Fast and Efficient Division Technique Using Vedic Mathematics in Verilog Code

Ugra Mohan Kumar[1], Sandeep Kumar[2,] Madan Pal Singh[3,] Ashok Kumar Yadav[4]

1 Assistant Professor, Uttaranchal University, Dehradun
2 Assistant Professor, JB Institute of Technology, Dehradun
3 Assistant Professor, JB Institute of Technology, Dehradun
4 Senior Lecturer, JB Institute of Technology, Dehradun

Abstract: Division is the most fundamental and commonly used operations in a CPU. These operations furthermore form the origin for other complex operations. With ever increasing requirement for faster clock frequency it becomes essential to have faster arithmetic unit. In this paper a new structure of Mathematics – Vedic Mathematics is used to execute operations. In this paper mainly algorithm on vedic division technique which are implemented for division in Verilog and performance is evaluated in Xilinx ISE Design Suite 13.2 platform then compared with different parameters like delay time and area (number of LUT) for several bits algorithms.
Key words: Vedic mathematics, multiplication, division, delay time, Verilog.

1. INTRODUCTION:

Division is an important essential function in arithmetic operations. Multiplication-based different operations considered as Multiply and Accumulate (MAC) and internal product are among some of the commonly used Computation- Intensive Arithmetic Functions (CIAF) presently implemented and designed in many Digital Signal Processing (DSP) appliances considered as convolution of two or more than two information, Fast Fourier Transform (FFT) of different sequences, filtering of signals or information and in microprocessors its used in arithmetical and logical unit (ALU) [1]. Since multiplying is the most important factor for the implementation time for most of the DSP algorithms or techniques, so there is a need of most efficient and high speed division. Currently, division time is still the most important factor in determining the instruction cycle time and the delay time of a DSP chip

The requirement for high speed processing has been growing as a result of increasing work for computer and signal processing applications. Higher throughput arithmetical and logical operations are important to accomplish the required performance in various real-time signal and image processing applications [2]. The main key of arithmetical and logical operations in these applications is multiplication and division techniques and the development and designing of fast and efficient multiplier circuit has been a subject of interest over last few years. Sinking the execution time and power consumption of required circuits are very necessary requirements for various applications such as in digital signal processing and in digital image processing [2, 3]**.** This work presents different division techniques and architectures. Multiplier based on Vedic or ancient Mathematics is one of the fast and efficient with low propagation delay and low power consumption multiplier**.**

2. VEDIC MATHEMATICS:

Vedic Mathematics introduces the magnificent applications to Arithmetical calculation and verification, theory of numbers, complex multiplications, fundamental algebraic operations, complex factorizations, simple quadratic and advanced order equations, concurrent quadratic equations, partial fractions, in differential calculus and integral calculus, squaring of complex number, cubing, square root of complex number, cube root, 2-Dimensional and 3-Dimensional coordinate geometry and brilliant Vedic Numerical code.

2.1 Vedic Mathematics Sutras and Up-sutras:

Entire mechanics of Vedic mathematics is based on 16 sutras – formulas and 13 up-sutras meaning – corollaries.

Sutras

1. *Ekadhikena Purvena*
2. *Nikhilam Navatascharamam Dashatah*
3. *Urdhva-tiryagbhyam*
4. *Paravartya Yojayet*

5. Shunyam Samyasamucchaye
6. Anurupye Sunyamanyat
7. Sankalana vyavakalanabhyam
8. Puranaprranabhyam
9. Calana – Kalanabhyam
10. Yavadunam
11. Vyastisamashtih
12. Sheshanynkena Charmena
13. Sopantyadvayamantyam
14. Ekanyunena Purvena
15. Ginitasamucchayah
16. Gunaksamucchayah

Up-sutras

1. *Anurupyena*
2. Shishyate Sheshsamjnah
3. Adyamadye Nantyamantyena
4. Kevalaih Saptakam Gunyat
5. Vestanam
6. Yavadunam Tavadunam
7. Yavadunam Tavadunikutya Varganka ch Yojayet
8. Antyayordhshakepi
9. Antyatoreva
10. Samucchayagunitah
11. Lopanasthapanabhyam
12. Vilokanam
13.Gunitasamucchyah Samucchayagunitah

