International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-8, Issue-1, May 2019

Floating Point Multiplier Implementation A Broader Perspective

S. Umadevi, T. Vigneswaran

Abstract: ***Floating point multiplication is a crucial and more frequently used arithmetic operations in high power computing applications such as signal processing, image processing etc. High speed and memory requirement of IEEE-754 standard floating point multiplier makes its implementation more complex in many systems which require fast computing. Hence bringing an efficient architecture to complete the floating point multiplication operation with less computation time and with less memory requirement turned into one of the main research area in almost all the field of electronics. In this research review paper detail explanation about each and every architecture proposed for floating point multiplier is presented and its pros and corns are discussed. Later, comparison study between five different parallel floating point multiplier architectures which has been implemented on tsmc 180nm technology node is presented. From the results it has been observed that among all five proposed floating point parallel multiplication, a New Computation Sharing High speed Multiplier (CSHM) architecture based floating point implementation can yield a better results interms of all the performance parameters is concerned and better suitable for Finite Impulse Response (FIR) filtering operation. Cadence® nclaunch, rc compiler and Encounter are the tools used for simulating, synthesizing and physical design implementation.***

Key words: Floating point, Parallel architecture, FIR Filter

I. INTRODUCTION

1.1. Floating point number:

A floating decimal point present in floating point number will not have any fixed number of digits after and before its decimal point. Computers recognize real numbers which consist of fractions as floating point number only. A floating-point number consists of Sign bit, Mantissa and Exponent bits. The value of the floating point number is obtained by multiplying its mantissa with the base raised to the power of the exponent. This operation may shift radix point to its right from its implied position by a number of places equal to the value of the exponent if it is positive and to the left from its implied position by a number of places equal to the value of the exponent if it is negative. Floating point number wildly represented with different magnitudes and provides relative accuracy at all magnitudes. A scientific notation of a floating point can be expressed in the form of F*r^E.

Revised Manuscript Received on 30 May 2019.
***** Correspondence Author
S. Umadevi*, Assistant Professor in the Department of ECE, SRM University, Chennai, India.
T. Vigneswaran, Professor, School of Electronics Engineering, VIT University, Chennai. India

Where F represents fraction, r represents radix and E represents an exponent. Binary number uses radix as 2 and Decimal numbers uses radix as 10 (F×10^E). Floating point number will not have unique representation always. For example, the number 59.67 can be represented as 5.967×10^1, 0.5967×10^2, 0.05967×10^3, and so on.

The fractional part of a floating point number can be normalized and there is only a single non-zero digit present before the radix point in the normalized form. For example 9.966×10^1 is the normalized form representation for the decimal number 99.66 and 1.1111011×2^3 is the normalized form representation for the binary number 1111.1011. When the floating point numbers are represented with fixed number of bits say 32 or 64 bit, it losses its precision. It is due to the fact that n-bit binary pattern can have a finite 2^n distinct numbers. Hence it is not possible to represent all the real numbers, instead nearest approximation will be used which results in loss of accuracy. Hence floating number arithmetic is very much less efficient than integer arithmetic.

Real numbers are used in most scientific computations. Floating point standard has been used to represent the real numbers in computer. But there was no uniform format was maintained to use floating point numbers in a computer during 1950's which results in programs were not portable from one manufacturer's computer to another. To address this, a committee was formed by the Institute of Electrical and Electronics Engineers to standardize the format to be used to represent floating point number in a computer. IEEE standard for floating point number was adopted in 1985 and the same was used by all computer manufacturers. This standard introduces format for 32-bit and 64 bit floating point representation. Due to advanced computer technology, nowadays it is possible to use larger number of bits for floating point numbers. The same standard has been updated in 2008.The updated version retained all the features of the 1985 standard and also introduces new standard for 16 and128-bit numbers.

1.2. Floating point number representation:

IEEE floating point representation for binary numbers consists of three parts. For a single precision number (32-bit), each parts having following allocation of bits.

1. Sign bit [1 bit] – To represent positive or negative number.
2. Mantissa or significand or coefficient [23 bits].
3. Exponent [8 bits] - Uses biased representation for both to represent positive and negative numbers and the chosen bias value is 127.For example in order to store the value 189, the equivalent exponent value will be 62(189-127). A normalized form of significand of IEEE754 standard implies that its most significant bit always will be 1 and it is virtually assumed to be on the left of the decimal point.

Thus in the IEEE Standard, the significand is 24 bits long on which 23 bits of the significand will be stored in the memory and an implied 1 will be the most significant 24th bit. IEEE 754 representation of 32-bit floating point number is shown in Table 1.

Table 1: IEEE 754 representation of 32-bit floating point number.

b_0	$b_1b_2b_3….b_8$	$b_9b_{10}b_{11}…….b_{30}b_{31}$
Sign	Exponent	Significand

Thus, a floating point number in the IEEE Standard can be expressed as,

$$(-1)^s \times (1.f)_2 \times 2^{\text{exponent}-127} \quad (1)$$

where, s is the sign bit; s = 0 is used for positive numbers and s=1 for representing negative numbers. f represents the bits in the significand. The implied 1 as the most significant bit of the significand is explicitly shown for clarity.

For example, the real number 52.125 will be represented as shown in Table 2

52.125= 110100. 001= 1.10100001* 2^5

Normalized significand =.10100001

Exponent: e = 132

0	10000100	10100001000000000000000
Sign bit (1)	Exponent bits (8)	Significand bits (23)

Table 2: IEEE 754 representation of real number 52.125.

1.2.1Representation of Special Values:

The special value's like +0, -0, +∞, -∞ can be represented as shown in Table 3, Table 4, Table 5 and Table 6 respectively in IEEE754-1985 single precision format.

0	00000000	00000000000000000000000
Sign bit (1)	Exponent bits (8)	Significand bits (23)

Table 3: IEEE 754 representation of +0.

1	00000000	00000000000000000000000
Sign bit (1)	Exponent bits (8)	Significand bits (23)

Table 4: IEEE 754 representation of -0.

0	11111111	00000000000000000000000
Sign bit (1)	Exponent bits (8)	Significand bits (23)

Table 5: IEEE 754 representation of + ∞.

1	11111111	00000000000000000000000
Sign bit (1)	Exponent bits (8)	Significand bits (23)

Table 6: IEEE 754 representation of - ∞.

1.2.2. Representation of NaN Numbers:

The operation performed on pair of operands yields Indeterminate value if the result obtained could not defined mathematically. For example, if zero is divided by zero then the result will be indeterminate. This kind of result is called Not a number (NaN) in the IEEE 754-1985 standard. There are two types of NaN. They are Quiet NaN (QNaN) and Signalling NaN (SNaN). If the result of an operation is not defined, it is called QNaN. When the result of operands is smaller or larger than the floating point number which can be stored, then the computer will give an error message and which will be represented as SNaN.The IEEE754-1985 standard representation for QNaN and SNaN is shown in Table 7 and Table 8 respectively.

1 or 0	11111111	00010000000000000000000
Sign bit (1)	Exponent bits (8)	Significand bits (23)

Table 7: IEEE 754 representation of QNaN.

1 or 0	11111111	10 000 000 000 001 000 000 000
Sign bit (1)	Exponent bits (8)	Significand bits (23)

Table 8: IEEE 754 representation of SNaN.

1.2.3. Representation of Largest and Smallest Positive Floating Point Number:

The largest possible positive floating point number that can be stored by the computer is shown in Table 9.

Significand: 1111…..1 + $(1\text{-}2^{-23})$ = $2\text{-}2^{-23}$

Exponent : (254 -127) = 127

Largest Number = 3.403 * 10^{38}

0	11111110	11111111111111111111111
Sign bit (1)	Exponent bits (8)	Significand bits (23)

Table 9: Largest floating point number representation in IEEE754-1985 Standard.

The smallest possible positive floating point number which will be stored by the computer is shown in Table 10.

Significand: 1.0

Exponent = 1-127 = -126

The smallest normalized number = 1.1755 * 10^{-38}

0	00000001	00000000000000000000000
Sign bit (1)	Exponent bits (8)	Significand bits (23)

Table 10: Smallest floating point number representation in IEEE754-1985 Standard.

