" Design Of High Speed Floating Point Mac Using Vedic Multiplier And Parallel Prefix Adder"

Dhananjaya A [1] Dr. Deepali Koppad [2]
Student IETE Member,Associate Professor
Mtech VLSI Design and Embedded SystemRVCE-Bangalore

ABSTRACT

Signal processing is a very hardware sensitive application hence to create high speed data processing systems like 3D rendering, 4G mobile internet, etc., we need better chips with high performance data path units and there is a growing need for research on alternative methods for signal processing hardware implementation. In most of DSP systems Multiply-Accumulate(MAC)is one of the main functions. The performance of the systems depends on the performance of the MAC units in place. More over these days real time signal processing systems require high through put and high performance MAC unit. In this work, an attempt is made to enhance the performance of the MAC unit using Urdhvatiryegbhyam sutra of the ancient Indian multiplication techniques and parallel prefix adder. In ancient times, this technique was used for decimal multiplication. There is not much clarity on.if this multiplier is best suited for high speed multiplication purpose in MAC units and in general, for VLSI implementation. Properties of the multiplication in concern, is studied, implemented, characterized and compared with convention multiplication techniques on the FPGA platform. Thus, a floating point MAC unit is designed and implemented using this technique and tested using different FPGAs like Spartan – 3E, Spartan-3A and vertex-2pro using Xilinx 13.2 ISE tool

Key Words:*Vedic Multiplier, Single Precision Floating point, MAC,*Kogge's stone adder

I. INTRODUCTION

Because of the high-precision, great dynamic range and easy operating rules, floating-point operations have found intensive applications in various fields that require high precision. In modern day computers, floating-point arithmetic operations are mainly performed by the co-processors. In systems without floating-point hardware, the CPU emulates it with a series of simpler fixed-point arithmetic operations that run on the integer arithmetic and logical unit [1,2].This saves the added cost of a floating-point unit (FPU) but is significantly slower. Coprocessors cannot fetch instructions from the main-memory, perform I/O, manage-memory and so on. These processors require the host main processor to fetch the coprocessor instructions from the memory and handle all operations aside from the coprocessor functions. High processor speeds demand high coprocessor operation speed.

High-Level Synthesis (HLS) is an emerging technology that synthesizes algorithms represented in high level languages (ANSI-C, Matlabetc) into an effective hardware (RTL). This is achieved by HLS compiler that analyses high level language features such as arithmetic resources, loops, branches and maps them into optimized hardware for data path and control elements. Arithmetic resources operate on both integer and real data. Most of the communication/DSP algorithms commonly use either real or integer operations. These algorithms can be easily represented in high level languages as they contain rich set of data types. Conversion of these algorithms (with real and integer arithmetic) into optimum hardware requires mapping of each operator to an effective hardware resource. As arithmetic operation on real operands is complex over integer operands, design and implementation of optimum hardware resources for real arithmetic is a challenging task. Integer arithmetic resources in-general, consume negligible hardware and can be freely used in the HLS synthesis process. Real arithmetic can be done either by fixed or floating-point methods. Fixed-point method is better in hardware performance than floating-point method, but the precision of operations and range of numbers that can be handled are limited. On the other hand, floating-point method offers high-precision and a great dynamic range at the expense of hardware-requirement and latency. Floating-point arithmetic resources alone consume 50 to 70 percent of the total hardware.

The hardware requirement in case of floating-point arithmetic or logical operation is high because of high complexity. The reason for high complexity is that, that every floating-point operation is divided into three stages namely; pre-normalization, arithmetic-operation and post-normalization; each of these stages in turn include a number of arithmetic and logical operations. Also, each floating-point operand is first divided into three parts namely sign, exponent and the fraction; which are then separately operated in each of the above mentioned stages. On the other hand, the operations on integer operands do not require grouping or normalization hence, are relatively less complex.

There are lots of similarities with respect to the resources being used in the architectures of floating-point addition, subtraction and multiplication. For example, concatenation of hidden-bits to the fraction-bits, addition or subtraction of exponents in the pre-normalization stage, shifting in the post-normalization stage are some of the common operations.

