



# 4X4 VEDIC MULTIPLIER

USING VERILOG HDL

---

COURSE: FPGA BASED SYSTEM DESIGN (2EC202)  
MADE BY – HARSHVARDHAN SINGH (22BEC120)  
SEMESTER IV (DIV: 4BECB)

# ABSTRACT

The 4 x 4 Vedic Multiplier is a crucial component in digital arithmetic operations, offering advantages such as reduced computational complexity and faster multiplication compared to traditional methods.

This report delves into the literature surrounding Vedic Multipliers, discusses their limitations, proposes an innovative solution, presents the methodology with diagrams and flowcharts, analyzes results in terms of RTL, TTL, and waveforms, and concludes with insights and references.



# SPECIFICATIONS

Software used – Quartus II 64-bit

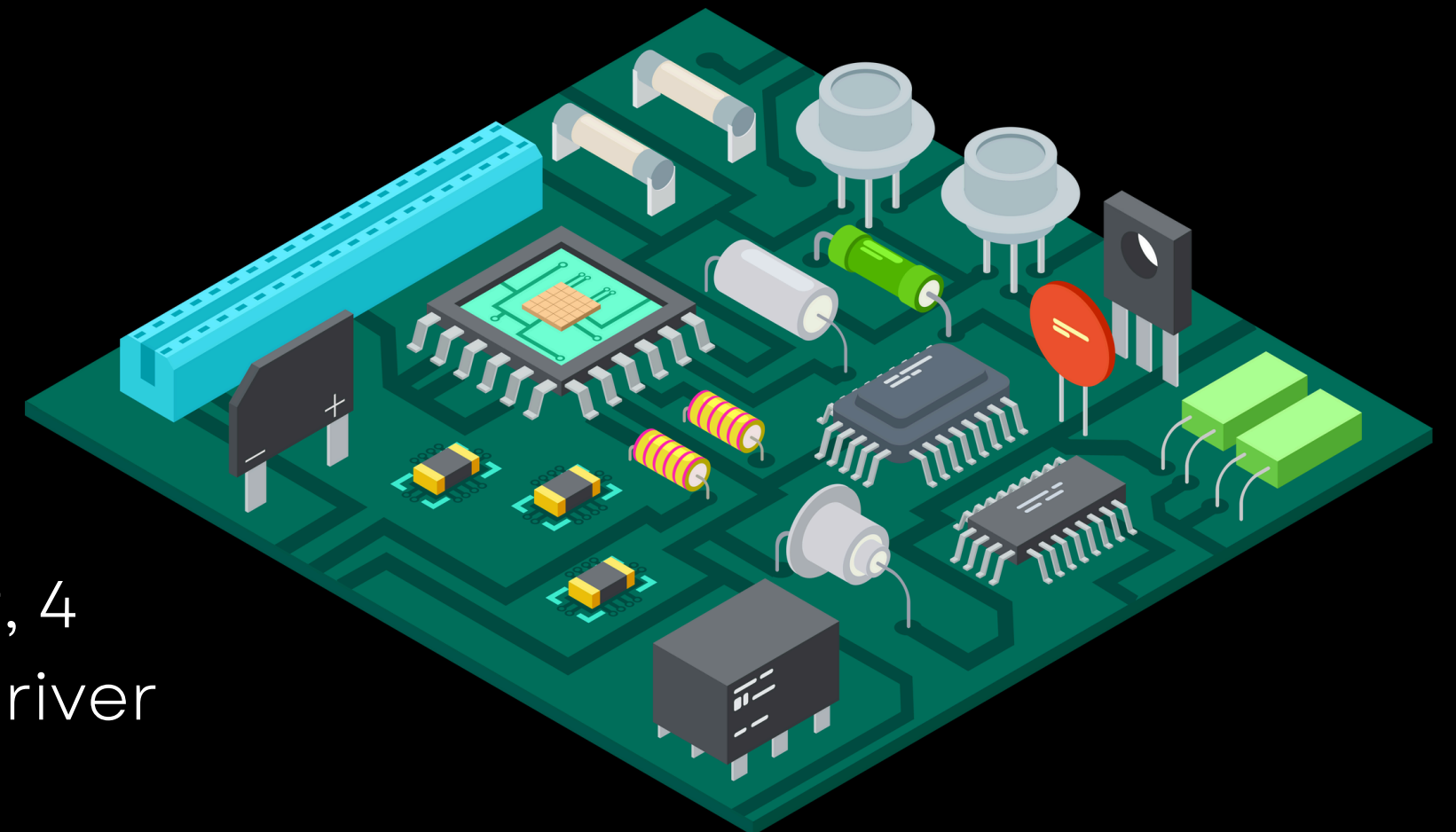
Chip used – Cyclone II EP2C35F672C6

Hardware Description Language – Verilog HDL

Number of input pins utilized – 8

Number of output pins utilized – 21

Modules used – Half Adder, 2x2 Vedic Multiplier, 4 bit Adder, 6 bit Adder, seven segment display driver



# METHODOLOGY

**2x2 Vedic  
Multiplier  
module**

**6 Bit Adder**

**Half Adder  
Module**

**4 Bit Adder**

**Seven  
Segment  
Display  
Driver**



The ha module implements a half adder, which is a digital circuit that performs the addition of two bits. It has two inputs, a and b, and two outputs, sum and carry. The sum output is the result of adding a and b, while the carry output is set to 1 if there is a carry out from the addition.

The vedic\_2\_x\_2 module implements a 2x2 Vedic multiplier, which is a digital circuit that multiplies two 2-bit numbers using the Vedic multiplication method. It has two 2-bit inputs, a and b, and a 4-bit output, c. The module uses four instances of the ha module to calculate the partial products and then adds them together to obtain the final result.

The add\_4\_bit module implements a 4-bit adder, which is a digital circuit that adds two 4-bit numbers. It has two 4-bit inputs, a and b, and a 4-bit output, sum. The module uses the + operator to perform the addition. The add\_6\_bit module is similar to the add\_4\_bit module, but it adds two 6-bit numbers instead. It has two 6-bit inputs, a and b, and a 6-bit output, sum.

The vedic\_multiplier module implements a Vedic multiplier that multiplies two 4-bit numbers using the Vedic multiplication method. It has two 4-bit inputs, a and b, and three 4-bit outputs, hundreds, tens, and ones. The module uses four instances of the vedic\_2\_x\_2 module to calculate the partial products and then adds them together using several instances of the add\_4\_bit and add\_6\_bit modules. The final result is then split into three 4-bit outputs.