1.2.4. IEEE 754 – 64 bit Floating Point standard – 1985:

The IEEE 754 standard for 64 bit [3] floating point number representation is otherwise called as double precision representation and it is very similar to 32 bit floating point representation. The allocation of bits for the exponent has been increased from 8 to 11 bits in 64 bit standard compared to 32bit standard and allocation of bits for the significand has been increased from 23 to 52 bits in 64 bit standard compared to 32 bit standard. As like single precision format, double precision format also uses biased representation with the bias value of 1023.

IEEE 754-1985 [7] standard for 64-bit floating point number is shown in Table 11.

Sign bit	Exponent bits	Significand bits
(1)	(11)	(52)

Table 11: IEEE 754-1985 standard of 64-bit floating point number

Thus, a floating point number in the IEEE 754-64 bit standard can be expressed as,

$$(-1)^{s}.(1.\ f\ )_2 \times 2^{\text{exponent}\ -\ 1023} \quad (2)$$

where, s is the sign bit; s = 0 is used for positive numbers and s=1 for representing negative numbers.

f represents the bits in the significand. The largest possible positive number which can be represented using 64 bit standard is given in Table 12 and the value of it approximately equal to $[10]^{3083}$

	1111111110	111.....1111
ign bit	xponent bits	ignificandbits
(1)	(11)	(52)

Table 12: Largest possible positive number in IEEE754-64bit standard

The smallest possible positive number which can be represented using 64 bit standard is given in Table 13 and the value of it approximately equal to 2^{-1022}

	0000000001	000.....0000
ign bit	xponent bits	ignificand bits
(1)	(11)	(52)

Table 13: Smallest possible positive number in IEEE754-64bit standard

The definitions of ± 0, ± ∞, QNaN, SNaN remain same as IEEE754 standard for 32 bit except that the number of bits in the exponent and significand are now 11 and 52 respectively.

1.2.5. IEEE 754 Floating Point Standard – 2008

To enhance the scope of IEEE 754 floating point standard of 1985, this new standard has been introduced. This standard satisfies the following demands of professionals.

- Professionals who require exact decimal computation in financial transactions.
- Professionals who are working in graphics area.
- Professionals who use high performance computers for numeric intensive computation.

This standard introduces floating point format for 16 and 128 bits long without altering the format of 64 and 32 bits numbers. These are respectively called as half and quadruple precision. For pixel storage in graphics, the 16 bit format is used which is shown Table 14. The 16 bit format is not used for computation. Exponent uses biased representation with the bias value of 15.

b_0	$b_1b_2b_3b_4b_5$	$b_6b_7b_8$.....$b_{14}b_{15}$
Sign bit	Exponent bits	Significand bits
(1)	(5)	(10)

Table 14: IEEE 754-2008, 16 bit format.

The maximum and minimum positive integer numbers that will be stored with this format are 65504 and $0.61* 10^{-4}$ respectively. The definitions of ± ∞, ± 0, SNaN and NaN will be same as in 32-bit format. The floating point number representation with 128 bits is possible nowadays with improvements in computer technology which is shown in Table 15.

Sign Bit	Exponent Bits	Significand Bits
(1)	(14)	(113)

Table 15: IEEE 754-2008, 128 bit format

The maximum and minimum positive integer numbers which have been stored in 128 bits formats are 10^{4932} and 10^{-4931} respectively. The definitions of ± ∞, ± 0, SNaN and NaN remain same as in 1985 standard.

1.3 Floating point: Arithmetic operations:

Arithmetic operation on floating point numbers [2],[4] includes addition, subtraction, multiplication and division.

1.3.1. Addition operation:

Addition operation on floating point number consists of following steps.

- Align radix points – Bring both the numbers exponent to be same
- Perform Addition – Hidden bit of mantissa also to be considered
- Normalize the result- The result will be normalized to get the "hidden bit" to be a 1 in mantissa.

Consider an example 100_{10} to be added with 0.25_{10}. Table 16 explains the floating point addition between the numbers 100_{10} and 0.25_{10}.

IEEE754-32 bit representati for a given number	100_{10} - 0 10000101 10010000000000000000000 0.25_{10} -0 01111101 00000000000000000000000
Step 1: Align Radix so that both the numbers exponent are same	100_{10} - 0 10000101 10010000000000000000000 0.25_{10} -0 10000101 00000001000000000000000 [Right shifted eight times]
Step 2: Performing addition	0 10000101 1.10010000000000000000000 (100) + 0 10000101 0.00000001000000000000000 (.25) 0 10000101 1.10010001000000000000000 - Result
Step3: Normalizing result to get hidden bit to be in the mantissa	Not required for this example ,since Mantissa already hidden bit as "1"
Final Result	0 10000101 10010001000000000000000 – 100.25_{10}

Table 16: Floating Point Addition operation

1.3.2. Subtraction operation:

Subtraction operation on floating point number consists of following steps.

- Align radix points – Bring both the numbers exponent to be same
- Perform Subtraction – Hidden bit of mantissa also to be considered.

The sign bit value of the resultant will be obtained by comparing the sign bit of both the numbers. When big number is subtracted from small number sign bit to be changed as 1.

- Normalize the result- The result will be normalized to get the "hidden bit" to be a 1 in mantissa.

Consider an example 0.25_{10} to be subtracted from 100_{10} . Then the steps to be followed have been given in Table 17.

IEEE754-32 representation for given number	100_{10} -0 10000101 10010000000000000000000 0.25_{10} -0 01111101 00000000000000000000000	
Step 1: Align R so that both numbers exponent same	100_{10} -0 10000101 10010000000000000000000 0.25_{10} -0 10000101 00000001000000000000000 [Right shifted eight times]	
Step 2: Perform Subtraction	0 10000101 1.10010000000000000000000 -0 10000101 0.00000001000000000000000	(100) (.25)
	0 00000000 1.10001111000000000000000 - Result	
Step3: Normalizing result to get hidden bit to be in the mantissa	Not required for this example ,since Mantissa already hidden bit as "1"	
Final Result	0 10000101 10010001000000000000000 – 99.75_{10}	

Table 17: Floating Point Subtraction

1.3.3. Multiplication operation:

Multiplication operation on floating point number obtained based on the flowchart given in Figure 1.

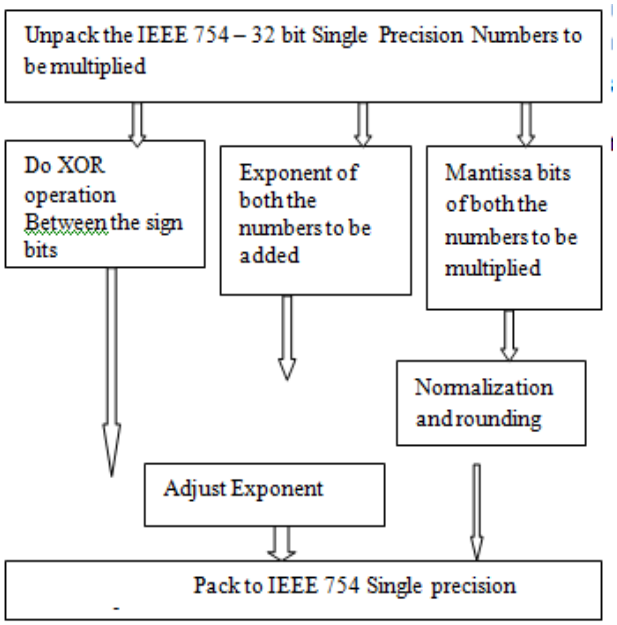


Figure 1: Floating point multiplier operation

Multiplication operation on floating point number obtained based on the flowchart is given in Figure 16

1.3.4. Division operation:

Floating point division operation is similar to multiplication. Unsigned division operation to be performed on mantissas and exponents will be subtracted.

1.3.5. Rounding

The result of any floating point operation needed to be rounded since arithmetic operations on floating point values compute results that cannot be represented in the given amount of precision. There are many types of rounding methods [6] are there. They are

- Round towards Zero: Otherwise called as truncation.

