Low Power Floating Point Computation Sharing Multiplier for Signal Processing Applications

Sivanantham S[#], Jagannadha Naidu K[#], Balamurugan S[*], Bhuvana Phaneendra D[#]
[#]ASIC Design Laboratory, School of Electronics Engineering
VIT University, Vellore - 632014, Tamilnadu, India
{ssivanantham, jagannadhanaidu.k}@vit.ac.in
[*]School of Electrical Engineering
VIT University, Vellore - 632014, Tamilnadu, India
sbalamurugan@vit.ac.in

Abstract - **Design of low power, higher performance digital signal processing elements are the major requirements in ultra deep sub-micron technology. This paper presents an IEEE-754 standard compatible single precision Floating-point Computation SHaring Multiplier (FCSHM) scheme suitable for low-power and high-speed signal processing applications. The floating-point multiplier used at the filter taps effectively uses the computation re-use concept. Experimental results on a 10-tap programmable FIR filter show that the proposed multiplier scheme can provide a power reduction of 39.7% and significant improvements in the performance compared to conventional floating-point carry save array multiplier implementations.**

Keywords — low-power design, IEEE-754 standard, Floating-point multiplier, Digital FIR filter, VLSI implementation.

I. INTRODUCTION

The finite impulse response (FIR) filters are used in signal processing applications ranging from video and image processing to wireless communications. In some applications, such as video processing, the FIR filter circuit must be able to operate at high-frequencies, while in other applications, such as cellular telephony, the FIR filter circuit must be a low-power circuit, capable of operating at moderate frequencies. These demands led the designers to focus on the algorithmic as well as numerical strength reduction techniques for the low-complexity design of FIR filter. Strength reduction at the algorithmic level can be used to reduce the number of computations (additions and multiplications). Numerical strength reduction improves the performance of a computation. In this paper, numerical strength reduction is the area of interest.

Many previous efforts like Common sub-expressions elimination [4], [5] and differential coefficients method [6], [7] explore low-complexity design of FIR filters by minimizing the number of additions in filtering operations. Canonical signed digit (CSD) [8] is used to reduce the number of the required additions and subtractions for filtering operation by reducing the total number of nonzero bits in coefficients. In [6] the differences between absolute values of filter coefficients were employed to reduce the complexity of computation. All these techniques are limited to the optimization of hardware for a particular fixed coefficient set. A computation sharing multiplier (CSHM) architecture, which identifies common computations and shares them between different multiplications was suggested in [1], [2] overcomes this drawback and applicable for applications with programmable filter coefficients. CSHM achieves high- performance programmable filtering operation by reusing the optimal precomputations and low-power consumption, since, redundant computations are removed. In addition to signal processing applications, the multipliers can also be used to test data compression/decompression VLSI test applications [16]. The reconfigurable multiplier design based on reordering of partial product [14] and row-bypassing technique [15] are proposed to reduce the switching power. In the literature [2], [10] and [11] CSHM is used only in fixed-point FIR filter implementation.

The main contribution of this paper is proposition of a floating-point multiplier based on the CSHM technique, and effective implementation of floating-point FIR filter. Henceforth, in this paper this multiplier will be referred as FCSHM. The floating-point input values are taken in single-precision IEEE-754 standard format. In the rounding stage of multiplier and adder “round to nearest number” technique is used. The significance of our proposition should be understood in the context of wide usage of FIR filter in real time applications such as satellite communications and signal equalizers.

In the remaining of this paper organized as follows. Section II presents the architecture and operation of 8x8 computation sharing multiplier. In section III, we present the proposed floating-point computation sharing multiplier architecture (FCSHM) and the architecture of floating-point carry save array multiplier (FSHM) scheme is described in section IV for comparison purpose. In section V, the FCSHM based FIR filter