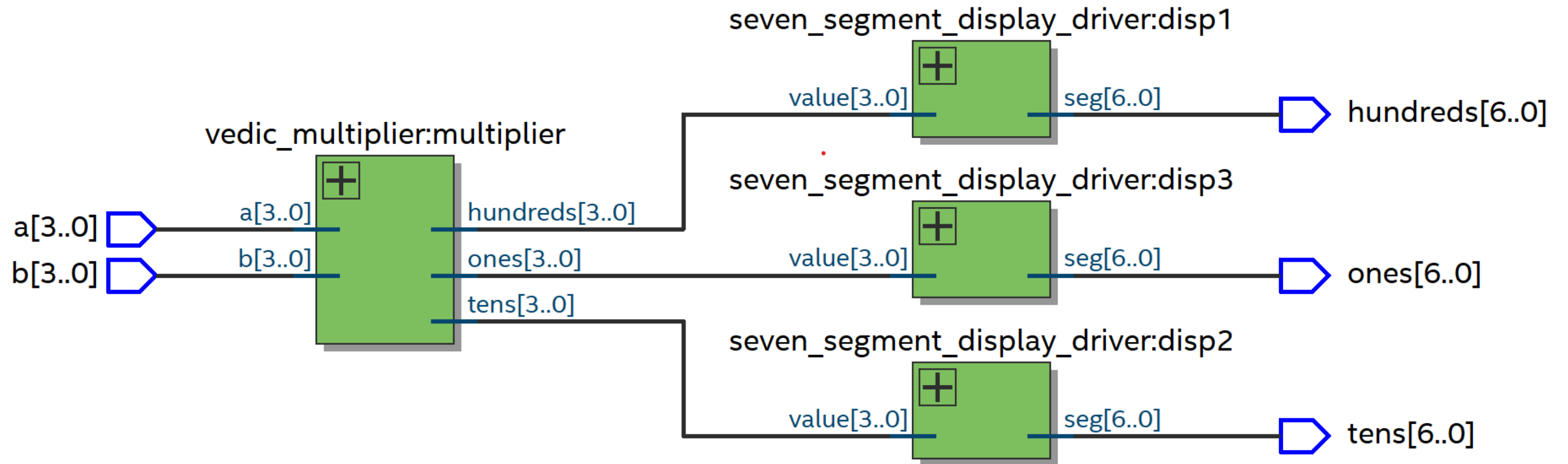


The `seven_segment_display_driver` module is a digital circuit that drives a seven-segment display, which is a type of display that consists of seven segments that can be turned on or off to display a digit. The module has a 4-bit input, `value`, and a 7-bit output, `seg`. The module uses a case statement to map the input value to the corresponding output pattern for the seven-segment display.

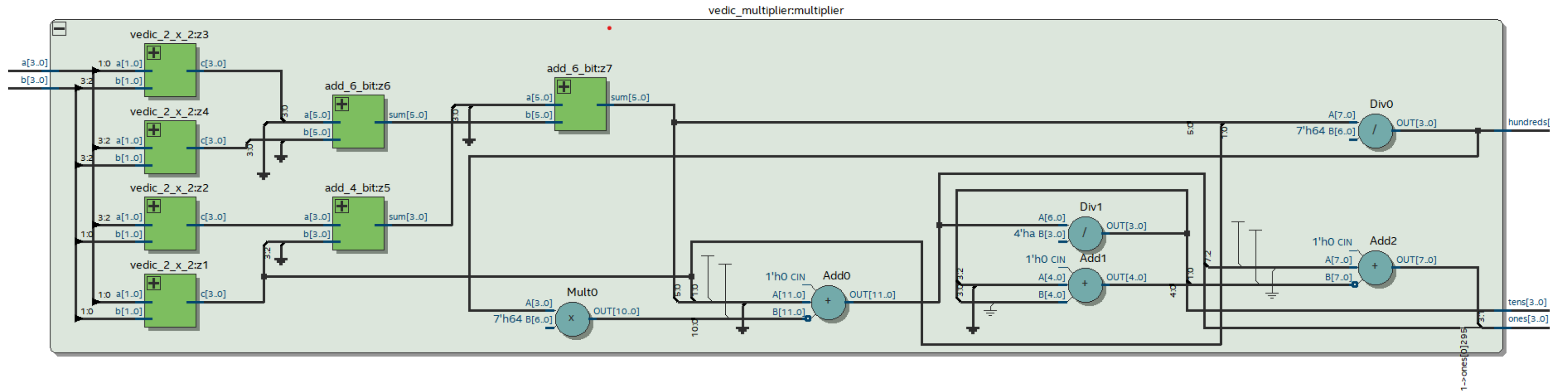
The `vedic_multiplier_with_display` module instantiates the `vedic_multiplier` and `seven_segment_display_driver` modules and connects them together. It has two 4-bit inputs, `a` and `b`, and three 7-bit outputs, `hundreds`, `tens`, and `ones`. The module uses the `vedic_multiplier` module to calculate the product of `a` and `b` and then uses three instances of the `seven_segment_display_driver` module to display the result on three seven-segment displays.



