Design and Implementation of a 4-Bit Wallace Tree Multiplier (RTL Level)

1 Introduction

This project implements a 4×4 Wallace Tree Multiplier using Verilog HDL.
The Wallace Tree algorithm improves multiplication speed by reducing partial products using multiple parallel layers of:

Half Adders

Full Adders

Carry-Save Reduction

The goal is to minimize the critical path delay compared to the conventional array multiplier.

2 Architecture

A 4×4 Wallace Tree multiplier consists of three major steps:

1. Partial Product Generation

Each bit of operand *B* AND each bit of *A*.

2. Reduction Tree Using Wallace Algorithm

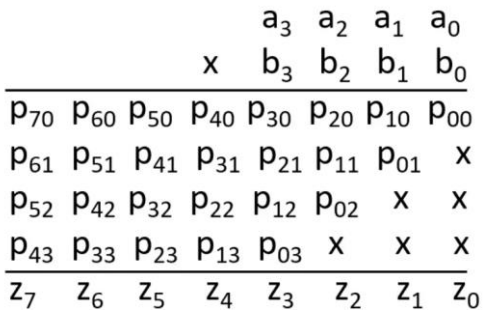
Reduces the height of partial product rows using HAs and FAs:

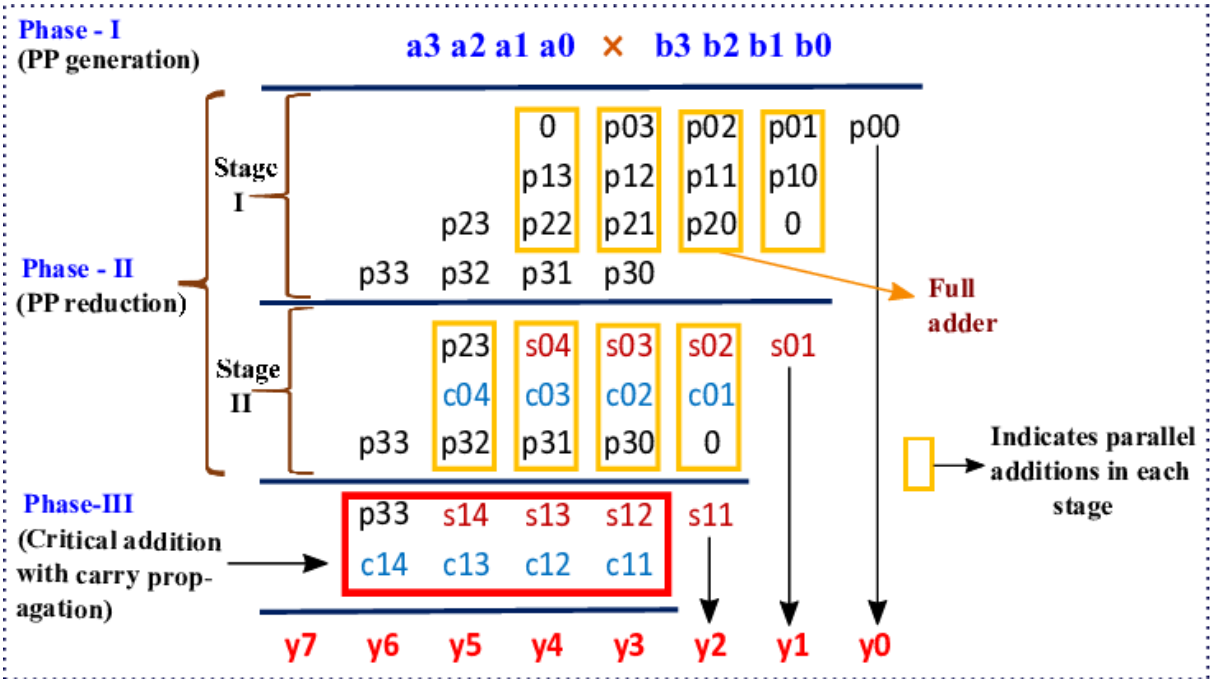
- Stage 1 reduction
- Stage 2 reduction

3. Final Addition

Adds the reduced results using a final ripple carry adder (implemented using sum1 + sum2 in my code).

