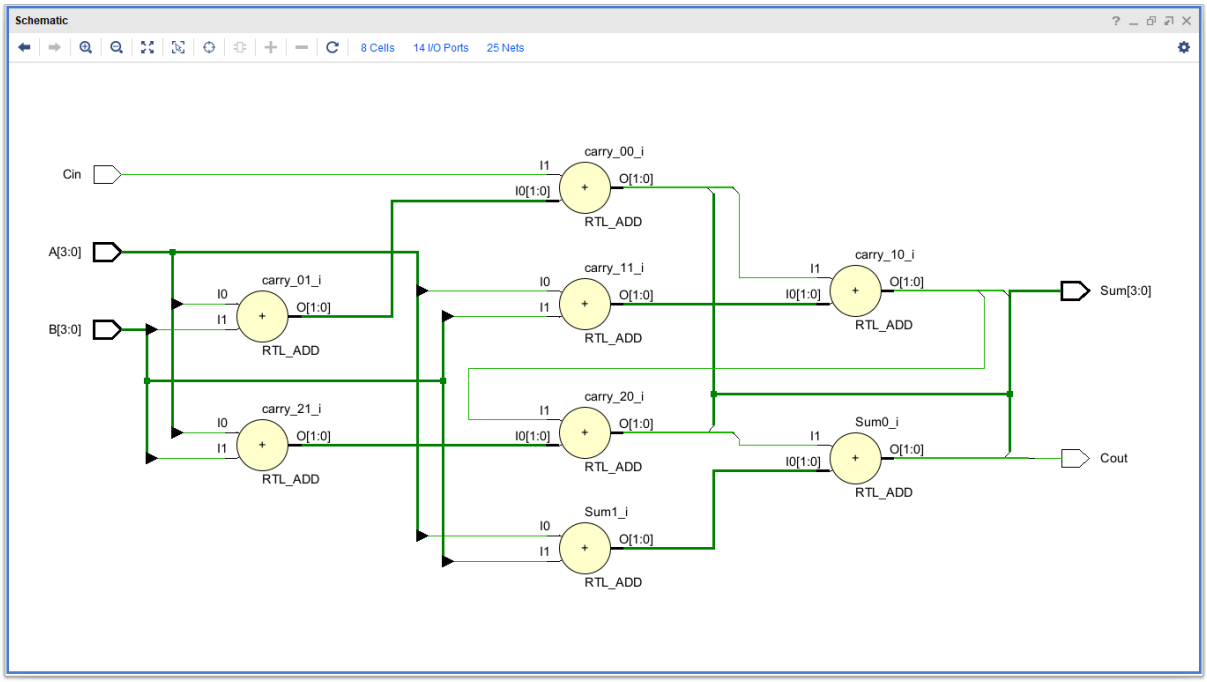

Figure 1: Schematic of Parallel Adder

6 SystemVerilog Code

Listing 1: Parallel Adder RTL Code

```
module parallel_adder (
    input  logic [3:0] A,   // 4-bit input A
    input  logic [3:0] B,   // 4-bit input B
    input  logic       Cin, // Carry-in
    output logic [3:0] Sum, // 4-bit Sum output
    output logic       Cout // Carry-out
);

    // Intermediate signals for carry
    logic [3:0] carry;

    // Full adder for each bit
    assign {carry[0], Sum[0]} = A[0] + B[0] + Cin;
    assign {carry[1], Sum[1]} = A[1] + B[1] + carry[0];
    assign {carry[2], Sum[2]} = A[2] + B[2] + carry[1];
```

```
    assign {carry[3], Sum[3]} = A[3] + B[3] + carry[2];

    // Final carry-out
    assign Cout = carry[3];

endmodule
```

7 Testbench

Listing 2: Parallel Adder Testbench

```
module tb_parallel_adder;

    // Testbench signals
    logic [3:0] A_tb;
    logic [3:0] B_tb;
    logic       Cin_tb;
    logic [3:0] Sum_tb;
    logic       Cout_tb;

    // Instantiate the parallel adder
    parallel_adder uut (
        .A(A_tb),
        .B(B_tb),
        .Cin(Cin_tb),
        .Sum(Sum_tb),
        .Cout(Cout_tb)
    );

    // Test stimulus
    initial begin
        // Initialize inputs
        A_tb = 4'b0000;
        B_tb = 4'b0000;
        Cin_tb = 1'b0;
        #10;

        // Test case 1
        A_tb = 4'b0011; // 3
        B_tb = 4'b0101; // 5
        Cin_tb = 1'b0;
        #10;
        $display("Test Case 1: A=3, B=5, Cin=0 => Sum=%b, Cout=%b",
            Sum_tb, Cout_tb);

        // Test case 2
        A_tb = 4'b1111; // 15
        B_tb = 4'b0001; // 1
        Cin_tb = 1'b0;
        #10;
        $display("Test Case 2: A=15, B=1, Cin=0 => Sum=%b, Cout=%b",
            Sum_tb, Cout_tb);

        // Test case 3
        A_tb = 4'b1001; // 9
        B_tb = 4'b0110; // 6
        Cin_tb = 1'b1;
```

```
45          #10;
46          $display("Test Case 3: A=9, B=6, Cin=1 => Sum=%b, Cout=%b",
                Sum_tb, Cout_tb);
47
48          // Test case 4
49          A_tb = 4'b0110; // 6
50          B_tb = 4'b0111; // 7
51          Cin_tb = 1'b1;
52          #10;
53          $display("Test Case 4: A=6, B=7, Cin=1 => Sum=%b, Cout=%b",
                Sum_tb, Cout_tb);
54
55          // Finish the simulation
56          $stop;
57      end
58
59  endmodule
```