2.2Urdhva-tiryagbhyam:

The *Nikhilam* and *Anurupyena* are for special cases, whereas Urdhva-tiryagbhyam is general formula applicable to all [4]. Its algebraic principle is based on multiplication of polynomials. Consider we want to multiply two 4th degree polynomials

$$Ax^4 + Bx^3 + Cx^2 + Dx + E$$
$$Zx^4 + Yx^3 + Xx^2 + Wx + V$$

$$AZ\,x^8 + (AY+BZ)\,x^7 + (AX+BY+CZ)\,x^6 + (AW+BX+CY+DZ)\,x^5 + (AV+BW+CX+DY+EZ)\,x^4 + (BV+CW+DX+EY)\,x^3 + (CV+DW+XE)\,x^2 + (DV+EW)\,x + EV$$

Figure 1 - Multiplication of two fourth degree polynomials

Highest degree coefficient can be obtained by multiplication of two highest degree coefficients of individual polynomial namely A and Z. A next degree coefficient is obtained by addition of cross multiplication of coefficients of 4th degree and 3rd degree of other polynomials[5]. It means *A* which is 4th degree coefficient of polynomial-1 is multiplied by 3rd degree coefficient of polynomial-2 is added to 4th degree coefficient of polynomial-2 multiplied by 3rd degree coefficient of polynomial-1 to get (AY+BZ).

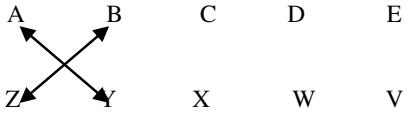

Figure 2 - Vertically Crosswise First Cross Product

Similar logic of cross multiplication and addition can be extended till all 5 coefficients of both polynomials are used as follows. Every iteration gives a coefficient of product.

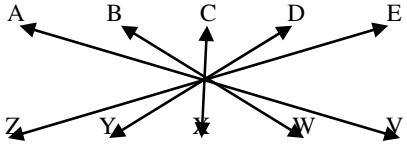

Figure 3 - Vertically Crosswise Intermediate Cross Product

In this iteration, coefficient of degree 4 of product is obtained. For next iteration we drop A and Z which are the highest degree polynomial coefficients. The resulting operation gives coefficient of the degree 3 of multiplication of polynomials. As follows

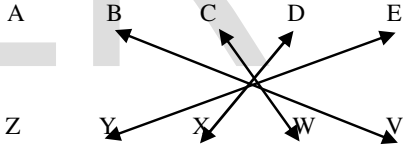

Figure 4 - Vertically Crosswise Intermediate Cross Product

Continuing with this process last coefficient is obtained by multiplication of 0th degree terms of both polynomials as E*V. This process can be done is both ways as it is symmetric. In summary the process can be stated as, process of addition of product of coefficients of two polynomials in crosswise manner with increase and then decrease in number of coefficients from left to right with crosswise meaning product of coefficients for one polynomial going rightwards while for other leftwards.

Any decimal number can be thought as a polynomial with unknown or *x* equal to 10. Being said that, formula stated above can be utilized to calculate product of two decimal numbers. Each digit of decimal number is though as coefficient of power of 10. Only restriction in

this case is each cross product should be only one digit, if not it is added to the next power of 10.

3. PROPOSED ALGORITHM:

The algorithms will be compared to conventional algorithm which is considered as restoring algorithm and/or Non-restoring algorithm of division. In restoring algorithm shifted divisor is repeatedly subtracted from dividend and result of subtraction is stored temporarily. The algorithm can be formulated as [6].

$$N = Q \times D + R$$

Where N dividend, D is divisor, Q is quotient and R is remainder.

Restoring Algorithm

1. R(m) = N m as width of N
2. Repeat for i from m-1 to 0

$Z = R(i+1) - D \times 2i$

If $Z \geq 0$ then Q(i) = 1, R(i) = Z

Else Q(i) = 0, R(i) = R(i+1)

Verilog implements restoring algorithm for its division block. For restoring algorithm worse case N subtractions has to be performed to get N digits of quotient. Each subtraction is equal to width of divisor.

