Research Journal of Applied Sciences, Engineering and Technology 10(7): 816-823, 2015
DOI:10.19026/rjaset.10.2435
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2015 Maxwell Scientific Publication Corp.
Submitted: February 22, 2015 Accepted: March 12, 2015 Published: July 10, 2015

Research Article

A Novel Less Area Computation Sharing High Speed Multiplier Architecture for FIR Filter Design

[1]S. Umadevi, [1]T. Vigneswaran, [1]S. Kadam Vinay and [2]V. Seerengasamy
[1]School of Electronics Engineering, VIT University, Chennai, India
[2]Deparment of Mathematics, PSNA College of Engineering and Technology, Anna University, Dindigul, India

Abstract: High performance multiplier designs are the prime need of emerging digital filtering operations. This research study presents a novel architecture of reduced area computation sharing multiplier for Finite Impulse Response (FIR) filter. The same architecture is extended for the floating point applications. The chosen pre-computer alphabet set is the most prominent feature of this architecture. The proposed integer based Computation Sharing High speed Multiplier (CSHM) efficiently computes the vector scalar product based on the distributed arithmetic. The proposed CSHM (8*8) shows 29.81% of area and 46% of power optimization over existing CSHM style. The experimental results for Look up Table (LUT) based implementation shows 57% improvement than the LUT required to implement a existing 8*8 CSHM based FIR filter. The proposed design style is also extended for Floating Point (FP) multiplication. The 4 tap Floating Point Finite Impulse Response (FP FIR) filter is designed in Xilinx environment (No. of LUT's 5919) and TSMC 180 nm technology (power 29.5 mW and area 212636.79 um^2) using proposed CSHM. The performance results get improves in terms of power and area over conventional design style.

Keywords: Computation Sharing High speed Multiplier (CSHM), Finite Impulse Response (FIR), Floating Point Finite Impulse Response (FP FIR), Look up Table (LUT)

INTRODUCTION

Due to the rapid growth in multimedia applications and popularity of the portable battery-powered systems, there is a high demand of high performance and low power signal processing devices. Since many telephony and data communications applications have been moving to digital, the need for digital filtering methods continues to grow. In digital VLSI systems, filtering operation is widely employed in various applications such as video processing, image processing and wireless communication. In the filtering operation multiplier unit is an essential and extensively used element. Complexity reduction of multiplier unit present in FIR filter implementations has also been of particular interest since lower computational complexity leads to low-power as well as high performance design. Multiplication operation is nothing but generation and addition of partial products. Generation of partial products in the multiplier consumes huge power as well as area.

Many research proposals have been devoted to high speed, low power, less area multipliers for various applications. Previously Booth's multiplier (Alan *et al*., 2000) Wallace tree multiplier (Kernhof *et al*., 1989), Array multiplier (Mark, 1989) for filtering application explored the low power and area efficient design. FPGA implementation of such design consumes large LUT area. Also computation performed has large power consumption. Some algorithmic techniques like CSD format (Mamatha and Ramchandram, 2012) is used to reduce the addition operation in multiplication, which benefits in less arithmetic operations but again a number system conversion itself increases the area. The reconfigurable multiplier design based on the reordering of partial product (Wu *et al*., 1998) and row-bypassing technique (Kuo and Chou, 2010) is proposed to reduce the switching power. But row bypassing technique is useful only when numbers of zeros are more in multiplicand. A Computation Sharing Multiplier (CSHM) (Jongsun *et al*., 2004) architecture has been proposed in which common computations (alphabets) are identified and all the partial products been achieved only by shift and add method. Also CSHM overcomes the drawback of area and power and applicable for applications with programmable filter coefficients. Since, redundant computations are removed; CSHM achieves high-performance in filtering operation by reusing the optimal pre-computations and low-power consumption. In addition to

Corresponding Author: S. Umadevi, School of Electronics Engineering, VIT University, Chennai, India

