Project 25 : Redundant Binary Adder

A Comprehensive Study of Advanced Digital Circuits

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

Contents

1 Project Overview

A Redundant Binary Adder (RBA) is a type of binary adder used in digital arithmetic to eliminate the problem of carry propagation, which can slow down the speed of addition in conventional binary adders. RBAs use a redundant number representation, meaning they allow multiple representations of the same number to simplify and speed up the addition process.

2 Redundant Binary Adder

2.1 Description

A Redundant Binary Adder (RBA) is a specialized form of digital adder that uses redundant binary representation to perform addition operations more efficiently, especially in high-speed applications. The key feature of this adder is that it eliminates the need for carry propagation between the bits, which is a limiting factor in traditional binary adders like ripple-carry adders. This makes RBAs faster and more suitable for applications requiring rapid computation, such as digital signal processing (DSP), floating-point arithmetic, and processors.

2.2 Key Concepts of Redundant Binary Adder:

- **Redundant Binary Representation:**

In traditional binary systems, each digit (bit) can only take values 0 or 1. In a redundant binary system, each digit can take values from a larger set, typically -1, 0, or 1. This redundant encoding eliminates the need for carry propagation across multiple bit positions, as each digit can locally handle small overflows.

- **Benefits of Redundancy:**

Carry-Free Addition: In a redundant system, the addition of two numbers can be done in parallel without needing to propagate carries from lower to higher-order bits, as opposed to conventional binary adders where carries need to ripple through all bits.
Speed: By eliminating the need for carry propagation, RBAs enable faster addition, especially in large-width numbers or in high-speed applications like digital signal processing (DSP) and floating-point units in CPUs.

- **Number Representation:**

Redundant binary numbers are represented using signed digits. A typical signed-digit set could be {-1, 0, 1}, represented as 1(for -1), 0, and 1.
Example: In a redundant binary system, the decimal number 3 can be represented as either 11 (conventional binary) or 101, where 1 is -1.

- **Addition Process:**

When two numbers are added in a redundant binary system, each digit position is handled independently, and no carry propagation is required between digit positions. The result may be in an intermediate form that can later be converted to a standard binary form if needed. Intermediate sums can have multiple representations, making it easier to balance overflow conditions without propagating a carry.

- **Redundant Binary to Standard Binary Conversion:**

After performing operations in the redundant binary format, the result might need to be converted back into standard binary form. This process is typically done using a separate circuit but can be done quickly and efficiently.

2.3 RTL Code

Listing 1: Redundant Binary Adder

```
1
2  module Redundant_Binary_Adder #(parameter WIDTH = 16) (
3      input  logic [WIDTH-1:0] a,     // Input operand A
4      input  logic [WIDTH-1:0] b,     // Input operand B
5      input  logic               cin, // Carry in
6      output logic [WIDTH:0] sum,     // Redundant sum output (1 extra
           bit)
7      output logic               cout // Carry out
8  );
9
10     // Internal signals
11     logic [WIDTH:0] carry;          // Carry chain
12     logic [WIDTH-1:0] g, p;         // Generate and propagate
13
14     assign carry[0] = cin;
15
16     // Step 1: Generate and Propagate signals
17     genvar i;
18     generate
19         for (i = 0; i < WIDTH; i++) begin
20             assign g[i] = a[i] & b[i];      // Carry generate
21             assign p[i] = a[i] ^ b[i];      // Propagate signal (xor)
22         end
23     endgenerate
24
25     // Step 2: Carry and sum calculation (Redundant representation)
26     generate
27         for (i = 0; i < WIDTH; i++) begin
28             assign carry[i+1] = g[i]  (p[i] & carry[i]); // Carry
                   propagation
29             assign sum[i] = p[i] ^ carry[i];              // Sum
                   calculation
30         end
31     endgenerate
32
33     // Step 3: Assign carry out
34     assign sum[WIDTH] = carry[WIDTH];  // Redundant carry-out bit
35     assign cout = carry[WIDTH];        // Final carry-out
36
37 endmodule
```

