FPGA Implementation of 160- bit *Vedic* Multiplier

Ravi Kishore Kodali, Sai Sourabh Yenamachintala and Lakshmi Boppana
Department of Electronics and Communication Engineering
National Institute of Technology, Warangal
WARANGAL 506004, INDIA
E-mail: ravikkodali@gmail.com

***Abstract*—The rapid growth of technology influenced the need for the design of highly efficient digital systems. Multipliers have been playing a crucial role in every digital design. It is necessary to make use of an efficient multiplier. Many algorithms came into existence aiming at the reduction of execution time and area. Taking us back to the *Vedic* (ancient Indian) era, the *sutras* or algorithms described in *Vedic* mathematics rendered high degree of efficiency. *Vedic* mathematics describes 16 different sutras which involve multiplication operation. This work discusses one of the 16 sutras, *urdhva tiryakbhyam* sutra for multiplication. Two other multiplication algorithms namely, *Booth* and *Karatsuba* have been considered for the purpose of performance comparison. Elliptic Curve Cryptographic applications require repeated application of higher key size multiplication operation. All the three algorithms have been implemented using Xilinx FPGA and a resource utilization and timing summary comparison has been made.**

***Index Terms*—Vedic multiplication, FPGA, ECC**

I. Introduction

Multipliers constitute a vital block of hardware used in many applications like design of various signal, image and cryptographic processors. Speed, area and configuration are the factors determining the efficiency of multipliers. Many algorithms have been proposed to improve the speed and reduce the area occupied by multipliers. Dadda and Wallace tree multipliers, array multipliers, *Sunar-Koc* multipliers [1], [2] are some of them.

The multiplier architecture can be divided into serial multipliers, parallel multipliers, serial parallel multipliers and details of each are described in [3].

Recent investigations have focussed into the application of *Vedic* (ancient Indian) mathematics for the design of components of various mathematical computations. *Vedic* mathematics can be deemed as a tool to simplify most of the highly complex mathematical computations.

Using the techniques described in *Vedic* mathematics, even a cube of a number can easily be calculated [4]. The multipliers designed based on Vedic mathematics techniques have been extensively used in arithmetic/ logic unit (ALU) and multiply and accumulate (MAC) units which proved to be highly efficient than the conventional multipliers [5].

Many digital signal processing operations such as convolution, correlation require the use of multipliers, wherein the speed of computation can be increased by using high speed multipliers. *Vedic* multipliers, which are used for multiplication of floating point numbers, have greatly reduced the time delay and the area occupied [6]. This work introduces a multiplication algorithm based on *Vedic* mathematics and compares the results with *Booth* and *Karatsuba* multiplication algorithms for larger key lengths.

II. Literature Review

The basic operations involved in any multiplication are (i) generation of partial products (ii) accumulation of these partial products. Reduction in the number of partial products generated or increase in the speed of accumulation improves the speed of multiplication. Booth multiplication algorithm is implemented using state machine approach. Booth multiplication algorithm reduces the number of partial products generated but fails at minimizing the circuit complexity.

Karatsuba algorithm allows multiplication of two large numbers by splitting of the given numbers and performing certain shifting and addition operations. The *Vedic* multiplier algorithm and *Karatsuba* algorithm possess a close resemblance. The most fundamental operation of signal processing, convolution can be performed easily by *Vedic* multiplication algorithm [7].

Booth multiplication algorithm is designed using data path and controller technique. The data path utilizes shift registers, accumulator and a counter. The flow of data among the various components of data path is controlled by means of a controller. For every one clock cycle the controller generates appropriate signals required for next operation depending on the outputs of the present state. After a particular number of clock cycles, final product is obtained. A power dissipation based comparison of Booth multiplier, array and column bypass multiplier is given [8].

In *Karatsuba* algorithm any n- digit number is split into two $\frac{n}{2}$- digit numbers and this process continues recursively until a lower limit is reached on which the multiplication is performed by direct application of the algorithm. These recursive terms are then combined to produce the final result.

978-1-4799-6052-1/14/$31.00 ©2014 IEEE

If a number is split into two and is multiplied by the other number which is also split into two parts then usually four partial products are expected and addition of these products produces the final result.