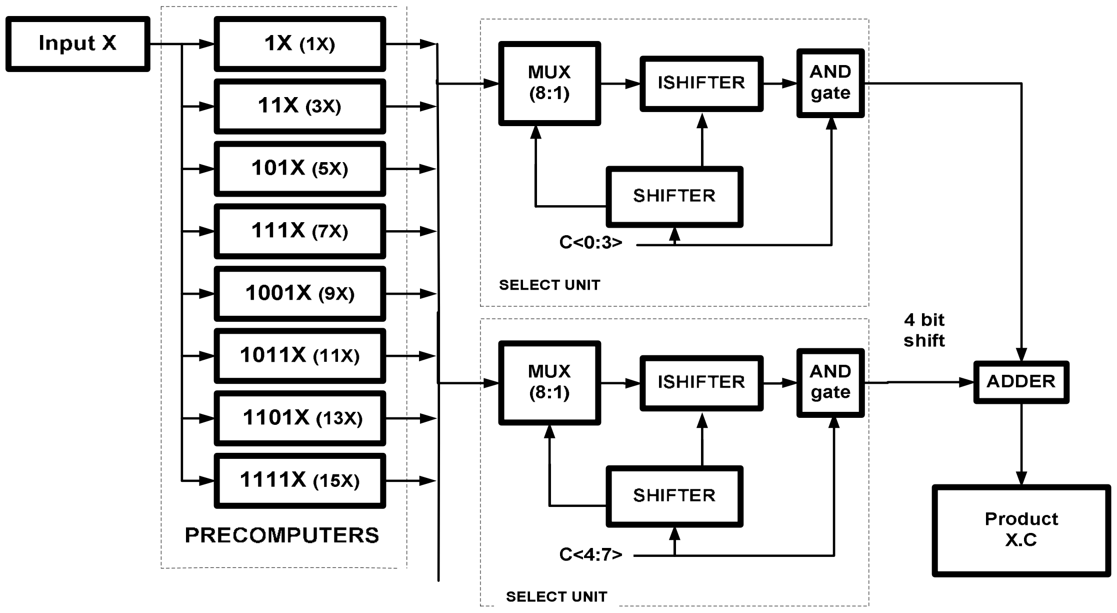

Fig. 1: Block diagram of existing CSHM

signal processing applications, CSHM can also be used in data compression/decompression for testing and VLSI testing applications. In the literature (Jongsun *et al*., 2004; Sravanthi *et al*., 2012) CSHM is used only in fixed-point FIR filter implementation.

But existing CSHM uses fixed-size look-up rule to select the alphabet. According to this rule, if the alphabet length is increases then the number of alphabet set also increases which increases the amount of computation in pre-computer unit. This leads to large area and power consumption. Another disadvantage of existing CSHM is that it compulsorily requires SHIFTER and ISHIFTER which occupies more area in LUT based designs.

This drawback is wisely overcome in proposed CSHM. The proposed CSHM architecture, consist of pre-computer having constant alphabet set count irrespective of length of the coefficient, input and number of bits to represent the coefficient. The chosen alphabet set is easy and efficient to compute as compare to existing one. In the existing design select unit consist of 8:1 MUX, SHIFTER and ISHIFTER which is used to generate the partial products. In the proposed design the same operation is been implemented by using two 4:1 MUX and an adder. The proposed architecture is extended for signed number also.

This proposed work is intended to give less area and low power implementation of FIR filter. So that this novel proposed technique can be used in many DSP applications.

Existing CSHM: Computation sharing scheme is highly efficient where common computations are frequently performed. In signal processing, FIR filtering can be expressed as multiplication of vector by scalars C.X. Expression (1) shows representation of FIR filter using difference equation form:

$$Y(n) = C_0X(n) + C_1X(n-1) + \cdots + C_NX(n-N) \quad (1)$$

Distributed arithmetic avails the easy implementation method of above expression. The existing computation sharing multiplier (Jongsun *et al*., 2004) shown in Fig. 1 which presents parallel multiplication of input signal X with all coefficients. In this algorithm, the multiplication operation is significantly simplified as add and shift operations. Complexity reduction in the vector scalar product is been achieved by using the concept of computation sharing.

In the vector scaling operation, we can carefully select a set of small bit sequence so that the same multiplication result can be obtained by only add and shift operation. For example, C = 01011110 has to multiply with input X. The multiplication of X.C is obtained by (0101.X) <<4+ (0111.X) <<1. If both (0101).X and (0111).X are available, then the entire multiplication process is reduced to few addition and shift operations. The chosen basic bit sequences are called as alphabets. An alphabet set is a set of alphabets that spans all the coefficients in vector C.

Since alphabets are small bit sequence the multiplication of alphabets with operand X can be done without seriously compromising the performance. In the existing computation sharing multiplier design fixed size look up rules been used. In the fixed sized lookup rules multiplication, the maximum alphabet length L is fixed. Let the coefficient length is W then it is been divided into W/L parts, each part consist of L bits. W/L should be an integer. Once L is determined, an alphabet set should be able to express any L bit number by one of its alphabet multiplied by the power of two. In existing algorithm, the used alphabet set consist of odd numbers less than or equal to 2^L -1.

