Design and Simulation of a 4x4 High-Speed Vedic Multiplier using Urdhva-Tiryakbhyam Sutra

Vishnu Vardhan Reddy Yerramala
Rajiv Gandhi University of Knowledge Technologies,
Nuzvid, Andhra Pradesh, India
Email: n220291@rguktn.ac.in

Abstract – This report details the design and simulation of a high-speed 4x4 Vedic Multiplier. The design is based on the ancient Indian mathematical sutra "Urdhva-Tiryakbhyam," which translates to "Vertically and Crosswise." The architecture is implemented using Verilog HDL and simulated within the eSim open-source EDA environment. The project aims to demonstrate the efficiency of Vedic algorithms in reducing propagation delay by generating partial products in parallel. Functional verification was achieved through transient analysis in Ngspice, proving the design's suitability for high-speed digital signal processing applications.

Keywords – Urdhva-Tiryakbhyam, eSim, Verilog HDL, Hierarchical Architecture, Propagation Delay, Digital-Analog Bridge, Partial Product Generation

1. Introduction

In the realm of VLSI design, the multiplier is a fundamental building block for ALUs, DSPs, and microprocessors. Traditional multipliers like the Booth or Wallace-Tree multipliers often involve complex carry propagation chains that limit performance. Vedic mathematics offers a unique approach to arithmetic that simplifies complex calculations into smaller, concurrent steps. This project focuses on the 4x4 Vedic Multiplier, which decomposes the multiplication process into smaller 2x2 blocks. By utilizing eSim—an integrated tool combining KiCad, Ngspice, and Verilog—this project provides a complete hardware-to-simulation workflow for validating Vedic logic.

2. Architecture and Working Principle

2.1 The Urdhva-Tiryakbhyam Algorithm

The primary working principle is the parallel generation of partial products. For a 4-bit multiplication, the inputs A and B are divided into lower and upper 2-bit segments:

- A = [A_{high}, A_{low}] where A_{high} = A[3:2] and A_{low} = A[1:0]
- B = [B_{high}, B_{low}] where B_{high} = B[3:2] and B_{low} = B[1:0]

2.2 4x4 Structural Architecture

The architecture consists of four 2x2 multiplier blocks, as implemented in the provided Verilog code:

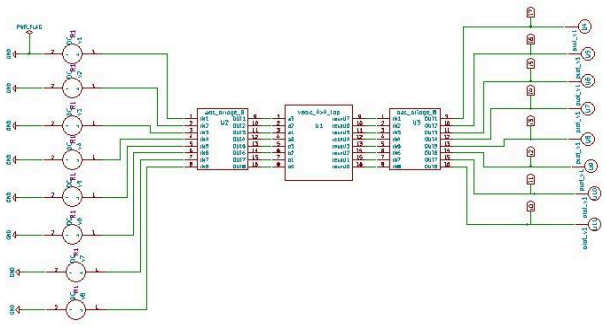
1. **Block 0 (q0):** Multiplies (A_{low} x B_{low}).
2. **Block 1 (q1):** Multiplies (A_{high} x B_{low}).
3. **Block 2 (q2):** Multiplies (A_{low} x B_{high}).
4. **Block 3 (q3):** Multiplies (A_{high} x B_{high}).

2.3 Summation and Alignment

To obtain the final 8-bit result, the partial products are shifted according to their binary weight:

- q0 is the LSB part (no shift).
- q1 and q2 are middle-order products, shifted left by 2 positions (<<2).
- q3 is the MSB part, shifted left by 4 positions (<<4).
  The final result is obtained by the summation: Result = q0 + (q1<<2) + (q2<<2) + (q3<<4).

3. Implementation in eSim



The project was developed following the standard FOSSEE eSim workflow:

- **Schematic Entry:** In KiCad, a schematic was created using a sub-circuit block for the Verilog code.
- **Bridge Integration:** DAC (Digital-to-Analog) and ADC (Analog-to-Digital) bridges were added to interface the logic with analog components and power sources.
- **Model Generation:** The Verilog file was converted into a Ngspice-compatible model using the "Makerchip" or "GHDL" utility within eSim.

