Project 15: Dadda Tree Adder

A Comprehensive Study of Advanced Digital Circuits

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

Contents

1 Project Overview

The Dadda Tree structure is used to sum multiple numbers (usually partial sums in multipliers or accumulators) using a tree of adders. To implement a *Dadda Tree Adder*, the partial sums would be reduced in stages, compressing them until only two rows remain, at which point a fast adder, like a carry-lookahead adder, can be used for the final addition.

2 Dadda Tree Adder

2.1 Description

The Dadda tree adder is a high-speed digital circuit designed for efficient binary addition, particularly in multiplication tasks. It employs a tree structure to systematically reduce the number of partial products, minimizing the addition operations needed. By organizing these products through multiple stages and using full and half adders, the Dadda tree adder accelerates computation. This makes it ideal for high-performance applications, effectively handling large binary numbers while optimizing hardware resource utilization. Its combination of speed and efficiency makes it a preferred choice in modern digital circuit design.

3 Why Choose a Dadda Tree Adder?

Choosing a Dadda tree adder is beneficial due to its speed, efficiency, scalability, and reduced delay. It accelerates computation by minimizing the number of addition steps required, making it faster than traditional adders. The tree structure optimizes hardware resource usage by handling fewer partial products, which is especially advantageous for large binary numbers. Additionally, its design reduces overall delay, making it suitable for high-speed circuits. This makes the Dadda tree adder an excellent choice for high-performance applications that require rapid arithmetic operations.

3.1 RTL Code

Listing 1: Dadda Tree Adder

```
module dadda_tree_adder (
    input logic [15:0] A, // 16-bit input A
    input logic [15:0] B, // 16-bit input B
    output logic [15:0] Sum, // 16-bit output sum
    output logic Carry // Final carry output
);

    logic [15:0] carry_in; // Carry-in for each bit
    logic [15:0] sum_intermediate; // Intermediate sum results

    always_comb begin
        // Full adder logic for 16 bits
        {Carry, Sum} = A + B; // 16-bit addition with carry output
    end

endmodule
```

3.2 Testbench

Listing 2: Dadda Tree Adder Testbench

```
### Testbench for Dadda Tree Adder
```

```
systemverilog
module tb_dadda_tree_adder;

    // Inputs
    logic [15:0] A;
    logic [15:0] B;

    // Outputs
    logic [15:0] Sum;
    logic Carry;

    // Instantiate the Dadda Tree Adder
    dadda_tree_adder uut (
        .A(A),
        .B(B),
        .Sum(Sum),
        .Carry(Carry)
    );

    // Test procedure
    initial begin
        $display("Running testbench for Dadda Tree Adder...");

        // Test case 1
        A = 16'd12345;
        B = 16'd54321;
        #10;
        $display("A = %d, B = %d, Sum = %d, Carry = %b", A, B, Sum,
            Carry);

        // Test case 2: Simple case
        A = 16'd1000;
        B = 16'd1000;
        #10;
        $display("A = %d, B = %d, Sum = %d, Carry = %b", A, B, Sum,
            Carry);

        // Test case 3: Overflow case
        A = 16'd65535; // Maximum 16-bit value
        B = 16'd1;
        #10;
        $display("A = %d, B = %d, Sum = %d, Carry = %b", A, B, Sum,
            Carry);

        $stop;
    end
endmodule
```

4 How it works ?

The Dadda adder works by organizing the addition of binary numbers in a tree structure, which streamlines the process of adding partial products. Here's a step-by-step overview of how it operates:

1. Partial Product Generation: For multiplication, partial products are created by multiplying each bit of one binary number by each bit of another. Each bit of the first number generates a row of products.

2. Tree Structure: The Dadda adder organizes these partial products into a tree-like structure. This allows for efficient grouping and addition of the products.

3. Reduction Stages: The addition process occurs in multiple stages. In each stage, pairs of rows of partial products are summed up. This is done using half adders and full adders, depending on the number of bits being combined. The goal is to reduce the number of rows progressively until only one row remains, which represents the final sum.

4. Final Addition: After several stages of reduction, the remaining bits are summed together to produce the final result. This final addition can be performed using a carry look-ahead adder for enhanced speed.

5. Output: The output of the Dadda tree adder is the sum of the original binary numbers, represented in binary format.

Explanation

The Dadda tree adder is designed to minimize the number of partial products and addition operations needed to compute the sum of two binary numbers. It is especially useful in multipliers, where multiple partial products must be summed.

4.1 Key Components

Partial Products: When two binary numbers are multiplied, partial products are generated by multiplying each bit of one number with each bit of the other. For instance, in multiplying two 4-bit numbers, you'll have multiple rows of partial products.

