



National Institute of
Technology
Warangal

Department of Electronics and
Communication Engineering

Digital Systems and
Design -||

Mini Project

**Topic:32 bit Vedic
multiplier**

Submitted by
164138- Princy Jacob Mesapam(ECE 2nd year)

Guided by
Dr B. Lakshmi

INDEX

1.Title

2.Problem Statement

3.Abstract

4.Introduction

5.Top Modules

6.Algorithm for Multiplier

7.Design Environment

8.Code for Multiplier

9.Simulation Results

10.Synthesis Report

11.Concusion

Problem Statement:

To design and develop a structural VHDL model of 32bit Vedic Multiplier and simulate using Xilinx tool. Synthesize to generate its RTL schematic.

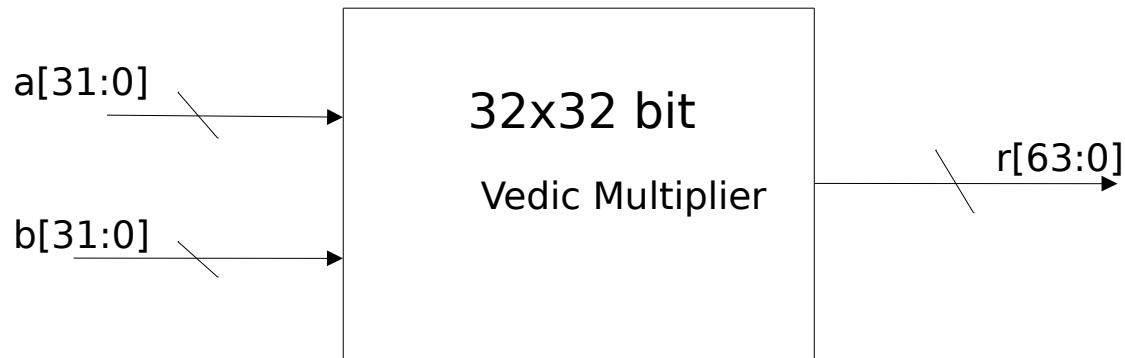
Abstract

This describes the design of a fully functional 32-Bit vedic multiplier that accepts two 32-bit floating point single basic IEEE format operands and generates a 32-bit IEEE format result. This work discusses one of the ancient Indian Vedic mathematic sutra called “***Urdhva Tiryakbhyam***” for multiplication. The thesis presents the system level description to transistor- level circuit schematics and simulations of the multiplier. A description of the IEEE standard is presented for reference. Next, the architectural/functional level description of the design is examined in detail. A "e-level" programming model was developed to implement the function .

Introduction

Multiplication is a very essential arithmetic operation and extensively used in microprocessors, microcontrollers and digital signal processors. It is a time consuming operation because it takes more time and clock cycles as compared to other arithmetic operations. There are number of multiplication algorithms proposed in literature which include array, booth and Vedic algorithms. It is found from the various proposed architectures in literature that Vedic multipliers are faster than non-Vedic multiplier architectures. Different architectures have been proposed in literature to improve the efficiency of multiplication using Vedic mathematics. These architectures are based on conventional Vedic. The drawback of conventional Vedic architecture is that it works fine at 2 bit level but when we increase the order of multiplier, it becomes more complex. This thesis proposes a new architecture for Vedic multiplier which is more efficient in terms of both cost and area.

Top Module for Multiplier



Algorithm for Multiplication

Vedic multiplier is based on the Vedic mathematics . Vedic mathematics is the ancient method of mathematics. The Vedic mathematics is based on the 16 sutras of Urdhva Tiryagbhyam. It simply means that “vertically and crosswise multiplication”.it involves minimum number of calculations, it reduces the space, save the computational time and it is applicable in all cases of multiplication. This method is most efficient when the number of bit increases in multiplication. The structure of this method is shown in the figure from this we get clear idea of “vertical and crosswise multiplication”