3.1 Binary *Dhwajanka*:

In binary number system, similar to decimal system, MSB of divisor is kept aside and remaining digits are used for cross-products[7].

In binary number system, similar to decimal system, MSB of divisor is kept aside and

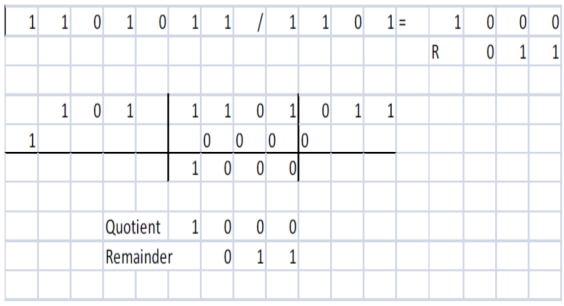

Figure 5 - Complete Example of *Dhwajanka*

remaining digits are used for cross-products. As digits in binary can only be 0 or 1, the process of *Dhwajanka* becomes simpler as division has to be carried out with 1 always. Hence MSB of dividend becomes the MSB of quotient. Cross-product is taken between quotient and rest bits. Again, in cross-product digits being only 0 and 1 multiplication is replaced by AND logic. After the addition of cross-product of bits the sum is subtracted from combination of previous remainder and next digit of dividend. In the example above first bit of quotient is equal to MSB of dividend and hence the remainder is 0. This 0 is combined with next digit of dividend 1, to form 01. Cross-product of quotient and rest bits of divisor gives 1 as only one bit in quotient at this point of the process. Cross-product is subtracted from partial dividend 01 to get 0. Now when 0 is divided by 1 we get quotient 0 and remainder 0. Quotient 0 of this partial division forms the next bit of quotient and remainder as next prefix [8]. In short, *Dhwajanka* formula for binary is further simplified by the nature of the binary numbers. Despite being very easy to solve by *Dhwajanka,* there are some limitations of the process which will be described in next section.

3.1.1 Limitations and Solutions:

A combinational model of this process was designed in Verilog [9]. The reason to choose combinational method was to compare it with equivalent Verilog implementation block and a sequential model will require varying number of clock cycles to finish the division depending on the input – dividend and divisor. The nature of problem in building the combinational model and respective solutions on them are described in next sections.

3.1.2 Negative Subtraction:

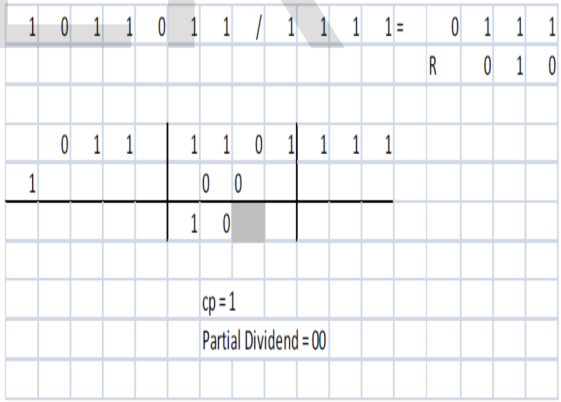

Figure 6 - *Dhwajanka* – Problem of recalculation

During the calculation of third bit of quotient partial dividend is 0 and cross-product is 0 which results in negative 1 as partial dividend which is unacceptable according to algorithm. Solution for this is given as recalculate previous iteration with smaller quotient digit to result in sufficiently large partial remainder. In this case previous quotient bit being 0, it cannot be further reduced to make remainder bigger. Hence former to this iteration has to be recalculated shown below.