This method identifies the number of bits available for the representation and keep only that many bits in the result. Remaining bits in the result will be thrown away.

- Round towards +∞: The result obtained will be rounded towards +∞.

Example:

1.43 will be rounded to 1.5, if two decimal places have been chosen.

-2.66 will be rounded to -2.6, if two decimal places have been chosen.

- Round toward -∞: The result obtained will be rounded towards -∞.

Example:

1.43 will be rounded to 1.4, if two decimal places have been chosen.

-2.66 will be rounded to -2.7, if two decimal places have been chosen.

In this review paper, different multiplier architectures or algorithms have been proposed till now to perform mantissa multiplication is presented in section 2 and analytical comparison of different parallel floating point multiplier architecture implemented on same technology node is presented in section 3.

II. LITERATURE REVIEW ON DIFFERENT MULTIPLICATION ALGORITHM

Nowadays, Digital Signal Processing (DSP) consists of various techniques for improving the reliability and accuracy of digital systems. High speed multipliers and adders are of very important component in DSP to perform Digital filtering, convolution and Discrete Fourier Transform. Accuracy of any digital system is mostly depends on multipliers used in the system. Different types of applications expects different performance measures from the multipliers such as high speed, low power consumption, scalability, reconfigurability, regularity of layout and less area etc. In general, Algorithm used to perform the operation of multiplication consists of two steps. One step is for generating partial products and the other one that accumulates it with previous partial products. The general scheme used for unsigned multiplication is shown in Figure 2.

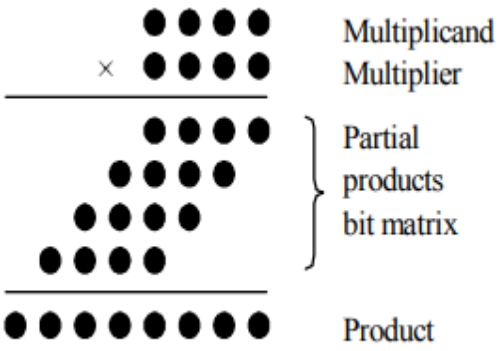


Figure 2: General multiplication technique

Each row or partial product is obtained by multiplying one digit of the multiplier times the multiplicand. The lower order digits of partial product is determined from just one multiplicand digit, but other digits include the effects of the carry from the digits to the right. The sum of the partial products gives the product results. The multipliers are classified as serial and parallel multipliers based on various ways of performing multiplication, range from, treating one pair of digits at a time, to generate and summing all partial products simultaneously with parallel hardware units. Serial multipliers have increased delay while parallel multiplier eliminates this problem by processing the bits simultaneously. Array multipliers can be implemented by directly mapping the manual multiplication process into hardware. Array of adder circuits will be used to accumulate the partial products in array multiplier. An n x n array multiplier requires 'n (n-1)' adders and 'n^2' AND gates, where 'n' represents the number of bits. With all these available multiplier structures and algorithms, it is essential to choose a particular type of multiplier suited for a given application based on certain parameters like delay, power consumption, and complexity of the circuit. In the Baugh wooley multiplier algorithm [8], to accumulate partial products an array of carry-save adders has been used. To generate the final product, Carry-Propagate Adder has been used. Since carry-save adders propagate the carry to the next level of adder rather than to the adjacent ones, critical path delay of the multiplier gets reduced. But the major drawback is that the array multipliers are not suitable for signed binary multiplication, because of the absence of sign bit. Booth algorithm [9] treats both negative and positive numbers uniformly and it is a powerful signed-number multiplication algorithm.It is a method that will reduce the number of multiplicand multiples. Booth's algorithm effectively skips over runs of 1's and 0's that it encounters in the multiplier and thus optimizes the number of add and shift operations. For large multipliers, the modified Booth recoding technique [10] is used. This converts the multiplier into higher radix numbers thereby reducing the number of stages of partial products to be added, by half. The scheme is modified from the original Booth's recoding to avoid a variable-size partial product array. Generally, array multipliers have larger delay but offer the benefits of regular layout with simpler interconnects. Wallace multiplier [11] is a fast technique to operate multiplication of larger operands. Compared to array multiplier, in Wallace tree multiplier both the number of adder cells and critical path have been reduced. The Wallace tree multiplier is a column compression multiplier and its works based on the principle of achieving partial product accumulation through successively reducing the number of bits of information in each .The Wallace tree consists of numerous levels of such column compression structures until finally, only two full-width operands remain. Those two operands will be added using fast carry-propagate adder to obtain the product result. The final product obtained by compressing the partial product matrix as quickly as possible using as much as hardware possible.Dadda multiplier [12] is similar to Wallace multiplier and it is a hardware based multiplier design. In Dadda multipliers fewer columns are compressed in the first few steps of the column compression tree, and more columns in the later levels of the multiplier. Tree multipliers have the shortest logic delay but has irregular layouts with complicated interconnects. Increased number wiring capacitances introduce crosstalk noise and implementing tree multiplier requires more physical design effort.Novel binary multiplication architecture [13] has been proposed for high speed and low power applications. Computation speed has been improved by generating all partial products in one step and by using binary tree network, all partial products are added. The number of levels needed to create this binary tree structure is $\log_2(n)$. The architecture for 4*4 MBT structure [13] is shown in Figure 3. The steps followed to get the multiplication result of 4*4 are shown below. In the below example, there are four partial products and all the partial products generated at a time. Then two partial sums will be obtained from addition of each two partial products as mentioned in the above example. The final product formed from adding two partial sums with the second level of adder.

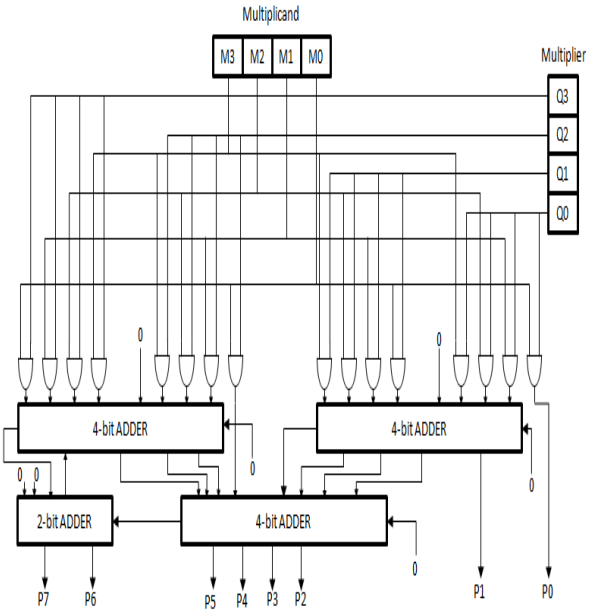
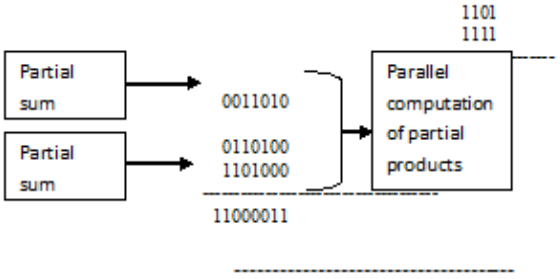


Figure 3: 4*4 MBT structure



Since this architecture requires (n-1) of n-bit adders, it requires total of n (n-1) full adder cells to complete the operation. An efficient low power basic cell [14] has been used as full adder. The Full adder has been developed based on combining XOR gates and transmission gates which yields low power dissipation. The worst case delay identified for multiplier is given in the below equation.

$d' + nd[log2(n)]$ (3)

Where,

d' – Propagation delay of AND gate

d - Propagation delay of full adder cells.