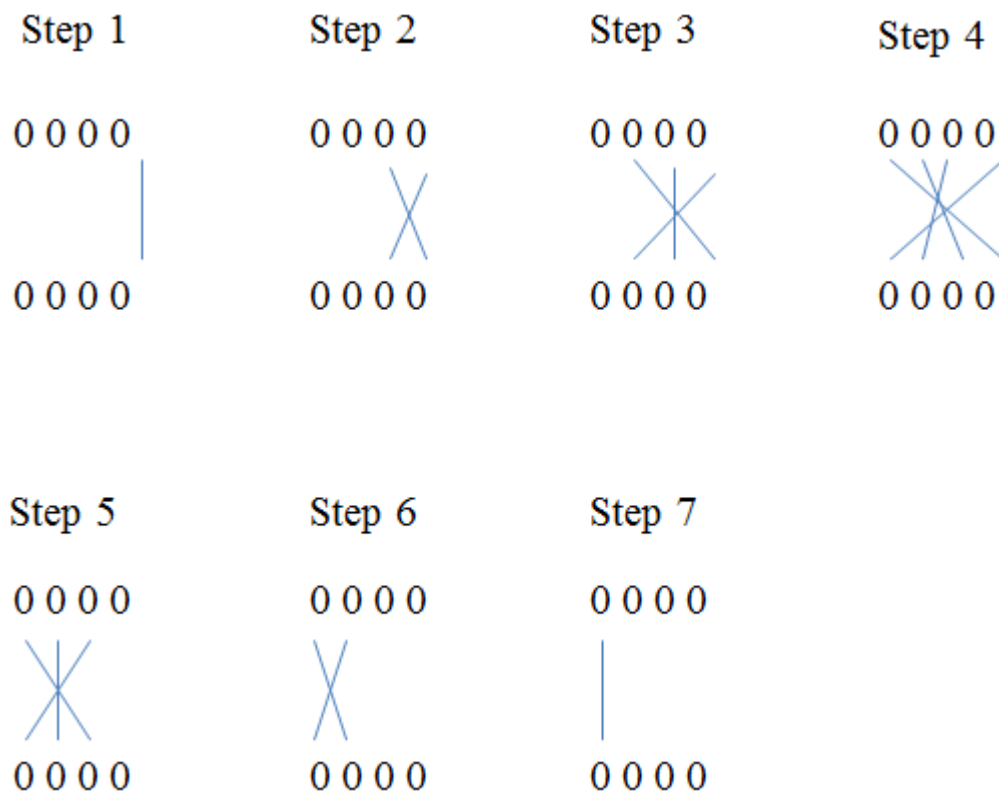


Fig 2: Vedic method for multiplication of 4 bit binary Number.

For multiplication of two 4 bit numbers the multiplication process is divided into 7 steps. The first number is $a_3a_2a_1a_0$ which is multiplier and second number is $b_3b_2b_1b_0$ which is multiplicand as shown in the step1 of the figure. The LSB (least significant bit) of multiplier is multiplied with LSB of the multiplicand and the result of this multiplication is stored as the LSB of the final result. Then as shown in the step2 of the above figure The LSB of multiplier is multiplied with second higher bit of multiplicand and second higher bit of multiplier is multiplied with LSB of multiplicand and then these two partial products are added, after adding these two numbers the LSB of addition is taken as the second higher bit of the final result and remaining bits of the addition are taken as carry bit and this carry can be of multi-bit. Then follow the steps which are given in the above figure and after that we get the result of the 4 bit multiplication.

DESIGN ENVIRONMENT

This method can be used for any $N \times N$ bit multiplication and this Vedic multiplier is independent on clock frequency because partial product and their sum calculated in parallel. And because of we don't require high clock frequencies for multiplication and that's why less switching takes place, because of less switching delay and power minimizes and the result of this we get an efficient processor in delay and power. This thesis proposes 32 bit multiplication and for that we require 2 bit, 4 bit, 8 bit ,16 bit multiplier. The detailed explanations are given below.

Vedic Multiplier for 2X2 Bit Module

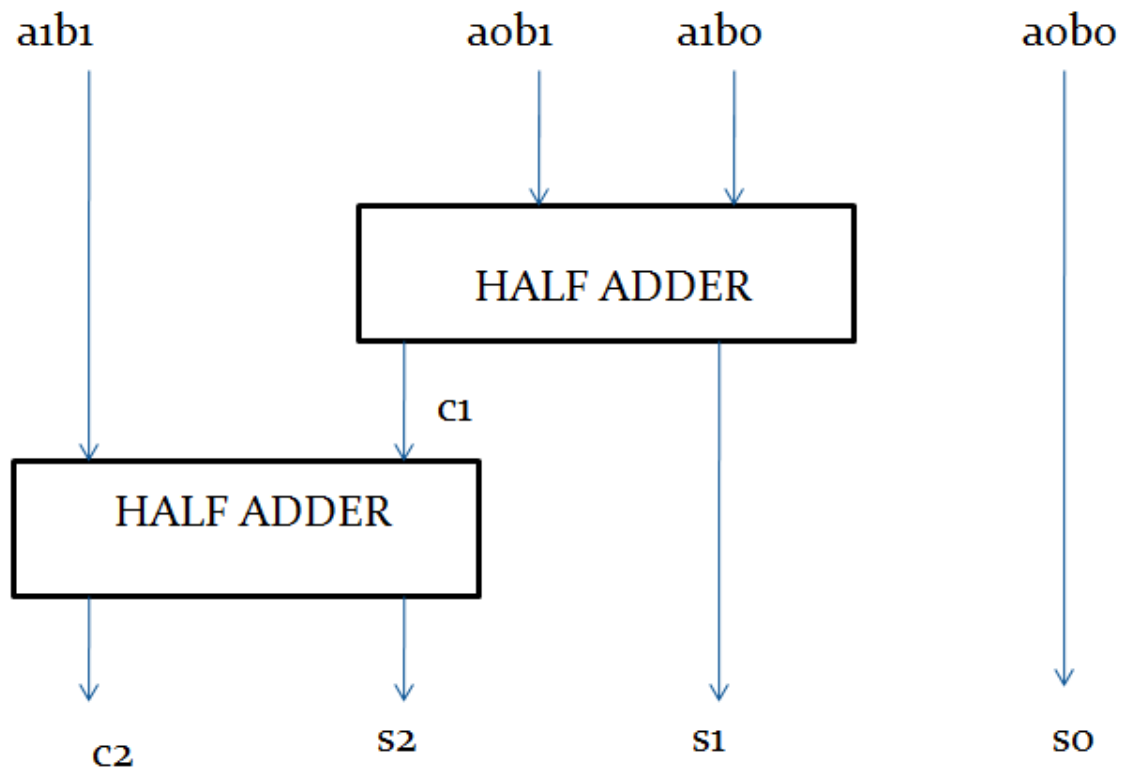


Fig: 2x2 bit Vedic multiplier

A basic block diagram of 2X2 multiplier is as shown. In this diagram a_0, a_1 are the bits of first digit and b_0, b_1 are the bits of second digit. In this multiplier we are taking AND of respective bits and then as shown in block diagram we are following the procedure. The multiplication is done on the sutra of Urdhva-Tiryagbhyam. The result obtained is of 4 bits