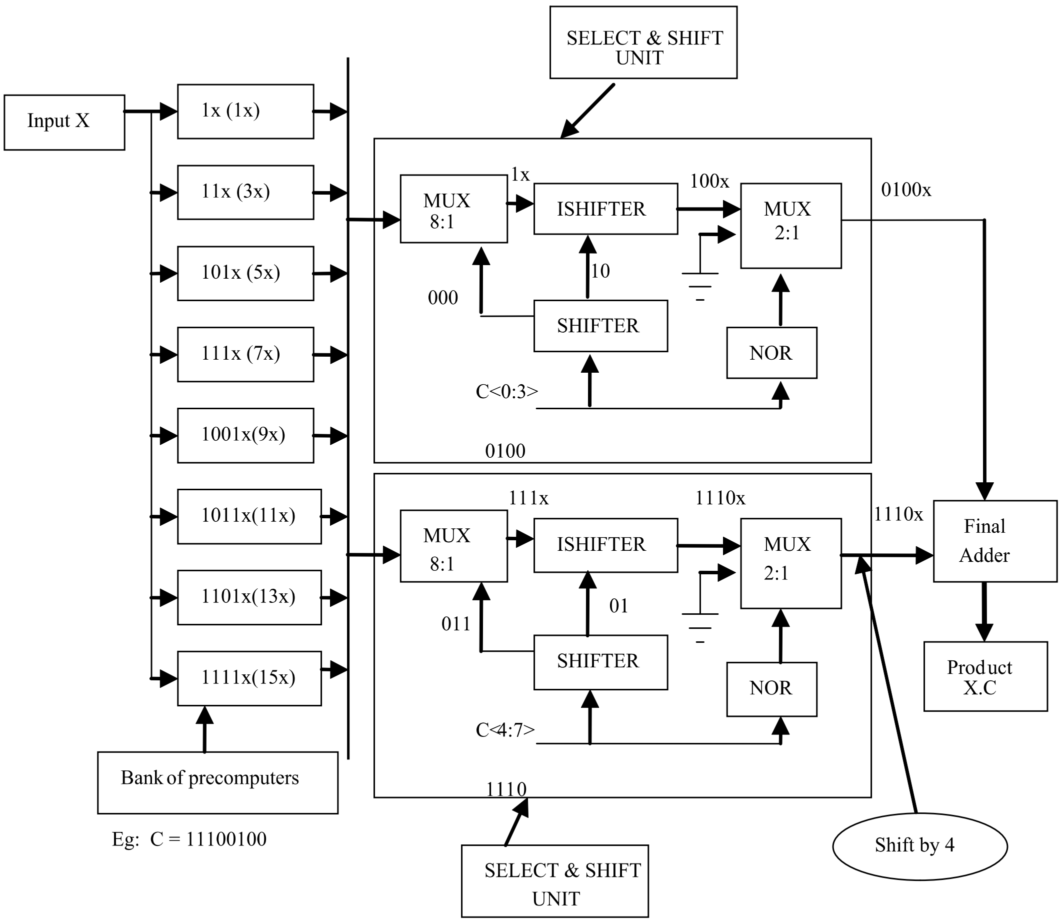


Fig. 1. Architecture of 8 x 8 Parallel Computation Sharing Multiplier

implementation are explained. The implementation results are discussed in section VI. Finally, section VII concludes this paper.

II. COMPUTATION SHARING MULTIPLICATION

The FIR filtering can be expressed as multiplications of vectors by scalars *C.x*. In vector scaling operations, we can carefully select a set of small bit sequences so that the same multiplication result can be obtained by only add and shift operations. For instance, *(1011).x* can be decomposed as *(0011).x* $+2^3$*.(0001).x* . If both *(0011).x* and *x* are available, the entire multiplication process is reduced to a few add and shift operations. These chosen basic bit sequences as *alphabet*s [9]. Also, an *alphabet set* is a set of *alphabets* that spans all the coefficients in vector *C*.

It is easy to figure out that, as the number of coefficients in increases, there can be many choices for *alphabet sets* on the coefficients and each *alphabet set* gives rise to a different combination of add and shift operations to obtain *C.x*. Obviously, an alphabet set should cover all the coefficients in coefficient vector. In addition, there are two other desirable characteristics of ‘good’ alphabet set. First, total number of *add* operations should be minimized. Multiplication operation can be simplified to add and shift operations with the computation sharing multiplier algorithm. As will be shown later in the multiplier implementation, the add operations lie on the critical path and incurs the largest delay. Second, the number of alphabets in alphabet set should be minimized.

In the computation sharing multiplier scheme, the multiplied value of should be available before the decomposition. They are computed at the first stage. As the number of alphabets increase, the amount of the computations also increases, which results in large area and power consumption. It also increases delay due to large length of individual alphabet (*L*). The alphabet sets obtained for *L=2* and *L=4* are *{1,3}* and *{1,3,5,7,9,11,13,15}*, respectively [1], where each alphabet is represented with *4*-bits. If the values are precomputed using the input and the above set the final value of the multiplication can be easily obtained by the combination of simple add and shift operations.