As shown in Fig. 1 8*8 CSHM structure based on computation sharing scheme, the pre-computer performs the computations α_k.X, for k = 0, 1, 2, …7. Where α_k is the alphabet set. As a result, the outputs of pre-computer are 1X, 3X, 5X, 7X, 9X, 11X, 13X and 15X for L = 4. To find the correct alphabet shifter performs the right shift operation until it encounters 1 from LSB side and sends an appropriate select signal to 8:1 MUX along with the exact shift signal to ISHIFTER.

The 8:1 MUXs select the correct alphabet among the eight values received from the pre-computed value α_k.X, for k = 0, 1, 2…7. ISHIFTER simply inverse the operation performed by SHIFTER. AND gates are used to deal with zero coefficients. SHIFTER-ISHIFTER-AND gate forms the select unit. The upper select unit generates the multiplication of 4 LSB bits of coefficient with input X. The lower select unit produces the product of 4 MSB bits with input X. A shift of 4 bits is performed when those two values are fed to the final adder. A simple adder produces the final result.

Let us consider an example, X = 00100011 and C = 01100011. The coefficient is divided into two parts consisting of 4 bits. 0011 is fed to SHIFTER of upper select unit and 0110 to that lower one. In the upper select unit, shifter shifts 0011 to the right until it encounters 1 in the LSB and sends the select signal 001 to the MUX which chooses 3X among pre-computer output. Shifter also sends (00) to ISHIFTER so final result of the first select unit is 3X.

For second select unit, the input will be 0110. MUX select line is 001 and shifted signal value to ISHIFTER is 01. The MUX output is 3X which is left shifted once by ISHIFTER.

When 0011X and 0110 X reached to the adder, 0110X should be shifted 4 times to the left because it is the multiplication of 4 MSB bits.

For 16*16 multiplication, if L = 4, 4 select units required to compute the operation. Considering VLSI implementation of this structure, select unit logic is not area efficient.

PROPOSED CSHM

The proposed computation sharing multiplier architecture is based on reduced alphabet set and grouping the bits of coefficient to drives the select signals of MUX. Figure 2 shows proposed CSHM. It consists of pre-computer, select unit and an adder. Pre-computer produces multiplication of alphabets with input X. Two distinct alphabet set's are formed namely set 1 and 2. Set 1 consist of {1X, 2X and 3X} and set 2 consist of {4X, 8X and 12X}. Compared to existing CSHM pre-computer alphabet count, this pre-computer has less count and also it is constant. Hence it computes multiplication result along with positive area overheads. The main advantage of select unit is select signals for MUX are obtained without any lookup rules.

With this idea, 8*8 and 16*16 computation sharing multiplier has been implemented. Figure 2 shows modified 8*8 computation sharing multiplier architecture for unsigned integers. Pre-computer