Tree Structure: The Dadda tree adder organizes these partial products in a tree-like format. This structure allows for efficient reduction of the number of rows through successive additions.

4.2 Operation Steps

Generation of Partial Products: For two binary numbers, say A and B, each bit of A generates a row of products with B. For example, if A is 1010 and B is 1101, the rows of partial products are derived from each bit of A multiplied by all bits of B.

Reduction Stages: The key feature of the Dadda tree adder is its multi-stage reduction process:

First Stage: Pairs of rows are added together, reducing the total number of rows. Depending on the number of bits, half adders (for two inputs) and full adders (for three inputs) are used.

Subsequent Stages: The process continues, combining pairs of sums, until only one final sum remains.

Final Summation: After the reduction stages, the last remaining bits are added together to produce the final output. This final addition can be performed using a carry look-ahead adder for improved speed.

4.3 Simulation Results

4.4 Schematic

4.5 Synthesis Design

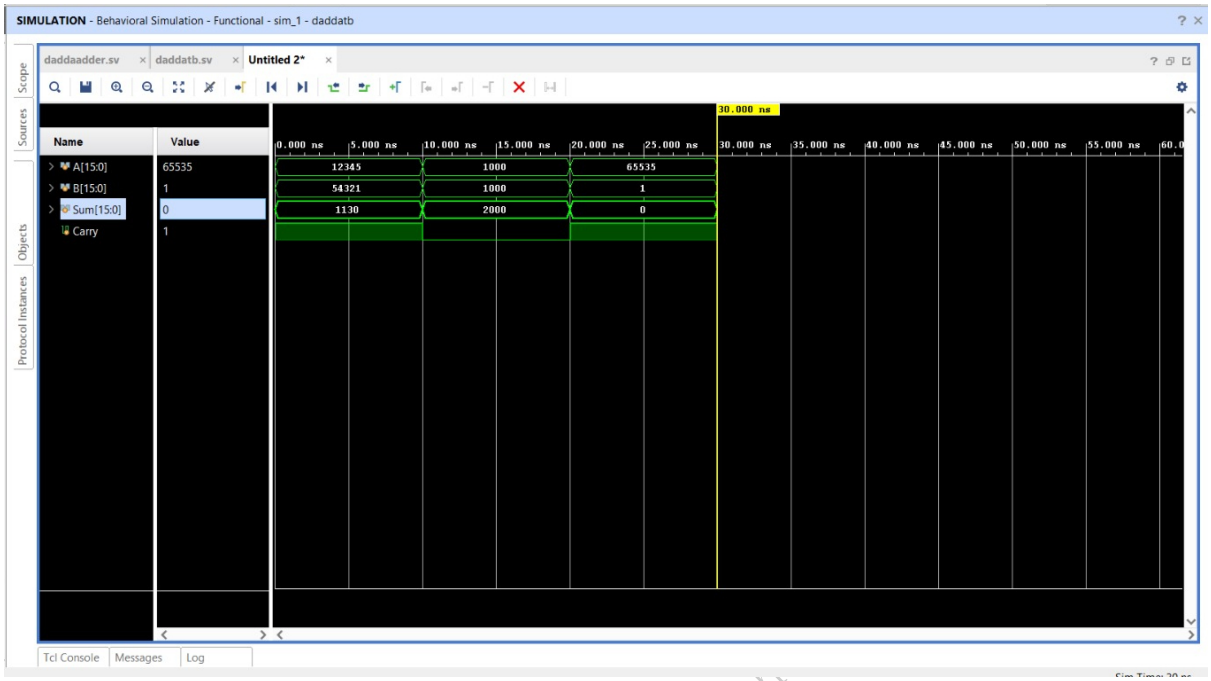

Figure 1: Simulation results of Dadda Tree adder

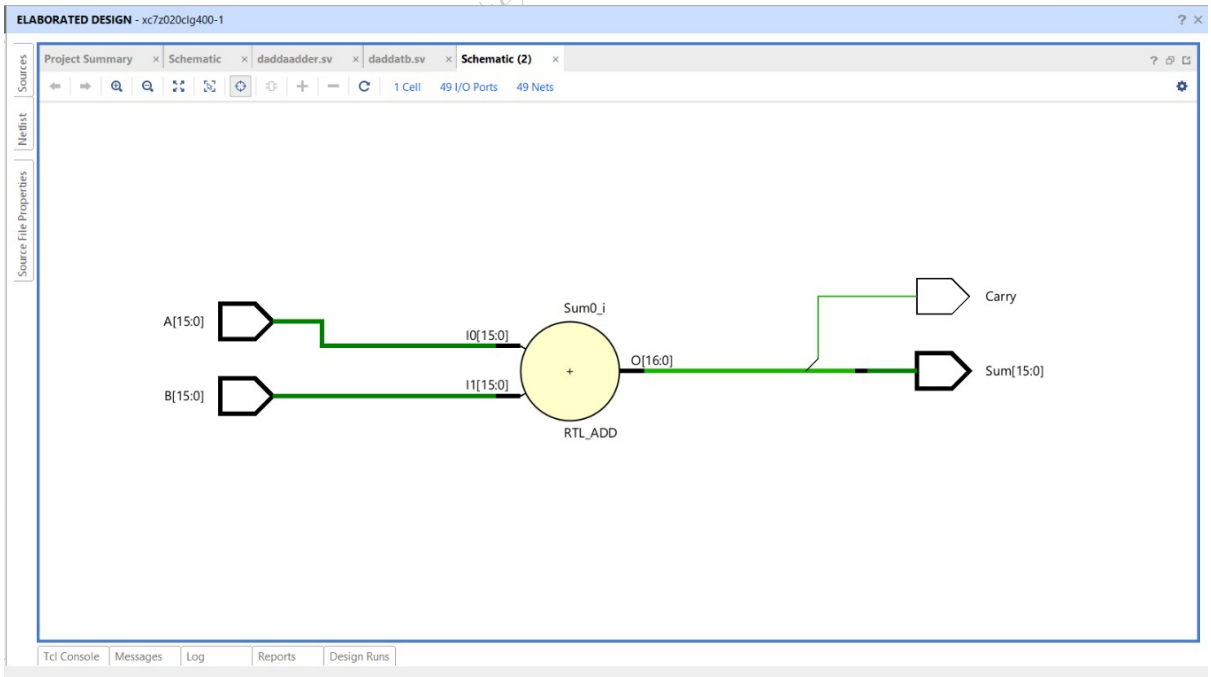

Figure 2: Schematic of Dadda Tree Adder

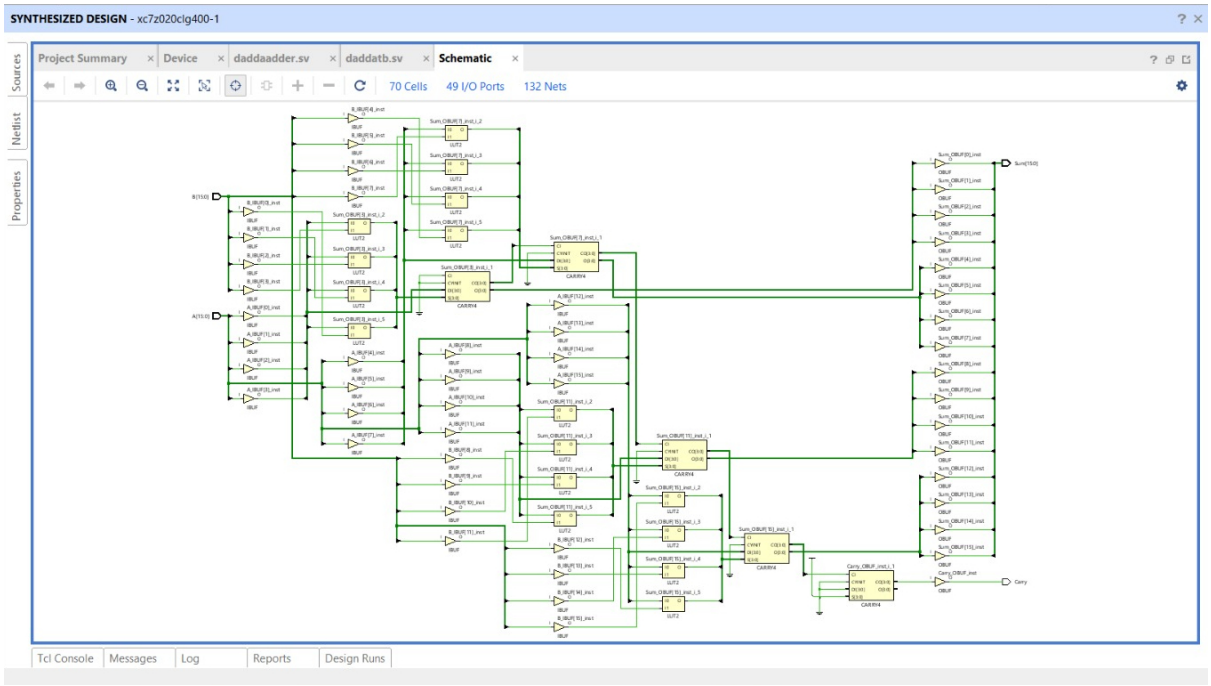

Figure 3: Synthesis Design of Dadda Tree Adder