But *Karatsuba* algorithm reduces the number of partial products to three by using simple addition and shifting operations. This reduction in the number of partial product generation increases the speed. If n represents the number of operands and T(n) represents number of one bit operations then for normal multiplication algorithm T(n) is n^2, whereas for *Karatsuba* algorithm it is $n^{1.58}$ [9].

Let X and Y represent two numbers and $X = X_H + X_L$, $Y = Y_H + Y_L$.

$$\begin{aligned} X.Y &= (X_H + X_L).(Y_H + Y_L) \\ &= X_H.Y_H + X_L.Y_L + X_H.Y_L + X_L.Y_H \end{aligned} \quad (1)$$

According to *Karatsuba* algorithm, the term $X_H.Y_L + X_L.Y_H$ can be computed by using equation (2) and reducing the number of partial products from 4 to 3 [10].

$$\begin{aligned} X_H.Y_L + X_L.Y_H = (X_H + X_L).(Y_H + Y_L) \\ - X_H.Y_H - X_L.Y_L \end{aligned} \quad (2)$$

In the design of analog and digital filters, non-linear circuit analysis, and various interpolation techniques make use of *Chebyshev* polynomials. In order to multiply two *Chebyshev* polynomials *Karatsuba* algorithm is used [11] and it is proved to be more efficient after comparing with direct multiplication and direct cosine form techniques.

The *Vedic* multiplier *sutra* or algorithm is proved to be more efficient in terms of area as well as speed when compared to *Karatsuba* algorithm. The Sanskrit word veda is derived from the root word vid meaning to know without limit. The veda is a repository of all knowledge, fathomless, ever revealing as it is delved deeper.

Swami Bharati Krishna Tirtha, former Jagadguru Sankaracharya of Puri compiled a set of 16 sutras and 13 sub-sutras, which together are known as *Vedic* mathematics. Using *Vedic* mathematics multiplication of numbers, which are nearest to powers of 10 can be achieved easily without following the lengthy conventional form of multiplication [12].

Application of these sutras to various complex mathematical problems renders a simplified solution which is quick and easily implementable. This work deals with one of these 16 sutras called *URDHVA TRIYAKBHYAM* and its implementation using FPGA.

III. AN OVERVIEW OF VEDIC ALGORITHM

URDHVA TRIYAKBHYAM literally means vertically and crosswise. In this method bits of multiplicand and multiplier are multiplied either vertically or crosswise and the result is added to obtain the corresponding product bit. This procedure can be applied over any radix system. An example of multiplication performed for numbers using radix- 10 utilizing this sutra is given in [13].

The *Vedic* multiplication procedure for two 5- bit numbers is shown in Figure 1.

