Project 19: Carry Bypass Adder

A Comprehensive Study of Advanced Digital Circuits

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

Contents

1 Introduction

The Carry Bypass Adder (CBA) speeds up addition by allowing the carry to bypass sections of the adder when all bits in a block propagate the carry. By dividing the input into blocks, the CBA reduces the delay caused by ripple-carry propagation, making it faster than traditional ripple-carry adders while maintaining moderate hardware complexity. This method balances speed and efficiency, particularly in larger bit-width additions.

2 Key Concepts

- **Block Division**: The adder divides the input bits into smaller blocks to optimize carry propagation.
- **Generate and Propagate Signals**: Each bit in a block generates a "propagate" signal, which determines if the carry can bypass the block.
- **Carry Skip Logic**: If all propagate signals in a block are true, the carry skips that block, reducing the delay.
- **Ripple Carry within Blocks**: Carry is computed using ripple-carry logic inside each block, ensuring accuracy.
- **Trade-off Between Speed and Complexity**: The CBA achieves faster operation compared to ripple-carry adders with moderate hardware complexity, making it a practical design for larger adders.

3 Steps in Carry Bypass Adder

- **Generate and Propagate Signals:**
  - For each bit i, calculate the generate (G_i) and propagate (P_i) signals:

$$G_i = A_i \wedge B_i$$

$$P_i = A_i \oplus B_i$$

- **Block Division:**
  - Divide the input bits into smaller blocks of size B. Each block processes a group of bits, computing carry and sum for that block.
- **Carry Calculation within Blocks:**
  - Within each block, compute the carry using ripple-carry logic:

$$C_{i+1} = G_i \vee (P_i \wedge C_i)$$

  - Here, C_i is the carry-in for the block.
- **Carry Bypass Logic:**
  - For each block, determine if the carry can be skipped. If all propagate signals in a block are true, the carry is bypassed:

$$\text{Skip}_{block} = \text{All P signals in block}$$

  - Update the carry for the next block based on the skip condition:

$$C_{next} = \text{Skip}_{block} ? C_{current} : C_{block}$$

- **Sum Calculation:**

  - Compute the sum bits for each bit i in the block:

$$S_i = P_i \oplus C_i$$

  - Here, C_i is the carry for that bit.

- **Final Carry-Out:**
  - The final carry-out of the adder is derived from the last block's carry output.

4 Why to Choose It

- **Improved Speed:**
  - The Carry Bypass Adder significantly reduces propagation delay by allowing the carry to skip over sections of the adder where it is not needed, resulting in faster addition operations compared to traditional ripple-carry adders.
- **Moderate Hardware Complexity:**
  - While it speeds up computation, the Carry Bypass Adder does not require as much hardware complexity as more advanced adders like Kogge-Stone or Brent-Kung adders. This makes it a good balance between performance and hardware resources.
- **Efficient for Larger Bit-widths:**
  - The adder is particularly effective for larger bit-widths where the carry delay can become substantial. By reducing the number of carry computations, it scales well with larger data sizes.
- **Trade-off Between Speed and Complexity:**
  - The CBA offers a trade-off between the speed of addition and the complexity of hardware implementation. It provides a good balance for applications where speed is crucial, but hardware resources are limited.
- **Versatility:**
  - The Carry Bypass Adder can be used in various digital systems and is suitable for applications where performance improvements are necessary without a significant increase in complexity.