Vedic Multiplier for 4X4 Bit Module

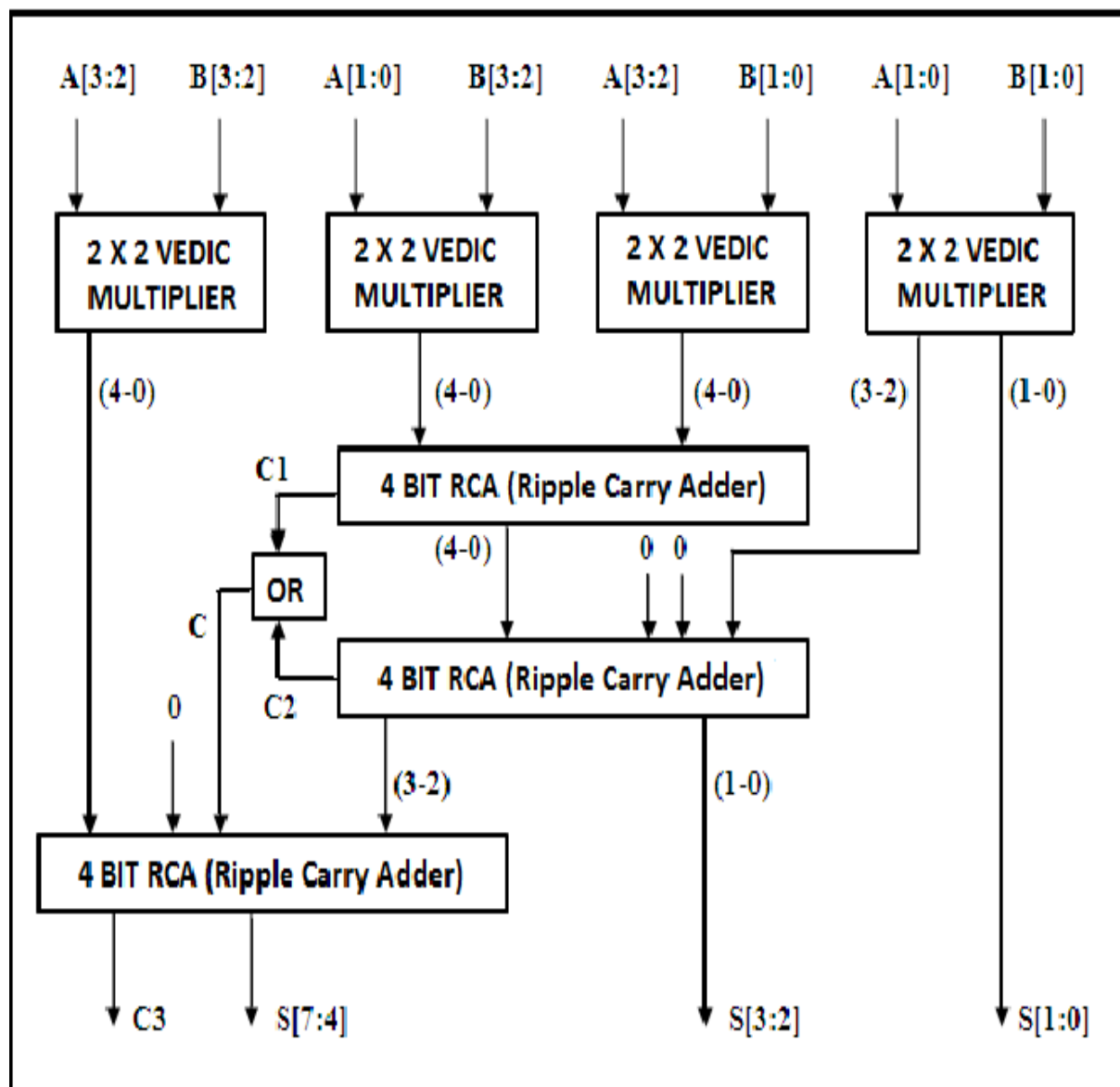


Fig: 4x4 bit Vedic multiplier

Vedic Multiplier for 8X8 Bit Module

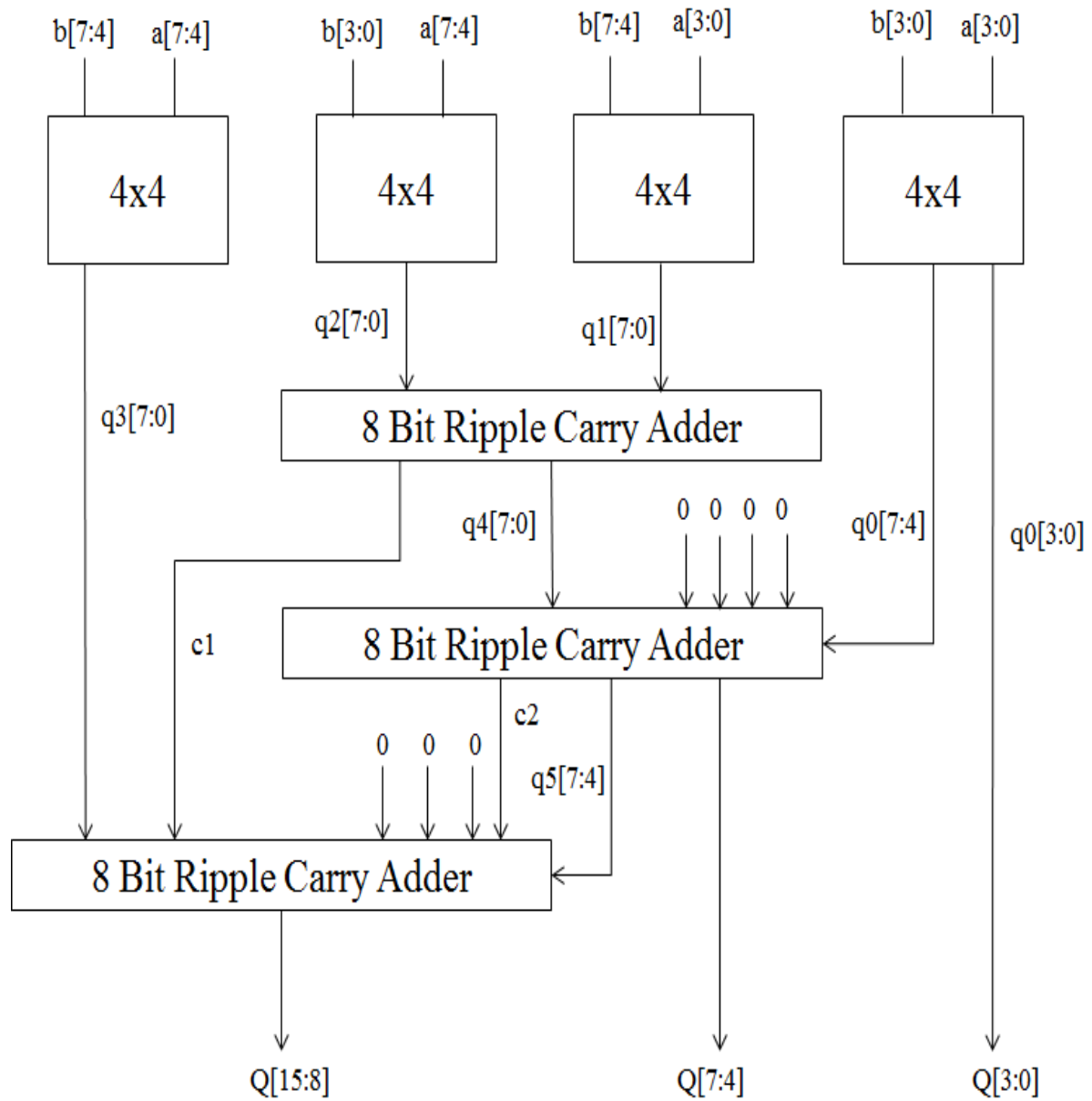


Fig 5: Vedic multiplier for 8X8

bit

Vedic Multiplier for 16X16 Bit Module

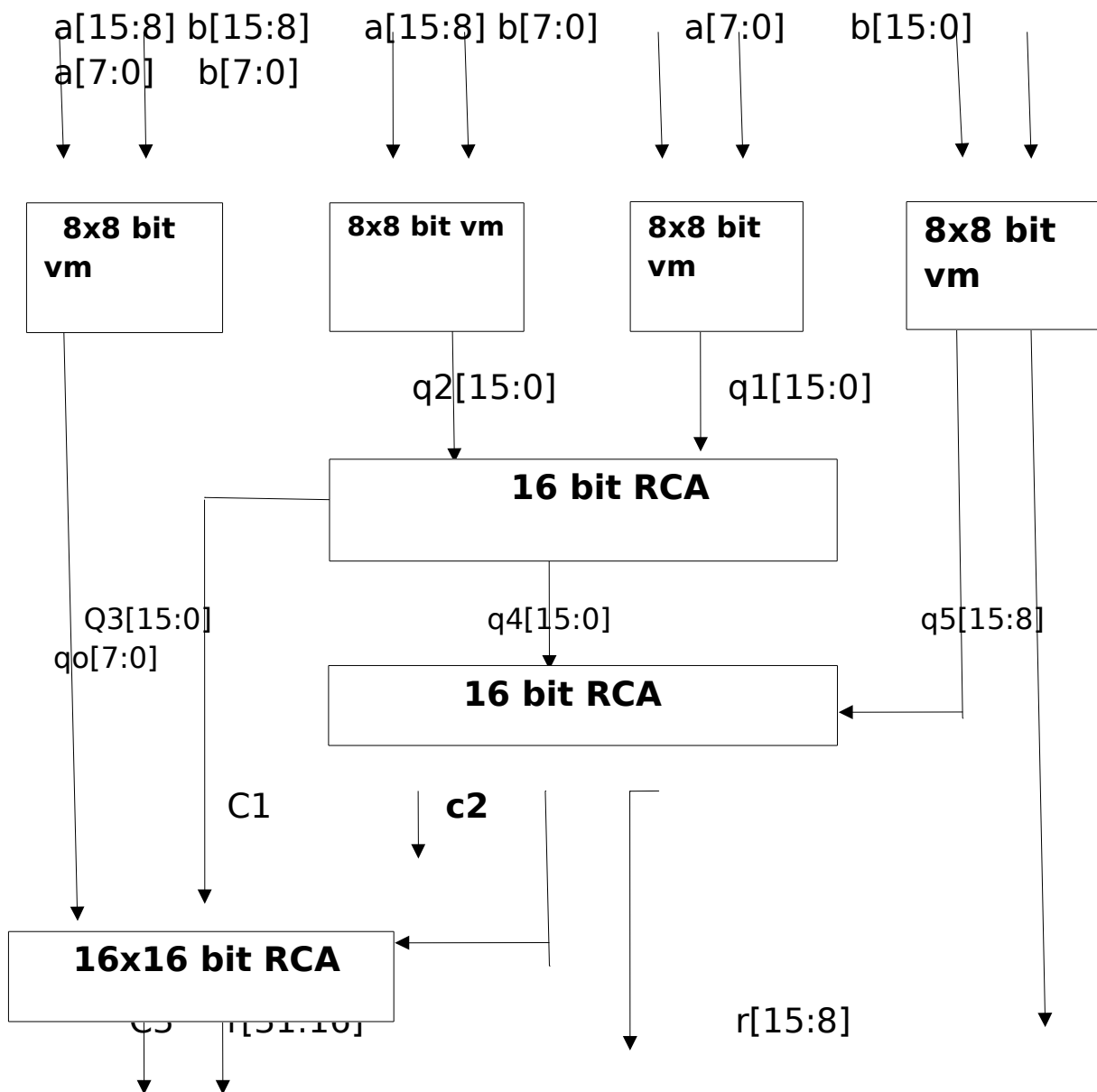


Fig: 16x16 bit Vedic Multiplier

Vedic Multiplier for 32x32 bit Module

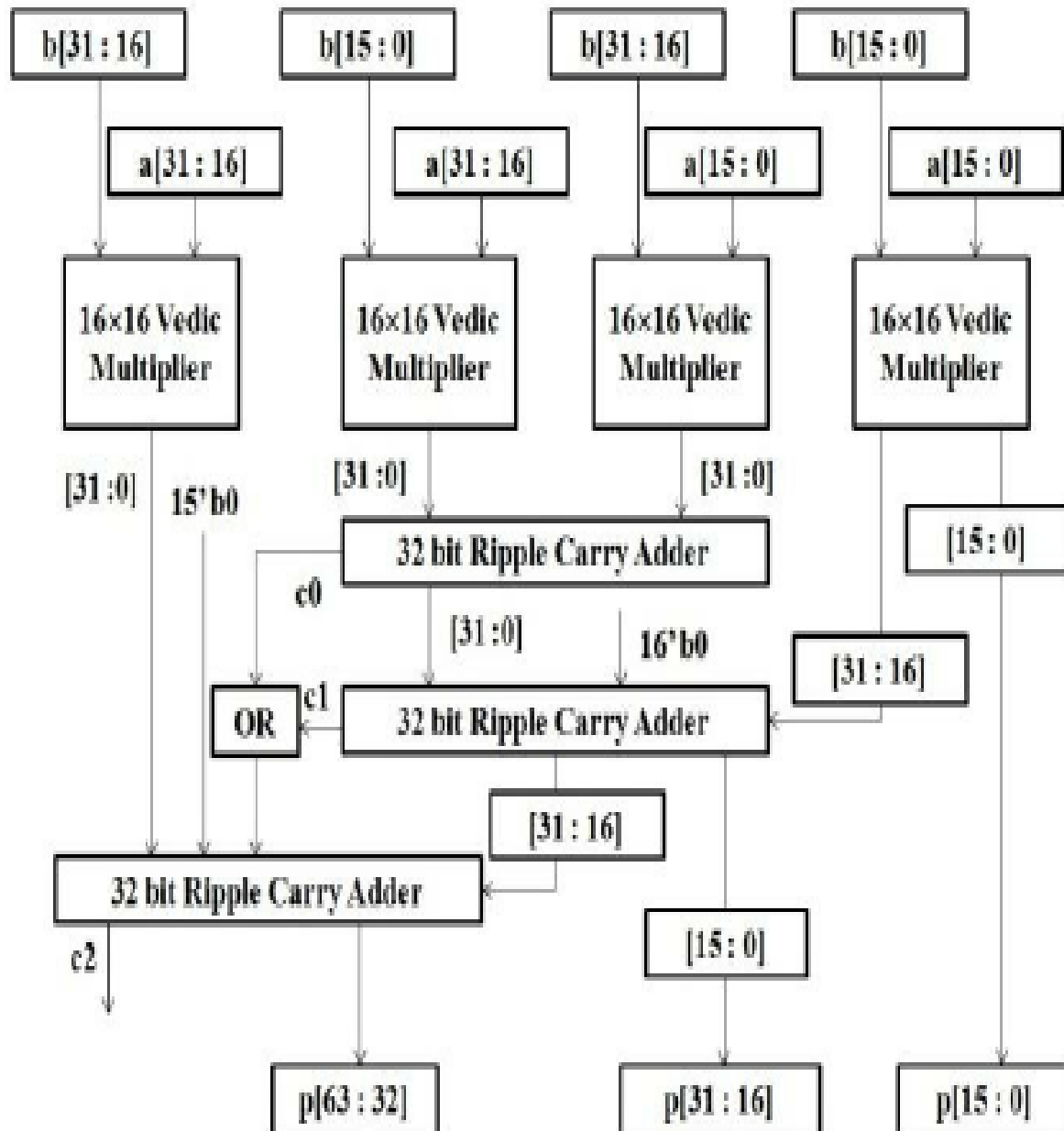


Fig: 32x32 bit Vedic Multiplier

VHDL code for 2x2bit Vedic Multiplier:

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity vm2 is
```

```