3 Block Diagram





4 Verilog Implementation

4.1 Top Module:

```
// Wallace Tree 4x4 Multiplier

module parallel_multiplier(
   input  [3:0] A,
   input  [3:0] B,
   output [7:0] P
);
   wire [3:0] pp0, pp1, pp2, pp3;
   wire [7:0] p;

   // Step 1: Generate Partial Products
   assign pp0 = A & {4{B[0]}};
   assign pp1 = A & {4{B[1]}};
   assign pp2 = A & {4{B[2]}};
   assign pp3 = A & {4{B[3]}};

   //sum using wallce tree
   //stage 1:
   wire s11,c11,s12,c12,s13,c13,s14,c14,s15,c15;
   half_adder ha0(pp0[1],pp1[0],s11,c11);
   full_adder fa0(pp0[2],pp1[1],pp2[0],s12,c12);
   full_adder fa1(pp0[3],pp1[2],pp2[1],s13,c13);
   half_adder ha1(pp1[3],pp2[2],s14,c14);
   half_adder ha2(pp2[3],pp3[2],s15,c15);
```

```
    //stage 2:
    wire s21,c21,s22,c22,s23,c23,s24,c24,s25,c25;
    half_adder ha3(c11,s12,s21,c21);
    full_adder fa3(pp3[0],c12,s13,s22,c22);
    full_adder fa4(pp3[1],c13,s14,s23,c23);
    half_adder ha4(c14,s15,s24,c24);
    half_adder ha5(pp3[3],c15,s25,c25);

    //after reduction of two rows calculating the sum
    wire [7:0] sum1,sum2;
    assign sum1 = {1'b0, c24, c23, c22, c21, 3'b000};
    assign sum2 = {c25, s25, s24, s23, s22, s21, s11, pp0[0]};
    assign P = sum1 + sum2;

endmodule
```

4.2 Half adder module

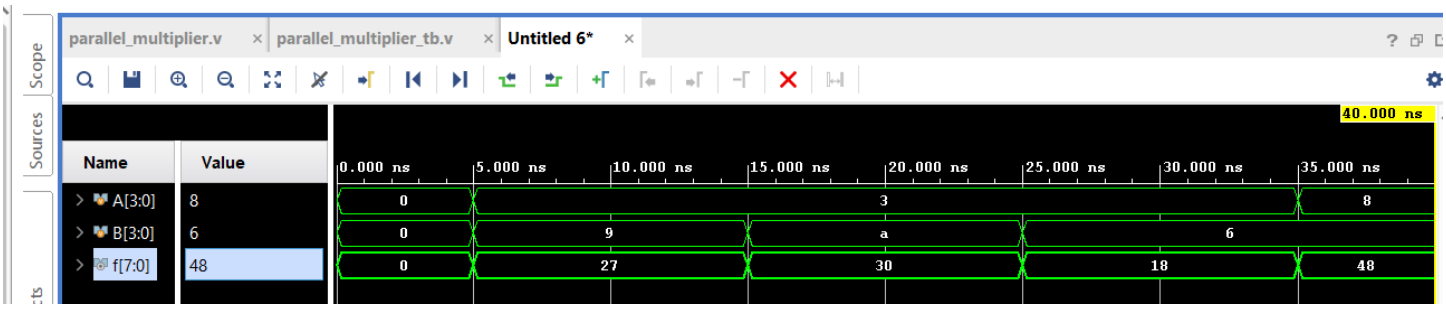
```
// half Adder Module
module half_adder(input a, input b, output sum, output carry);
    assign sum = a ^ b;
    assign carry = a & b;
endmodule
```

4.3 Full adder module

```
// full Adder Module
module full_adder(input a, input b, input cin, output sum, output carry);
    assign sum = a ^ b ^ cin;
    assign carry = (a & b) | (b & cin) | (a & cin);
endmodule
```

5 Result:

5.1 Simulation waveform:



5.2 Synthesis result in Vivado:

Timing analysis:

Maximum Path Delays (Slow Corner):

Path	Delay (ns)	Logic / Route Breakdown
B[1] → P[6]	11.092 ns	Logic 4.96 ns (45%), Route 6.13 ns (55%), 7 logic levels
B[1] → P[7]	10.166 ns	Logic 5.05 ns, Route 5.11 ns, 7 logic levels
A[2] → P[2]	7.507 ns	Logic 4.10 ns, Route 3.41 ns, 4 logic levels
A[0] → P[0]	7.267 ns	Logic 3.88 ns, Route 3.39 ns, 3 logic levels

