

8 BIT VEDIC MULTIPLIER

Objective :

To implement and test 8 Bit Vedic Multiplier using Verilog. This module include adders of different bit widths and Vedic multipliers for efficient multiplication. The objective is to understand digital arithmetic concepts, explore Vedic multiplication techniques, and validate module functionality through simulation. This project serves as a practical exercise in Verilog coding and digital design principles.

Software Detail :

Xilinx provides a comprehensive environment for FPGA design, including synthesis and implementation, while **ModelSim** offers powerful simulation capabilities, allowing us to verify the functionality of your Verilog designs. This combination enables us to design, simulate, and validate our digital circuits efficiently.

Introduction :

This project explores efficient multi-bit addition algorithms inspired by Vedic mathematics principles using Verilog HDL. Vedic mathematics offers fast and simple techniques for arithmetic operations. By implementing these techniques in digital circuits, the project aims to achieve faster and more efficient addition operations compared to traditional methods. The project covers various bit-widths and demonstrates practical applications of Vedic multiplication in digital circuit design.

Abstract :

This mini project implements various multi-bit addition algorithms in Verilog HDL. The project consists of several modules for performing addition operations of different bit widths using Vedic mathematics principles. Vedic mathematics is a system of ancient Indian mathematics that offers fast and efficient techniques for arithmetic operations.

The implemented modules include:

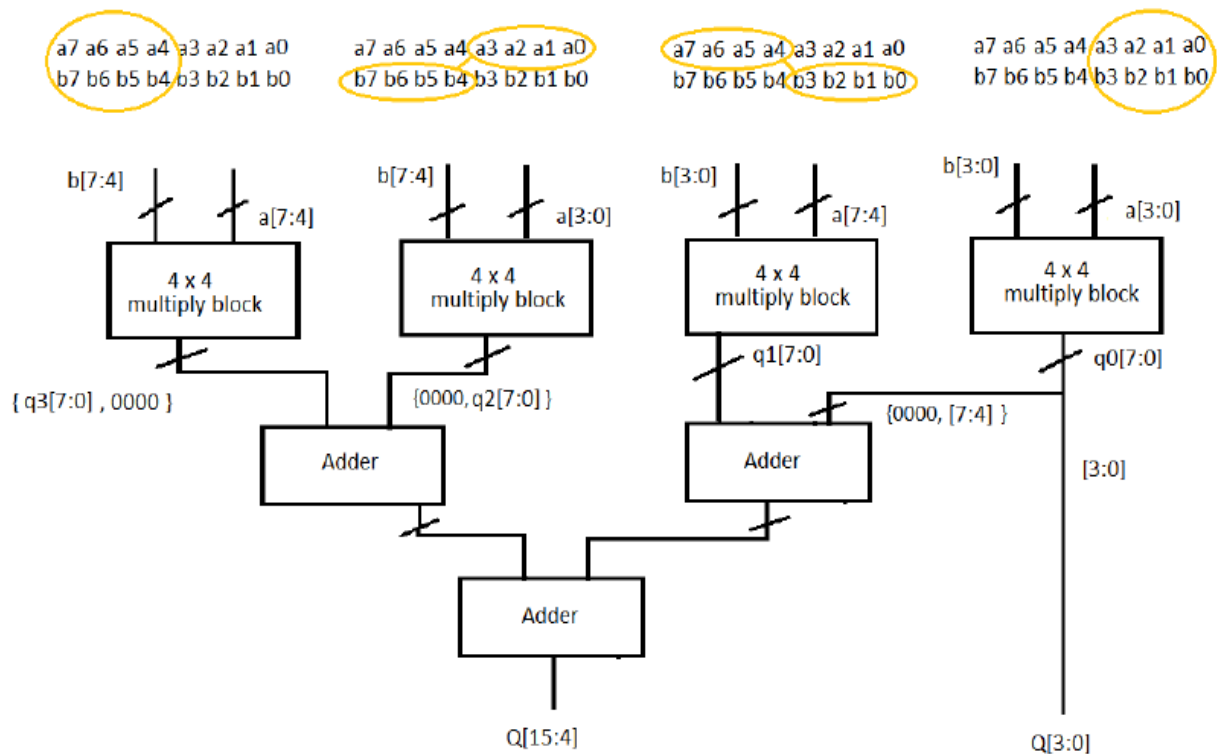
- Half Adder (HA): A basic building block for addition, providing the sum and carry outputs for two input bits.
- 4-bit Adder: Adds two 4-bit numbers using the + operator.
- 6-bit Adder: Adds two 6-bit numbers using the + operator.
- 8-bit Adder: Adds two 8-bit numbers using the + operator.
- 12-bit Adder: Adds two 12-bit numbers using the + operator.
- 2x2 Vedic Multiplier: Multiplies two 2-bit numbers using Vedic multiplication techniques.
- 4x4 Vedic Multiplier: Multiplies two 4-bit numbers using Vedic multiplication techniques.