Fig. 1 shows a parallel *8*×*8* CSHM structure [2]. The bank of precomputer performs the computations and the outputs of the precomputer bank are *1x, 3x, 5x, 7x, 9x, 11x, 13x* and *15x*. To find the correct *alphabet*,

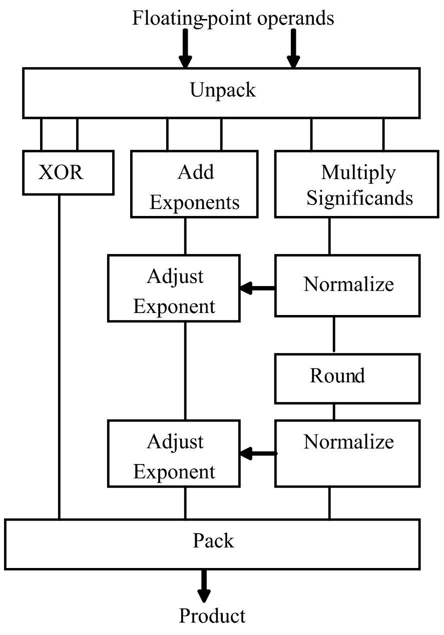


Fig. 2. Block diagram of Floating-point Multiplier.

SHIFTERs perform the right shift operation until it encounters *'1'* and send an appropriate *select* signal to 8-to-1 (8:1) MUXes. They also send the exact shifted values (*shift* signal) to ISHIFTER's. The 8-to-1 (8:1) MUXes select the correct answer among the eight values received from precomputer. ISHIFTERs simply inverse the operation performed by SHIFTERs. When the coefficient input is *0000*, we cannot obtain a zero output with shifted value of the precomputer outputs. Simple NOR gates and MUX (2:1) are used to deal with the zero (0000) coefficient input. We refer to SHIFTER-MUX (8:1) – ISHIFTER - NOR gate-MUX (2:1) as the *select unit*. The upper select unit generates the multiplication of *4* LSBs of the coefficient with the input. The lower select unit produces the product of upper *4*- bits with input. A shift of the upper 4 bits is performed when those two values are fed to the adder. A simple adder produces the final result.

Let us consider an example shown in Fig. 1. If the coefficient is *11100100*, it is divided into two parts consisting of *4* bits. *0100* is fed to SHIFTER of the upper select unit and *1110* to that of the lower select unit. In the upper select unit, SHIFTER shifts *0100* to the right twice until it encounters *1* and it sends *000* (*select* signal) to MUX (8:1), which chooses *1* among the precomputer outputs. SHIFTER also sends *10* (*shift* signal) to ISHIFTER. ISHIFTER shifts to the left input from the MUX(8:1) *1* twice. Finally, output of ISHIFTER is *0100.* In the lower select unit, SHIFTER shifts *1110* to the right once and sends *011* (select signal) to MUX (8:1), which chooses *111* among the outputs of precomputer. Like the one in the upper select unit, SHIFTER sends *01* (shift signal) to ISHIFTER, which shifts *111* to the left once. Because none of the inputs of select units are *0000*, the MUX (2:1) of both select units just pass their inputs. The outputs of the upper select unit and the lower select unit are *0100* and *1110*, respectively. When these values reach the adder, *1110* should be shifted four times to the left because it is the multiplication of the four MSBs. The precomputer, MUX (8:1), ISHIFTER and ADDER lie on the critical path in this multiplier structure. When the input of the SHIFTER is *0000*, select and shift signal go to don't care state and the NOR gates (Fig. 1) simply generate the zero outputs.

III. Proposed Floating-Point Computation Sharing Multiplication (FCSHM) Architecture

In the IEEE-754 single precision [13] (*32*-bit) floating-point representation the LSB *23*-bits give fraction, the next *8*-bits represent the biased exponent (bias value is *127*) and the MSB is the sign-bit. The general floating-point multiplication algorithm can be represented as

$$(\pm S_1 \times b^{E1}) \times (\pm S_2 \times b^{E2}) \quad = \quad \pm (S_1 \times S_2) \times b^{E1+E2} \tag{1}$$

where S_1, S_2 are significands (including the hidden '*1*') of the input values and *E1, E2* are the biased exponents of the input values.