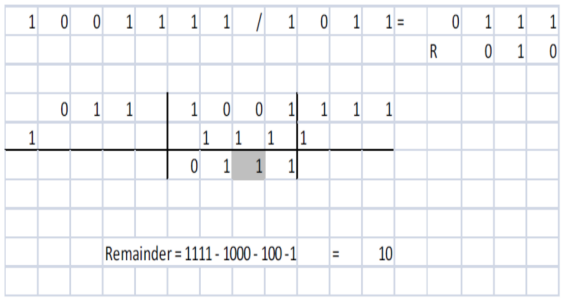

Figure 7 - *Dhwajanka* – Solution of recalculation

In case of combinational design reiteration would result in feedback loop which is unacceptable and in sequential design this would lead to a design which is data dependant and hence undesirable. Solution on this is to allow bits of quotient as well as partial remainders to be negative. This obviously is an overhead of calculation as the state of each bit of quotient and partial remainders has to be maintained, but this enables us to build a combinational design. The complete illustration is as follows.

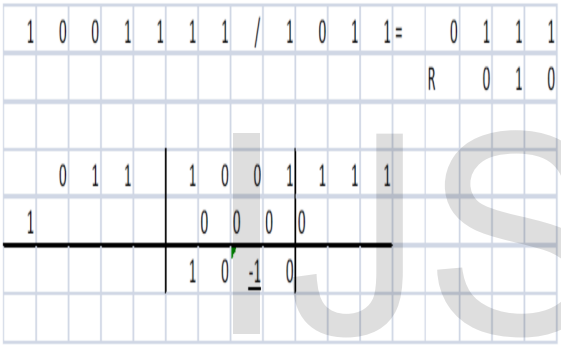

Figure 8 - *Dhwajanka* – Problem of Negative Quotient

During the calculations of third bit of quotient partial dividend 00 is subtracted by cross-product 01 to get quotient as -1 and next partial remainder as 0. If the subtraction is more than 1 then both the quotient bit and partial remainder would be negative.

3.1.3 Correct Remainder

There is another problem in the illustration above. While calculating the remainder we subtract cross-products from right part of dividend prefixed with last partial remainder. Cross-product consists of rest digits of divisor and quotient with first cross-product contains all bits and is also shifted left by one less than bits in rest digits of divisor. If any cross-product is negative then it is added. If last partial remainder is negative then right part of dividend becomes inherently negative. After all the calculations for remainder if it is more than divisor or less than zero it is illegal. Also, it is imperative to have legal remainder to get correct quotient. In the illustration above calculations for remainder are as follows.

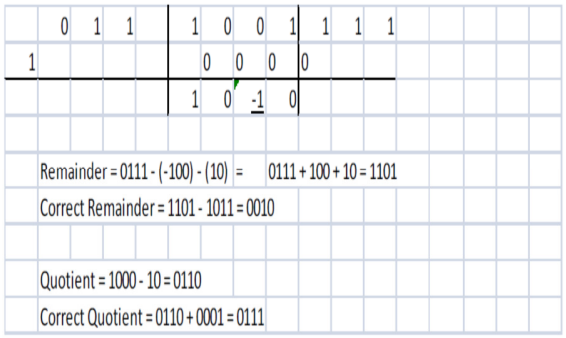

Figure 9 - *Dhwajanka* – Solution for Negative Quotient

Correct remainder is obtained by subtracting divisor from remainder. If the subtraction gives remainder more than divisor, process is repeated. Above correct remainder is obtained by subtracting divisor once, so correct quotient is obtained by adding 1.

3.1.4 Partial remainder overflow:

As the width of dividend and divisor increases, in some cases it is observed that last partial remainder is itself a large number which when combined with right part of dividend becomes a number which may sometimes exceed the width of dividend or divisor itself. This results in large correction logic and hence is undesirable. There can be different approaches to deal with this like checking the correctness of remainder after every 3-4 bit calculation of quotient, use of sequential design model. To check correctness of remainder after every 3-4 bits can work for combinational design but has huge overhead of calculation partial quotient repeatedly and again results is considerably large logic. As seen previously sequential model results in a design which would depend on data to calculate the answer.