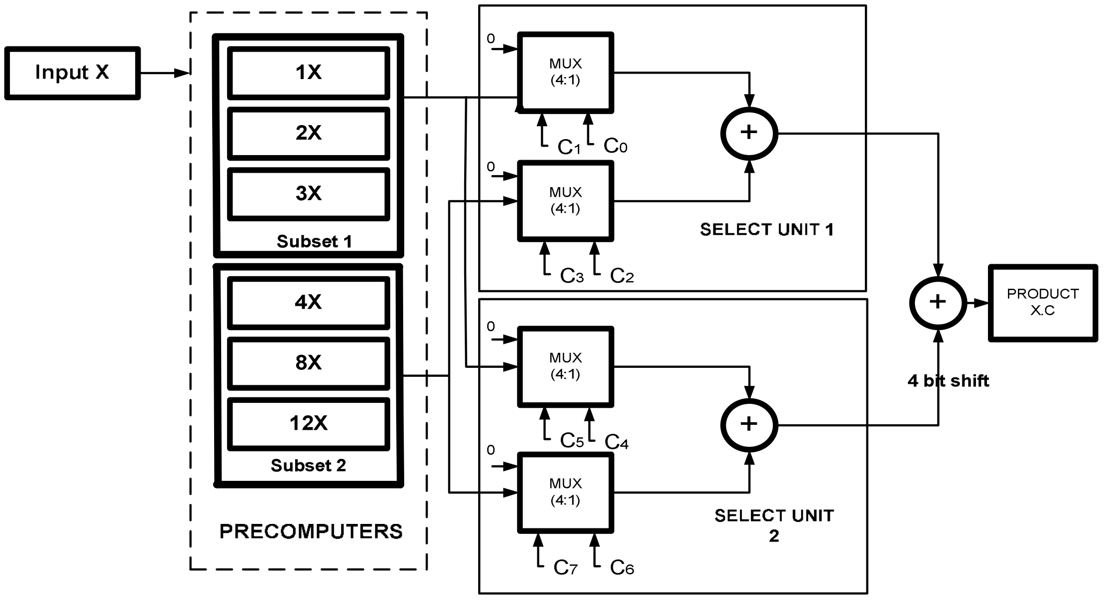

Fig. 2: Block diagram of proposed CSHM

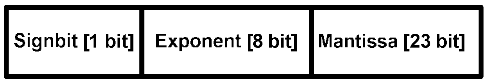

Fig. 3: IEEE 754 single precision format

alphabet sets are of only 6 significant values and easy and efficient to compute. It is implemented by shift and add logic only 2 addition operations are needed in pre-computer block. The coefficient of length W is divided into groups of 2 bits {b_0b_1, b_2b_3,} and the number of required MUX's is W/2. Each coefficient group will act as a select line to the 4:1 Mux's. All odd 4:1 MUX's has input signal as (0, Set 1 alphabets). All even 4:1 MUX's has input signal as (0, Set 2 alphabets). In each select unit's output's of MUX's are added as shown in Fig. 2. Independent select signals benefited in parallel operations.

If coefficient is C = $01100011_2 = 99_{10}$, starting from LSB of coefficient, 11 will be the select line for MUX 1, 00 for MUX 2 and so on. Hence values selected by MUX's are 3X, 0, 2X, 4X. The final result Z = X*C is obtained by the following expression:

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

$$Z = \{3X + 96X\} \quad (3)$$

$$Z = 99 * X \quad (4)$$

For the final multiplication result, the addition of outputs of MUX 3 and MUX 4 needs left shift of 4 bits because it is the multiplication of 4 most significant bits of coefficients. If 3X, 2X, 4X are available, the multiplication result will be of few shifts and add operation. The select unit of proposed CSHM is highly area and power efficient compared to existing design.

The same idea can be expanded for 16*16 multiplication. A 16*16 unsigned integer CSHM has been implemented with the same pre-computer alphabet sets. The critical path has addition operation which incurs the largest delay. But compared to select unit of existing CSHM, proposed design has less area overhead in LUT based designs.

Floating point CSHM: The practical value in digital application may be a fraction or an unsigned number. If CSHM indulge same gains for signed number then its application area can be wider. One of the formats used for signed number representation is IEEE-754 single precision format. To represent a number in floating point representation, a double word is divided into 3 fields {S, E, M}, representing as follows: 1 bit for the Sign (S), 8 bits for the Exponent (E) and 23 bits for the Mantissa (M). Figure 3 shows the format, since the exponent field is 8 bits, it can be used to represent exponents between -128 and 127. The significant field can store the first 23 bits of the binary representation of m, namely b_0, b1, b2, b3…… b23. But for the normalized number the mantissa is always 1.m ($b_0 = 1$). In single precision representation 1 is implicit. Hence only 23 bits (b_1, b_2, b_3…… b_{23}) are stored in single precision format.

Figure 4 shows algorithm and expression for floating point multiplier. To get the multiplication result, sign bits are XORed and exponent is obtained by simple addition. For biased exponent 127 has to be subtracted from the addition. Also for final exponent, overflow and underflow has to be checked. The mantissa bits are multiplied and rounding is done with normalization for the final 23 bits. Here 24×24 multiplier is required and hence proposed CSHM suits very well.

If $F_0 = \pm M_0 \times 2^{E0}$ and $F_1 = \pm M_1 \times 2^{E1}$, Then the multiplication of F_0 with F_1 is obtained by the following expression:

$$F_0 * F_1 = \pm M_0 * M_1 * 2^{E0+E1} \quad (5)$$

The implementation of IEEE 754 floating point multiplier using 24*24 CSHM is presented in this study. The 24×24 significant multiplication is the main block in which redundant computations take place frequently and is the main obstacle in achieving high-performance and low-power consumption in many DSP applications. The redundant computations can be reduced by identifying common computations and sharing them among different arithmetic units in the application. Figure 5 shows Floating Point multiplier using proposed CSHM. It consists of the following block.

Unpack: Sign bit, exponent bit and mantissa bit are extracted from the packed IEEE 754 floating point number.

XOR: Sign bits are XORed for the final result.

Exponent adder: Exponents are added and 127 is subtracted for the bias result.

CSHM multiplier: It is basically 24×24 CSHM. Two mantissa's are multiplied by computation sharing style and final result will be of 48 bits. For packing the result in IEEE format, the lower 24 bits is been truncated i.e., rounding number to 24 bits.