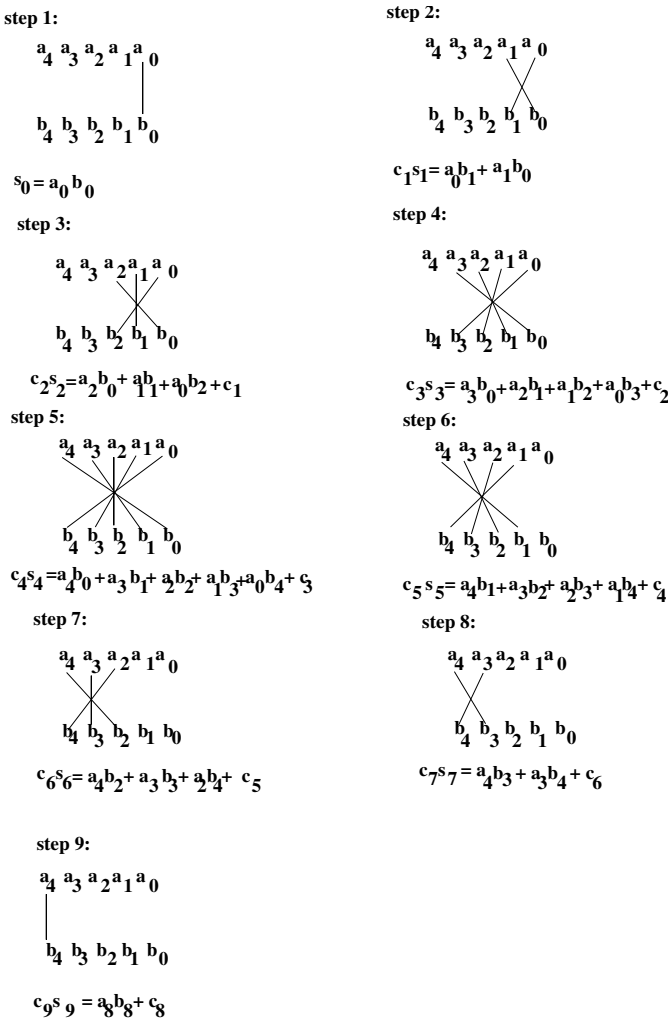


Fig. 1: Steps in 5- bit Vedic multiplication

step-1 the least significant bits (LSB) of both the operands are multiplied resulting in the LSB of the product.

step-2 the bits a_0 and a_1 are multiplied crosswise with the bits b_0 and b_1 and the resulting products are added together with their carry being forwarded to step3. The sum bit becomes the corresponding bit of the product.

step-3 the above procedure is repeated using the bits a_0, a_1, a_2 and b_0, b_1, b_2 generating a_0b_2, a_1b_1, a_2b_0. These terms are added together with the carry generated during

step2 and sum and carry are obtained.

step-4 the first four bits of both the operands are multiplied resulting in the terms $a_0b_3, a_1b_2, a_2b_3, a_3b_0$, which are added with the carry generated during step3.

step-5 In a similar manner all the 5- bits are used for vertical and crosswise multiplication.

step-6 the terms multiplied are a_4, a_3, a_2, a_1 with b_4, b_3, b_2, b_1 in vertical and crosswise manner.

step-7 multiply the bits a_4, a_3, a_2 with b_4, b_3, b_2 generating the product terms a_4b_2, a_3b_3, b_4a_2 whose sum is added with the carry generated during step6.

step-8 only two bits a_4, a_3 and b_4, b_3 from each operand are used in multiplication.

step-9 the bits a_4 and b_4 are multiplied and the result is added with the carry of step8. The sum generated will be the 9th bit of the product term. The carry forms the most significant bit (MSB) of the final product.

Algorithm 1 VEDIC ALGORITHM

INPUT: 5- bit Multiplicand and Multiplier
OUTPUT: 10- bit product
$k \leftarrow 0$
S(k): 10- bit vector initialized to 0
for $i = 0$ **to** 4 **do**
  for $j = 0$ **to** i **do**
    $S(k) = S(k) + a(i) \times b(i - j)$
  end for
  $k = k + 1$
end for
for $i = 4$ **to** 1 **do**
  for $j = 4$ **to** i **do**
    $S(k) = S(k) + a(i) \times b(5 - (i - j))$
  end for
  $k = k + 1$
end for
for $i = 0$ **to** $(k - 1)$ **do**
  $P = P + S(i)$
end for

The procedure for 5- bit multiplication is illustrated in figure 1, where s_i and c_i represent the sum and carry bits respectively. The final product obtained is $c_8s_8s_7s_6s_5s_4s_3s_2s_1s_0$. This sutra is also known by the name Mental Multiplication technique [14].

Based on the Algorithm-1, a 5- bit multiplier has been designed. Using this 5- bit multiplier as a building block, a 160- bit multiplier is built in a recursive manner. A structural view of this N- bit multiplier is shown in figure 2.

The multiplier and the multiplicand are split into two parts (i) lower $\frac{N}{2}$ bits and (ii) upper $\frac{N}{2}$ bits. In figure 2 X_L represents the lower $\frac{N}{2}$- bits and X_H represents the upper $\frac{N}{2}$- bits. Four $\frac{N}{2}$ bit multipliers are used and input to these multipliers are all possible combinations of X_L and X_H.