4. RESULTS:

4.1 simulation results of 8 bits Vedic Division

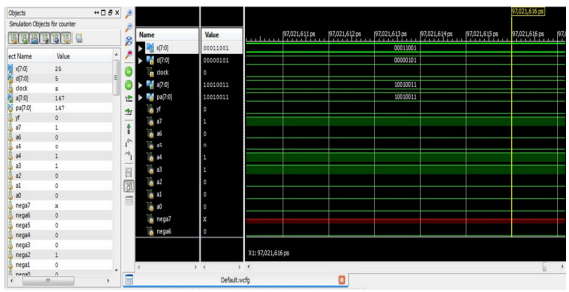

Figure 10 – Simulation result of 8 bits Vedic Division

Description:-

x		: Input data 8 – bit
d		: Input data 8 – bit
clk		: Input clock
a		: Output data 8 – bit
x	=	00011001
d	=	00000101
a	=	10010011

Thus simulated result and calculated result match correctly.

4.2 Synthesis Results

Device utilization summary:

Selected Device: 3s500efg320-4

Number of Slices: 248 out of 4656 5%

Number of 4 input LUTs: 450 out of 9312 4%

Number of IOs: 25

Number of bonded IOBs: 25 out of 232 10%

IOB Flip Flops: 8

Number of GCLKs: 1 out of 24 4%

Total memory usage is 198936 kilobytes

4.3 Timing Results

Minimum input arrival time before clock: 98.119ns

Maximum output required time after clock: 4.283ns

Total REAL time to Xst completion: 13.00 secs

Total CPU time to Xst completion: 12.90 secs

5. CONCLUSION:

The designs of 8 bits Vedic division have been implemented on Spartan3E (3s500efg320-4) device. The computation delay for 8 bits Vedic division is 98.119ns. 10 IEEE.

It is therefore seen that the Vedic division is much faster than the conventional division for higher order bits. The algorithms of Vedic mathematics are much more efficient than of conventional mathematics.

6. FUTURE SCOPE:

In future this work can be extended to higher bit Division which can be implemented using Vedic Mathematics. Floating Point Vedic Processor could be also a good extension of this work.

Figure 12 – Simulation result of 16X16 bits Vedi

7 REFERENCES:

[1] Purushottam D. Chidgupkar and Mangesh T. Karad, "The Implementation of Vedic Algorithms in Digital Signal Processing", Global J. of Engng. Educ., Vol.8, No.2 © 2004 UICEE Published in Australia.

[2] Himanshu Thapliyal and Hamid R. Arabnia, "A Time-Area-Power Efficient Multiplier and Square Architecture Based On Ancient Indian Vedic Mathematics", Department of Computer Science, The University of Georgia, 415 Graduate Studies Research Center Athens, Georgia 30602-7404, U.S.A.

[3] E. Abu-Shama, M. B. Maaz, M. A. Bayoumi, "A Fast and Low Power Multiplier Architecture", The Center for Advanced Computer Studies, The University of Southwestern Louisiana Lafayette, LA 70504.

[4] Honey Durga Tiwari, Ganzorig Gankhuyag, Chan Mo Kim, Yong Beom Cho, "Multiplier design based on ancient Indian Vedic Mathematics", 2008 International SoC Design Conference.

[5] Parth Mehta, Dhanashri Gawali, "Conventional versus Vedic mathematical method for Hardware implementation of a multiplier", 2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies.

[6]Prabir Saha, Arindam Banerjee, Partha Bhattacharyya, Anup Dandapat, "High Speed ASIC Design of Complex Multiplier Using Vedic Mathematics", Proceeding of the 2011 IEEE Students' Technology Symposium 14-16 January, 2011, lIT Kharagpur.

[7] Ramalatha M, Thanushkodi K, Deena Dayalan K, Dharani P, "A Novel Time and Energy Efficient Cubing Circuit using Vedic Mathematics for Finite Field Arithmetic", 2009 International Conference on Advances in Recent Technologies in Communication and Computing.

[8] Shamim Akhter, "VHDL Implementation of Fast NxN Multiplier Based on Vedic Mathematics", 2007 IEEE.

[9] Anvesh Kumar, Ashish Raman, Dr. R.K. Sarin, Dr. Arun Khosla, Small area Reconfigurable FFT Design by Vedic Mathematics", 20