32*32 binary multiplication implemented with this new architecture gives 51.6% improved speed compared to array multiplier. The MBT structure based multiplication has good repeatability in its physical layout and computation time of this architecture is a function of O (log2N) whereas in array multiplier it is O (Nlog2N). Leapfrog multiplier [15] has been proposed to fill up the research gap present in conventional array multiplier and which is shown in Figure 4.The conventional array multiplier is one of the most popular architectures due to its regular interconnect and simplicity. Array multiplier has architectural disadvantage due to its signal transition activity[16].This is mainly due to non uniform path delays in the structure, which results in multiple signal transitions on internal nodes before they settle to a final value. These multiple transitions are redundant and dissipate unnecessary power. It has been proved that [17], 50 % of the power consumption in the array multiplier is because of spurious transitions. In the past, improvements in power of array multipliers have been obtained as a result of bottom-up analysis. Given an array topology, the characteristics of the constituents or its structure can be modified to yield lower power dissipation. By equalizing path delays from input to outputs either by using latches or by using inverters [18], unnecessary transition can be avoided. The spurious transitions can be avoided by providing a signals just when they were needed .But these techniques introduces area and power penalty due to extra logic. Later to improve the speed of the array multiplier, equalizing delays among sum and the carry path was proposed [19]. This was achieved by rearranging multiplier cells into three groups and which works in parallel to produce the carry outputs almost at the same time as sum outputs. But in Temporal tiling architecture [15], instead of achieving delay balancing either through modifying components or by introducing delay elements, overall delay-balanced structure is achieved through usage of existing components with delay imbalances. This research work proves that temporally tiled array multiplier achieves 50% and 30% improvement in delay and power dissipation compared to conventional array multiplier when it was implemented for 16*16 bit multiplication and on 0.6 µm technology with supply voltage 2.5 V. The leapfrog multiplier is the one on which the sum output S_{out} from row 'i' skips the next row "i+1" of adders to the adder input in row "i+2". Through this concept delay has been reduced compared to conventional array multiplier.

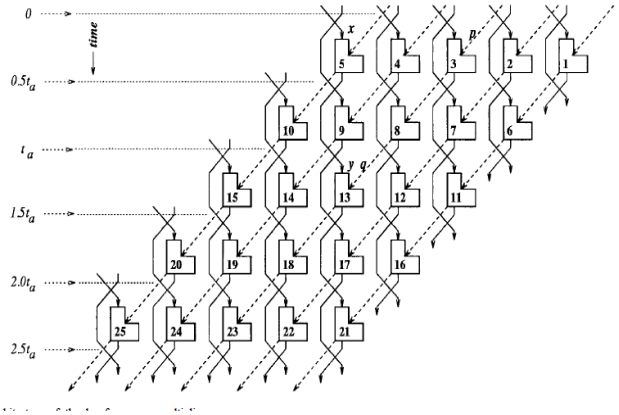


Figure 4: Leapfrog Multiplier [15]

A five-counter addition unit based new iterative array [2] parallel multiplier has been proposed. It was faster than the previous iterative-type multiplier due to its regular interconnection structure. Implementation of this multiplier requires approximately $n^2/2$ cells which are half compared to n^2 cells of the previous iterative array multiplier. However, circuit complexity of five-counter cell gets doubled compared to basic cell of a other iterative parallel multiplier. This proposed multiplication algorithm permits efficient VLSI realization and also suitable for both signed and unsigned numbers. This multiplexer based multiplier array can be used in a pipeline form which requires fewer delay elements compared to previously reported pipeline multipliers. The research work, speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach [21] is a method and an algorithm for generation of a parallel multiplier which optimizes the speed. The algorithm used in this research work applicable to any multiplier size and adaptable to technology node. The algorithm was implemented in 1µm CMOS ASIC technology with the partial product reduction tree with the minimum number of cells.It takes advantages of the uneven delays through a fulladder in order to build a global compressor that minimizes the critical path of the multiplier. In the Karatsuba algorithm for fast multiplication process [22], if the two numbers say X and Y to be multiplied then X and Y will be divided into two halves. Let us assume that the number of bits (n) in X and Y is equal. The numbers X and Y is divided into two halves [23] as per equation number 4 and 5.

$X = X_1 * 2^{n/2} + X_r$

$Y = Y_1 * 2^{n/2} + Y_r$

The product XY can be obtained as expressed in equation

$XY = 2^n X_1Y_1 + 2^{n/2} *[(X_1+X_r).(Y_1+Y_r)-X_1Y_1 - X_rY_r] + X_rY_r$ (6)

Example

X = 1111 (Divide X into two halves, last two MSB is named as X_l and first two LSB is called X_r)

Y = 1000 (Divide Y into two halves, last two MSB is named as Y_l and first two LSB is called Y_r)

Multiplication of X and Y following the below steps.

$X_l \times Y_l = 110$;

$(X_l + X_r) \times (Y_l + Y_r) = 01100$;

$X_r \times Y_r = 000$;

$(X_l + X_r) \times (Y_l + Y_r) - (X_l \times Y_l) - (X_r \times Y_r) = 110$

$$X.Y = ((X_l \times Y_l) * 2^4 + \big((X_l + X_r) \times (Y_l + Y_r)) - (X_l \times Yl - Xr \times Yr * 22) + Xr \times Yr = 1111000.$$

In karatsuba multiplication method number of multiplication steps required to compute the product result gets reduced .They require three multiplications rather than four for accomplishing multiplication between two 4 bit numbers. It is helpful when it is implemented on FPGA because the number of DSP blocks [23] usage can be reduced. But this algorithm can be applicable if and only if both multiplicand and multiplier has same number of bits and complexity of architecture of the algorithm increases if the number of bits to be multiplied gets increases.

A Computation sharing scheme [24] is the one on which common computations are frequently performed. The existing 8*8 computation sharing multiplier is shown in Figure 20.In this algorithm, computation time of multiplication operation significantly reduced by add and shift operation. The computation sharing scheme concept is very much useful in vector scalar product[25] to achieve the complexity reduction.

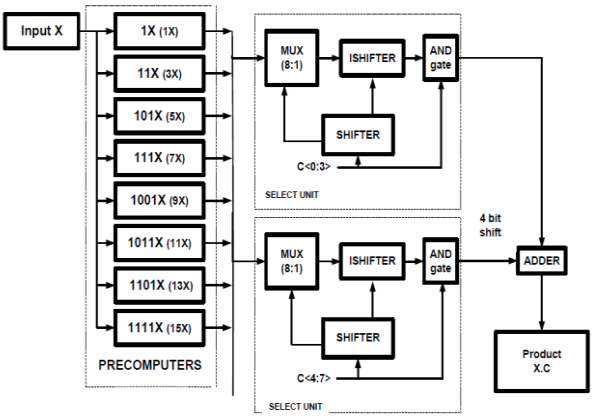


***Figure 5: Block diagram of 8*8* CSHM**

In this computation sharing high speed multipliers, alphabets in the precomputer unit has been selected using fixed size look up rule. In the fixed look up rule coefficient decomposition length is fixed as Y. Let the coefficient length is Z then it is been divided into Z/Y parts, each part consist of Y bits. Y/Z should be an integer. In existing algorithm, the used number of alphabet entries in precomputer unit is equivalent to odd numbers which are less than (or) equal to 2^Y.

For an example, X=10101011 and C=11100011, the logical flow starts with coefficient decomposition. If the decomposition length Y is chosen as 4, then the coefficient divided into two parts. The coefficient LSB bits, 0011 given as input to SHIFTER of upper select unit and MSB bits, 1110 given as input to SHIFTER of lower select unit. Each shifter in the individual select unit does circular right shift with a given input until it finds "1" in the LSB bit. From the resultant output of individual SHIFTER, first three bits starting from MSB will act as a select line for the respective MUX present in the select units. The counts SHIFTER took to get "1" as LSB while shifting the coefficient act as an ISHIFTER input. For a taken example, MUX output of upper and lower select is going to be 3X and 7X respectively. SHIFTER present in each select unit does left circular shift to an output of an MUX with number of times equal to number of counts it is getting as an input. Then lower select unit ISHIFTER output left shifted 4 times and added with upper select unit ISHIFTER output to get the final product X.C. Later Floating point multiplier has been implemented using computation sharing multiplication [26].The disadvantage of this computation sharing multiplication method is that, number of alphabet entries in the precomputer unit follows fixed size look up rule which indicates if decomposition length gets increase (L) then number of alphabet entries(2^L -1) in the precomputer unit also increases which may leads to computational complexity. In real world digital signal processing application both coefficient and input signal will be represented with more number of bits and in that case decomposition length will be more to compute the operation quickly. To address this problem, later a modified computation sharing high speed multiplier has been proposed [27].