In this thesis, the similarities of different floating-point arithmetic operations are explored to implement a High Speed Floating Point Multiply Accumulate Hardware architecture based on Vedic multiplier and Kogge's stone adder , resulting in a decrease in the amount of hardware (number of LUTs) and logical delay. In other words, the amount of hardware used in the MAC architecture is found

to be less than the hardware required in floating-point units based Wallace, Braun , Serial Parallel and array multiplier. A large number of arithmetic and logical components are required in the design of any floating-point unit, the same holds good for the design of unified floating-point hardware architecture. To make sure that the most optimized architectures of individual components are used; architectures of all individual arithmetic and logic units are explored and the performance with respect to area and speed is found. The architectural explorations include: Kogge-Stone Adder, Carry-ripple adder, Carry-look-ahead adder, Wallace-Tree multiplier, Serial -Parallel Multiplier, Braun multiplier, Array Multiplier, and Barrel shifter.

It is found that the Floating-Point MAC Architecture results in an improvement in latency by a factor of 3 and an improvement in area (number of LUTs) by a factor of 1.5 as compared to the Wallace and array based floating-point MAC units.

II RELATED WORK:

Similar work was presented [1][2] for mac in which they used, array multiplier and modified booth multiplier which is not so fast method and in the base work they have not considering floating point number . In a MAC unit multiplier are the essential block that determine the combinational path delay and area required to implement the hardware, so in this project we used high speed and area optimized multiplier i.e Vedic multiplier are being used . Mainobjective of this work is to design a unit at RTL level that is capable of handling floating-point addition, subtraction and multiplication of single-precision by replacing it with a new type of multiplier based on Urdhva tiryakbhyamand the adders are replaced by the Kogge-Stone adders in the sparse mode. Using SPFP-unit new floating point MAC is build and characterizes on FPGA and further, investigates the new MACs feasibility for handling signed numbers and floating point values. It should also be able to handle various exceptions like not-a-number, infinity and overflow. The new MAC is implemented on an FPGA for hardware proof and characterization and evaluate MAC performance on FPGA.

III.DESIGN METHODOLOGY

Multiplication is an important fundamental function in arithmetic operations. Multiplication-based operations such as Multiply and Accumulate(MAC) and inner product are among some of the frequently used Computation- Intensive Arithmetic Functions(CIAF) currently implemented in many Digital Signal Processing (DSP) applications such as convolution, Fast Fourier Transform(FFT), filtering and in microprocessors in its arithmetic and logic unit . Since multiplication dominates the execution time of most DSP algorithms, so there is a need of high speed multiplier and in many DSP algorithms, the multiplier lies in the critical delay path and ultimately determines the performance of algorithm. Currently, multiplication time is still the dominant factor in determining the instruction cycle time of a DSP chip. So in this project we try to minimize the critical path delay of multiplier using vedic mathematics concept and finally MAC is implemented to evaluate the performance.

III. aVedic Multiplier (Urdhav-Triyak method)

The multiplier A[n] is of size ‘n’ words and the multiplicand B[m] is of size ‘m’ words, where A and B are given by equation 1 and 2.[4]

$$A[n] = \sum_{i=0}^{n-1} \left(A_i * X^i \right) \dots\dots\dots (1)$$

$$B[m] = \sum_{i=0}^{m-1} \left(B_i * X^i \right) \dots\dots\dots (2)$$

Product of A and B is given by equation 3.

$$P[n+m]=A[n]* B[m] \dots\dots\dots (3)$$

$$\sum_{i=1}^{n} \left(CP[0,0,i] * X^{i-1} \right) + \sum_{j=1}^{n-1} \left(CP[0,j,n] * X^{j+n-1} \right) + \sum_{k=1}^{n-1} \left(CP[k, k+m-n, n-k] * X^{m+k+1} \right) + \dots\dots\dots$$

Where

$$CP[n,m,q] = \sum_{i=n}^{q+n-1} \left(A_i * B_j \right) \dots\dots\dots (4)$$

Equation 4 gives the cross-product of two numbers. Where j = (m+ n + q - i -1).Figure 1 shows the N XN Vedic Multiplier with Parallel prefix adder.The NxN bit multiplier is structured using N/2XN/2 bit blocks as shown in Figure 1. In this figure the N bit multiplicand A can be decomposed into pair of N/2 bits AH-AL. Similarly multiplicand B can be decomposed into BH-BL. The 2Nbit product can be written as: P= A x B= (AH-AL) x (BH-BL)

III b. Floating Point Adder Architecture

In FP adder Architecture shown in Figure 2and the inputs go through the several stages that is discussed below

1. **Pre-processor:**This component will take the floating point input. Detect whether the number is Nan (Not a number), ‘0’, infinity and denormalized.
2. **Floating Point Alignment:**This component aligns the mantissa of the inputs according to their exponents and prepends leading 1
3. **Mantissa Adder:** which adds (subtracts) the aligned mantissa, determines presticky bit (or of discarded mantissa)
4. **Normalizer:** Finds the leading one and shifts it to the front producing a normalized sum. Determines if result is de-normal and does proper de-norm shifting. Also calculates round, and sticky bits as well as whether the sum is 0 or inexact.
5. **Rounder:** This component rounds normalized mantissa according to selected rounding mode and calculates the final exponent.
6. **Finalizer:** This component finally assembles all the pieces and uses special case information to determine correct final result. Also determines all exception flags.

