EFFICIENT FIXED-POINT DLMS ADAPTIVE FILTER FOR LOW ADAPTATION-DELAY AREA-DELAY POWER.

P.RAJESH[1], B.MALLIKARJUNA NAIK[2], M.MOUNIKA[3], H.SREENIVASULU[4]

1. CRIT COLLEGE OF ENGG, ANANTAPUR, 9985786099, rajesh.crit@gmail.com
2. CRIT COLLEGE OF ENGG, ANANTAPUR, 9700713699, malli.nitr@gmail.com
3. CRIT COLLEGE OF ENGG, ANANTAPUR, 8985977110, midathala.mounika@gmail.com
4. CRIT COLLEGE OF ENGG, ANANTAPUR, 9885742603, sree_skv@yahoo.co.in

ABSTRACT:

Multiplication of a variable by a set of constants, generally known as Multiple Constant Multiplications (MCM), is essential in many Digital Signal Processing (DSP) applications such as, digital Finite Impulse Response (FIR) filters, Fast Fourier Transforms (FFT), and Discrete Cosine Transforms (DCT). An important objective is the reduction of the adder depth (AD), which is defined as the number of adder stages needed to compute a coefficient. In this paper, an efficient architecture for the implementation of a [DLMS] DELAYED LEAST MEAN SQUARE Adaptive Filter. For achieving lower adaptation-delay and area-delay-power efficient implementation, work modifies a novel partial product generator and proposes a strategy for optimized balanced pipelining across the time-consuming combinational blocks of the structure. From synthesis results, it is identified that the proposed design offers nearly 17% less area-delay product (ADP) and nearly 14% less energy-delay product (EDP) than the best of the existing systolic structures, on average, for filter lengths N=8, 16, and 32. This work proposes an efficient fixed-point implementation scheme of the proposed architecture, and derives the expression for steady-state error. This Work show that the steady-state mean squared error obtained from the analytical result matches with the simulation result. Moreover, we have proposed a bit-level pruning of the proposed architecture, which provides nearly 20% saving in ADP and 9% saving in EDP over the proposed structure before pruning without noticeable degradation of steady-state-error performance.

***Keywords*: Multiple constant multiplication, DLMS architecture, Balanced Pipelining, Partial product generation, reduction Adder depth**

I.INTRODUCTION

THE LEAST MEAN SQUARE (LMS) adaptive filter is the most popular and most widely used adaptive filter. The direct-form LMS adaptive filter involves a long critical path due to an inner-product computation to obtain the filter output. The critical path is required to be reduced by pipelined implementation when it exceeds the desired sample period. Since the conventional LMS algorithm does not support pipelined implementation because of its recursive behavior, it is modified to a form called the delayed LMS (DLMS) algorithm [3]–[5], which allows pipelined implementation of the filter. A lot of work has been done to implement the DLMS algorithm in systolic architectures to increase the maximum usable

frequency [3], [6], [7] but, they involve an adaptation delay of ~ N cycles for filter length N, which is quite high for large order filters.

Since the convergence performance degrades considerably for a large adaptation delay, Visvanathan *et al*. [8] have proposed a modified systolic architecture to reduce the adaptation delay. A transpose-form LMS adaptive filter is suggested in [9], where the filter output at any instant depends on the delayed versions of weights and the number of delays in weights varies from 1 to N. Van and Feng [10] have proposed a systolic architecture, where they have used relatively large processing elements (PEs) for achieving a lower adaptation delay with the critical path of one MAC operation. Ting *et al*. [11] have proposed a fine-grained pipelined design to limit the critical path to the maximum of one addition time, which supports high sampling frequency, but involves a lot of area overhead for pipelining and higher power consumption than in [10], due to its large number of pipeline latches. Further effort has been made by Meher and Maheshwari [12] to reduce the number of adaptation delays. Meher and Park have proposed a 2-bit multiplication cell, and used that with an efficient adder tree for pipelined inner-product computation to minimize the critical path and silicon area without increasing the number of adaptation delays [13], [14].