The sticky-bit generation methods for floating point multipliers [28] research work addresses the critical delay introduced by rounding techniques when two IEEE 754 standard floating point numbers are multiplied. Except one or two rounding techniques all other IEEE rounding modes test the (n-2) least significant product bits and result of the test is indicated by the sticky bit. Since fast generation of the sticky-bit is required to improve the speed of the floating point multiplication process, various sticky bit generation designs are developed. Hence a novel sticky bit generation technique introduced in this research work which is independent of multiplier hardware and fastest among all other methods. The proposed implementation is very much useful for implementing floating point multiplications in FPGA and requires less hardware resources. This method will be applicable to any floating point multiplier without changing the mantissa multiplier design. Architecture for multiplication based on bypassing the redundant operations of multiplier [30] have been proposed and proved it is taking less computation time. The best results are obtained if the number of zeros in the multiplicand section has more than half of the number of zeros than in the multiplier section. This leads to extra hardware which helps in bypassing the operations and thereby increasing the operational speed of the system by significantly reducing the critical path of the system. The concurrent operation happening due to parallel architecture adopted leads to faster processing. The working of bypassing technique involves the disabling of the operations where it is not required. The disabling action is based on multiplier bits bj which ranges from (zero <= j <= n-1). If the multiplicand has zeros, then the adders are made non-functional which leads to saving of power and enhancement of speed. This disabling action in the multiplier unit requires some alterations in the existing conventional adder structure. Bypassing technique for four bit multiplier is shown in Figure 6. In adder section, the main adder cells in the intermediate rows consist of CSA [30] adders. The ripple carry adder (RCA) used in the final row. In this architecture, the multiplier array uses 3N adders for the multiplication operation. However, delay time required for the worst case is (N+2) full adders delay. Three tristate buffers [30] and two MUXs are utilized in the modified carry save adder which does the bypassing technique.

The tristate buffer takes the decision that whether to disable the adder cell or enable it. It is done based on the multiplier bit (bj).Two MUXs are designed to pick up the appropriate outputs. Suppose, if bit b2 is zero, the adder in the third row of multiplier is disabled which leads to the output of adders of second row can be passed to adders of fourth row directly.

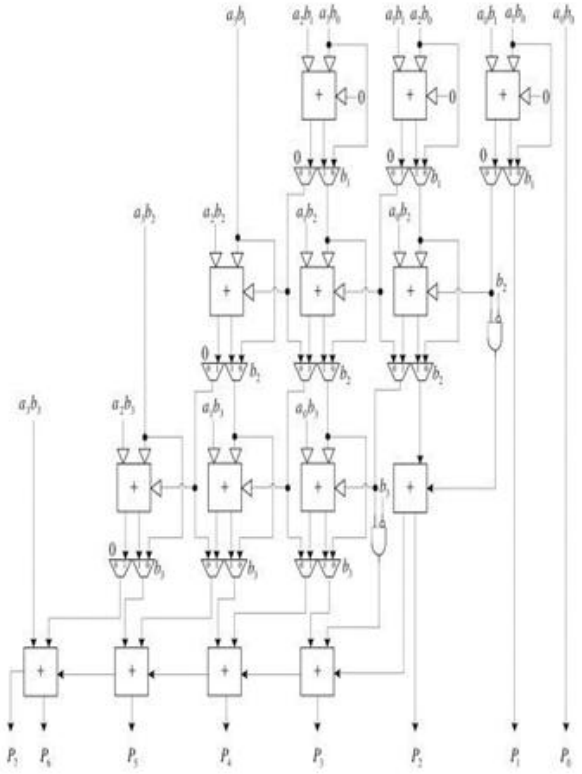


Figure 6: Row Bypassing Multiplier Using Carry save Adder

In order to multiply two binary numbers efficiently, Integer multiplication using Nikihilam method [31] of Vedic mathematics [32] has been used to get the efficient multiplication between two binary numbers. In this research work, researcher compares Nikihilam algorithm with karatsuba algorithm and it has been proved that number of multiplication steps required to compute the multiplication operation gets reduced compared to karatsuba algorithm and due to that computation time gets reduced. The below tables shows how nikhilam method is efficient than karatsuba both for inter multiplication and for binary multiplication.

Karatsuba algorithm for decimal multiplication Algorithm: 1.Consider a two digit decimal numbers and the numbers to be multiplied will be divided as shown below a1a2*b1b2 2. Multiplication result will be obtained using below equation. Result = 100 * A + 10 * D + B and D = C - A - B Where, A = a1 * b1, B = a2 * b2, C = Ha1 + a2H* Hb1 + b2H	**Nikilam sutra for decimal multiplication** Algorithm: Integer | Base Difference Multiplicand | A | x-(x-a) Multiplier | B | x-(x-b) (x-a-b) | a.b Result | x.(x-a-b)+a.b
Example: 95 *96 = 9120 A= 81; B= 30; C= 210 ; D=99 and the Result = 8100+990+30= 9120.	Example: 95*96=9120 Integer | Base Difference Multiplicand | A (95) | x-(x-a) =5 Multiplier | B (96) | x-(x-b)=4 (x-a-b) | a.b=20 Result | x.(x-a-b)+a.b | 100*(91)+20=912
Inference :3 multiplication steps are required	Inference : Only one Multiplication step is involved

Table 18: Decimal multiplication using Karatsuba and Nikilam algorithm

Karatsuba algorithm for binary multiplication Algorithm: 1.Consider a four digit binary numbers and the numbers to be multiplied will be divided as shown below a1a2*b1b2 2. Multiplication result will be obtained using below equation. ??????? = 2^4 * ?? + 2^2 * ?? + ?? ????? ?? = ?? - ?? - ?? ??h?????,?? = ??1 * ??1, ??= ??2 * ??2 , ?? = H??1 + ??2H* H??1 + ??2H	**Nikilam sutra for binary multiplication** Algorithm: 1. Let us consider 4-bit multiplication of 1 (Multiplicand M) with 1111(Multiplier N). 2. Compute A=1111-1000; Subtract from the near base 3. Compute B=1111-1000; Subtract from the sa base 4. Compute C=A-100=11 5.Compute D= B-100=11 6.Compute E= C-10=1 7.Compute F=D-10=1 8.Compute G=E*F=1 9.Comput H= (C+F)*10 +G 10.Compute I = (A+D)*100+H 11. Result J= (M+B)*1000+I.
Example: 1111*1111 = 11100001 A= 1001; B= 1001; C= 100100 ; D=10010 and the Result = 11100001. Inference :3 multiplication steps are required	Inference: Only one Multiplication step is involvec step number 8. Remaining steps computation will achieved through add/shift operation.

Table 19: Binary multiplication using Karatsuba and Nikilam algorithm

In this research work, performing binary number multiplication using nikhilam sutra of Vedic mathematics has been introduced. This method gives more efficient result when both multiplicand and multiplier are near to same base power. It has been also proved that compared to all other multipliers, nikhilam sutra based multiplication takes less computation time. The disadvantage of this research work can be listed as follows

- Can be applicable if the same number of digits present in both multiplicand and multiplier
- There was no clear idea was explored if the numbers to be multiplied is not near to the base.

The modified CSHM architecture [27] has been proposed with fixed alphabet entry in the precomputer unit. Compared to the existing CSHM, this architecture [24] proved that it takes less chip area and power without compromising delay. Figure 7, Shows the block diagram of 8*8 modified CSHM.