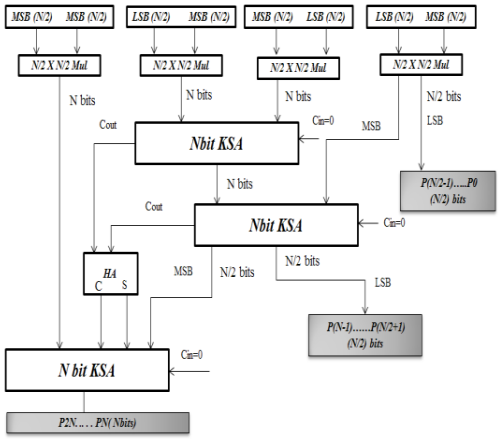

Figure1 :N X N Vedic multiplier

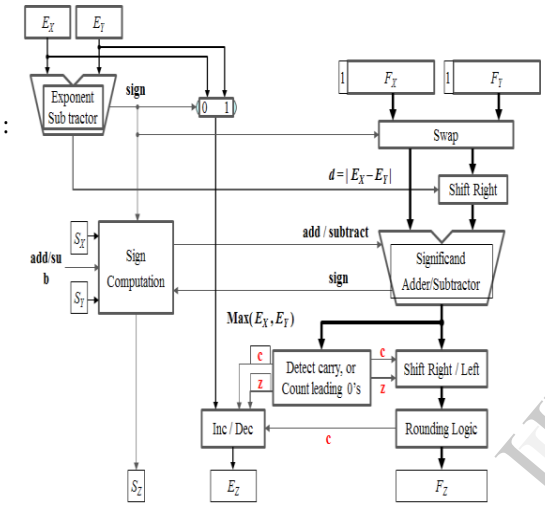

Figure 2: Floating Point Adder Architecture

III c. Floating Point Multiplier Architecture

Multiplication is one of the operations which require more number of steps in computation [9]. However, floating point multiplication is somewhat simpler than addition to implement because the significands are represented in sign- magnitude format, which is similar to the integer format. In FP Multiplier Architecture shown in Figure 3 and the operation of multiplication discussed below:

1.**Preprocessor:** This component will take the floating point input. Detect whether the number is Nan (Not a number), '0', infinity and denormalized.

2. **Pre-normalizer:** A normalized number is passed as it is by appending a 1 in the MSB.In case, any number is detected to be denormalized in the previous stage, then they are normalized here.

3. **Multiplier:** Here the product is calculated. If the product is having its MSB (or 48th bit) as a 1, then overflow is said to have occurred & this is indicated in a variable.

4. **Exponenter:**In this stage, the exponent of the result is calculated using the following formula:

Exponent_result = exponent_of_a + exponent_of_b – 127.

5. **Shifter**:From the exponent obtained above, see if the shifting needs to be done in this stage.If yes, then look at how much amount of shifting is to be done for the obtained product. Also indicate whether there is any loss in the precision because of the shifting.If no shifting is needed, then transfer the product as it is.

6. **Rounder:** Firstly the MSB of the product obtained from the above component is taken. Here we check the rounding modes as given in the last 2 bits of the control input & then round off the product according to the following:

00 = RN; 01 = RZ; 10 = RP; 11 = RM

7. **Flagger:**This component is used to indicate whether there is an overflow, underflow or inexact result after rounding.