The first two multipliers taking inputs X_L, Y_H and Y_H, X_L give their outputs to a N- bit adder. The lower $\frac{N}{2}$ bits of the output of multiplier with inputs X_H, Y_H and upper $\frac{N}{2}$- bits of the output of the multiplier with inputs X_L, Y_L are concatenated together and the resulting N-bits are given to N-bit adder, the other input being the output of the first adder. The output produced from this adder gives the middle n-terms of the final product.

The lower $\frac{N}{2}$ bits obtained from the product of the multiplier with inputs X_L, Y_L gives the lower $\frac{N}{2}$ bits of the final product. The carry term generated from two N- bit adders are provided as inputs to the half adder. The resultant sum and carry term are concatenated together with $\frac{N}{2}$ -2 zeroes on the lower side and the result is given as input to an $\frac{N}{2}$- bit adder. The other input of this adder being the upper $\frac{N}{2}$ bits of the multiplier with input X_H, Y_H. The output of this $\frac{N}{2}$- bit adder gives the upper $\frac{N}{2}$- bits of the final product.

Internally every $\frac{N}{2}$ bit multiplier used in the above process utilizes *Vedic* multiplication algorithm similar to the one described above.

The fundamental block used to construct this multiplier is a 5×5 *Vedic* multiplier. The N- bit adders used are ripple carry adders which take N- bit inputs with a carry-in which is zero in for all adders in this algorithm and generates a N- bit sum and a carry- out bit. The above algorithm also utilizes a half adder circuit. Thus a 160- bit *Vedic* multiplier circuit is constructed using simple full adder and half adder components along with lower order *Vedic* multipliers.

IV. RESULTS AND SIMULATION

The *Vedic* multiplier algorithm is synthesized using 6vlx760ff1760-2 a device in *Xilinx* Virtex-6 family.

The synthesis results, device utilization summary and time delay for *Vedic*, *Karatsuba* and *Booth* multipliers for a key length of 160- bits are presented in Table -I.

Table -I gives a comparison between *Vedic* multiplier and *Karatsuba* multiplier and it reveals that *Vedic* multiplier occupies less number of resources and has higher speed than the latter. Thus *Vedic* multiplier is more efficient than *Karatsuba* multiplier.