The existing work on the DLMS adaptive filter does not discuss the fixed-point implementation issues, e.g., location of radix point, choice of word length, and quantization at various stages of computation, although they directly affect the convergence performance, particularly due to the recursive behavior of the LMS algorithm. Therefore, fixed-point implementation issues are given adequate emphasis in this paper. Besides, we present here the optimization of our previously reported design [13], [14] to reduce the number of pipeline delays along with the area, sampling period, and energy consumption. The proposed design is found to be more efficient in terms of the power-delay product (PDP) and energy-delay product (EDP) compared to the existing structures. In the next section, we review the DLMS algorithm, and in Section III, we describe the proposed optimized architecture for its implementation. Section IV deals with fixed-point implementation considerations and simulation studies of the convergence of the algorithm. In Section V, we discuss the synthesis of the proposed architecture and comparison with the existing architectures. Conclusions are given in Section VI.

II. REVIEW OF DELAYED LMS ALGORITHM

The weights of LMS adaptive filter during the nth iteration are updated according to the following equations

$$\mathbf{w}n+1 = \mathbf{w}n + \mu \cdot en \cdot \mathbf{x}n \rightarrow (1a)$$

where

$$en = dn - yn \; yn = \mathbf{w}Tn \cdot \mathbf{x}n \rightarrow \quad (1b)$$

where the input vector $\mathbf{x}n$, and the weight vector $\mathbf{w}n$ at the nth iteration are, respectively, given by

$$\mathbf{x}n = [xn, xn-1, \ldots, xn-N+1]T$$
$$\mathbf{w}n = [wn(0), wn(1), \ldots, wn(N-1)]T,$$

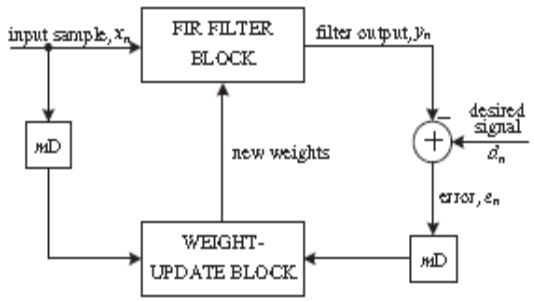


Fig. 1. Structure of the conventional delayed LMS adaptive filter.

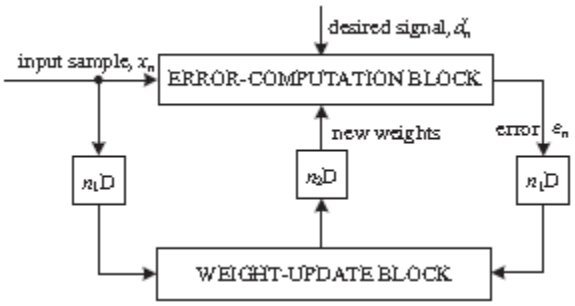


Fig. 2. Structure of the modified delayed LMS adaptive filter.

dn is the desired response, *yn* is the filter output, and *en* denotes the error computed during the *n*th iteration. μ is the step-size, and *N* is the number of weights used in the LMS adaptive filter. In the case of pipelined designs with *m* pipeline stages, the error *en* becomes available after *m* cycles, where *m* is called the "adaptation delay." The DLMS algorithm therefore uses the delayed error *en*−*m*, i.e., the error corresponding to *(n* − *m)*th iteration for updating the current weight instead of the recent-most error. The weight-update equation of DLMS adaptive filter is given by

$$\mathbf{w}n+1 = \mathbf{w}n + \mu \cdot en-m \cdot \mathbf{x}n-m \rightarrow. \quad (2)$$

The block diagram of the DLMS adaptive filter is shown in Fig. 1, where the adaptation delay of *m* cycles amounts to the delay introduced by the whole of adaptive filter structure consisting of finite impulse response (FIR) filtering and the weight-update process. It is shown in [12] that the adaptation delay of conventional LMS can be decomposed into two parts: one part is the delay introduced by the pipeline stages in FIR filtering, and the other part is due to the delay involved in pipelining the weightupdate process. Based on such a decomposition of delay, the DLMS adaptive filter can be implemented by a structure shown in Fig.2

Assuming that the latency of computation of error is *n*1 cycles, the error computed by the structure at the *n*th cycle is *en*−*n*1 , which is used with the input samples delayed by *n*1 cycles to generate the weight-increment term.