The Fig. 2 shows the block diagram of floating point multiplier. Initially the input values are unpacked into sign-bit, exponents and significands and applied to the sub-modules. The sign of the multiplication result is obtained by EX-OR operation on the input sign-bits. The exponents of the inputs are added and bias value is subtracted from the result (in order to make the final exponent result in biased format). This resultant exponent is adjusted in the subsequent normalization steps and checked for overflow or underflow before final exponent is obtained. The *24*×*24* significand 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 the filter design. The redundant computations can be reduced by identifying common computations and sharing them among filter taps. Since each alphabet is represented with *4*-bits, for the *24*×*24* significand multiplication we need six select & shift units. We have implemented an IEEE-754 compliant FCSHM based on the proposed architecture. Also, we have implemented *24*×*24* CSAM for comparison purpose.

1) Result Sign: The result sign is obtained by XOR operation of the sign bits of the input operands.

2) Exponent Addition: We have used Carry-Look ahead adders for the exponent addition and for the subtraction of bias (valued 127) from the added result.

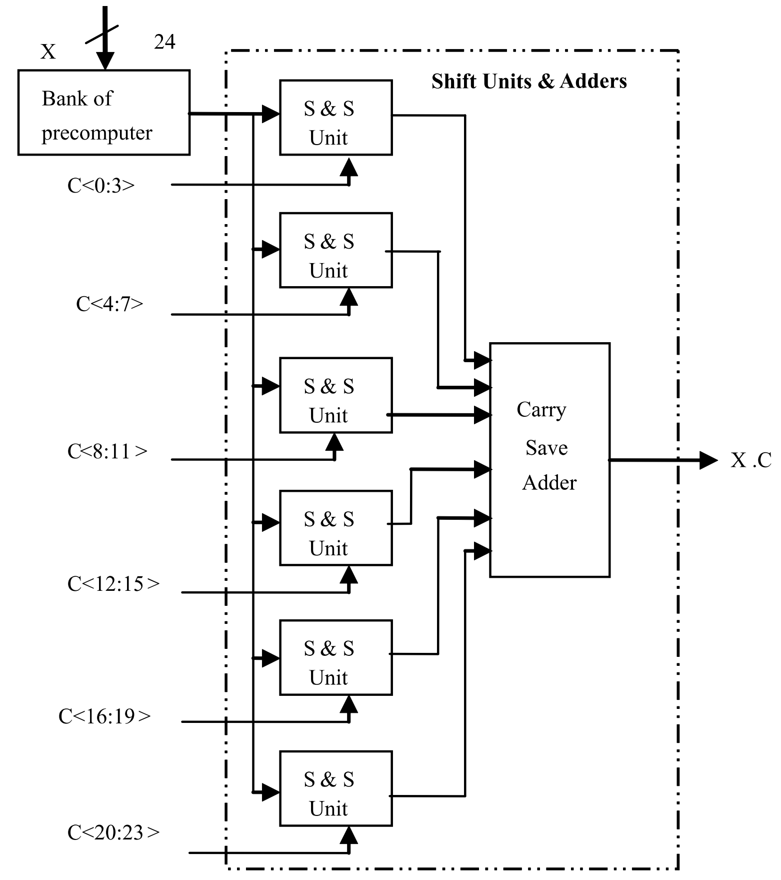


Figure 3. 24x24 Significand Multiplication using CSHM

IV. CSHM Implementation

The architecture for *8* × *8* multiplier using parallel *24*×*24* CSHM scheme is shown in Fig. 3. In this structure, the S & A of *8*×*8* multiplier, which is shown in Fig. 2, are connected in parallel to the precomputer and a carry save adder is used to generate the final output. As the number of coefficient increases, only the number of inputs to the final carry save adder increases and this does not incur large delay. The CSHM scheme contains the following blocks:

a) Precomputation Block: The multiplications *1x, 3x, 5x, 7x, 9x, 11x, 13x* and *15x* are performed by the precomputer and this is implemented using adders.

b) Select unit: It consists of *SHIFTER, MUX (8:1), ISHIFTER, NOR gates* and *MUX (2:1).*

c) Carry save adder: As shown in Fig. 3, carry save adder is used to obtain final output in the parallel *24*×*24* CSHM. The carry save adder is the largest component in the S & A, which sums the outputs of four select units.

d) Normalization block: After *24x24* 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.