5 SystemVerilog Code

Listing 1: Carry Bypass Adder RTL Code

```
module carry_bypass_adder #(parameter WIDTH = 16, parameter BLOCK_SIZE = 4) (
    input  logic [WIDTH-1:0] A, B,   // Inputs: A and B are binary numbers
    input  logic Cin,                // Carry in
    output logic [WIDTH-1:0] Sum,    // Output: Sum of A and B
    output logic Cout                // Carry out
);

    // Internal signals for carry, propagate, generate
    logic [WIDTH-1:0] P, G;
    logic [(WIDTH/BLOCK_SIZE)-1:0] carry_skip; // Signals for carry skip condition
    logic [WIDTH:0] carry;    // Carry signals (including final carry out)

    assign carry[0] = Cin;  // Initial carry input

    // Step 1: Generate and propagate signals
    always_comb begin
        for (int i = 0; i < WIDTH; i++) begin
            P[i] = A[i] ^ B[i];  // Propagate signal
            G[i] = A[i] & B[i];  // Generate signal
        end
    end

    // Step 2: Ripple carry and bypass carry logic
    always_comb begin
        for (int i = 0; i < WIDTH; i++) begin
            if (i % BLOCK_SIZE == 0) begin
                // First bit of the block: carry ripple from the
                    previous block or Cin
                carry[i+1] = G[i] | (P[i] & carry[i]);
            end else begin
                // Ripple carry within the block
                carry[i+1] = G[i] | (P[i] & carry[i]);
            end
        end

        // Carry bypass for each block
        for (int j = 0; j < (WIDTH / BLOCK_SIZE); j++) begin
            carry_skip[j] = &P[j*BLOCK_SIZE +: BLOCK_SIZE];  // Skip
                condition: all bits propagate
            if (carry_skip[j]) begin
                carry[(j+1)*BLOCK_SIZE] = carry[j*BLOCK_SIZE];  //
                    Skip carry to the next block
            end
        end
    end

    // Step 3: Compute the sum bits
    always_comb begin
        for (int i = 0; i < WIDTH; i++) begin
            Sum[i] = P[i] ^ carry[i];  // Sum = Propagate XOR Carry
        end
    end

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

endmodule
```

6 Testbench

Listing 2: Carry Bypass Adder Testbench

```
module tb_carry_bypass_adder;

    // Parameters
    parameter WIDTH = 16;
    parameter BLOCK_SIZE = 4;

    // Inputs
    logic [WIDTH-1:0] A, B;
    logic Cin;

    // Outputs
    logic [WIDTH-1:0] Sum;
    logic Cout;

    // Instantiate the Unit Under Test (UUT)
    carry_bypass_adder #(WIDTH, BLOCK_SIZE) uut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .Sum(Sum),
        .Cout(Cout)
    );

    // Test stimulus
    initial begin
        // Test case 1
        A = 16'h1234;
        B = 16'h5678;
        Cin = 1'b0;
        #10;
        $display("TC1: A = %h, B = %h, Cin = %b -> Sum = %h, Cout =
            %b", A, B, Cin, Sum, Cout);

        // Test case 2
        A = 16'hFFFF;
        B = 16'h0001;
        Cin = 1'b0;
        #10;
        $display("TC2: A = %h, B = %h, Cin = %b -> Sum = %h, Cout =
            %b", A, B, Cin, Sum, Cout);

        // Test case 3
        A = 16'hAAAA;
        B = 16'h5555;
        Cin = 1'b1;
        #10;
        $display("TC3: A = %h, B = %h, Cin = %b -> Sum = %h, Cout =
            %b", A, B, Cin, Sum, Cout);

        // Test case 4
        A = 16'h0FFF;
        B = 16'hF000;
        Cin = 1'b0;
        #10;
        $display("TC4: A = %h, B = %h, Cin = %b -> Sum = %h, Cout =
            %b", A, B, Cin, Sum, Cout);

        // End simulation
        $stop;
    end

endmodule
```