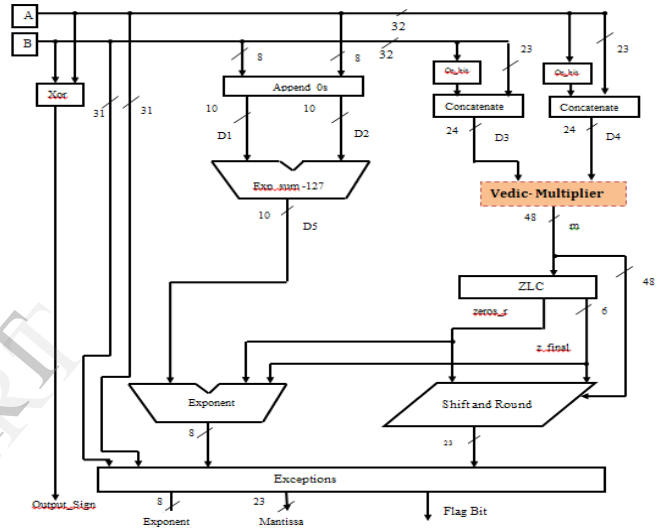

Figure 3: Floating Point Multiplier Architecture

III d. Floating Point MAC Architecture

A basic MAC architecture consists of multiplier and an accumulate adder organized as in figure. The MAC unit computes the product of two numbers and addsthe product to an accumulator register. The output of the register is fed back to one input of the adder as shown in figure. On each clock, the output of the multiplier is added to the register. It is known that combinational multiplier require a large amount logic, but can compute a product much quickly than the conventional method of shifting and add.

Overflow occurs when the number of Mac operations large. It is noteworthy to mention here that overflow in a signed adder occurs when two operands with same signal produce a results with a different signal from them. In this situation, the largest value (+ve or -ve) should be assigned to the result obtained. To be more specific, if eight bits are used to encode the values the addition of two numbers must fall in the interval from 0 to 127. On other hand, the addition of

two negative numbers must fall between -128 and -1. This is considered in our design approach.Floating Point MAC Architecture is shown in Figure 4

IV. Simulation Results and Discussion

The resultis grouped in Table 1 and Table 2 show the difference in combinatonal delay between various flaoting point adder and multiplier scheme. Table 3 show the performance of SPFP MAC units in term of delay, device utilization (number LUT's used) and maximum operating frequency. the highest performancefor all FP adder and multiplierare seen on the device spartan 3A with a speed grade of -4.The results suggest that the MAC design using Vedic matchematics is an exterem faster MAC and well ahead than other conventinal multiplier. Simulation results and graphs show below in Figure(5),(6)(7),8(a),8(b)respectively

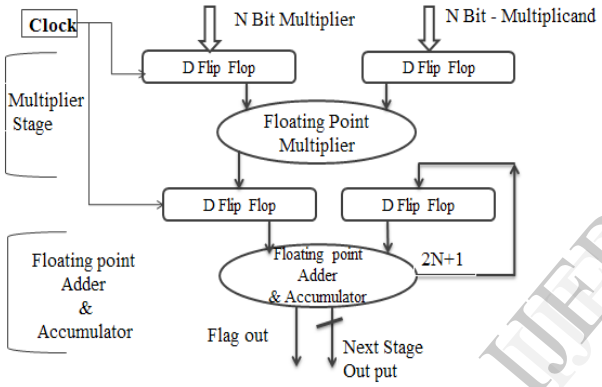

Figure 4: Floating Point MAC Architecture

Figure 5: Simulation results Single precision floating point Adder/Subtractor

Figure 6: Simulation results Single precision floating point Multiplier

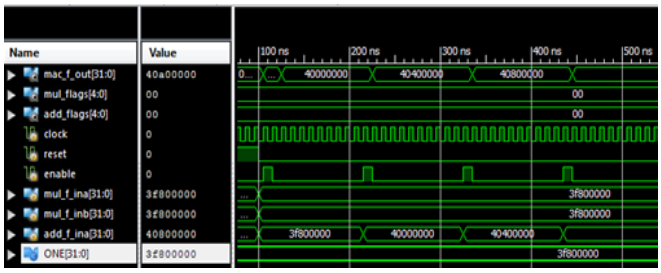

Figure7: Simulation results Single precision floating point MAC

Table 1: Synthesis result of Single Precision Floating Point Adder/Subtractor on Spartan 3A-XC3S500E

Component	Type	Combinational Path Delay (ns)
Single Precision Floating Point Adder	Kogge Stone Adder	47.230
	Carry– Ripple	72.469
	Carry Look –Ahead	63.427

Table 2: Synthesis result of Single Precision Floating Point Multiplier on Spartan 3A-XC3S500E

Component	Type	Combinational Path Delay (ns)
Single Precision Floating Point Multiplier	Proposed Vedic	85.294
	Wallace	110.864
	Array	115.251

Table 3: Synthesis result of Single Precision Floating Point MAC on Spartan 3A-XC3S500E