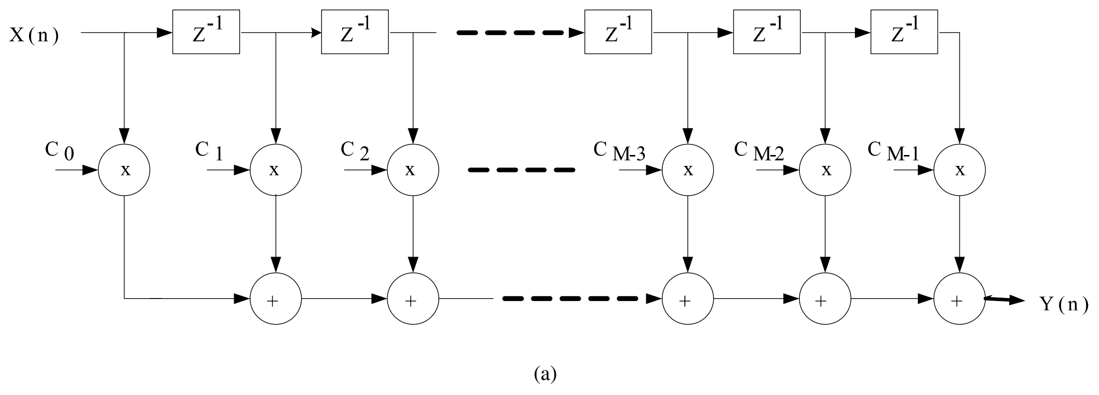
e) Rounding block: Among the rounding techniques we use "Round to nearest" technique. A '*1*' is added to the *LSB* position of the bits to be retained if there is a '*1*' in the *MSB* position of the bits to be removed. Thus, $0.b_{-1}b_{-2}b_{-3}1\ldots$ is rounded to $0.b_{-1}b_{-2}b_{-3}+0.001$, and $0.b_{-1}b_{-2}b_{-3}0\ldots$ 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 an unbiased rounding technique, because the error range is approximately -1/2 to +1/2 in the LSB position of the retained bits.

V. FIR Filter Implementation using FCSHM

The input-output relationship of linear time invariant (LTI) FIR filter can be described as

$$y(n) = \sum_{k=0}^{M-1} c_k \cdot x(n-k) \tag{2}$$

where n represents the length of FIR filter, c_k 's are the filter coefficients, and $x(n-k)$ denotes the data sample at time instance $(n-k)$.Fig. 4(a) shows a direct form (DF) implementation of an FIR filter. An equivalent architecture is the *transposed direct form* (TDF) as shown in Fig. 4(b). The TDF implements a product of the coefficient vector $C=[c_0,c_1,\ldots,c_{M-1}]$ with the scalar $x(n)$ at time n. The input $x(n)$ is multiplied by all the coefficients $c_0,c_1,\ldots,c_{M-1}$ simultaneously. In the sequel, such product will be referred to as a *vector scaling* operation. Expressing the filtering operation in terms of a vector scaling operation allows opportunity to share computations between multiplication operations.



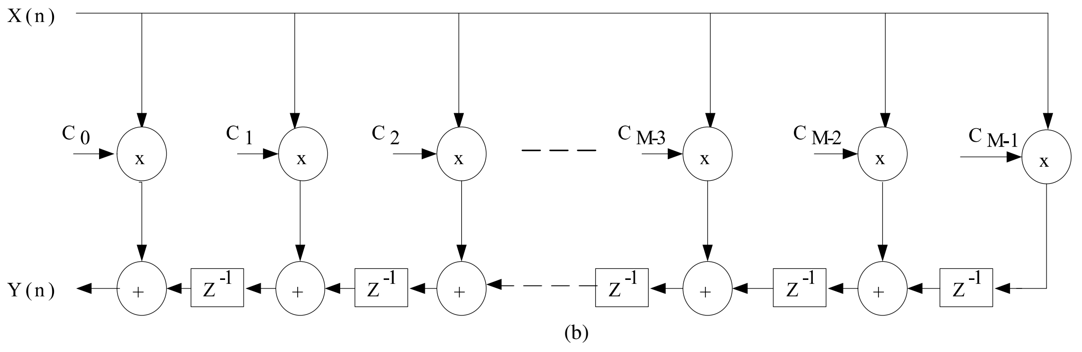


Figure 4. Implementation of FIR filters architecture in (a) Direct Form (b) Transposed Direct Form.