Rounding: Here round to nearest technique is used (Sivanantham *et al*., 2013). The value '1' is added to the LSB position of the bits to be retained if there is a '1' in the MSB position of the bits to be removed. Thus $0.b_{-1}b_{-2}b_{-3}1$ is rounded to $0.b_{-1}b_{-2}b_{-3}$ +0.001 and $0.b_{-1}b_{-2}b_{-3}0$ is rounded to $0.b_{-1}b_{-2}b_{-3}$. When the bits to be removed are 10…0, a tie occurs. In this case $0.b_{-1}b_{-2}0100$ is truncated to the value $0.b_{-1}b_{-2}0$ and the value $0.b_{-1}b_{-2}1100$ is truncated to $0.b_{-1}b_{-2}1$+0.001. This is unbiased rounding technique, because the error range is -½ to +½ in the LSB position of the retained bits.

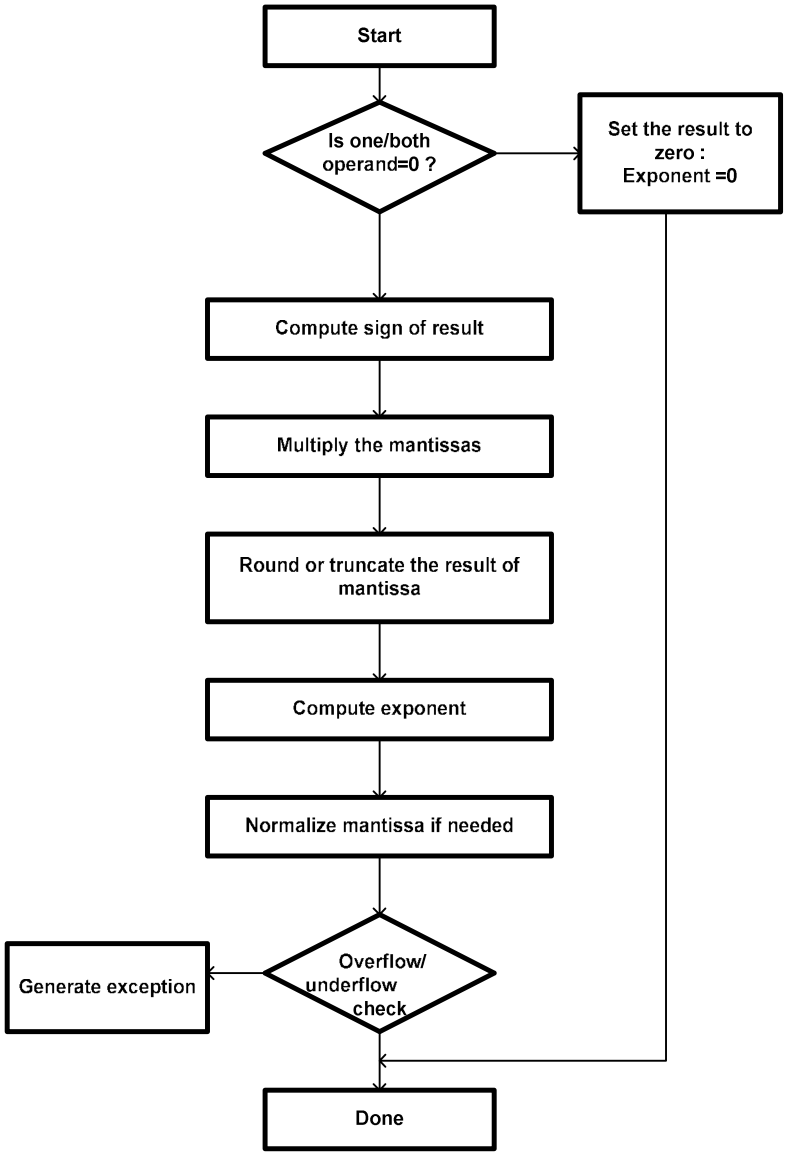
Start
Is one/both operand=0 ?
Set the result to zero : Exponent =0
Compute sign of result
Multiply the mantissas
Round or truncate the result of mantissa
Compute exponent
Normalize mantissa if needed
Overflow/ underflow check
Generate exception
Done

Fig. 4: Flowchart of floating point multiplication

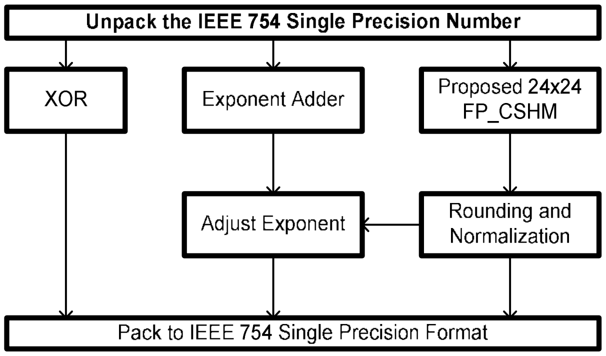
Unpack the IEEE 754 Single Precision Number
XOR
Exponent Adder
Proposed 24x24 FP_CSHM
Adjust Exponent
Rounding and Normalization
Pack to IEEE 754 Single Precision Format

Fig. 5: Proposed CSHM based floating point multiplier architecture

Normalization: After 24×24 multiplication the 48 bit result has to be normalized such that there will be at least one non zero digit left to the binary point.