	Delay in ns	LUT's	Maximum Frequency in Mhz
Proposed Vedic Multiplier	121.402	4398	8.25
Braun Multiplier	138.023	4483	7.21
Wallace Multiplier	141.766	4029	7.054
Array Multiplier	195.581	5750	5.113
Serial parallel Multiplier	233.254	5557	4.287

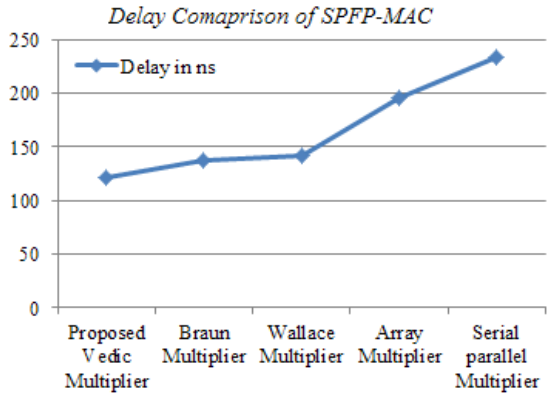

Figure 8(a): Delay Comparison of Various SPFP-MAC

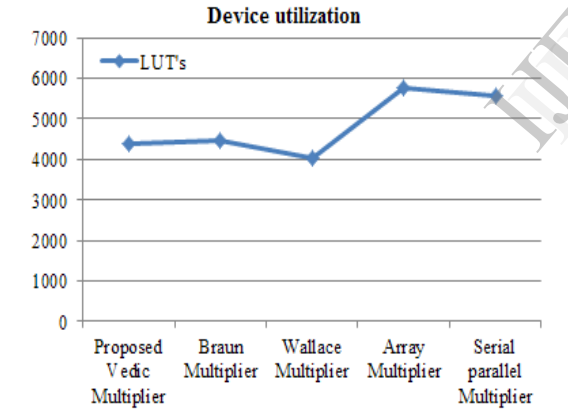

Figure 8(b): Device utilization of Various SPFP-MAC

V.CONCLUSION AND FUTURE WORK

The proposed floating point MAC based on Vedic mathematics based kogges stone adder proves to be highly efficient in term speed. Due to its regular and parallel structure , it can be realized on silicon as well. The benefit of using kogges stone adder is its high operational speed with higher number of bits. Pipelining the structure will improvise the performance by greater extent, can be enhanced to double precision and adjustable precisions also. Accumulator can be improvised to hold larger and more precise values, extra logic would be necessary to cope with increasing accumulator size.Precise power estimation can be done

REFERENCE

[1] John L. Hennessy and David A. Patterson. Computer Architecture A Quantitative Approach, Second Edition. Morgan Kaufmann, 1996.

[2] Deschamps, Jean-Pierrie and Sutter, D. Gustavo, Synthesis of Arithmetic Circuits, FPGA, ASIC and Embedded Systems, John Wiley & sons Inc. Publication (2006).

[3]Perry, Douglas, VHDL Programming by Example, McGraw Hill Publication (2002).

[4]Dinesh Kumar and Girish Chander Lall "Simulation and synthesis of 32-bit multiplier using configurable devices" IJAET Jan. 2013.
ISSN: 2231-1963

[5]Jagadguru Swami Sri Bharati Krishna Tirthaji Maharaja, "Vedic Mathematics: Sixteen simple Mathematical Formulae from the Veda", Delhi(1965).

[6]Anitha R, Alekhya Nelapati,Bagyaveereswaran4 Comparative Study of High performance Braun's Multiplier using FPGAs, IOSR Journal of Electronics and Communication Engineering ,Vol.1, PP 33-37,2012

[7]C. S. Wallace, "A suggestion for a fast multiplier," IEEE Trans.Electronic Computers, vol. EC13.

[8] P.D, Chidgupkar, and M.T. Karad , " The Implementaion of Vedic Algorithm in DSp ", Global Journal of Engg Edu., vol.8 pp.153-158,2004.

[9]VHDL Programming by example by Douglas Perri second edition, 1997.

[9]Gong Renxi, Zhang Shangjun,Zhang Hainan, MengXiaobi, Gong Wenying, XieLingling, Huang Yang "Hardware Implementation of High-Speed Floating-Point Multiplier Based on FPGA" Proceedings of 2009 4th International Conference on Computer Science & Education, IEEE, 1902-1906.

[10]O. L. MacSorley, "High-speed arithmetic in binary computers," Proc. OfIRE, vol. 49, January 1961.