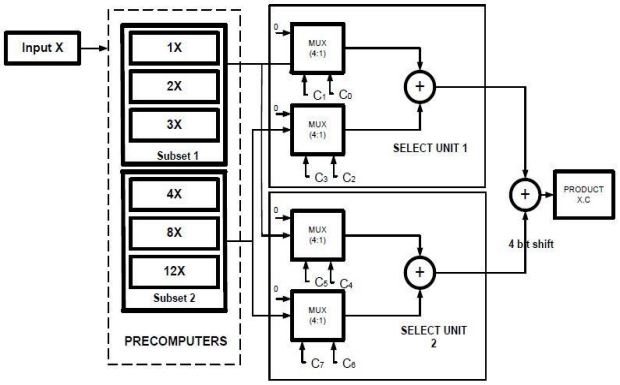


Figure 7: Block diagram of 8*8 Modified CSHM

The coefficient of length Y is always divided into W bits and each W bits again sub divided into groups of 2 bits $\{c_0c_1, c_2c_3,\}$ and the number of 4:1 MUX required in each select unit is W/2. All odd 4:1 MUX present in select unit has input signal as (0, 1x, 2x and 3x). All even 4:1 MUX present in select unit has input signal as (0, 4x, 8x and 12x). All select unit's completes its computation on same time.If the coefficient is $C=11100011_2 = 227_{10}$, then the final result Q=X*C is obtained by the following expression,

$$Q = \{3X + 0\} + \{4\ bit\ Leftshift(2X + 12X)\} \quad (4)$$

$$Q = \{3X + 224X\} \quad (5)$$

$$Q = 227 * X \quad (6)$$

The resultant adder output of select unit 2 is left shifted "4" times and added with resultant adder output of select unit 1 to get the final product of X.C. The select unit of proposed CSHM is highly area and power efficient compared to existing design. Research's [8] have proved that modified computation sharing multiplier takes less area and low power compare to existing CSHM by designing both the architecture on 180nm technology. Though modified CSHM reduces area and power, computation time remain almost same as existing CSHM.

The dual way architecture [33] has been introduced which composed of two 53*27 size subtrees.The same architecture can be used for both single precision multiplication as well as double precision multiplication. The mantissa of the multiplicator is divided into two halves. For a double precision multiplication, both the trees are used but in a parallel manner hence reduces the computation time. For a single precision multiplication, one tree alone can be used and another one has been kept idle. But the disadvantage of this method is increase in area. Area is increased due to increase in pipeline stages which is bringing greater number of register and also increase in pipeline stages reduces the efficiency of instruction. But in order to balance both area and delay, partial products are reduced firstly by modified booth encoding technique and divided into two parts and calculated in separate multiplier tree. The usage of Urdhva-Tiryabhyam sutra of Vedic mathematics [34] for IEEE 754 double precision floating point multiplier implementation in Xilinx platform against conventional multiplier synthesized result proves that number of Look Up Table [LUT] , computation time required and chip area gets reduces in double precision floating point multiplication implemented using Vedic mathematics than conventional method

A very high speed and low power multiplier architecture has been proposed with improved column bypassing scheme [36].Improved column bypassing scheme [ICBS] achieved through power gating approach. In this research work new adder cell with optimized hardware is proposed for full adder and it combines both row [30] and column bypassing [35] schemes. The architecture of this adder reduces the propagation delay and power consumption, even when ICBS is not in use. Simulation result [UMC 90nm and 0.9 V CMOS technology] shows that the proposed multiplier architecture facilitates reduction of switching transitions and leakage power. It is also found better in terms of area occupancy and propagation delay. The input test patterns are taken in such a way that number of zero's and one's present in the multiplicand is exactly 50%. The proposed multiplier can achieve more power saving if the input test pattern has more no. of zero's than the no. of one's. It has been verified that proposed multiplier outperforms previously designed multipliers more effectively at all frequencies and ranks much higher in performance when used for low frequency applications.

In order to achieve more efficient floating point multiplication with respect to complexity and speed both karatsubha and Urdhva-Tiryabhyam sutra of Vedic mathematics algorithm has been used together [37]. Whereas for exponent addition different parallel prefix adders has been used. Comparison analysis of floating point multiplier implementation has been presented using Sklansky [38], Brent-Kung [39] and Knowles [40] parallel prefix adders. For simulation Xilinx ISE package has been used. From the results it is observed that, Knoules prefix adder along with karatsuba and Vedic technique has the best performance in terms of delay and area compared Multiplication using Sklansky and Brent-Kung adders.Though Modified CSHM [27] reduces area and power compare to existing CSHM, computation time remain almost same as existing CSHM [24]. So there is a new CSHM architecture has been proposed [41] with the few modifications in the modified computation sharing multiplier architecture. In this new scheme along with reduction in chip power and chip area, the reduction in computation time also achieved compare to existing CSHM [24]. The proposed 8*8 new computational sharing multiplier architecture is shown in Figure 8. This architecture consists of precomputer, select unit and an adder.

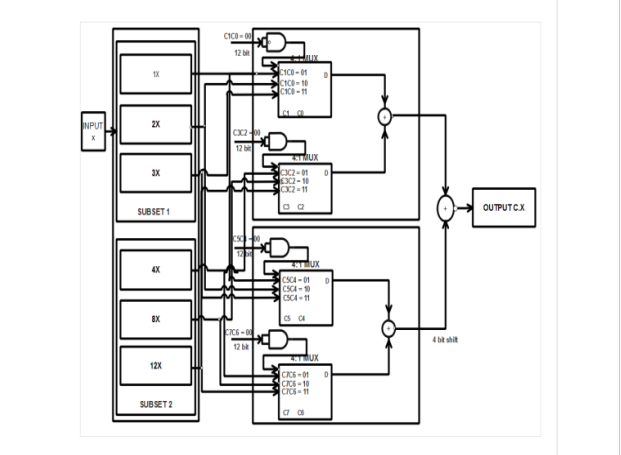


Figure 8: 8*8 newly proposed CSHM

The precomputer consists of two different alphabet sets, first set consists of {1X,2X,3X} and second set consists of {4X,8X,12X} respectively as in modified CSHM. These sets produce multiplication of alphabets with the input X. The select unit consists of 4:1 MUX, AND gate and an adder. The coefficient C is grouped into 2 bits and given as select lines to the MUXs. Depending on the coefficient, MUX will take the input from the precomputer. The number of required MUXs will be half of the length of coefficient C. In this architecture, each 4:1 MUX is embedded with one 2 input AND gate (each of 12 bits) in such a way that whenever select line is "00" ,AND gate will give an output and whenever select line is other than "00" MUX will give an output. This architecture is having the advantage of reducing the computation time further if one or two sets of coefficient bits is 00 (worst case) or when all the coefficient bits are zero (best case). This happens because when "00" comes,

the output directly taken from the AND gate due to that gates involved in MUX will not be accounted for computation time calculation. Thereby increases the speed of the operation and reduces the power also. For example, If the coefficient C = 10000011_2, C_1C_0=11 will be the select line for MUX 1, C_3C_2=00 will be the select line for MUX2 and so on. Since the coefficients C_5, C_4, C_3 and C_2 are zeros, instead going to MUXs it will directly go to AND gate. For getting the final product, the output of second select unit is left shifted to 4 bits. This architecture has been extended for floating point multiplication.CSHM architecture with Karatsuba algorithm [42] has been proposed which combines the advantage of computation sharing multiplier and Karatsuba algorithm. Advantage of modified computation sharing multiplier is that fixed set of alphabet entry in the precomputer unit. All the alphabets in the precomputer unit is multiplied with given input once and stored in a memory. Remaining all the multiplication is achieved through shift and add method. The select unit present in the architecture does shift and add operation. All the select unit presents in the architecture works parallel manner. Hence area, power consumption and computation has been reduced to significant level compared to all other architecture which are proposed earlier. To reduce the computation time further, a novel idea has been proposed in this work. The idea is to replacing conventional multiplication with karatsubha multiplication in the precomputer unit which is shown in Figure 9.