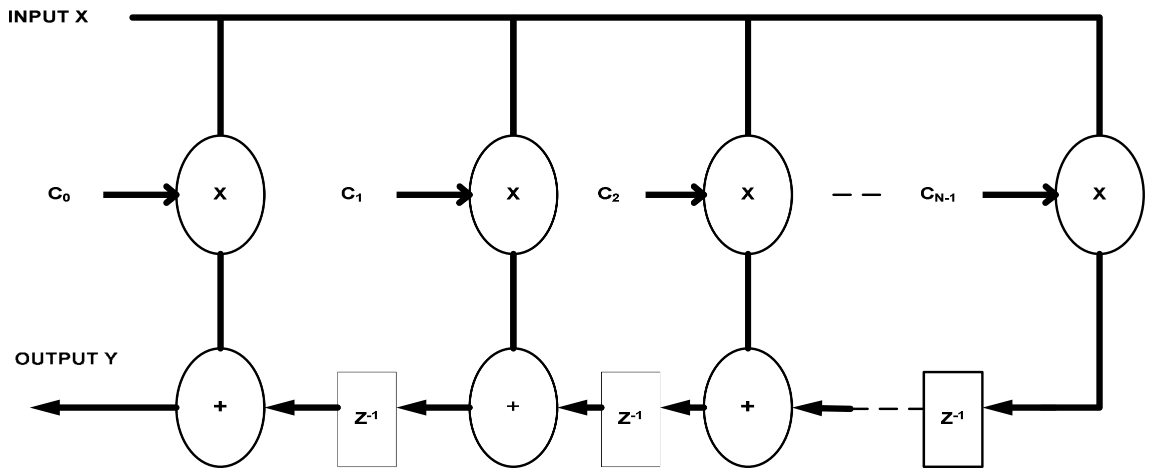

Fig. 6: Transposed form of FIR filter

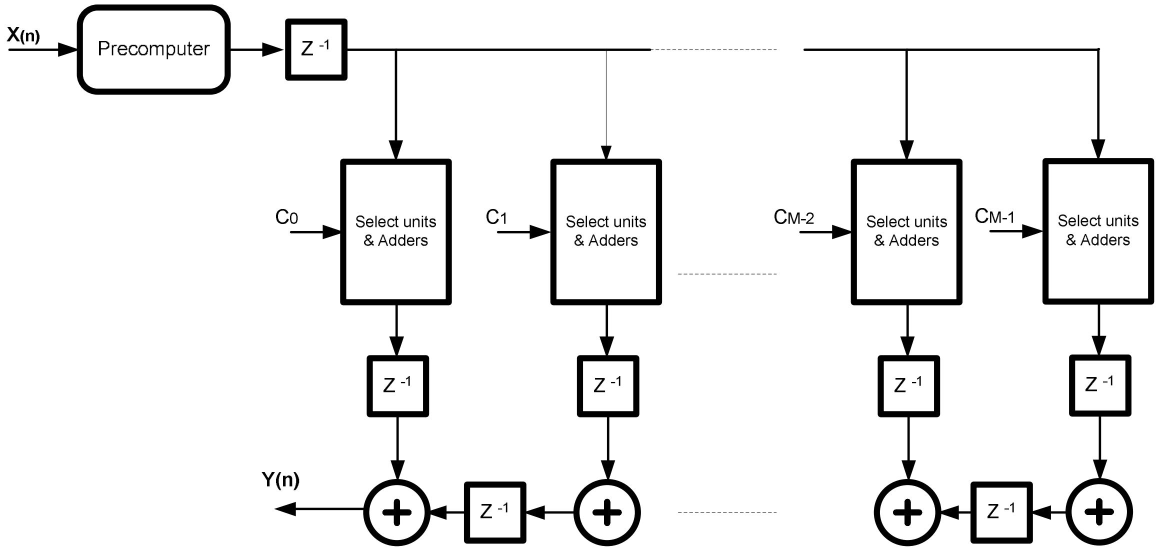

Fig. 7: FP_FIR filter using FP_CSHM

The final result is packed for standard IEEE 754 single precision format.