    Port ( x : in STD_LOGIC_VECTOR (1 downto 0);
```

```
          y : in STD_LOGIC_VECTOR (1 downto 0);
```

```
          z : out STD_LOGIC_VECTOR (3 downto 0));
```

```
end vm2;
```

```
architecture Behavioral of vm2 is
```

```
    signal s1,s2,s3,s4:std_logic;
```

```
begin
```

```
    z(0)<=x(0)and y(0);
```

```
    s1<=x(1)and y(0);
```

```
    s2<=x(0)and y(1);
```

```
    z(1)<=s1 xor s2;
```

```
    s3<=s1 and s2;
```

```
    s4<=x(1) and y(1);
```

```
    z(2)<=s3 xor s4;
```

```
    z(3)<=s3 and s4
```

```
end Behavioral;
```

simulation results:

[illegible]

Vhdl code for 4x4bit vedic multiplier

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity vm4 is

Port (a : in STD_LOGIC_VECTOR (3 downto 0);

 b : in STD_LOGIC_VECTOR (3 downto 0);

 r : out STD_LOGIC_VECTOR (7 downto 0));

end vm4;

architecture Behavioral of vm4 is

component rca is

Port (a : in STD_LOGIC_VECTOR (3 downto 0);

 b : in STD_LOGIC_VECTOR (3 downto 0);

 c : in STD_LOGIC;

 s : out STD_LOGIC_VECTOR (3 downto 0);

 co : out STD_LOGIC);

end component;

component vm2 is

Port (x : in STD_LOGIC_VECTOR (1 downto 0);

 y : in STD_LOGIC_VECTOR (1 downto 0);

 z : out STD_LOGIC_VECTOR (3 downto 0));