Minimum Path Delays (Fast Corner):

Path	Delay (ns)	Logic / Route Breakdown
B[0] → P[1]	2.16 ns	Logic 1.33 ns, Route 0.83 ns
A[0] → P[2]	2.212 ns	Logic 1.46 ns, Route 0.75 ns
B[0] → P[0]	2.222 ns	Logic 1.39 ns, Route 0.83 ns
B[3] → P[7]	2.346 ns	Logic 1.45 ns, Route 0.89 ns

Observation from timing analysis:

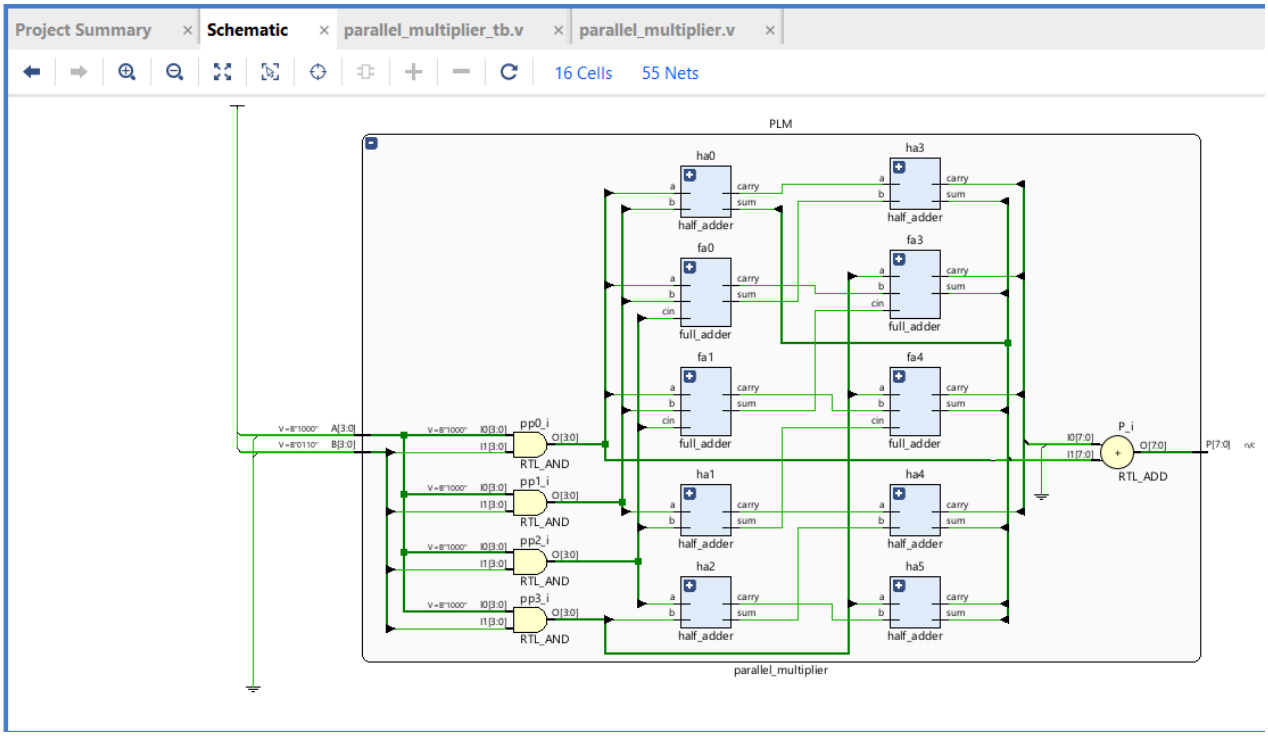
The critical path is B[1] → P[6] with 11.09 ns delay → maximum operating frequency ~90 MHz (1 / 11.092 ns).

Power Analysis

Total On-Chip Power: 4.181 W

- Dynamic Power: 4.112 W
- Static Power: 0.069 W

6 RTL Schematic of Wallace Tree:



7 Future Work

In the future, I plan to extend this design to an 8-bit Wallace Tree Multiplier.
This involves increasing the number of partial product rows and adding more stages of carry-save reduction to manage the increased bit-width. The design can also be enhanced by exploring optimizations such as Booth encoding, pipeline stages for higher throughput