FIR filter design using proposed CSHM: The input-output relationship of Linear Time Invariant (LTI) FIR filter can be described as:

$$Y(n) = \sum_{i=0}^{N} C_i . X(n-i) \quad (6)$$

where, N represents the order of FIR filter, C_i is the filter coefficients and X (n-i) denotes the data sample at time instance n. Figure 6 shows a transposed Direct Form (DF) implementation of an FIR filter. The multipliers are replaced by proposed Computation Sharing Multipliers (CSHM). The computations α_k. X (where k = 0, 1, 2, ... 5) are performed only once for all k's and all filter taps and these values are shared by all the select and shift units for generating C_i.X (where i = 0, 1, 2, 3). The input X (n) is multiplied by all the coefficients C_0 C_1 C_2…. C_{M-1} simultaneously and are added for final FIR response. Expressing the filtering operation in terms of a vector scaling operation allows opportunity to share computations between multiplication operations.

As the computer world emerging with accuracy and precise response, the signal processing techniques must support such a complex system. Floating point system based FIR filter provide high reliability in critical and security based application. The implementation of the transposed direct form floating point FIR filter is presented in this section on which multiplier is replaced with proposed FP_CSHM.

Figure 7 shows the proposed structure of the FIR filter using floating point CSHM. The floating point adder is also implemented to add all products. Final result of filter is represented in IEEE single precision format.

RESULTS AND DISCUSSION

The implementation result of existing CSHM, proposed CSHM (both for unsigned integers and IEEE single precision format), FIR filter (4 taps) design using existing CSHM and proposed CSHM in the environment of both Xilinx ISE 14.3 and Cadence® RC compiler is discussed and relevant comparison statement is also presented in this section.

Table 1 shows the comparison result of existing CSHM and proposed CSHM in the Xilinx Environment. Proposed CSHM shows 57% reduction (8*8) and 41% reduction (16*16) in the Look up table requirement. In an existing CSHM fixed size look up rule is used and which chooses the number of alphabets in the alphabet set. With the L = 2, number of alphabets needed in the pre-computer unit is 2 and number of select unit required is 4 and 8 for 8*8 and 16*16 multiplication respectively. With the L = 4 number of alphabets needed in the pre-computer unit is 8 and number of select unit required is 2 and 4 for 8*8 and 16*16 multiplication respectively. So increase In L causes more multiplication requirement in the pre-computer unit which automatically increases the computational complexity. Though there is a reduction in a number of select unit requirement compared to previous disadvantage, advantage we gain due to reduction in a select unit is less. But in proposed CSHM, for 8*8 and 16*16 multiplication always requires 6 constant alphabet sets in the pre-computer unit and 2 and 4 select unit's requirement respectively. Though the number of bits to multiply gets increases also always pre-computer entries remains same and even better result can be achieved by using proposed CSHM technique. For example, With L = 5, Existing CSHM requires 16 entries in the pre-computer unit and 5 select unit's for 20*20 multiplication. But proposed CSHM requires only 6 entries in the pre-computer unit and 5 select units for 20*20 multiplication. The timing result shows no big difference in the both the architecture. Since all the select units are working in parallel manner in the both the architecture, the expected final result also almost at the same time. With the small architecture modification, proposed CSHM is modified to work for signed number also.

Table 2 refers the comparison result of existing and modified CSHM in the cadence® RC compiler environment. Due to the reduction in the number of entries in the pre-computer unit of proposed CSHM, there is a reduction in the hardware requirement. For example, if we take 8*8 CSHM, there is a 46% area and power reduction is achieved compared to existing CSHM style. As mentioned above if number of bits to multiply get increases too, proposed design reduces the hardware requirement still in better way.

Table 1: Xilinx analysis for CSHM multiplier

8*8 CSHM 16*16 CSHM FP_CSHM	Existing CSHM design style	Proposed CSHM design style
Slices	118	48
	370	208
	860	629
No. of LUTs	222	94
	702	410
	1617	1211
Timing (nsec)	20.20	14.93
	20.16	17.94
	27.65	24.67

Table 2: CSHM results in CADENCE® RC compiler

8*8 CSHM 16*16 CSHM FP_CSHM	Existing CSHM design style	Proposed CSHM design style
No. of cell	439	207
	890	652
	2094	1331
Power (mW)	0.533	0.288
	2.440	1.650
	5.800	4.900
Area (um^2)	8528.890	4580.450
	24828.250	17716
	53538.400	43047.900
Timing (nsec)	6.397	6.050
	11.714	12
	21.310	20.800

Table 3: FIR filter implementation results (implemented in CADENCE® RC compiler)

8*8 FIR 16*16 FIR FP_FIR	Existing CSHM based style	Proposed CSHM based style
No. of cell	1288	904
	3211	2841
	10265	8143
Power (mW)	2.098	1.299
	9.700	7.250
	29.880	29.500
Area (um^2)	30093.900	24854.027
	84231.100	68184.500
	229035.940	212636.790
Timing (nsec)	9	8.151
	14.381	14.503
	36	35.270

The FIR filter implementation is done using existing and proposed CSHM in the both the environment. Multiplier is an essential element in FIR filtering operation. So the FIR filter has been chosen as an application to implement a proposed CSHM.