```
end component;
```

```
signal s1,s2,s3,s4,s5:std_logic_vector(3 downto 0);
```

```
signal c:std_logic_vector(7 downto 0);
```

```
signal ca1,ca2,ca3:std_logic;
```

```
signal co1,co2:std_logic_vector(3 downto 0);
```

```
begin
```

```
v1:vm2 port map(a(3 downto 2),b(3 downto 2),s5);
```

```
v2:vm2 port map(a(3 downto 2),b(1 downto 0),s1);
```

```
v3:vm2 port map(a(1 downto 0),b(3 downto 2),s2);
```

```
v4:vm2 port map(a(1 downto 0),b(1 downto 0),s4);
```

```
a1:rca port map(s1,s2,'0',s3,ca1);
```

```
co1<=(ca1 & '0' & c(3 downto 2));
```

```
a2:rca port map(s5,co1,'0',c(7 downto 4),ca3);
```

```
co2<=("00"& s4(3 downto 2));
```

```
a3:rca port map(s3,co2,'0',c(3 downto 0),ca2);
```

```
r(1 downto 0)<=s4(1 downto 0);
```

```
r(3 downto 2)<=c(1 downto 0);
```

```
r(7 downto 4)<=c(7 downto 4);
```

```
end Behavioral;
```

Simulation Results:

				1,000,000 ps	
Name		Value	999,998 ps	999,999 ps	1,000,000 ps
a[3:0]	0010		0010		
b[3:0]	0001		0001		
r[7:0]	00000010		00000010		

VHDL code for 8x8 bit Vedic Multiplier

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity vm8 is

Port (a : in STD_LOGIC_VECTOR (7 downto 0);

 b : in STD_LOGIC_VECTOR (7 downto 0);

 r : out STD_LOGIC_VECTOR (15 downto 0));

end vm8;

architecture Behavioral of vm8 is

component vm4 is

Port (a : in STD_LOGIC_VECTOR (3 downto 0);

 b : in STD_LOGIC_VECTOR (3 downto 0);

 r : out STD_LOGIC_VECTOR (7 downto 0));

end component;

component rca8 is

Port (a : in STD_LOGIC_VECTOR (7 downto 0);

 b : in STD_LOGIC_VECTOR (7 downto 0);

 c : in STD_LOGIC;

 s : out STD_LOGIC_VECTOR (7 downto 0);

 co : out STD_LOGIC);

end component;

signal q0,q1,q2,q3,q4,q5:std_logic_vector(7 downto 0);

signal c1,c2,c3:std_logic;

signal co1,co2:std_logic_vector(7 downto 0);

begin

v1:vm4 port map(a(7 downto 4),b(7 downto 4),q3);

v2:vm4 port map(a(7 downto 4),b(3 downto 0),q2);

v3:vm4 port map(a(3 downto 0),b(7 downto 4),q1);

v4:vm4 port map(a(3 downto 0),b(3 downto 0),q0);