The weight- -800 100 200 300 400 500 600 700 -60 -40-200 Iteration Number Mean Squared Error (dB)

LMS (n1=0, n2=0)

DLMS (n1=5, n2=1)

DLMS (n1=7, n2=2)

Convergence performance of system identification with LMS and modified DLMS adaptive filters. update equation of the modified DLMS algorithm is given by

$$\mathbf{w}n+1 = \mathbf{w}n + \mu \cdot en-n1 \cdot \mathbf{x}n-n1 \rightarrow (3a)$$

where

$$en-n1 = dn-n1- yn-n1 \rightarrow (3b)$$

and

$$yn = \mathbf{w}Tn-n2 \cdot \mathbf{x}n. \rightarrow (3c)$$

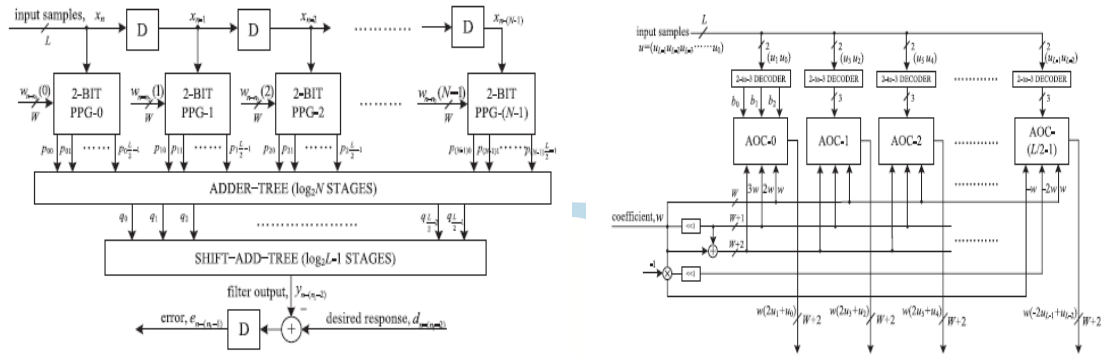
We notice that, during the weight update, the error with *n*1 delays is used, while the filtering unit uses the weights delayed by *n*2 cycles. The modified DLMS algorithm decouples computations of the error-computation block and the weight-update block and allows us to perform optimal pipelining by feedforward cut-set retiming of both these sections separately to minimize the number of pipeline stages and adaptation delay. The adaptive filters with different *n*1 and *n*2 are simulated for a system identification problem. The 10-tap band-pass filter with impulse response

$hn = \sin(wH(n - 4.5))\, \pi(n - 4.5) - \sin(wL(n - 4.5))\pi(n - 4.5)$ for $n = 0, 1, 2, \ldots, 9,$

otherwise *hn* =0 (4) is used as the unknown system. *wH* and *Wl* represent the high and low cutoff frequencies of the passband, and are set to $wH = 0.7\pi$ and $wL = 0.3\pi$, respectively. The step size μ is set to 0.4. A 16-tap adaptive filter identifies the unknown system with Gaussian random input *xn* of zero mean and unit variance. In all cases, outputs of known system are of unity power, and contaminated with white Gaussian noise of −70 dB strength. Fig. 3 shows the learning curve of MSE of the error signal *en* by averaging 20 runs for the conventional LMS adaptive filter *(n*1 = 0, *n*2 = 0*)* and DLMS adaptive filters with *(n*1 = 5, *n*2 = 1*)* and *(n*1 = 7, *n*2 = 2*)*. It can be seen that, as the total number of delays increases, the convergence is slowed down, while the steady-state MSE remains almost the same in all cases. In this example, the MSE difference between the cases *(n*1 = 5, *n*2 = 1*)* and *(n*1 = 7, *n*2 = 2*)* after 2000 iterations is less than 1 dB, on average.

III. PROPOSED ARCHITECTURE

As shown in Fig. 2, there are two main computing blocks in the adaptive filter architecture:
1) the error-computation



A. Pipelined Structure of the Error-Computation Block

The proposed structure for error-computation unit of an *N*-tap DLMS adaptive filter is shown in Fig. 4. It consists of *N* number of 2-b partial product generators (PPG) corresponding to *N* multipliers and a cluster of *L*/2 binary adder trees, followed by a single shift–add tree. Each subblock is described in detail.