8 Synthesis Results

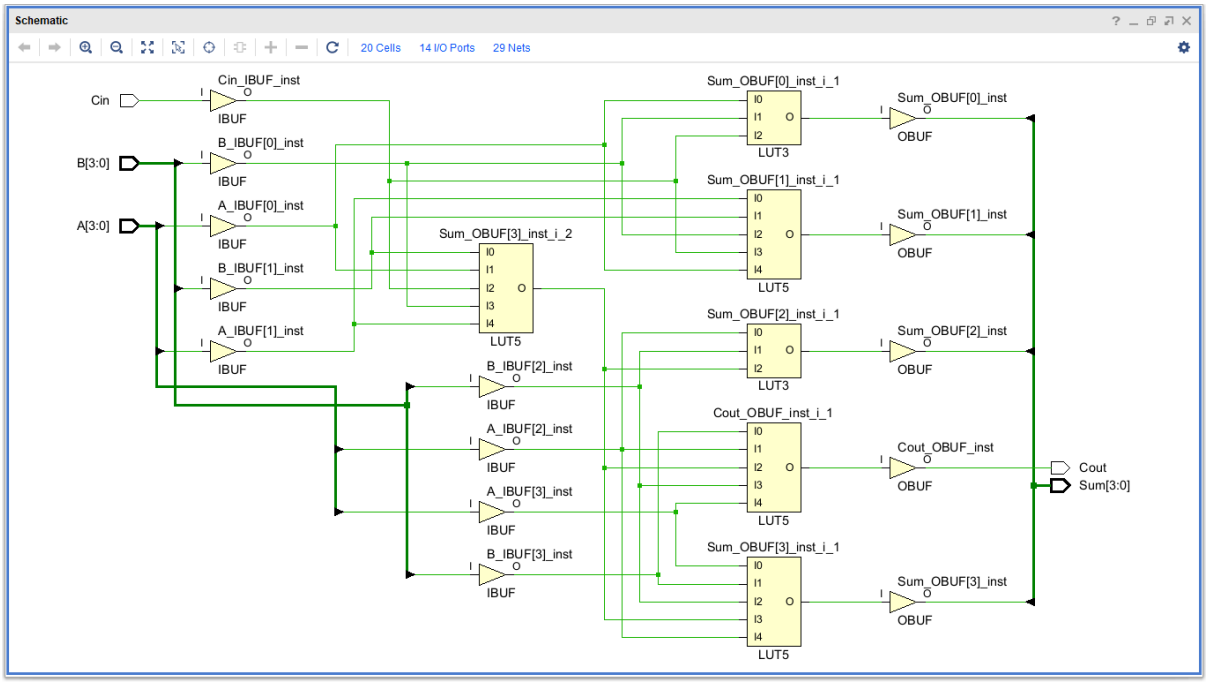

Figure 2: Synthesis of Parallel Adder

[h]

9 Simulation Results

10 How It Works in Depth

1. **Input Initialization:**
   - The parallel adder takes two binary numbers and optionally a carry-in value. Each number is divided into individual bits, corresponding to the bit positions (e.g., for a 4-bit adder, there are four bits for each number).