- 8x8 Vedic Multiplier: Multiplies two 8-bit numbers using Vedic multiplication techniques.

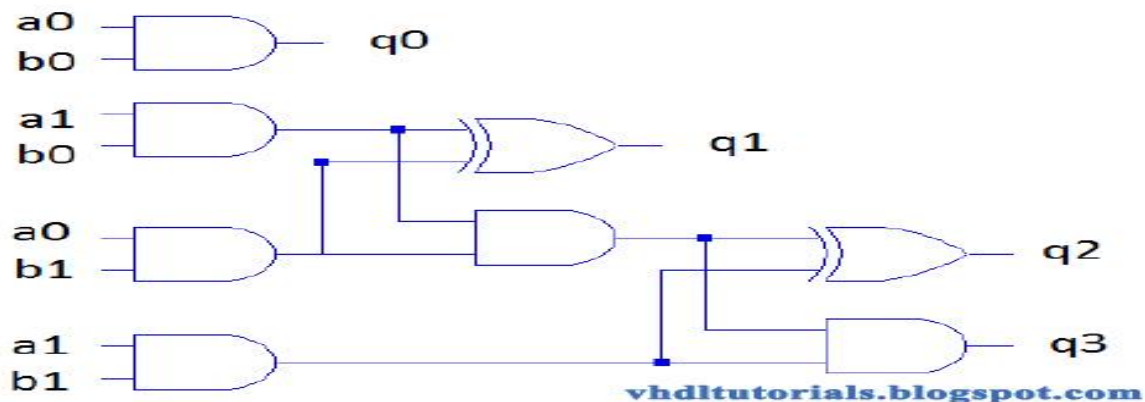
The top-level module test_vedic_8 is used to test the functionality of the vedic_8X8 module, which performs 8x8 multiplication using the Vedic multiplication technique. Test vectors are provided to verify the correctness of the implemented algorithm.

Overall, this project demonstrates the implementation of Vedic multiplication techniques in Verilog HDL for efficient multi-bit addition operations.

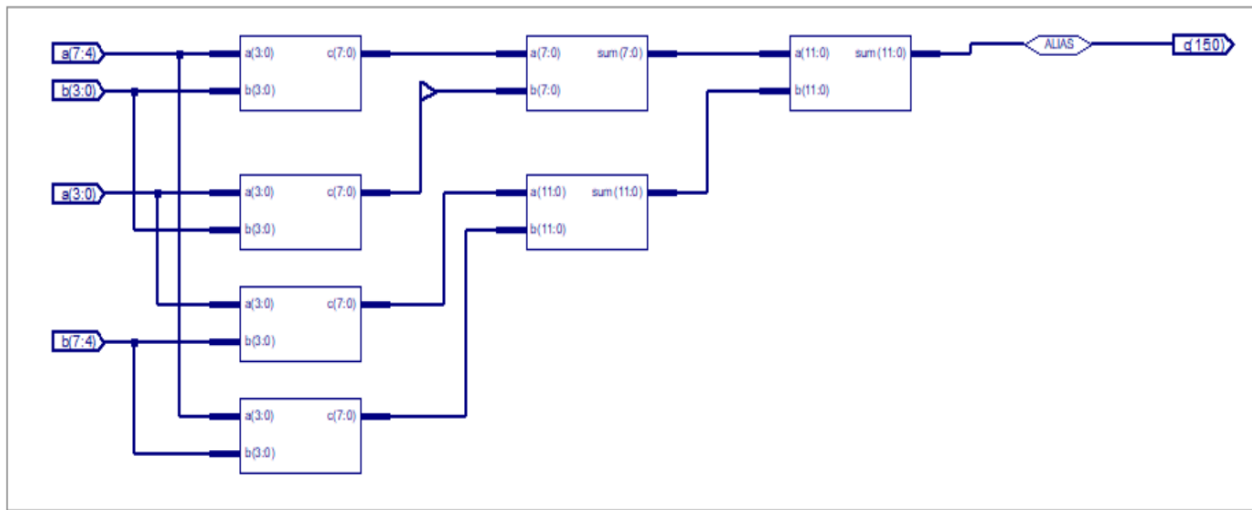
Block Diagram :