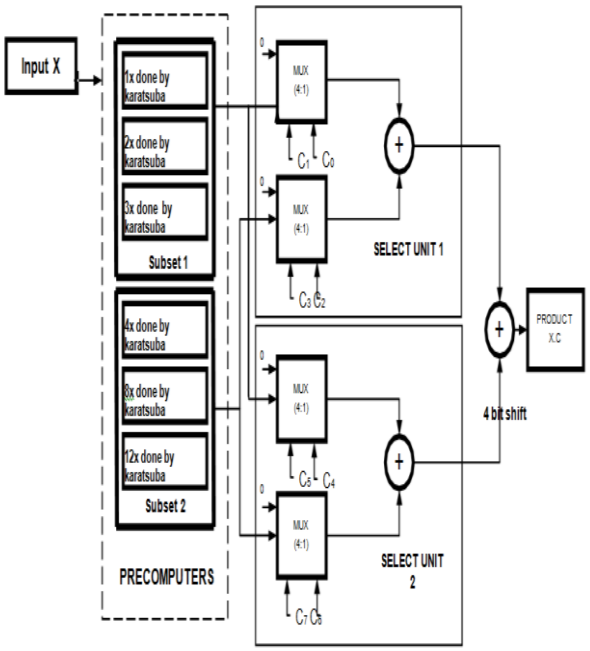


Figure 9: 8X8 Block diagram of proposed CSHM using karatsuba algorithm

III. RESULTS AND DISCUSSION

In general, multipliers are classified into serial, parallel and array multiplier based on the way it works. In serial multiplier, only one pair of digits will be treated at a time and hence takes much computation time. In parallel multipliers, generation and summing of all partial products will be happening simultaneously and hence computation time gets reduced compared to serial multiplier. In many DSP applications, FIR filtering [24] playing a major role to filter out the unwanted signals or noise. The performance of the FIR filter is fully depends on the floating point multiplier involved in each and every tap of the filter. So mostly parallel floating point multiplier will be preferred rather than serial multiplier which has more computation time and array multiplier which requires more hardware. Hence in this research work, many parallel floating point multipliers has been implemented in tsmc 180nm technology node and its performance comparison is listed in Table 20. Cadence® nclaunch, Cadence® RC compiler and Cadence® Encounter tools has been used for simulation, synthesis and physical design implementation.From the literature, it is clear that compared to all other multiplier architecture/method proposed earlier, computation sharing high speed multiplier [24] architecture implementation finishes its multiplication operation very quickly. It is because of minimum multiplication steps only involved in the architecture and also due to parallel computation of partial products. Later Modified computation sharing multiplier has been proposed [27] which reduces chip area and power consumption further without compromising computation time compared to existing CSHM. Later New CSHM [41] was proposed with small architecture changes so that along with chip area and power, computation time also gets reduced. Later to reduce the computation time further, there was an attempt made to do fast multiplication [Karatsubha [42] and Row bypassing] in alphabet entries of precomputer present in the CSHM instead of conventional multiplication.All these architectures implemented in 180nm technology and obtained results for various performance parameters after post layout simulation is mentioned in Table 6. Compared to all other architecture CSHM with row by passing technique yield less computation time with respect to worst case analysis. Here worst case represents multiplicand bits with minimum number of zeros present in it. This has been achieved due to the usage of bypassing multiplication instead of conventional one in the precomputer unit. Same way usage of karatsuba algorithm in the precomputer unit reduces the computation time compared to existing CSHM. But it is observed that CSHM with row bypassing and CSHM with Karatsuba, though they are reducing the computation time, due to the extra logic usage for bypassing and fast multiplication there is an increase in cell count and chip area. Chip power has been calculated by the tool in such a way that, for a taken worst case test pattern, power consumption of the gates which are active. Again from Figure 10,it is inferred that modified CSHM, new CSHM and CSHM with row bypassing almost having equal power consumption and which are 10% lesser than existing CSHM.This has been achieved for one floating point multiplication operation. Large improvement in the power consumption can be arrived if we use this for the application where many number of floating point operation involved like FIR filtering. Both in modified CSHM and new CSHM chip area and cell count has been reduced to approximately 8.5% and 24% compared to existing CSHM which is shown in Figure 11.

But in case of CSHM with Row bypassing and CSHM with karatsuba, both chip area and cell count has been increased due to extra hardware present in the bypassing logic and fast multiplication unit respectively. So from the all five proposed floating point parallel multiplication, New CSHM [41] can be the best one interms of all the performance parameters concerned and better suitable for FIR filter operation.

Parameter	Existing CSHM floating point multip -lier	Modified CSHM floating point multip -lier	New CSHM floating point multip -lier	CSHM with Karatsuba floating Point multiplier	CSHM with Row by passing floating point multiplier
Computatio time[ns] [Worst case analysis]	12.126	11.614	10.712	8.92	6.12
Chip power[mW]	4.36	3.97	3.94	4.5	3.89
Chiparea [μm^2]	64634	59160	59400	72129	67549
Cell count	3010	2280	2287	3020	4121

Table 20: Parameter comparison of different Parallel

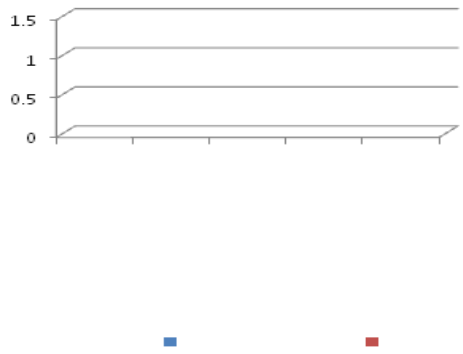


Figure 10: Computation time and Chip Power Consumption comparison of different multipliers

Multiplier

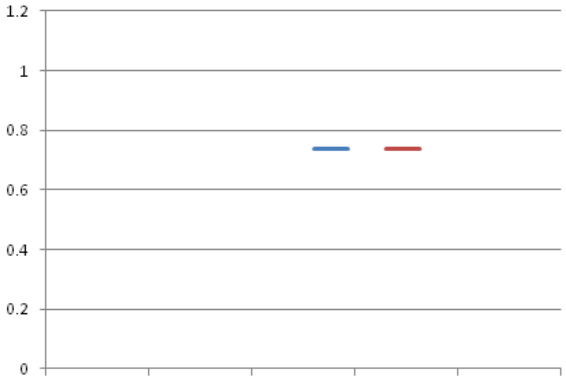


Figure 11: Chip area and Cell count comparison of different multipliers

IV. CONCLUSION

In this survey paper detail explanation about floating point number, floating point representation standards and different floating point operations possible are presented in section 1. In section 2, methodology used in each and every architecture proposed for mantissa multiplication of floating point multiplier is discussed along with its pros and corns. In section 3, comparison study between five different parallel floating point multiplier architectures which has been implemented on tsmc 180nm technology node is analyzed with its performance parameters. From the post layout simulation results it has been observed that among all five proposed floating point parallel multiplication, a New Computation Sharing High speed Multiplier (CSHM) architecture based floating point implementation [41] can yield a balanced performance interms all the parameters is concerned and better suitable for Finite Impulse Response (FIR) filtering operation.