# RTL VIEW

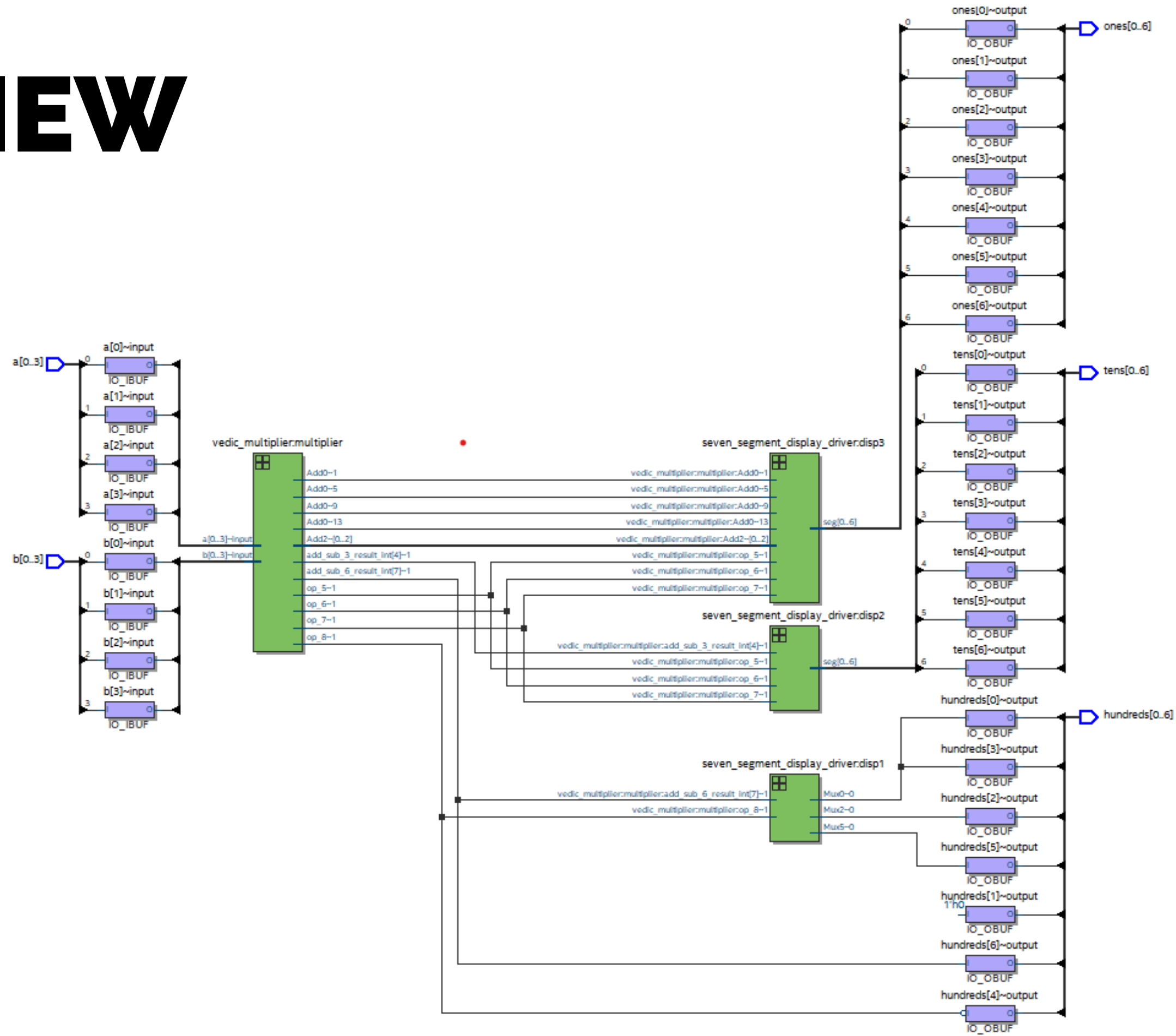


# RTL VIEW

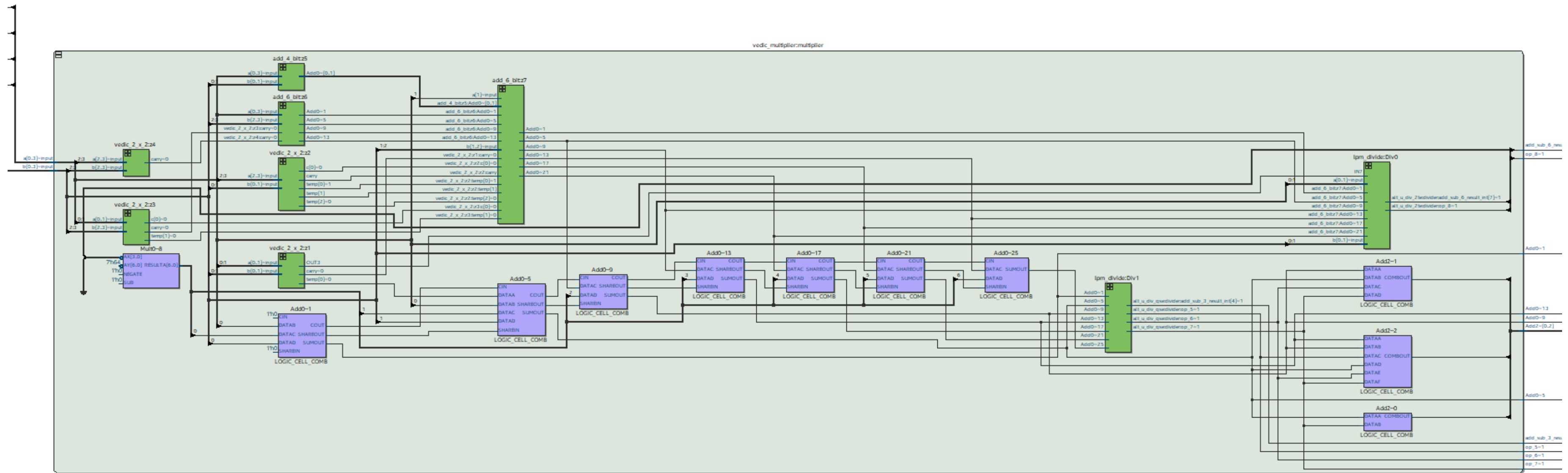




# TTL VIEW



# TTL VIEW



# COMPILATION REPORT

vedic\_multiplier\_with\_display.v

Compilation Report - vedic\_multiplier\_with\_display

Table of Contents

Flow Summary

Flow Settings

Flow Non-Default Global Setti

Flow Elapsed Time

Flow OS Summary

Flow Log

Analysis & Synthesis

Fitter

Assembler

Timing Analyzer

EDA Netlist Writer

Flow Messages

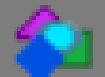
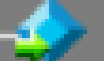
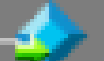
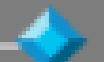
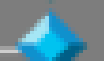
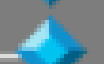
Flow Suppressed Messages

Flow Summary

<<Filter>>

Flow Status	Successful - Mon Apr 22 23:06:40 2024
Quartus Prime Version	22.1std.2 Build 922 07/20/2023 SC Lite Edition
Revision Name	vedic_multiplier_with_display
Top-level Entity Name	vedic_multiplier_with_display
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	66 / 56,480 ( < 1 % )
Total registers	0
Total pins	29 / 268 ( 11 % )
Total virtual pins	0
Total block memory bits	0 / 7,024,640 ( 0 % )
Total DSP Blocks	1 / 156 ( < 1 % )
Total HSSI RX PCSs	0 / 6 ( 0 % )
Total HSSI PMA RX Deserializers	0 / 6 ( 0 % )
Total HSSI TX PCSs	0 / 6 ( 0 % )
Total HSSI PMA TX Serializers	0 / 6 ( 0 % )
Total PLLs	0 / 13 ( 0 % )
Total DLLs	0 / 4 ( 0 % )