Logic Diagram : 2 BIT VEDIC MULTIPLIER



RTL Schematic :



Code :

```
module ha(a, b, sum, carry);
    input a, b;
    output sum, carry;
    xor(sum, a, b);
    and(carry, a, b);
endmodule
```

```
module add_4_bit(a, b, sum);
    input [3:0] a, b;
    output [3:0] sum;
    assign sum = a + b;
endmodule
```

```
module add_6_bit(a, b, sum);
    input [5:0] a, b;
    output [5:0] sum;
    assign sum = a + b;
endmodule
```

```
module add_8_bit(a, b, sum);
    input [7:0] a, b;
    output [7:0] sum;
    assign sum = a + b;
endmodule
```

```
module add_12_bit(a, b, sum);
```

```

    input [11:0] a, b;
    output [11:0] sum;
    assign sum = a + b;
endmodule

```

```

module vedic_2_x_2(a, b, c);
    input [1:0] a;
    input [1:0] b;
    output [3:0] c;
    wire [3:0] c;
    wire [3:0] temp;
    assign c[0] = a[0] & b[0];
    assign temp[0] = a[1] & b[0];
    assign temp[1] = a[0] & b[1];
    assign temp[2] = a[1] & b[1];
    ha z1(temp[0], temp[1], c[1], temp[3]);
    ha z2(temp[2], temp[3], c[2], c[3]);
endmodule

```

```

module vedic_4_x_4(a, b, c);
    input [3:0] a, b;
    output [7:0] c;
    wire [3:0] q0;
    wire [3:0] q1;
    wire [3:0] q2;
    wire [3:0] q3;
    wire [7:0] c;
    wire [3:0] temp1;
    wire [5:0] temp2;
    wire [5:0] temp3;
    wire [5:0] temp4;
    wire [3:0] q4;
    wire [5:0] q5;
    wire [5:0] q6;

    vedic_2_x_2 z1(a[1:0], b[1:0], q0[3:0]);
    vedic_2_x_2 z2(a[3:2], b[1:0], q1[3:0]);
    vedic_2_x_2 z3(a[1:0], b[3:2], q2[3:0]);
    vedic_2_x_2 z4(a[3:2], b[3:2], q3[3:0]);

    assign temp1 = {2'b0, q0[3:2]};
    add_4_bit z5(q1[3:0], temp1, q4);
    assign temp2 = {2'b0, q2[3:0]};
    assign temp3 = {q3[3:0], 2'b0};
    add_6_bit z6(temp2, temp3, q5);