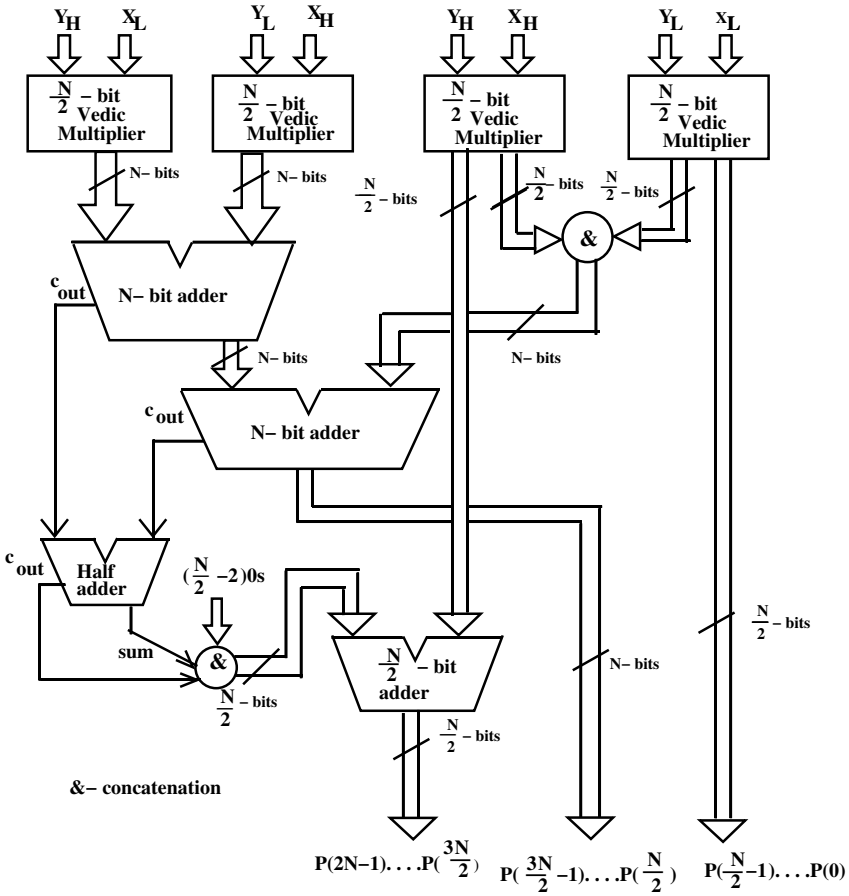
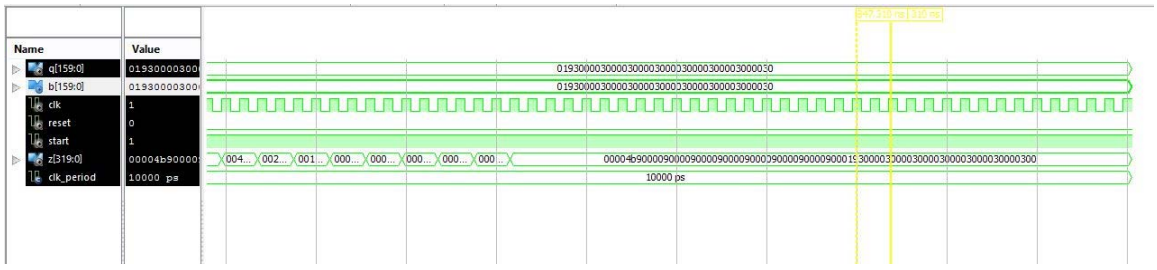
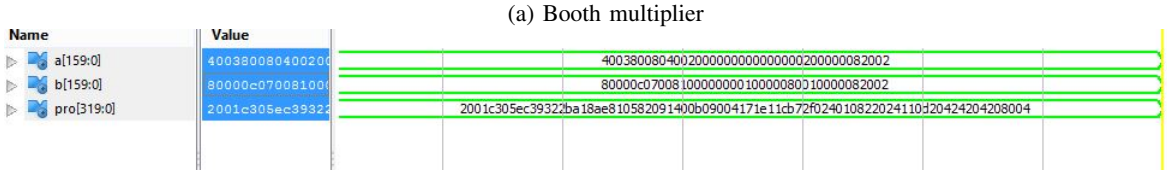


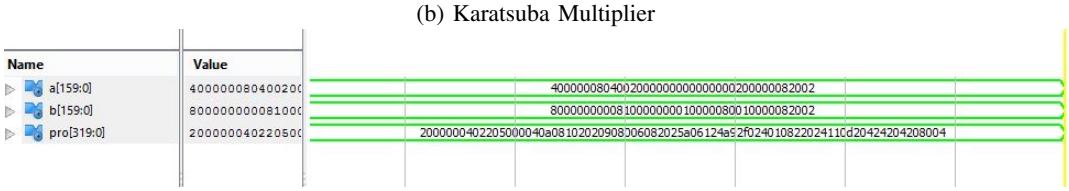
Fig. 2: Block diagram of 160- bit Vedic multipler



(a) Booth multiplier



(b) Karatsuba Multiplier



(c) *Vedic* Multiplier

Fig. 3: Simulation results for 160- bit multiplication using three algorithms

TABLE I: DEVICE UTILIZATION AND TIMING SUMMARY

LOGIC UTILISATION	VEDIC MULTIPLIER	KARATSUBA MULTIPLIER	BOOTH MULTIPLIER
No. of slice LUTs	51761	103246	1937
No. of LUT FF pairs used	51761	103246	2442
No. of IOBs	640	640	643
No. of fully used LUT-FF pairs	0	0	662
No. of slice registers	0	0	1167
Time Delay	27.172 ns	34.123 ns	287.840 ns

Even though Booth multiplier utilizes less number of components, each component occupies more area and takes the longest execution time, it cannot be used in most of the applications.