1. ***Structure of AOCs:*** The structure and function of an AOC are depicted in Fig. 6. Each AOC consists of three AND cells and two OR cells. The structure and function of AND cells and

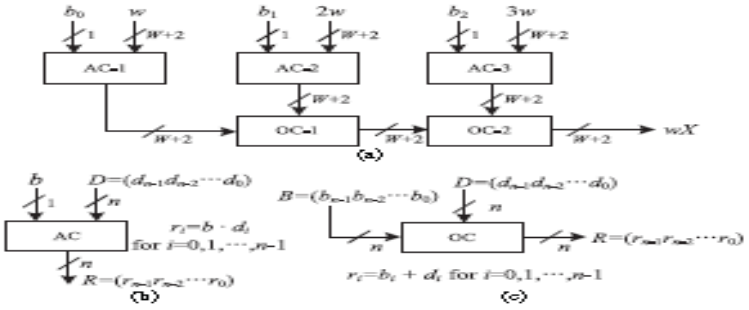


Fig-3.:structure and function of AND/OR cell.binary operation

OR cells are depicted by Fig. 3 respectively. Each AND cell takes an *n*-bit input *D* and a single bit input *b*, and consists of *n* AND gates. It distributes all the *n* bits of input *D* to its *n* ANDgates as one of the inputs. The other inputsof all the *n* AND gates are fed with the single-bit input *b*. As shown in Fig. 3, each OR cell similarly takes a pair of *n*-bit input words and has *n* OR gates. A pair of bits in the same bit position in *B* and *D* is fed to the same OR gate.

The output of an AOC is *w*, 2*w*, and 3*w* corresponding to the decimal values 1, 2, and 3 of the 2-b input (u_1u_0), respectively. The decoder along with the AOC performs a multiplication of input operand *w* with a 2-b digit (u_1u_0), such that the PPG of Fig. 5 performs

$L/2$ parallel multiplications of input word w with a 2-b digit to produce $L/2$ partial products of the product word wu.

2.Structure of Adder Tree:

Conventionally, we should have performed the shift-add operation on the partial products of each PPG separately to obtain the product value and then added all the N product values to compute the desired inner product. However, the shift-add operation to obtain the product value increases the word length, and consequently increases the adder size of $N-1$ additions of the product values. To avoid such increase in word size of the adders, we add all the N partial products of the same place value from all the N PPGs by one adder tree.All the $L/2$ partial products generated by each of the N PPGs are thus added by $(L/2)$ binary adder trees. The outputs of the $L/2$ adder trees are then added by a shift-add tree according to their place values. Each of the binary adder trees require $\log_2 N$ stages of adders to add N partial product, and for the error-computation block for a four-tap filter and input word size $L= 8$. For $N= 4$ and $L= 8$, the adder network requires four binary adder trees of two stages

3. Pipelined Structure of the Weight-Update Block

The proposed structure for the weight-update block is shown in Fig. 3. It performs N multiply-accumulate operations of the form $(\mu \times e) \times x_i + w_i$ to update N filter weights. The step size μ is taken as a negative power of 2 to realize the multiplication with recently available error only by a shift operation. Each of the MAC units therefore performs the multiplication of the shifted value of error with the delayed input samples $x_i$followed by the additions with the corresponding old weight values w_i . All the N multiplications for the MAC operations are performed by N PPGs, followed by N shift– add trees. Each of the PPGs generates $L/2$ partial products corresponding to the product of the recently shifted error value $\mu \times e$ with $L/2$, the number of 2-b digits of the input word x_i , where the sub expression $3\mu \times e$ is shared within the multiplier. Since the scaled error $(\mu \times e)$ is multiplied with the entire N delayed input values in the weight-update block, this subexpression can be shared across all the multipliers as well. This leads to substantial reduction of the adder complexity. The final outputs of MAC units constitute the desired updated weights to be used as inputs to the error-computation block as well as the weight-update block for the next iteration.

4. AdaptationDelay

As shown in Fig. 3, the adaptation delay is decomposed into $n_1$and n_2. The error-computation block generates the delayederror by n_1-1 cycles as shown in Fig. 4.6, which is fed to the weight-update block after scaling by μ; then the input is delayed by 1 cycle before the PPG to make the total delay introduced by FIR filtering be n_1. In Fig. 8, the weight-update block generates $\mathbf{w}_{n-1-n2}$, and the weights are delayed by n_2+1 cycles. However, it should be noted that the delay by 1 cycle is due to the latch before the PPG, which is included in the delay of the error-computation block, i.e., n_1. Therefore, the delay generated in the weight-update block becomes n_2.