2.4 Testbench

Listing 2: Redundant Binary Adder

```
1
2  module tb_Redundant_Binary_Adder;
3
4      parameter WIDTH = 16;
5
6      // Testbench signals
7      logic [WIDTH-1:0] a, b;
8      logic cin;
```

```
    logic [WIDTH:0] sum;  // Redundant sum output (1 extra bit)
    logic cout;

    // Instantiate the Redundant Binary Adder
    Redundant_Binary_Adder #(WIDTH) uut (
        .a(a),
        .b(b),
        .cin(cin),
        .sum(sum),
        .cout(cout)
    );

    // Test stimulus
    initial begin
        $display("Starting Testbench for Redundant Binary Adder");

        // Test Case 1
        a = 16'hA5A5; // 1010010110100101
        b = 16'h5A5A; // 0101101001011010
        cin = 0;
        #10;
        $display("Test 1: a = %h, b = %h, cin = %b -> sum = %h, cout =
            %b", a, b, cin, sum, cout);

        // Test Case 2
        a = 16'hFFFF; // 1111111111111111
        b = 16'h0001; // 0000000000000001
        cin = 0;
        #10;
        $display("Test 2: a = %h, b = %h, cin = %b -> sum = %h, cout =
            %b", a, b, cin, sum, cout);

        // Test Case 3
        a = 16'h1234;
        b = 16'h4321;
        cin = 1;
        #10;
        $display("Test 3: a = %h, b = %h, cin = %b -> sum = %h, cout =
            %b", a, b, cin, sum, cout);

        $display("Testbench complete.");
        $finish;
    end

endmodule
```

3 How it works ?

Working of a Redundant Binary Adder:

- The redundant binary adder takes two input binary numbers in redundant form and adds them bit by bit.
- Each bit position in the result is computed independently of others, reducing the need for carry propagation.
- The intermediate result may use signed digits, such as -1, 0, or 1, and is typically stored in a carry-save format.

- The final result, after all additions are complete, can be converted into a standard binary format if needed.

