IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 9, Issue 5, Ver. V (Sep - Oct. 2014), PP 35-39
www.iosrjournals.org

Design and FPGA Implementation of a simple time efficient KOM Multiplier

Nagarjun B[1], Poonam Sharma[2]
[1]*M.Tech Student, Department of Electronics & Communication, DayanandaSagar Institutions, India*
[2]*Associate Professor, Department of Electronics & Communication, DayanandaSagar Institutions, India*

Abstract: *Multipliers are key components of many high performance systems such as FIR filters, microprocessors, digital signal processors, etc. A system's performance is generally determined by the performance of the multiplier. In most of the cases, the multiplier is the slowest element which affects the performance of the system. Hence, optimizing the speed of the multiplier is a major design issue. Karatsuba-Ofman Multipliers (KOM Multipliers) have less delay compared to other conventional multipliers which greatly increase the performance of a system. In this work we have designed and implemented 'n' bit KOM Multiplier by using FPGA. Using VHDL as the language we have implemented the hardware design and simulated using Xilinx ISE System edition.*
Keywords: *Binary Multiplier, Delay, FPGA, KOM Multiplier, VHDL, Xilinx ISE*

I. INTRODUCTION

A binary multiplier is an electronic circuit which is most commonly used in digital electronics, such as computer, to multiply two binary numbers. It is built using simple binary adders. A wide variety of computer arithmetic techniques can be used to design and implement a digital multiplier. Most of the conventional involve computing the set of partial products, and then summing the partial products together [1]. This process of summing the partial products together is a long process since it involves multiplying a long number by one digit as shown in the example 1.

```
  123
x 456
=====
  738 (this is 123 x 6)
  615 (this is 123 x 5, shifted one position to the left)
+ 492 (this is 123 x 4, shifted two positions to the left)
====
56088
```

Example 1: Method based on calculating partial products

Until late 1970's, most computers didn't had the option of multiplying instruction, and so programmers had used "Multiply routine" which repeatedly shifts and accumulates partial results which is often written using loop unwinding. Mainframe computers had then option of multiply instructions, but they used to do the same sort of shifts and add as a "Multiply routine". Early multiprocessors also had no multiply instruction. The most common method which was used to multiply decimal numbers was based on calculating partial products, shifting them to the left and then adding them together as shown in example 1. Obtaining the partial products is the most difficult task as it involves multiplying a long number by one digit [2].In binary encoding each long number is multiplied by one digit either 0 or 1, and that is much easier than decimal, as the product by 0 or 1 is just 0 or the same number. Therefore, the multiplication of two binary numbers comes down to calculating partial products which are 0 or the first number, shifting them to left, and then adding them together which is shown in example 2 given below.

```
    1011   (this is 11 in decimal)
  x 1110   (this is 14 in decimal)
  ======
    0000   (this is 1011 x 0)
    1011   (this is 1011 x 2, shifted one position to the left)
    1011   (this is 1011 x 4, shifted two positions to the left)
  + 1011   (this is 1011 x 8, shifted three position to the left)
========
10011010   (this is 154 in decimal)
```

Example 2: Method based on multiplication of two binary numbers.

The above method is much simpler than the decimal system, as in this method there is no table of multiplication to remember. Just shifts and additions will be present in the above method. This method is mathematically accurate and has the advantage that a small CPU may perform the multiplication by using the shift and add features of its arithmetic logic unit rather than a specialized circuit. This method is slower in process because it involves lot of intermediate additions. These additions take lot of time. Faster multipliers may be engineered in order to do fewer additions. For example, a modern processor can multiply two 64 bit numbers with 6 additions rather than 64, and can do several steps in parallel. The second problem that we face is that this conventional method handles the sign with a separate rule like "+ with + yield +" and "+ with – yields -". Most modern computers today embed the sign of the number in the number itself, usually in the two's complement representation. That forces the multiplication process to be adapted to handle two's complement numbers, and that makes the process more complicated.