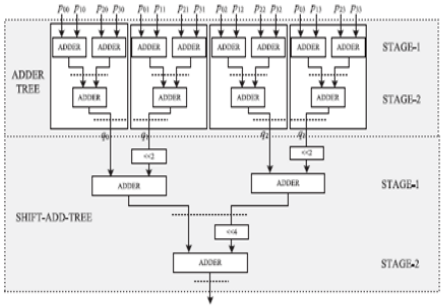


Fig-4 Adder structure of the filtering unit for N=4 and L=8

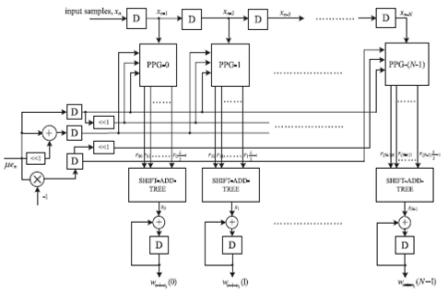
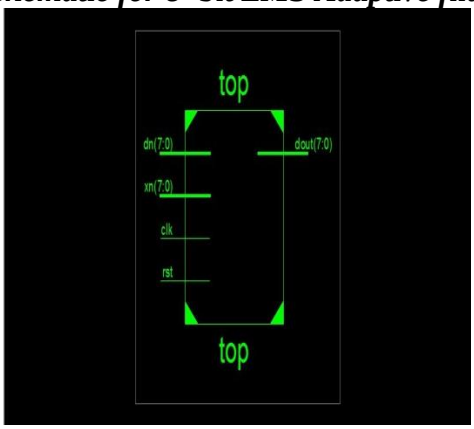


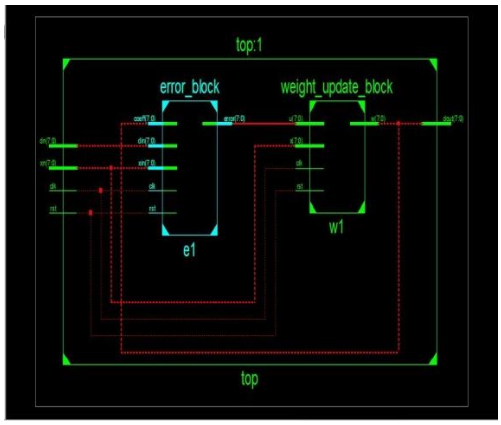
Fig-5:proposed structure of the weight update block

computation block, i.e., n_1. Therefore, the delay generated in the weight-update block becomes n_2.

IV.SIMULATION RESULTS

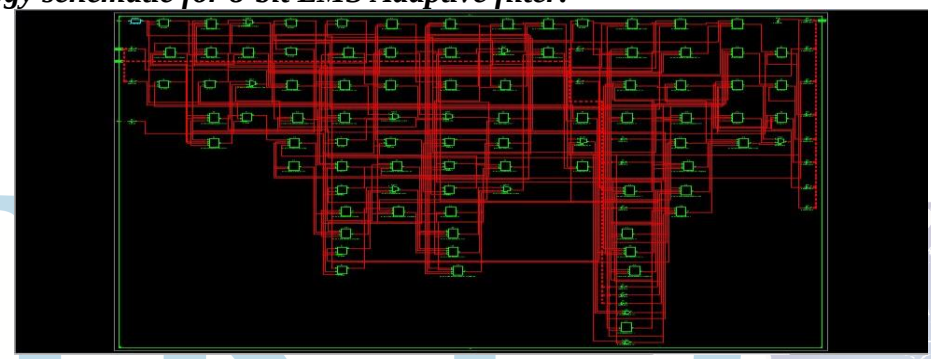
1.RTL schematic for 8- bit LMS Adaptive filter