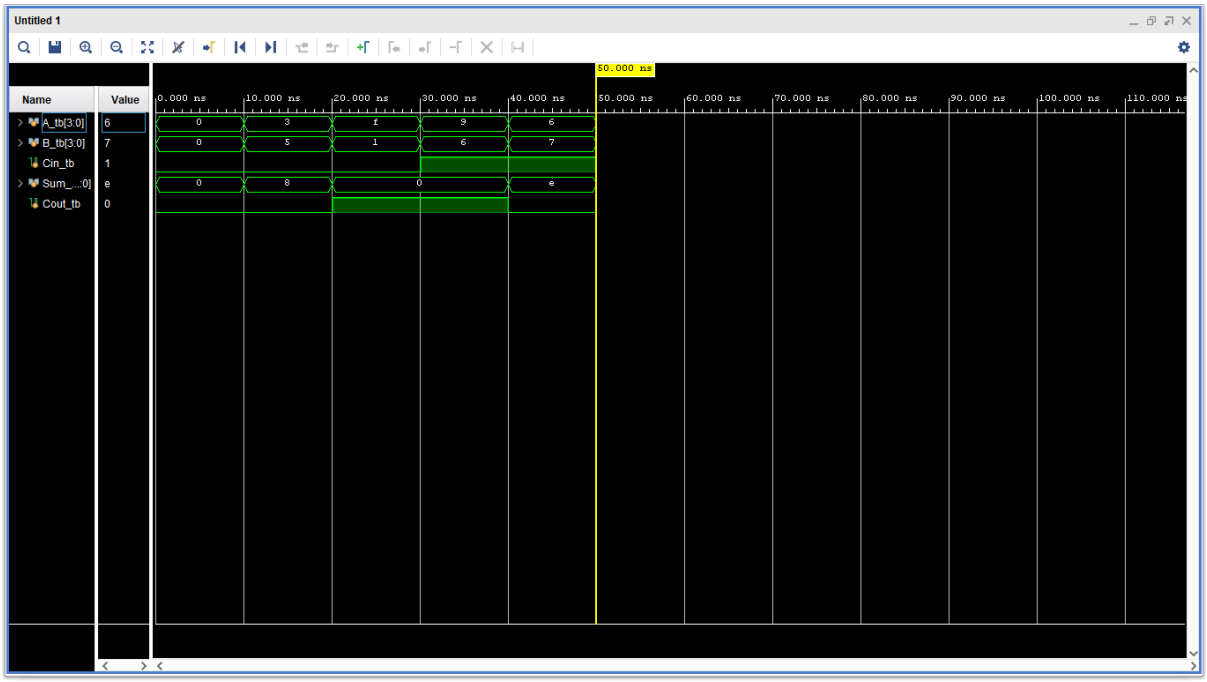

Figure 3: Simulation of Parallel Adder

2. **Bitwise Addition:**
   - Each bit position from the two input numbers is added together simultaneously. This is achieved using multiple full adder circuits, each responsible for one bit position.
3. **Full Adder Operation:**
   - Each full adder consists of three inputs: two data bits and a carry-in bit from the previous stage. It performs the following calculations:
     - **Sum Calculation:** The sum of the two input bits and the carry-in is computed using the XOR (exclusive OR) operation:

$$\text{Sum}_i = A_i \oplus B_i \oplus C_i$$

     - **Carry Generation:** The carry-out for the next stage is determined based on the input bits and carry-in. It is computed using:

$$\text{Carry}_{out} = (A_i \cdot B_i) + ((A_i \oplus B_i) \cdot C_i)$$

4. **Carry Propagation:**
   - The carry-out from each full adder is passed as the carry-in to the next full adder in the chain. This process continues from the least significant bit (LSB) to the most significant bit (MSB). Because each full adder processes bits in parallel, the delay due to carry propagation is minimized compared to serial adders.
5. **Result Compilation:**
   - After processing all bit positions, the sum outputs from each full adder are combined to form the final result. The carry-out from the last full adder is used as the final carry-out of the addition, indicating if there was an overflow beyond the most significant bit.
6. **Output Generation:**
   - The final sum and carry-out are output from the parallel adder. The sum represents the binary result of the addition, while the carry-out signifies whether there was an overflow beyond the bit-width of the adder.

11 Conclusion

A parallel adder is an essential component in digital arithmetic, designed to enhance the speed and efficiency of binary addition. By utilizing multiple full adders to process each bit position simultaneously, it significantly reduces addition time and minimizes carry propagation delays compared to serial adders. This parallel processing capability results in high throughput and fast computation, making parallel adders ideal for high-performance systems and applications requiring rapid arithmetic operations. Despite its increased hardware complexity, the benefits of reduced delay and improved speed make the parallel adder a valuable tool in modern digital systems.

12 References

1. M. Morris Mano, *Digital Logic Design*, 4th ed., Pearson, 2013.
2. J. R. Rice, *Mathematical Foundations of Computer Science*, Springer, 2017.
3. S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with Verilog Design*, 3rd ed., McGraw-Hill, 2013.
4. C. H. Roth and L. Kinney, *Fundamentals of Logic Design*, 7th ed., Cengage Learning, 2017.
5. W. I. Fletcher, *An Engineering Approach to Digital Design*, Prentice-Hall, 2018.