II. INTRODUCTION TO KOM MULTIPLIER

The Karatsuba-Ofman Multiplier or the Karatsuba algorithm is a fast multiplication algorithm which was discovered by AnatoliiAlexeevitchKaratsuba in 1960 and was published in 1962 [3][4][5]. This algorithm reduces the multiplication of two n bit digit numbers to at most single-digit multiplications in general. It is therefore faster than the classical algorithm, which requires 'n^2' single digit products. If $n=2^{10}=1024$, in particular, the exact counts are $3^{10}=59049$ and $(2^{10})^2 = 1048576$, respectively. The Toom-Cook algorithm is a faster generalization of this algorithm. For sufficiently larger 'n', Schonhage-Strassen algorithm is even faster. The Karatsuba algorithm was the first and foremost multiplication algorithm which was faster than the earliest algorithm. If n is four or more, the three multiplications in Karatsuba's basic step involve operands with fewer than n digits. Therefore, those products can be easily computed by recursive calls of Karatsuba algorithm. The recursion can be applied until the numbers are so small that they can be computed directly. In a computer with a full 32 bit by 32 bit multiplier, for example, on could choose $B=2^{31}=2147483648$ or $B =10^9=1000000000$, and store each digit as a separate 32 bit binary word. Then the sums x1+x0 and y1+y0 will not need an extra binary word for storing the carry over digit as in case of carry save adder, and the Karatsuba recursion can be applied until the numbers to multiply are only 1 digit long. It follows that, for sufficiently large n, Karatsuba's algorithm will perform fewer shifts and single digit additions than longhand multiplication, even though its basic step uses more additions and shifts than the straightforward formula. For small values of n, however, the extra shift and add operations may make it run slower than the longhand method. Hence, Karatsuba's algorithm is usually faster when the multiplicands are longer than 320-640 bits.

III. KOM MULTIPLIER ARCHITECTURE

Karatsuba-Ofman Multiplier has a high speed, parallel multiplier architecture as shown in Fig 3.1. The product of KOM is given in equation,

$$\mathbf{P_{KOM} = a_L b_L + (a_L b_H + a_H b_L)\, z^{n/2} + (a_H b_H)\, z^n} \qquad \mathbf{(1)}$$

Where z=2, for binary number system. The architecture of KOM used in the proposed model is shown in figure 3.1. The operands say a and b of size n bits are considered for multiplications. The each n bit operand is decomposed into two n/2 bits operands as a_L, $a_H$of operand a and b_L, b_H of operand b. the KOM product given in equation (1) is implemented using four multipliers and three additions.

Let,
$a = 0011_{(2)}$
$b= 1001_{(2)}$
z=2 for binary numbers
n=4 bits
$a_L = 11_{(2)}$ and $a_H = 00_{(2)}$
$b_L = 01_{(2)}$ and $b_H = 10_{(2)}$
So, $a_L b_L= 11_{(2)} \times 01_{(2)} = 0011_{(2)}$.
$(a_L b_H + a_H b_L)\, z^{n/2}= (11_{(2)} \times 10_{(2)} + 00_{(2)} \times 01_{(2)})\, z^2 = (0110_{(2)} + 0000_{(2)})\, z^2= 011000_{(2)}$; shifted left by 2 bits, hence barrel shifter used in architecture.
$(a_H b_H)\, z^n= (00_{(2)} \times 10_{(2)})\, z^4 = 00000000_{(2)}$
$(a_L b_H + a_H b_L)\, z^{n/2}= 011000_{(2)}$ is decomposed into $01_{(2)}$ and $1000_{(2)}$. Zero extension is used to make $01_{(2)}$ as 4 bit because n bit full adders are used in architecture. Therefore we get $0001_{(2)}$ as the other decomposed part.

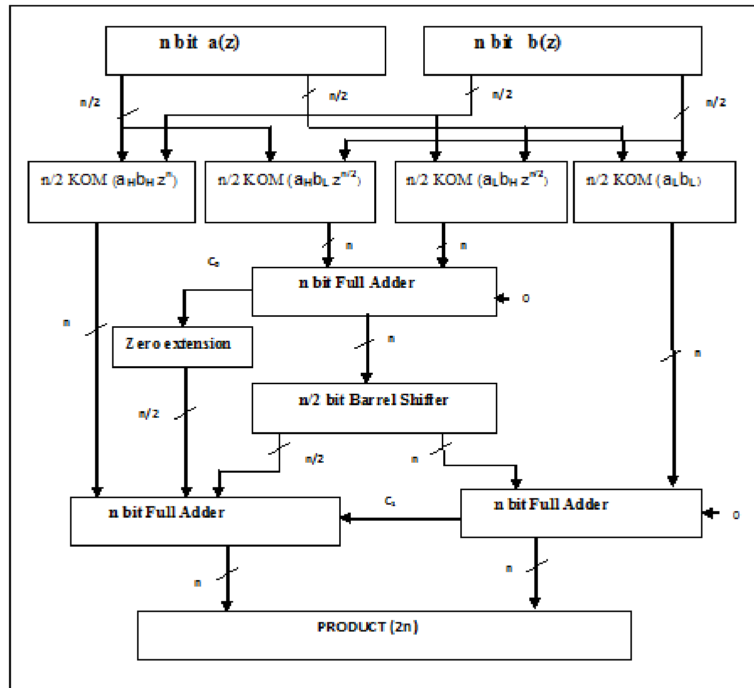

Figure 3.1 KOM in Proposed Model [6]

Now $a_L b_L + (a_L b_H + a_H b_L)\, z^{n/2} + (a_H b_H)\, z^n$ implemented as
$0011_{(2)} + 1000_{(2)} = 1011_{(2)}$; Lower product
$0001_{(2)} + 0000_{(2)} = 0001_{(2)}$; Higher product
$P_{KOM}= 00011011_{(2)}$; final product. ; Product aligned.