# RTL SIMULATION (MODELSIM)

Wave - Default								
		Msgs						
								
+  /vedic_multiplier_with_display/a	9		1	10	14	15	12	9
+  /vedic_multiplier_with_display/b	8		1	6	7	15	14	8
+  /vedic_multiplier_with_display/H	0		0			2	1	0
+  /vedic_multiplier_with_display/T	7		0	6	9	2	6	7
+  /vedic_multiplier_with_display/O	2		1	0	8	5	8	2

# VERILOG CODE

```
// HALF ADDER
module ha(a,b,sum,carry);

    input a,b;
    output sum,carry;

    xor(sum,a,b);
    and(carry,a,b);

endmodule

// 2X2 VEDIC MULTIPLIER
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

// 4 BIT ADDER
module add_4_bit(a,b,sum);
```

```
    input[3:0] a,b;
    output[3:0]sum;

    assign sum = a + b;

endmodule

// 6 BIT ADDER
module add_6_bit (a,b,sum);

    input[5:0] a,b;
    output[5:0] sum;

    assign sum = a + b;

endmodule

// VEDIC MULTIPLIER
module vedic_multiplier(a,b,hundreds,tens,ones);

    input [3:0]a;
    input [3:0]b;
    output[3:0]hundreds,tens,ones;
    wire [7:0]c;
    wire [3:0]q0;
    wire [3:0]q1;
    wire [3:0]q2;
    wire [3:0]q3;
    wire [3:0]temp1;
    wire [5:0]temp2;
    wire [5:0]temp3;
    wire [5:0]temp4;
    wire [3:0]q4;
```

# VERILOG CODE

```
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];

assign hundreds=c/100;
wire [6:0]next;
assign next=c-(hundreds*100);
assign tens=next/10;
assign ones=next-(tens*10);

endmodule
```

```
// Seven-segment display driver
module seven_segment_display_driver(value,seg);
input [3:0] value;
output reg [6:0] seg;
parameter ZERO= 7'b1000000; // 0
parameter ONE=7'b1111001; // 1
parameter TWO=7'b0100100; // 2
parameter THREE=7'b0110000; // 3
parameter FOUR=7'b0011001; // 4
parameter FIVE=7'b0010010; // 5
parameter SIX=7'b0000010; // 6
parameter SEVEN=7'b1111000; // 7
parameter EIGHT=7'b0000000; // 8
parameter NINE=7'b0010000; // 9

always @(*)
begin
case(value)
4'b0000 : seg = ZERO;
4'b0001 : seg = ONE;
4'b0010 : seg = TWO;
4'b0011 : seg = THREE;
4'b0100 : seg = FOUR;
4'b0101 : seg = FIVE;
4'b0110 : seg = SIX;
4'b0111 : seg = SEVEN;
4'b1000 : seg = EIGHT;
4'b1001 : seg = NINE;
endcase
end

endmodule
```

# VERILOG CODE

```
module vedic_multiplier_with_display(a,b,hundreds,tens,ones);
input [3:0] a,b;
output [6:0] hundreds,tens,ones;
wire [3:0] H,T,O;

// Instantiate the Vedic multiplier module
vedic_multiplier multiplier(a,b,H,T,O);

// Instantiate the seven-segment display driver
seven_segment_display_driver disp1(H,hundreds);
seven_segment_display_driver disp2(T,tens);
seven_segment_display_driver disp3(O,ones);

endmodule
```

# CONCLUSION

In conclusion, the 4x4 Vedic Multiplier represents a significant advancement in computational hardware design, blending ancient mathematical wisdom with modern digital circuitry principles. Its innovative methodology, efficient circuit architecture, and optimized algorithms offers tangible benefits in terms of performance, resource utilization and scalability compared to existing multiplication techniques. The Vedic multiplier's potential for application extends beyond 4x4 multiplication to various computational tasks requiring fast and reliable arithmetic operations.







**THANK  
YOU!**

---

COURSE: FPGA BASED SYSTEM DESIGN (2EC202)  
MADE BY – HARSHVARDHAN SINGH (22BEC120)  
SEMESTER IV (DIV: 4BECB)