3.1 Simulation

Figure 1: Simulation of Redundant Binary Adder

3.2 Schematic

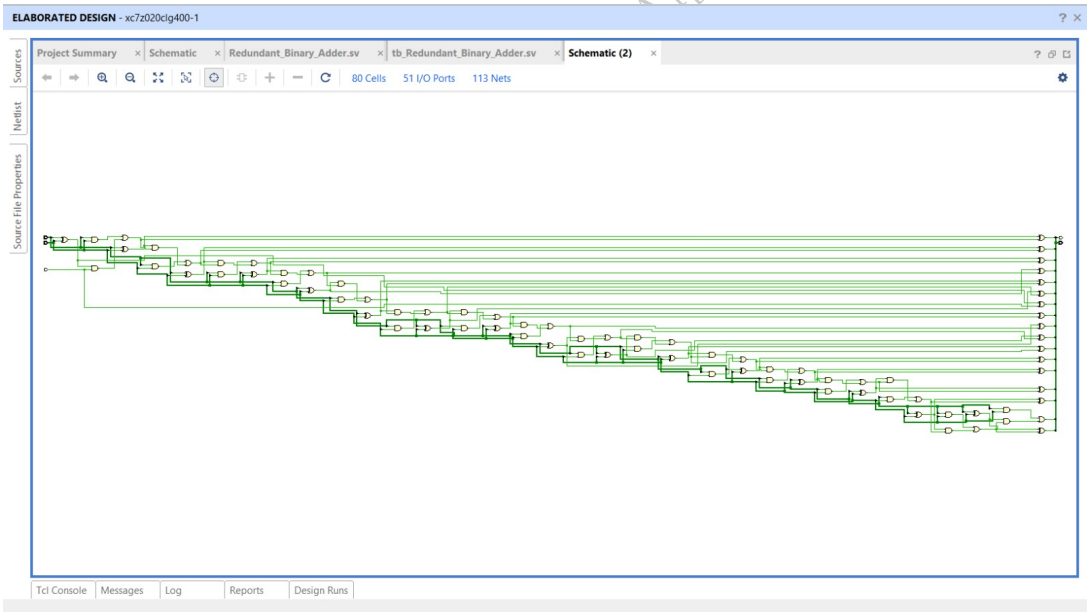

Figure 2: Schematic of Redundant Binary Adder

Explanation

A Redundant Binary Adder (RBA) is a high-speed adder designed to eliminate the delay caused by carry propagation in conventional binary adders. By using a redundant binary representation where digits can take values such as -1, 0, and 1, RBAs allow faster, carry-free addition. They are particularly useful in high-performance computing applications like digital signal processing, floating-point arithmetic, and situations where multiple operands must be added quickly. However, their increased complexity and the need for conversion back to standard binary form are trade-offs to consider.

3.3 Comparison with Other Adders:

- **Redundant Binary Adder (RBA):**

Carry Propagation: None; operates with redundant binary representation.
Speed: Very fast due to carry-free addition.
Complexity: High complexity, requires handling of signed digits.
Parallelism: Highly parallel as each bit is processed independently.
Area: Requires more hardware compared to simpler adders.
Applications: Suitable for high-speed arithmetic in DSP, floating-point units, and multi-operand addition.

- **Ripple Carry Adder (RCA):**

Carry Propagation: Full ripple propagation through all bit positions.
Speed: Slowest among adders due to carry propagation delay.
Complexity: Simple and easy to implement.
Parallelism: Very low, each bit depends on the carry from the previous bit.
Area: Requires the least hardware.
Applications: Used in low-complexity, low-speed applications.

- **Carry-Lookahead Adder (CLA):**

Carry Propagation: Partial carry propagation; generates carries in advance.
Speed: Faster than RCA but still limited by bit-width.
Complexity: More complex than RCA due to lookahead carry logic.
Parallelism: Moderate, as carry lookahead logic allows some parallelism.
Area: Higher hardware cost compared to RCA.
Applications: Used where medium-speed addition is required without excessive complexity.

- **Carry-Save Adder (CSA):**

Carry Propagation: Delays carry propagation until all operands are added.
Speed: Fast, especially in multi-operand addition scenarios.
Complexity: Moderate complexity; simpler than RBA but more complex than RCA.
Parallelism: High parallelism, suitable for multi-operand addition.
Area: More hardware than RCA but generally less than RBA.
Applications: Common in multiplier circuits, ALUs, and scenarios requiring multi-operand addition efficiency.

3.4 Advantages of the Redundant Binary Adder:

- **Speed:**

By eliminating carry propagation, RBAs can perform addition faster than conventional adders, especially for large bit-width numbers.

- **Parallel Processing:**

Since each bit position is processed independently, RBAs can be easily parallelized, making them ideal for high-performance computing tasks.

- **Multi-Operand Addition:**

RBAs are well-suited for adding multiple numbers simultaneously, as they avoid the cumulative delay that would otherwise result from carry propagation in standard adders.

3.5 Disadvantages of the Redundant Binary Adder:

- **Complexity:**

RBAs are more complex in terms of circuit design and implementation compared to simple adders like ripple-carry or carry-lookahead adders.

- **Conversion Overhead:**

After performing redundant binary addition, the result often needs to be converted to standard binary format, which adds an additional step in the process.

3.6 Applications of the Redundant Binary Adder:

- **High-Speed Arithmetic:**

RBAs are used in systems that require extremely fast addition, such as microprocessors, floating-point units (FPUs), and digital signal processors (DSPs).

- **Digital Signal Processing (DSP):**

In DSP applications, where many numbers must be added in real-time, RBAs are crucial for ensuring that the computations are performed quickly and efficiently.

- **Multiplication and Division:**

RBAs are often used in combination with other arithmetic units, such as multipliers and dividers, to speed up operations that involve multiple additions.

- **Floating-Point Arithmetic:**

RBAs are also used in floating-point arithmetic units, where large numbers are added or subtracted frequently, and avoiding carry propagation significantly boosts performance.