This KOM Architecture uses four n/2 KOM multipliers and barrel shifters for the radix operations and summers for the addition operation respectively. The shifter performs the left shift operation and the final product is obtained in the final summer as shown in the above figure. This KOM architecture is speedy in operation compared to other multipliers since all the operations in it takes place together in parallel. This greatly reduces the delay in the system and hence improves the performance.

IV. SIMULATION TOOL & LANGUAGE USED

We have implemented the hardware design using VHDL language and simulated using Xilinx ISE System edition. VHDL is a Hardware Description Language used in Electronic design automation to describe digital and mixed signal systems such as FPGA's and IC's. VHDL can also be used as a general purpose parallel programming language. The key advantage of VHDL, when used for systems design, is that it allows the behavior of the required system to be modeled and simulated before synthesis tools translate the design into real hardware like gates and wires. Xilinx ISE (Integrated Software Environment) is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. System Generator for DSP™ is the industry's leading high-level tool for designing high-performance DSP systems using FPGAs. The target device is xc3s50-5pq208 and product version of Xilinx ISE used is 13.1.

V. SIMULATION RESULTS

Table 5.1. Timing Report

Speed Grade	-5
Delay in Route	9.438ns
Delay in Logic	11.864ns
Delay (Levels of logic=17)	21.302ns

Table 5.2. Macro Statistics

No. of Multipliers	16
2x2 Bit Multiplier	16
No. of Adders/Subtractors	10
16 Bit Adders	2
8 Bit Adders	8
No. of XOR Gates	24
1 Bit XOR 2	5
1 Bit XOR 3	19

Table 5.3. Experimental Values

A	B	OUTPUT
73	195	14235
73	10	730
85	10	850
850	20	1700
12	12	144

Table 5.4. Device Utilization Summary

Logic Utilization	Used	Available	Utilization
No. of Slices	79	768	10%
No. of 4 input LUT's	137	1536	8%
No. of bonded IOB's	32	124	25%