REFERENCES

1. V.Rajaraman," IEEE standard for floating point number" in Resonance, January 2016.
2. David Goldberg, "What every computer scientist should know about Floating-point Arithmetic" in ACM computing surveys, Volume 23, Issue 1, March 1991.
3. W.J.Cody, "Floating point standards- Theory and Practice" in Reliability in computing, 1998, pp 99-107.
4. Liang-Kai Wang and Michel J.Schulte, "Decimal floating point division using Newton- Raphson Iteration", in Proceedings of the 15th IEEE International conference on application specific systems, Architectures and processing(ASAP'04),2004.
5. The Institute of Electrical and Electronics Engineering "IEEE standard for floating-point arithmetic", IEEE, Ny 10016-5997, USA, 29th August 2008.
6. David J.Kuck, Douglass Parker and Ahmed H. Sameh, "Analysis of Rounding methods in Floating point Arithmetic", in IEEE transactions on computers, Volume 26, Issue 7, July 1977.
7. The Institute of Electrical and Electronics Engineers, "IEEE Standard for Binary Floating - Point Arithmetic"ANSI/IEEE Std 754.
8. Charles R.Baugh and A.Wooley, "A two's complement parallel array multiplication algorithm", in IEEE transactions on computers, Volume 22, Issue 12, December 1973, pp 1045-1047.
9. A.D. Booth, "A Signed Binary Multiplication Technique," Quarterly Journal of Mechanics and Applied Mathematics, vol. 4, 1951, pp. 236-240, 1951.
10. L. P. Rubinfield, "A Proof of the Modified Booth's Algorithm for Multiplication." IEEE Transactions on Computers. Vol. C-24, pp. 1014-1015, 1975.
11. C. S. Wallace, "A suggestion for a fast multiplier," IEEE Transactions on Electronic Computers, vol. EC-13, no. 1, pp. 14–17, 1964.
12. L. Dadda, "Some Schemes for Parallel Multipliers", Alta Frequenza, vol. 34, pp. 349-356, 1965.
13. E.Abu shama, M.B.Maaz and M.A.Bayoumi, "A fast and low power multiplier Architecture", in proceedings of the 39th Midwest symposium on circuits and systems, August 1996.
14. E. Abu-Shama, A. Elchoumi, S. Sayed, M. Bayoumi,"An Efficient Low Power Basic Cell for Adders," in Proceeding of 38th Midwest Symposium on Circuit and Systems, Rio de Janeiro, Brazil, 1995.
15. Shivaling s. Mahant-shetti, Paras T. Balsara and Carl lemonds, "High Performance low power array multiplier using Temporal Tiling", in IEEE transactions on very large scale integration (VLSI) systems, Volume 7, Issue 1, and March 1999.
16. J. Leijten, J. van Meerbergen, and J. Jess, "Analysis and reduction of glitches in synchronous networks," in European Design Test Conf., 1995, pp. 398–403.
17. C. Lemonds and S. S. Mahant-Shetti, "A low power 16 by 16 multiplier using transition reduction circuitry," in Int. Workshop Low Power Design, 1994, pp. 139–140.
18. U. Ko, P. T. Balsara, and W. Lee, "A self-timed method to minimize spurious transitions in low power CMOS circuits," in Symp. Low Power Electron. 1994, pp. 62–63.

19. C. P. Lerouge, P. Girard, and J. Colardelle, “A fast 16-bit NMOS parallel multiplier,” IEEE J. Solid-State Circuits, vol. SC-19, pp. 338–342, Mar.1984.
20. Kiamal Z.Pekmestzi, “Multiplexer-Based Array Multipliers”, in IEEE transactions on computers, Volume 48, Issue 1, January 1999.
21. Vojin G. Oklobdzija, David villager and Simon S. Liu, “A method for speech optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach”, Volume 45, Issue 3, March 1996.
22. A. Karatsuba and Yu. Ofman "Multiplication of Many-Digital Numbers by Automatic Computers" in Proceedings of the USSR Academy of Sciences, pp 293–294, 1962.
23. A. Karatsuba, "The Complexity of Computations" in Proceedings of the Steklov Institute of Mathematics pp186–202, 1995.
24. Jongsun park, Woopyo Jeong, Hamid Mahmoodi-Meim , Yongtao Wang, Hunsoo Choo and Kaushik Roy, “ Computation Sharing Programmable FIR filter for Low-Power and High-Performance Applications,” IEEE Journal of Solid-State Circuits, Vol. 39, No. 2, 2004.
25. Kantamaneni. Sravanthi, Dr.V.V.D.Prasad Battula,Veera Vasantha Rao, “Design of FIR filter Design Using Sharing Multiplier With Low Delay” in International Refereed Journal of Engineering and Science (IRJES) 2319-183X, 2319-1821 Volume 1, Issue 1, 2012.
26. Shivanantham S, Jagannadha Naidu K, Balamurugan S, Bhuvana Phaneendra, “Low Power Floating Point Computation sharing Multiplier for signal processing applications” in International Journal of Engineering and Technology, Vol 5, No.2, 2013.
27. Umadevi.S, T. Vigneswaran, S. Kadam Vinay and V. Seerengasamy, “A Novel ,Less Area Computation Sharing High Speed Multiplier Architecture for FIR Filter Design,” in Research Journal of Applied Sciences, Engineering and Technology , Volume 10, Issue 7, July 2015,pp. 816-823.
28. Mustafa Gok.Metin Mete ozbilen, “Evaluation of sticky-bit generation methods for floating point multipliers” in Springer science+business media, 2008.
29. Muhammad K and Roy K, “ Reduced computational redundancy implementation of the DSP algorithms using computation sharing vector scaling,” IEEE Trans. Very Large Scale Integration (VLSI) Systems, vol.10, no. 3, pp.292-300, June 2002.
30. Ko-Chi Kuo, Chi-Wen Chou,”Low power and high speed multiplier design with row bypassing and parallel architecture” in Microelectronics Journal, Volume 41, Issue 10,pp 639-650,2010.
31. Shri Prakash dwivedi “An efficient Multiplication algorithm using nikhilam method”, in fifth international conference on advances in recent technologies in communication and computing , sep 2013.
32. Tirthaji, B.K.M.; “Vedic mathematics”, Motilal Banarsidass Publication, 1992.
33. Xu zhou and zhimin Tang, “A new architecture of a Fast Floating – point multiplier” in Advanced parallel processing Technologies, pp 23-30, 2003.
34. Mownika.V, Girish Gandhi S and Leele Mohan C, “International journal of Innovations in Engineering and Technology, 2016.
35. M.C. Wen, S.J. Wang, and Y.N. Lin, “Low-power parallel multiplier with column bypassing”, in Electronics Letters. vol. 41, pp. 581-583,2005.
36. Pankaj Kumar and Rajendra kumar Sharma, “Low power multiplier design with improved column bypassing scheme”, in Electrical and Electronics Engineering: An international journal, Volume 5, Issue 1, February 2016.
37. K V Gowreesrinivas and P.Samundiswary, “Design and analysis of single Precision Floating Point Multiplication using Karatsuba Algorithm and Parallel Prefix Adders”, in international conference on signal processing, communication and Networking (ICSCN-2017), March 2017.
38. Sklansky, J,” Conditional-Sum Addition Logic” in IEEE Transactions on Electronic Computers, 1960.
39. R.P. Brent and H.T.Kung,” A Regular Layout for Parallel Adders”, in IEEE Transactions on computers, Volume 31, Issue 3, March 1982, pp 260-264.
40. Knowles. S,” A family of adders” in Proceedings 15th IEEE Symposium on Computer Arithmetic. ARITH-15 2001.
41. Umadevi, S. & Vigneswaran. T, “Reliability improved, high performance FIR filter design using new computation sharing multiplier: suitable for signal processing applications” in Cluster Computing,2018.
42. P. K. Dhivya Gayathri, S. Umadevi, T.Vigneswaran, “A review on floating point multiplier architecture using semi custom vlsi design flow”, in journal of advanced research in Dynamical and control systems, issue 18, 2017,pp 823-831.

AUTHORS PROFILE

Ms.S.Umadevi received the graduate degree in Electrical and Electronics Engineering from PSNA college of Engineering and Technology affiliated to Anna University in 2006, M.E (Applied Electronics) from Kongu Engineering College affiliated to Anna University in 2008 and doing Ph.D degree in Low power VLSI design in VIT University. In Aug 2008 she joined as a Project Engineer in VLSI domain of WIPRO technologies Bangalore. Later in October 2009, she joined as an Assistant professor in the department of ECE, SRM University. Her research interests include different aspects of Low power VLSI circuit design and high speed algorithm, architecture development and post layout optimization. Presently she is working as an Assistant professor, School of Electronics engineering, VIT University, Chennai.

Dr. T.Vigneswaran received the graduate degree in Electrical and Electronics Engineering from Bharathidhasan University in 2000, M.E (VLSI Design) from College Of Engineering, Guindy, Anna University in 2003 and the Ph.D degree in Low power VLSI design from SRM University, India in 2009. In 2003, he joined as lecturer in the department of ECE, SRM University.His research interests include different aspects of Low power VLSI circuit design and high speed algorithm and architecture development. He is a member of Indian science congress.Presently he is working as a professor, School of Electronics engineering, VIT University, Chennai.