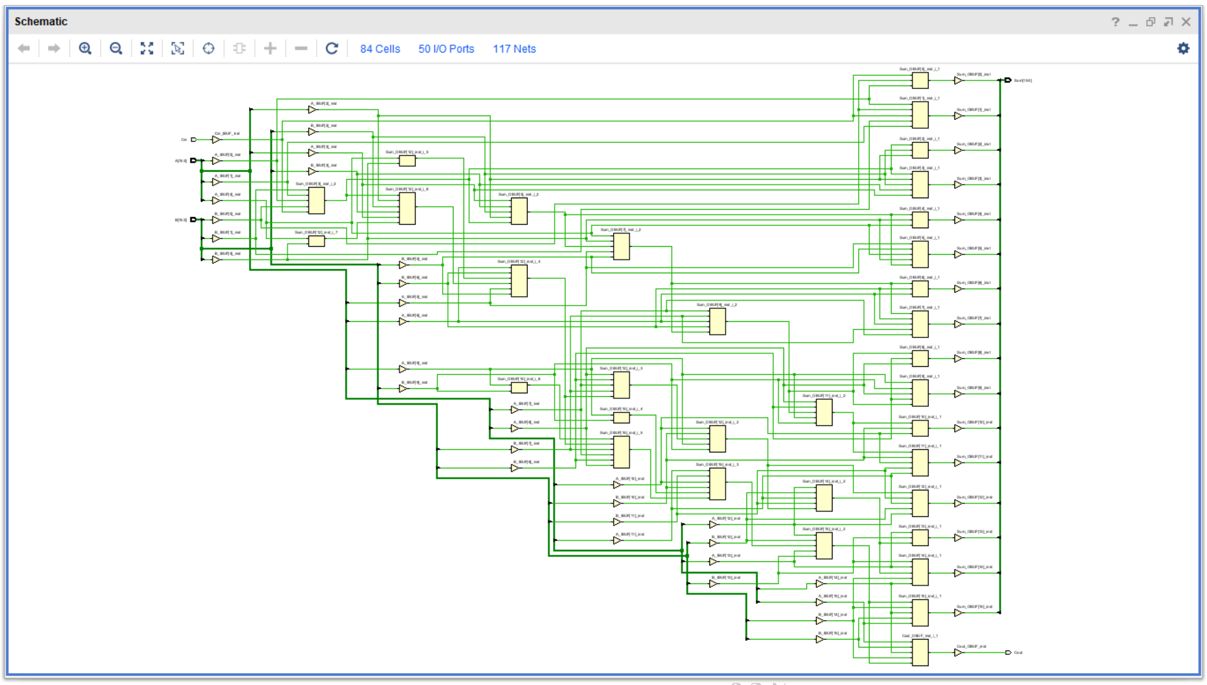

Figure 1: Schematic of Carry Bypass Adder

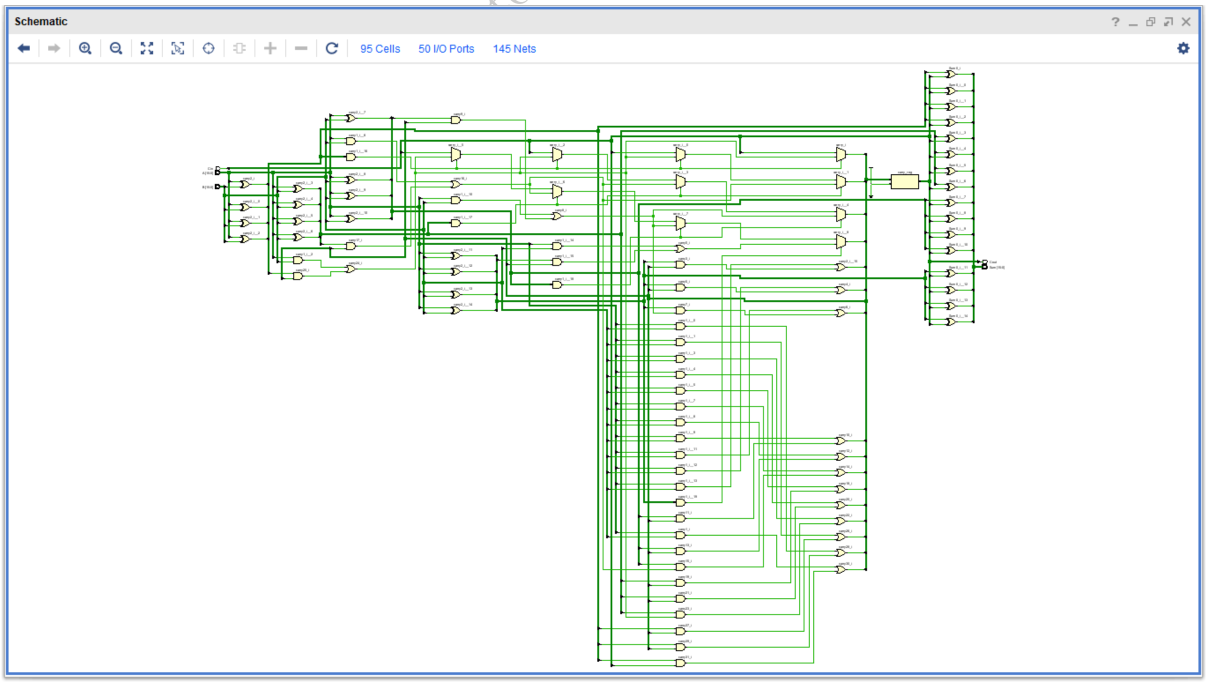

Figure 2: Synthesis of Carry Bypass Adder

7 Conclusion

The **Carry Bypass Adder** (CBA) is a highly effective addition technique that balances speed and hardware complexity. By allowing the carry to bypass certain sections of the adder, it significantly reduces propagation delay compared to traditional ripple-carry adders. This results in faster arithmetic operations, particularly beneficial for larger bit-width additions.