V. CONCLUSIONS

Vedic multiplier, *Karatsuba* multiplier and *Booth* multiplier have been synthesized on *Xilinx* Virtex-6 FPGA device for a key length of 160- bits. In *Karatsuba* multiplier, for smaller number of bits a shift and add multiplier is used as a terminating condition for a recursive algorithm and for higher number of bits, *Karatsuba* algorithm has been used recursively. In contrast, in *Vedic* multiplier, the basic multiplier utilizes *Vedic* multiplication algorithm and no other multiplier is needed. The synthesis results show that *Vedic* multiplier is more efficient in terms of speed and the area occupied. In the construction of *Vedic* multiplier the ripple carry adders have been used. In order to improve the speed further, high speed adders such as carry look ahead adders, carry save adders, carry skip adders may be used.

REFERENCES

[1] B. Sunar and C. Koc, "An efficient optimal normal basis type ii multiplier," *Computers, IEEE Transactions on*, vol. 50, no. 1, pp. 83–87, 2001.

[2] R. Kodali, P. Gomatam, and L. Boppana, "Implementations of Sunar-Koc multiplier using FPGA platform and wsn node," in *TENCON 2013 - 2013 IEEE Region 10 Conference (31194)*, Oct 2013, pp. 1–4.

[3] V. Kunchigi, L. Kulkarni, and S. Kulkarni, "High speed and area efficient Vedic multiplier," in *Devices, Circuits and Systems (ICDCS), 2012 International Conference on*, March 2012, pp. 360–364.

[4] M. Ramalatha, K. Thanushkodi, K. Deena Dayalan, and P. Dharani, "A novel time and energy efficient cubing circuit using Vedic mathematics for finite field arithmetic," in *Advances in Recent Technologies in Communication and Computing, 2009. ARTCom '09. International Conference on*, Oct 2009, pp. 873–875.

[5] M. E. Paramasivam and R. S. Sabeenian, "An efficient bit reduction binary multiplication algorithm using Vedic methods," in *Advance Computing Conference (IACC), 2010 IEEE 2nd International*, Feb 2010, pp. 25–28.

[6] A. Kanhe, S. Das, and A. Singh, "Design and implementation of floating point multiplier based on Vedic multiplication technique," in *Communication, Information Computing Technology (ICCICT), 2012 International Conference on*, Oct 2012, pp. 1–4.

[7] A. Itawadiya, R. Mahle, V. Patel, and D. Kumar, "Design a DSP operations using Vedic mathematics," in *Communications and Signal Processing (ICCSP), 2013 International Conference on*, April 2013, pp. 897–902.

[8] A. S. Prabhu and V. Elakya, "Design of modified low power Booth multiplier," in *Computing, Communication and Applications (ICCCA), 2012 International Conference on*, Feb 2012, pp. 1–6.

[9] M. Ramalatha, K. Dayalan, P. Dharani, and S. Priya, "High speed energy efficient ALU design using Vedic multiplication techniques," in *Advances in Computational Tools for Engineering Applications, 2009. ACTEA '09. International Conference on*, July 2009, pp. 600–603.

[10] G. Chow, K. Eguro, W. Luk, and P. Leong, "A Karatsuba-based Montgomery multiplier," in *Field Programmable Logic and Applications (FPL), 2010 International Conference on*, Aug 2010, pp. 434–437.

[11] J. Lima, D. Panario, and Q. Wang, "A Karatsuba-based algorithm for polynomial multiplication in Chebyshev form," *Computers, IEEE Transactions on*, vol. 59, no. 6, pp. 835–841, June 2010.

[12] V. Dave and C. Coulston, "Multiplication by complements," in *Electrical Communications and Computers (CONIELECOMP), 2012 22nd International Conference on*, Feb 2012, pp. 153–156.

[13] A. Kumar and A. Raman, "Low power ALU design by ancient mathematics," in *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, vol. 5, Feb 2010, pp. 862–865.

[14] A. Gupta, U. Malviya, and V. Kapse, "Design of speed, energy and power efficient reversible logic based Vedic ALU for digital processors," in *Engineering (NUiCONE), 2012 Nirma University International Conference on*, Dec 2012, pp. 1–6.