4. Verilog Description

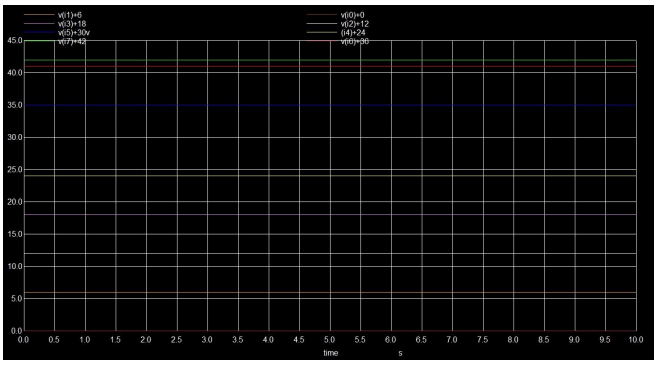
```
module vedic_4x4_top (
    input [3:0] a,
    input [3:0] b,
    output [7:0] result
);
    // Partial Products (2x2 Multiplier results)
    wire [3:0] q0, q1, q2, q3;
    // --- Block 0: a[1:0] * b[1:0] ---
    assign q0[0] = a[0] & b[0];
    assign q0[1] = (a[1] & b[0]) ^ (a[0] & b[1]);
    wire c0_1    = (a[1] & b[0]) & (a[0] & b[1]);
    assign q0[2] = (a[1] & b[1]) ^ c0_1;
    assign q0[3] = (a[1] & b[1]) & c0_1;
    // --- Block 1: a[3:2] * b[1:0] ---
    assign q1[0] = a[2] & b[0];
    assign q1[1] = (a[3] & b[0]) ^ (a[2] & b[1]);
    wire c1_1    = (a[3] & b[0]) & (a[2] & b[1]);
    assign q1[2] = (a[3] & b[1]) ^ c1_1;
    assign q1[3] = (a[3] & b[1]) & c1_1;
    // --- Block 2: a[1:0] * b[3:2] ---
    assign q2[0] = a[0] & b[2];
    assign q2[1] = (a[1] & b[2]) ^ (a[0] & b[3]);
    wire c2_1    = (a[1] & b[2]) & (a[0] & b[3]);
    assign q2[2] = (a[1] & b[3]) ^ c2_1;
    assign q2[3] = (a[1] & b[3]) & c2_1;
    // --- Block 3: a[3:2] * b[3:2] ---
    assign q3[0] = a[2] & b[2];
    assign q3[1] = (a[3] & b[2]) ^ (a[2] & b[3]);
    wire c3_1    = (a[3] & b[2]) & (a[2] & b[3]);
    assign q3[2] = (a[3] & b[3]) ^ c3_1;
    assign q3[3] = (a[3] & b[3]) & c3_1;
    // --- Final Addition Stages (Corrected Alignment) ---
    // The Vedic method requires specific shifting:
    // Result = q0 + (q1 << 2) + (q2 << 2) + (q3 << 4)

    wire [7:0] stage1 = q0 + (q1 << 2);
    wire [7:0] stage2 = stage1 + (q2 << 2);
    wire [7:0] final_res = stage2 + (q3 << 4);
    assign result = final_res;
endmodule
```

The code employs a structural and behavioral hybrid approach. Each 2x2 block is explicitly defined using gate-level logic:

- **AND Gates:** Generate the individual bit-products.
- **XOR Gates:** Perform the half-adder logic required for the 2x2 cross-multiplication.
- **Addition Stages:** The code uses intermediate wires (`stage1`, `stage2`) to ensure that carries from the shifts are correctly propagated to the final 8-bit output.
-

5. Result and Discussion

The design was verified using **Transient Analysis**. Inputs were applied to the digital pins, and the output was monitored with multimeters.



- **Example Case:** With inputs 1100_2 (12) and 1000_2 (8), the output waveform displayed 01100000_2 (96).

Test Case	Input A (Binary)	Input B (Binary)	Decimal Math	Expected Output (Result)	Why this test?
1. Zero Case	`0000`	`1010`	0×10	`00000000` **(0)**	Ensures AND gates in 2x2 blocks work correctly.
2. Identity	`1101`	`0001`	13×1	`00001101` **(13)**	Verifies signal pass-through.
3. Small Values	`0011`	`0010`	3×2	`00000110` **(6)**	Tests only the lower 2x2 block ($q0$).
4. Middle Range	`0101`	`0110`	5×6	`00011110` **(30)**	Tests the $<< 2$ shifts and intermediate adders.
5. Max Value	`1111`	`1111`	15×15	`11100001` **(225)**	Tests the $<< 4$ shift and full carry propagation.

6. Technology used in esim

The 4x4 Vedic Multiplier is implemented in **eSim** using **NgVeri**, a powerful tool that facilitates mixed-signal simulation by bridging Verilog-based digital logic with the analog Ngspice engine. By importing the Verilog code into the NgVeri tab, a Ngspice-compatible C++ model is automatically generated, allowing the **Urdhva-Tiryakbhyam** mathematical logic to function as a hardware block within the schematic. To ensure seamless operation, ADC and DAC bridges are utilized to translate continuous analog voltages into discrete digital signals and back, enabling high-accuracy transient analysis of the 8-bit output. This integration demonstrates a sophisticated top-down design flow where high-level behavioral modeling is verified through precise spice-level simulation.

7. Delay Analysis

In traditional multipliers, the delay increases linearly with the number of bits. In this Vedic implementation:

- **Partial Product Delay:** Constant (logic depth of one 2x2 block).
- **Summation Delay:** Minimized through the use of high-speed adders.
- The architecture reduces the number of steps required for multiplication, making it more efficient than the "Shift and Add" method.

8. Conclusion

The project successfully validates the 4x4 Vedic Multiplier using eSim. The structural hierarchy implemented in Verilog matches the theoretical Urdhva-Tiryakbhyam algorithm. The simulation results prove that the multiplier is functional, accurate, and provides a scalable framework for larger (8x8 or 16x16) multiplier designs. This project highlights the effectiveness of using open-source EDA tools for modern VLSI design verification.

9. References

M. Durga Madhuri et al., "Implementation of High Speed Vedic Multiplier," *International Journal of Innovative Science and Research Technology*, Vol. 2, Issue 3, March 2017.