In the direct form FIR filter, a large adder in the final stage lies on the critical path and it slows down the FIR filter. Since we focus on the design of a high performance FIR filter, the transposed direct form FIR filter is more appropriate for a high-performance filter structure. In the TDF of FIR filter shown in Fig. 4(b), multipliers are replaced by floating-point computation sharing multipliers (FCSHM). The precomputation block is shared by all floating-point multipliers. The input x(n) is applied precomputation block and the precomputed values shared by all the Select & Shift units of the significand multipliers.

Fig.5 shows the proposed structure of the FIR filter using FCSHM. The computations $\alpha_k \cdot x, k = 0, 1, 2, \ldots, 7$, are performed only once for all k's, and all filter taps and these values are shared by

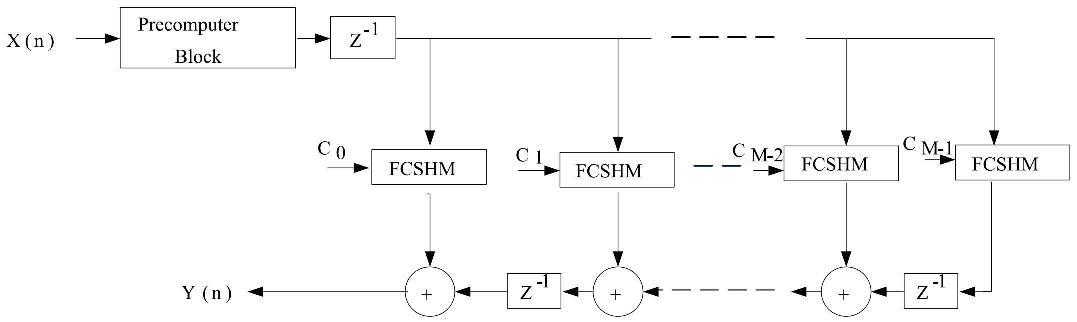


Figure 5. FIR filter (TDR) implementation based on FCSHM

all the select & shift units for generating $c_i \cdot x, i = 0, 1, 2, 3, \ldots$. This gives a reduced computational redundancy in FIR filter.

Since the precomputer lies on the critical path of the FCSHM one pipeline stage is introduced after the precomputation block. Because of this the latency of the proposed FIR filter increases by one clock cycle. If conventional multipliers (Wallace multiplier, Booth-encoded multiplier etc.) are used for filter implementation, flip-flops for pipelining should be placed in every tap of the filter. However, pipelining of the filter using FCSHM can be simply done by placing flip-flops right after the precomputation block, irrespective of the filter size, due to computation sharing and reuse. Therefore, the cost of pipelining (the number of flip-flops) is much smaller than using conventional multipliers. IEEE-754 complaint Floating-point adder is designed as explained in the previous section for the adder elements in the filter.

VI. Results and Discussions

We have described the proposed floating-point computation sharing multiplier (FCSHM) architecture and floating-point carry-save array multiplier (FCSAM) architecture using Verilog Hardware Description Language. To demonstrate the application of the proposed floating-point computational sharing multiplier (FCSHM), we have implemented the FCSHM architecture into a 10-tap programmable FIR filter with transposed direct form. Also, to show the effectiveness of the proposed FCSHM scheme, a 10-tap FIR filter with programmable coefficients is designed based floating-point multiplier (with carry-save array multiplier for the 24x24 significand multiplication (FCSAM)) and implemented. Both the multiplier scheme and FIR filter structures using both FCSHM and FCSAM are modeled using Verilog HDL and simulated using Cadence IUS. After functional validation, the structures are synthesized using Cadence RTL Compiler, targeted to TSMC 0.18µm, 1.8v CMOS standard cell library. Placement and Routing is done using Cadence SOC Encounter. The results of the filter implemented based on FCSHM and FCASM are tabulated in Table I.

TABLE I. Comparison of Results with FCSHM scheme in TSMC 0.18µm CMOS technology.