a1:rca8 port map(q2,q1,'0',q4,c1);

co1<=("0000"& q0(7 downto 4));

a2:rca8 port map(q4,co1,'0',q5,c2);

co2<=("000"& c1 & q5(7 downto 4));







a3:rca8 port map(q3,co2,'0',r(15 downto 8),c3);

r(3 downto 0)<=q0(3 downto 0);

r(7 downto 4)<=q5(3 downto 0);

end Behavioral;

Simulation Results:

Name		Value
  a[7:0]		93
  b[7:0]		33
  r[15:0]		1d49

	999,998 ps	999,999 ps	1,000,000 ps
	93		
	33		
	1d49		

VHDL code for 16x16 bit Vedic Multiplier:

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity vm16 is
```

```
    Port ( a : in STD_LOGIC_VECTOR (15 downto 0);
```

```
          b : in STD_LOGIC_VECTOR (15 downto 0);
```

```
          p : out STD_LOGIC_VECTOR (31 downto 0));
```

```
end vm16;
```

```
architecture Behavioral of vm16 is
```

```
    component rca16 is
```

```
        Port ( a : in STD_LOGIC_VECTOR (15 downto 0);
```

```
              b : in STD_LOGIC_VECTOR (15 downto 0);
```

```
              c : in STD_LOGIC;
```

```
              s : out STD_LOGIC_VECTOR (15 downto 0);
```

```
              co : out STD_LOGIC);
```

```
    end component;
```

```
    component vm8 is
```

```

Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
      b : in STD_LOGIC_VECTOR (7 downto 0);
      r : out STD_LOGIC_VECTOR (15 downto 0));
end component;

signal q1,q2,q3,q4,q5,q0:std_logic_vector(15 downto 0);
signal c1,c2,c3:std_logic;
signal co1,co2:std_logic_vector(15 downto 0);

begin

v1:vm8 port map(a(7 downto 0),b(7 downto 0),q0);
v2:vm8 port map(a(7 downto 0),b(15 downto 8),q1);
v3:vm8 port map(a(15 downto 8),b(7 downto 0),q2);
v4:vm8 port map(a(15 downto 8),b(15 downto 8),q3);

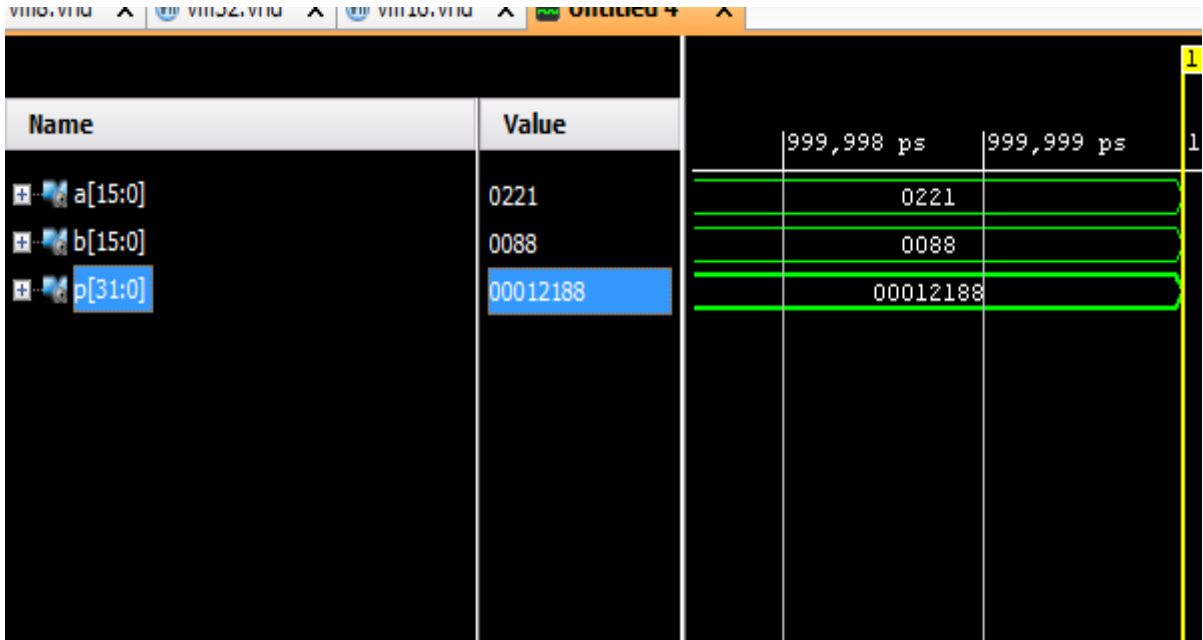
a1:rca16 port map(q2,q1,'0',q4,c1);
co1<=("00000000"& q0(15 downto 8));
a2:rca16 port map(q4,co1,'0',q5,c2);
co2<=("00000000"& c1 & q5(15 downto 8));
a3:rca16 port map(q3,co2,'0',p(31 downto 16),c3);

p(7 downto 0)<=q0(7 downto 0);
p(15 downto 8)<=q5(7 downto 0);

```

end Behavioral;

Simulation Results:



VHDL code for 32x32 bit Vedic Multiplier:

library IEEE;

```
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity vm32 is
```

```
    Port ( a : in STD_LOGIC_VECTOR (31 downto 0);
```

```
          b : in STD_LOGIC_VECTOR (31 downto 0);
```

```
          r : out STD_LOGIC_VECTOR (63 downto 0));
```

```
end vm32;
```

```
architecture Behavioral of vm32 is
```

```
component vm16 is
```

```
    Port ( a : in STD_LOGIC_VECTOR (15 downto 0);
```

```
          b : in STD_LOGIC_VECTOR (15 downto 0);
```

```
          p : out STD_LOGIC_VECTOR (31 downto 0));
```

```
end component;
```

```
component rca32 is
```

```
    Port ( a : in STD_LOGIC_VECTOR (31 downto 0);
```

```
          b : in STD_LOGIC_VECTOR (31 downto 0);
```

```
          c : in STD_LOGIC;
```

```
          s : out STD_LOGIC_VECTOR (31 downto 0);
```

```
          co : out STD_LOGIC);
```

```
end component;
```

```

signal q0,q1,q2,q3,q4,q5:std_logic_vector(31 downto 0);
signal c1,c2,c3:std_logic;
signal co1,co2:std_logic_vector(31 downto 0);

begin

v1:vm16 port map(a(15 downto 0),b(15 downto 0),q0);
v2:vm16 port map(a(15 downto 0),b(31 downto 16),q1);
v3:vm16 port map(a(31 downto 16),b(15 downto 0),q2);
v4:vm16 port map(a(31 downto 16),b(31 downto 16),q3);

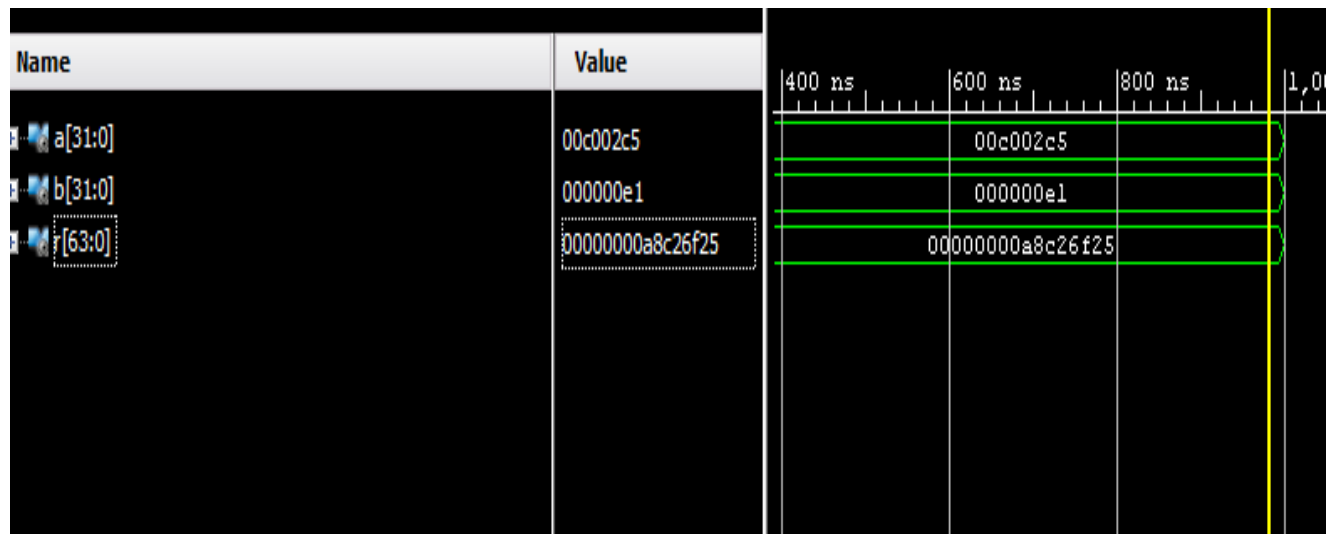
a1:rca32 port map(q2,q1,'0',q4,c1);
co1<=("0000000000000000"& q0(31 downto 16));
a2:rca32 port map(q4,co1,'0',q5,c2);
co2<=("0000000000000000"& c1 & q5(31 downto 16));
a3:rca32 port map(q3,co2,'0',r(63 downto 32),c3);

r(15 downto 0)<=q0(15 downto 0);
r(31 downto 16)<=q5(15 downto 0);

end Behavioral;

```

Simulation Results:



Synthesis Report:

Power

Total on- chip power: 68.869

Junction temperature: 103.2 C

Utilization

LUT's used : 0.41%

I/O : 15.06%

Timing

Critical Path delay - from a[3] to r[39] : 21.06ns

Conclusion

Vedic multipliers implemented in VHDL may be used in applications such as digital signal processors, general processors and controllers and hardware accelerators.

References

- IMPLEMENTATION OF 16x16 BIT MULTIPLICATION ALGORITHM BY USING VEDIC MATHEMATICS OVER BOOTH ALGORITHM by Pranita Soni, Swapnil Kadam, Harish Dhurape, Nikhil Gulavani.
- Conference paper on DESIGN AND IMPLEMENTATION OF 32-BIT VEDIC MULTIPLIER ON FPGA