While not as complex as advanced adders like Kogge-Stone or Brent-Kung, the CBA offers a practical compromise between performance and resource usage. Its efficient design makes it suitable for various digital systems where both speed and moderate hardware complexity are important. Overall, the Carry Bypass Adder stands out as a versatile and efficient choice for high-speed addition tasks.

8 Simulation Results

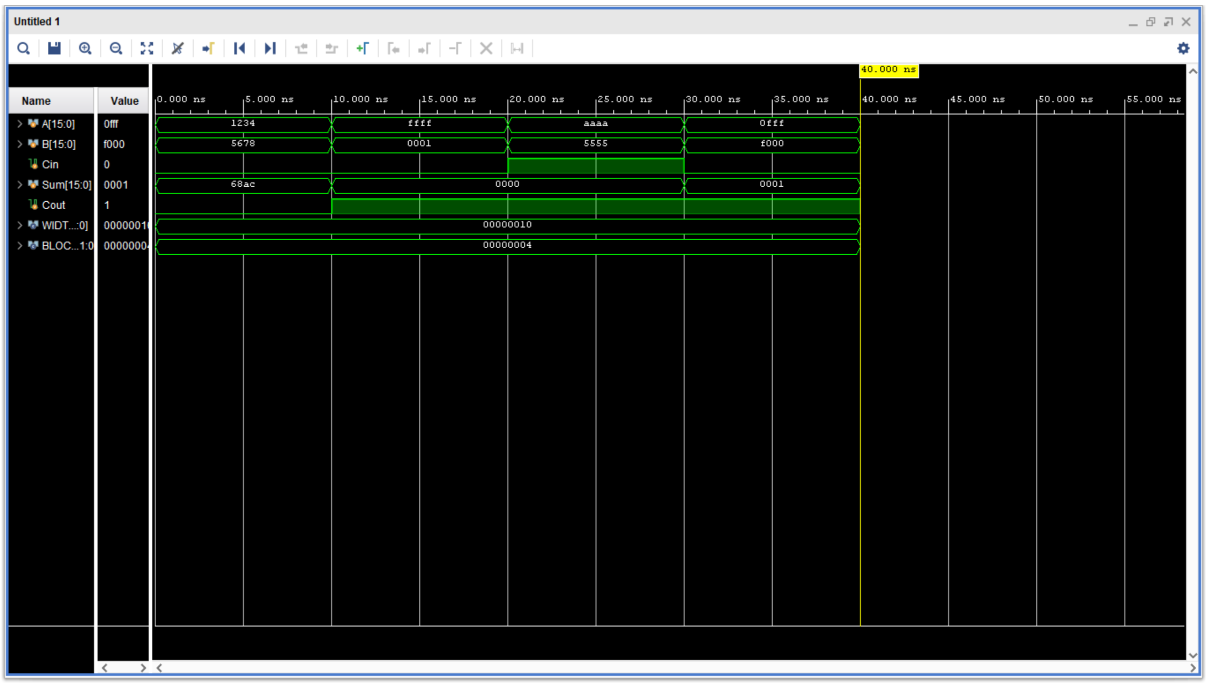

Figure 3: Simulation of Carry Bypass Adder

9 References

1. H. M. K. G. C. Smith, *Digital Design and Computer Architecture*, Wiley, 2008.
2. R. H. Katz, *Contemporary Logic Design*, 2nd ed., Addison-Wesley, 2008.
3. J. P. Hayes, *Digital Design: An Introduction to the Logic of Digital Circuits*, McGraw-Hill, 2004.
4. R. S. H. C. Chien, *High-Speed Digital Design: A Handbook of Black Magic*, Prentice-Hall, 1999.
5. D. P. H. M. Morris, *Digital Logic Design: Principles and Practices*, Springer, 2010.