Figure 8 shows comparison result of Slices, LUT requirement to implement a 4 tap FIR filter design using existing and proposed CSHM. From Fig. 8 it is clearly visible that proposed technique requires less number of slices and LUT than the existing technique due to its constant alphabet sets.

Table 3 shows comparison result obtained for a 4 tap filter design with 8*8, 16*16 and IEEE single precision format multiplier unit. Even for 4 tap itself

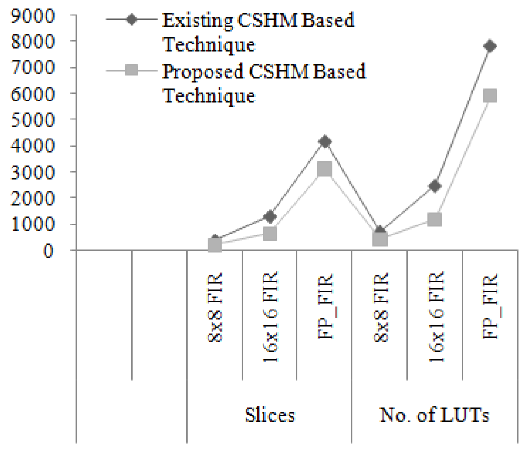

Fig. 8: FIR filter implementation results (Xilinx ISE 14.1)

proposed CSHM based FIR Filter design gave better result in terms of area as well as power (7% reduction for an FP_CSHM based FIR filter design) in the both the environment. The FIR Filter is implemented in RC compiler tool of cadence® design systems shows area and power requirement of ASIC implementation.

Real time application needs FIR filter with more taps in order to get the smooth amplitude and phase plot. So if the number of taps increases, it requires more multipliers unit then designing the FIR Filter with proposed CSHM will give significant reduction in terms of area without compromising the delay.

CONCLUSION

In this research study, successful optimization of existing computation sharing high speed multiplier is obtained by constant pre-computer alphabet set and the same work is extended for floating point multiplication also. The FIR filter implementation using proposed CSHM impacts positively on the area and power. The timing results shows in both the architecture requires very less delay to get the final output. The proposed floating point CSHM based FIR filter implementation also holds better results in terms of area, power compared to existing floating point CSHM based FIR filter implementation. So from all of the above results it is clear that the proposed CSHM suits well for filtering as well as processor applications.

REFERENCES

Alan, N., W. Wasserman and Y. Zhan, 2000. A painless way to reduce power dissipation by over 18% in booth encoded carry save array multipliers for DSP. Proceeding of IEEE Workshop Signal Processing System, pp: 571-580.

Jongsun, P., W. Jeong, H. Mahmoodi-Meimand, W. Yongtao, C. Hunsoo and K. Roy, 2004. Computation sharing programmable FIR filter for low-power and high-performance applications. IEEE J. Solid-St. Circ., 39(2): 348-357.

Kernhof, J., M.A. Beunder, B. Hoefflinger and W. Haas, 1989. High-speed CMOS adder and multiplier modules for digital signal processing in a semicustom environment. IEEE J. Solid-St. Circ., 24(3): 570-575.

Kuo, K.C. and C.W. Chou, 2010. Low power and high speed multiplier design with row bypassing and parallel architecture. Microelectr. J., 41: 639-650.

Mamatha, B. and V.V.S.V.S. Ramchandram, 2012. Design and implementation of 120 order FIR filter based on FPGA. Int. J. Eng. Sci. Emerg. Technol., 3(1): 90-97.

Mark, R.S., 1989. Design and clocking of VISI multipliers. Technical Report No, CSL-TR-89-397.

Sivanantham, S., K. Jagannadha Naidu, S. Balamurugan and D. Bhuvana Phaneendra, 2013. Low power floating point computation sharing multiplier for signal processing applications. Int. J. Eng. Technol., 5(2): 979-985.

Sravanthi, K., V.V.K.D.V. Prasad Battula and V.V. Rao, 2012. Design of FIR filter by using sharing multiplier with low delay. Int. Refereed J. Eng. Sci. (IRJES), 1(1): 031-038.

Wu, A.Y., K.J.R. Liu and A. Raghupathy, 1998. System architecture of an adaptive reconfigurable DSP computing engine. IEEE T. Circ. Syst. Vid., 8(1): 54-73.