VI. SIMULATION SNAPSHOTS

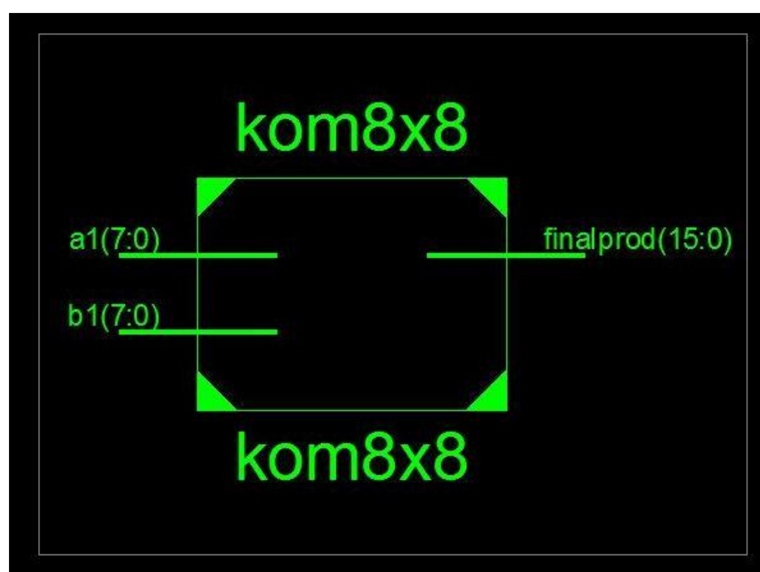

Fig 6.1 RTL Schematic

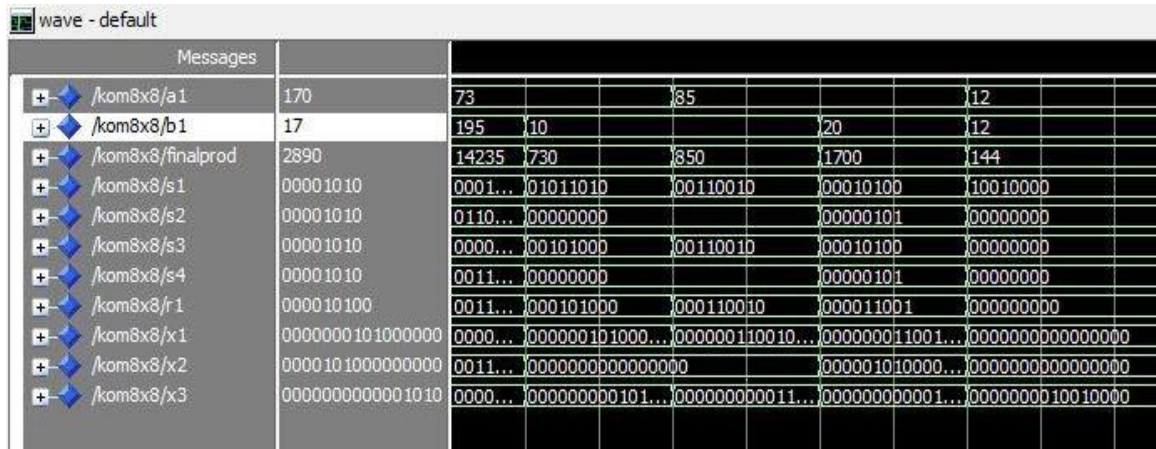

Fig 6.2 Simulation Output

VII. CONCLUSION

In this work, FPGA Implementation of KOM multiplier has been carried out using structural design of VHDL language in Xilinx ISE. We have used Xilinx ISE System edition to perform the simulations for the 8 bit binary numbers. Compared to other multipliers which are being used in the industry, KOM multiplier has low delay levels and high performance which makes it as the best choice by the designers. In this project we have simulated results for the 8 bit binary numbers which consists of four 4*4 KOM multipliers in the hardware level along with summers and barrel shifters for the radix operations. This KOM multiplier uses very few additions and shifting operations compared to other conventional multipliers which greatly enhance the speed.

VIII. ACKNOWLEDGEMENTS

I express my deepest gratitude and sincere thanks to my guide Ms. Poonam Sharma, Associate Professor, Dept of ECE, DayanandaSagar Institutions, for her valuable time and guidance throughout this work. I am immensely grateful to Prof. GirishAttimarad and DayanandaSagar Institutions for giving me an opportunity to carry out this work. I am highly indebted to my dad for his immense love and trust on me throughout my journey of life.I also thank all my M.Tech classmates &beloved friends (JASHD) for their continuous support and motivation throughout my career.

REFERENCES

[1] "The Evolution of Forth" by Elizabeth D. Rather et al.

[2] "Interfacing a hardware multiplier to a general-purpose microprocessor"

[3] A. Karatsuba and Yu.Ofman (1962)."Multiplication of Many-Digital Numbers by Automatic Computers". Proceedings of the USSR Academy of Sciences 145: 293–294. Translation in the academic journal Physics-Doklady, 7 (1963), pp. 595–596

[4] A. A. Karatsuba (1995). "The Complexity of Computations". Proceedings of the Steklov Institute of Mathematics 211: 169–183. Translation from Trudy Mat. Inst. Steklova, 211, 186–202 (1995)

[5] Knuth D.E. (1969) The art of computer programming. v.2. Addison-Wesley Publ.Co., 724 pp.

[6] E L Hall, D D Lynch and S J Dwyer III. "Generation of Products and Quotients Using Approximate Binary Logarithms for Digital Filtering Applications," IEEE Transactions on Computers, Vol. C-19, No. 2, pp. 97-105. February 1970.

[7] K H Abed and R E Sifred, "CMOS VLSI Implementation of a Low-Power Logarithmic Converter," IEEE Transactions on Computers, vol. 52, no. 11, pp. 1421-1433, November 2003.

[8] K H Abed and R E Sifred, "VLSI Implementation of a Low-Power Antilogarithmic Converter," IEEE Transactions on Computers, Vol. 52, No. 9, pp. 1221-1228, September 2003.

[9] D J McLaren, "Improved Mitchell-based Logarithmic Multiplier for low-power DSP applications," Proceedings of IEEE International System On Chip Conference, pp. 53-56, September 2003.

[10] Davide De Caro, Nicola Petra and Antonio G M "Efficient Logarithmic Converters for Digital Signal Processing Applications," IEEE Transactions on Circuits and Systems. vol. 58, pp. 667-671, October 2011.

[11] H Mahrous and A P James, "An Artificial Cellular Convolution Architecture for Real-Time Image

[12] Processing," International Scholarly Research Network (ISRN) Mission work, pp. 1-7, 2012