4 Results

4.1 Synthesis Design

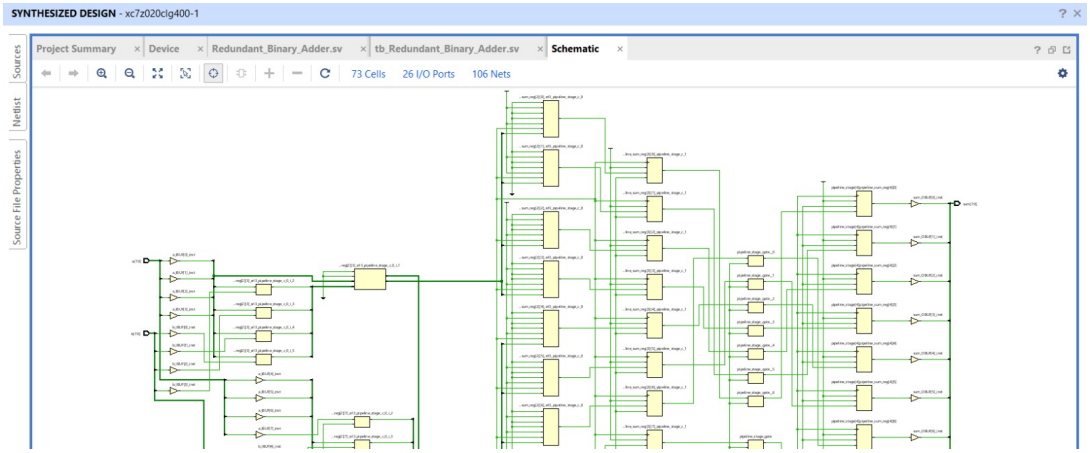

Figure 3: Synthesis Design of Redundant Binary Adder

5 FAQs

- **Q1. What is a Redundant Binary Adder?**

Answer: A Redundant Binary Adder is a digital circuit that performs addition using a redundant binary representation, allowing for faster addition by eliminating carry propagation delays.

- **Q2. How does this Redundant Binary Adder differ from a conventional binary adder?**

Answer: Unlike conventional binary adders that propagate carry bits through all bit positions, this adder uses redundant binary representation where each bit can represent -1, 0, or 1, reducing the need for carry propagation.

- **Q3. What are the inputs and outputs of the module?**

Answer: Inputs: a (operand A), b (operand B), cin (carry-in). Outputs: sum (redundant sum output), cout (carry-out).

- **Q4.How is carry propagation handled in this adder?**

Answer: Carry propagation is managed using generate blocks in Verilog. The adder computes generate (g) and propagate (p) signals for each bit position, and then calculates the carry chain and sum based on these signals.

- **Q5. Explain the purpose of the carry signal in the code.**

Answer: The carry signal forms the carry chain in the adder. It starts with carry[0] initialized by cin and is propagated through each bit position (carry[i+1]) based on the generate (g) and propagate (p) signals.

- **Q6. How is the sum calculated in this adder?**

Answer: The sum for each bit position (sum[i]) is computed using the propagate signal (p[i]) XORed with the current carry (carry[i]).

- **Q7. Why is an extra bit (sum[WIDTH]) used in the sum output?**

Answer: The extra bit (sum[WIDTH]) is used for redundant representation. It captures the final carry (carry[WIDTH]), ensuring the complete result is represented in a redundant binary format.

- **Q8. What are the advantages of using a Redundant Binary Adder?**

Answer: Advantages include faster addition due to reduced carry propagation, suitability for high-speed arithmetic operations, and the ability to handle multi-operand addition efficiently.

- **Q9. Can you describe the generate block in Verilog and its purpose in this code?**

Answer: The generate block allows for the creation of multiple instances of logic based on a parameter (WIDTH in this case). It is used here to generate logic for computing generate and propagate signals (g and p) and for calculating the carry chain and sum.

- **Q10. How would you modify this code to support signed binary addition?**

Answer: To support signed binary addition, you would modify the generate blocks to handle signed digits (-1, 0, 1). This typically involves adjusting how the propagate and generate signals (p and g) are computed and how the sum (sum) is derived based on these signals.