Component		**FIR-FCSHM (proposed work)**	**FIR-FCSAM**	**Performance Improvement of FCSHM over FCSAM**
Delay	Clock (ns)	44	47	**6.38**
Power	Switching Power (mW)	41.77	69.46	39.86
	Net Power (mW)	26.92	37.36	27.94
	Internal Power (mW)	14.85	32.09	53.72
	Leakage Power (nW)	21.12	22.69	6.92
	Total Power (mW)	83.75	138.93	**39.72**
Power-Delay Product (PDP)		3685.05	6529.84	**43.56**
Area	No. of Cells	46521	36703	-26.75
	Cell Area (μm^2)	475442	464179	**-2.43**

FIR filter based FCSHM structure achieves 39.7% savings in Power and an improvement of 6.38% in terms of speed and 43.56% savings in terms of PDP (Power Delay product) compared to the FIR filter based on FCSAM with 2.43% area overhead. This is achieved by computation sharing which reduces the computational redundancy in the filtering operation.

VII. Conclusion

We have implemented Floating-point FIR filter based on Floating-point Computation sharing Multiplier (FCSHM) and Carry-save Array Multiplier. In FCSHM scheme, the precomputed values are shared among the multipliers in the filter. Since redundant computations are removed in the 24×24 significand multiplication, the FCSHM technique results in Low-power and high-performance compared to CSAM. The proposed FIR filter based on FCSHM architecture can be used in the design of adaptive filter and signal equalizers.

References

[1] Muhammad K & Roy K, "Reduced computational redundancy implementation of DSP algorithms using computation sharing vector scaling," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol.10, no.3, pp.292-300, June 2002

[2] Jongsun Park, Muhammad K & Roy K, "High-performance FIR filter design based on sharing multiplication," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol.11, no.2, pp.244-253, April 2003

[3] J.L Hennessy & D.A Patterson, "Computer Architecture - A Quantitative Approach," Morgan Kaufmann, CA, 1996.

[4] Carl Hamacher, Zvonko Vranesic & Safwat Zaky, "Computer Organization," Mc Graw Hill, Intl. Edition, 2002.

[5] R. I. Hartley, "Sub expression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II*, vol. 43, pp. 677–688, Oct.1996.

[6] M. Potkonjak, M. Srivastava & A. P. Chandrakasan, "Multiple constant multiplications: Efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 151–165, Feb. 1996.

[7] N. Sankarayya, K. Roy & D. Bhattacharya, "Algorithms for lowpower high speed FIR filter realization using differential coefficients," *IEEE Trans. Circuits Syst. II*, vol. 44, pp. 488–497, June 1997.

[8] K. Muhammad & K. Roy, "A graph theoretic approach for synthesizing very low-complexity high-speed digital filters," *IEEE Trans. Computer-Aided Design*, vol. 21, pp. 204–216, Feb. 2002.

[9] H. Samueli, "An improved search algorithm for the design of multiplierless FIR filter with powers-of-two coefficients," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1044–1047, July 1989.

[10] Jongsun Park, Woopyo Jeong, Mahmoodi-Meimand H, Yongtao Wang, Choo H & Roy K, "Computation sharing programmable FIR filter for low-power and high-performance applications," *IEEE Journal of Solid-State Circuits,* vol.39, no.2, pp. 348- 357, Feb. 2004.

[11] Hunsoo Choo, K Muhammad & K Roy, "Two's Complement Computation Sharing Multiplier and Its Applications to High Performance DFE," *IEEE Trans. on Signal processing,* Vol.51, No.2, Feb'03.

[12] Behrooz Parhami, "Computer Arithmetic- Algorithms and Hardware Designs," Oxford Univ. Press, 2001.

[13] "IEEE Standard for Binary Floating - Point Arithrnetic", ANSI/IEEE Std 754-1985, New York, The Institute of Electrical and Electronics Engineers Inc., August 12, 1985.

[14] Praveen Kumar M V, Sivanantham S, Balamurugan S & Mallick P.S, "Low power reconfigurable multiplier with reordering of partial products," *Proc International conference on Signal Processing, Communication, Computing and Networking Technologies (ICSCCN)*, 2011, vol., no., pp.532-536, 21-22 July 2011.

[15] Balamurugan, S., Sneha Ghosh, S. Balakumaran, R. Marimuthu, and P. S. Mallick. "Design of low power fixed-width multiplier with row bypassing." *IEICE Electronics Express* 9, no. 20 (2012): 1568-1575.

[16] Sivanantham,S., Padmavathy, M., Divyanga, S. & Anitha Lincy, P. V. 2013 "System-On-a-Chip Test Data Compression and Decompression with Reconfigurable Serial Multiplier", *International Journal of Engineering and Technology*, vol. 5, no. 2, pp. 973-978.