```

```

    assign temp4 = {2'b0, q4[3:0]};
    add_6_bit z7(temp4, q5, q6);
    assign c[1:0] = q0[1:0];
    assign c[7:2] = q6[5:0];
endmodule

module vedic_8X8(a, b, c);
    input [7:0] a, b;
    output [15:0] c;

    wire [15:0] q0;
    wire [15:0] q1;
    wire [15:0] q2;
    wire [15:0] q3;
    wire [15:0] c;
    wire [7:0] temp1;
    wire [11:0] temp2;
    wire [11:0] temp3;
    wire [11:0] temp4;
    wire [7:0] q4;
    wire [11:0] q5;
    wire [11:0] q6;

    vedic_4_x_4 z1(a[3:0], b[3:0], q0[15:0]);
    vedic_4_x_4 z2(a[7:4], b[3:0], q1[15:0]);
    vedic_4_x_4 z3(a[3:0], b[7:4], q2[15:0]);
    vedic_4_x_4 z4(a[7:4], b[7:4], q3[15:0]);

    assign temp1 = {4'b0, q0[7:4]};
    add_8_bit z5(q1[7:0], temp1, q4);
    assign temp2 = {4'b0, q2[7:0]};
    assign temp3 = {q3[7:0], 4'b0};
    add_12_bit z6(temp2, temp3, q5);
    assign temp4 = {4'b0, q4[7:0]};
    add_12_bit z7(temp4, q5, q6);

    assign c[3:0] = q0[3:0];
    assign c[15:4] = q6[11:0];
endmodule

module test_vedic_8;
    reg [7:0] a;
    reg [7:0] b;
    wire [15:0] c;

    // Instantiate the Unit Under Test (UUT)

```

```
vedic_8X8 uut (.a(a), .b(b), .c(c));
```

```
initial begin
```

```
    $monitor($time, " a=%b, b=%b, --- c=%b\n", a, b, c);
```

```
    a = 0;
```

```
    b = 0;
```

```
    #100;
```

```
    a = 8'd255;
```

```
    b = 8'd255;
```

```
    #100;
```

```
    a = 8'd5;
```

```
    b = 8'd3;
```

```
    #100;
```

```
    a = 8'd4;
```

```
    b = 8'd2;
```

```
    #100;
```

```
    a = 8'd2;
```

```
    b = 8'd2;
```

```
    #100;
```

```
    a = 8'd6;
```

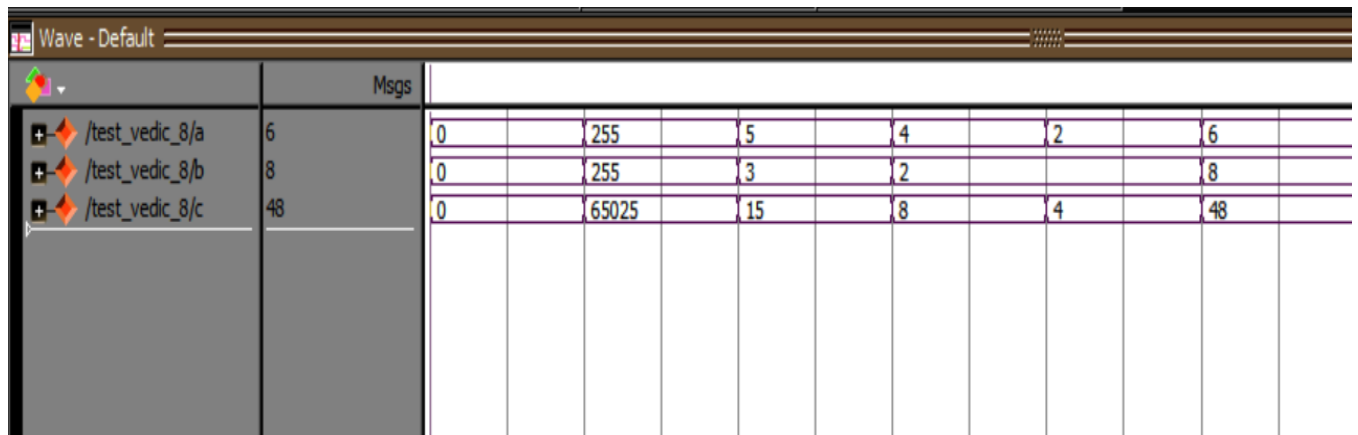
```
    b = 8'd8;
```

```
    #100;
```

```
end
```

```
endmodule
```

Result :



Time (ns)	a (6 bits)	b (8 bits)	c (48 bits)
0	0	0	0
100	255	255	65025
200	5	3	15

Conclusion :

We have successfully implemented multi-bit addition algorithms based on Vedic mathematics principles using Verilog HDL. Our goal was to achieve faster and more efficient addition operations by leveraging techniques like Vedic multiplication. This project serves as a concise introduction to alternative arithmetic algorithms for enhancing digital system performance.