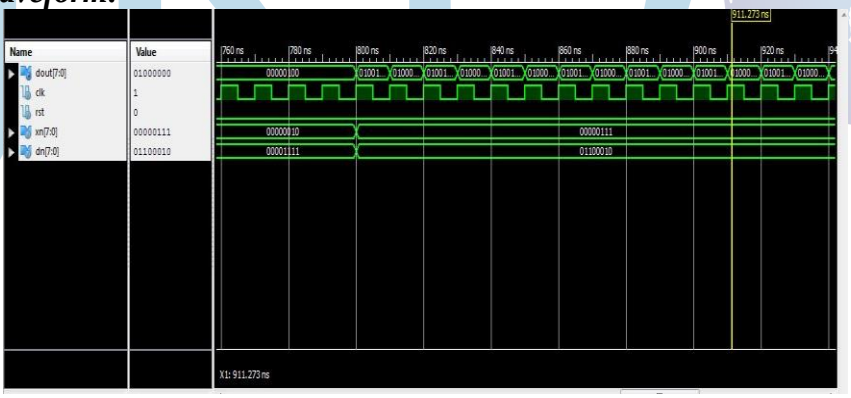


Internal view of RTL Schematic

2. Technology schematic for 8-bit LMS Adaptive filter:



3.Test bench waveform:



Direct form	Memory (in kbs)	Delay (in ns)	Power (in w)
Existing system	231992	13.04	2.144
Proposed system	189772	29.533	0.505

Table.6.2 : Comparison table for 8-bit existing and proposed system

V. CONCLUSION:

The work overviews an area–delay-power efficient low adaptation-delay architecture for fixed-point implementation of LMS adaptive filter. A novel PPG for efficient

implementation of general multiplications and inner-product computation by common sub expression sharing. Furthermore, we have proposed an efficient expansion plan for inner-product computation to reduce the adaptation delay significantly in order to achieve faster convergence performance and to reduce the critical path to support high input-sampling rates. Aside from this, work proposed a strategy for optimized balanced pipelining across the time-consuming blocks of the structure to diminish the adaptation delay and power consumption, as well. The proposed structure involved significantly less adaptation delay and provided significant saving of ADP and EDP compared to the existing structures. The work points out a fixed-point implementation of the proposed architecture, and derived the expression for steady-state error. We found that the steady-state MSE obtained from the analytical result matched well with the simulation result. We also discussed a pruning scheme that provides nearly 20% saving in the ADP and 9% saving in EDP over the proposed structure before pruning, without a noticeable degradation of steady-state error performance. The highest sampling rate that could be supported by the ASIC implementation of the proposed design ranged from about 870 to 1010 MHz for filter orders 8 to 32. When the adaptive filter is needed to be operated at a lower sampling rate, one can use the proposed design with a clock slower than the extreme usable frequency and a lower operating voltage to reduce the power utilization further.

VI. FEATURE SCOPE:

In future scope, the filter response considering the fear factors like delay and power as a important factors,which should not be increased so that it can affect the performance of the filter operations .Future work will include additional circuit optimizations to further reduce the power dissipation by adapting dynamic and analog implementations for the filter resolution module and a high-speed LMS adaptive filter module. Given that our filter is composed of two balanced timing modules, the structure can be divided into two or more pipeline stages with balanced delays, based on a set structure, to effectively increase the response of thee filter throughput.

VII.REFERENCES

[1] M. M. Peiro, E. I. Boemo, and L. Wanhammar, “Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm,” IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process., vol. 49, no. 3, pp. 196–203, Mar. 2002.

[2] C.-H. Chang, J. Chen, and A. P. Vinod, “Information theoretic approach to complexity reduction of FIR filter design,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 55, no. 8, pp. 2310–2321, Sep. 2008.

[3] F. Xu, C. H. Chang, and C. C. Jong, “Contention resolution—A new approach to versatile subexpressions sharing in multiple constant multiplications,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 55, no. 2, pp. 559–571, Mar. 2008.

[4] F. Xu, C. H. Chang, and C. C. Jong, “Contention resolution algorithms for common subexpression elimination in digital filter design,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 52, no. 10, pp. 695–700, Oct. 2005.

[5] I.-C. Park and H.-J. Kang, “Digital filter synthesis based on an algorithm to generate all minimal signed digit representations,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 21, no. 12, pp. 1525– 1529, Dec. 2002

[6] C.-Y. Yao, H.-H. Chen, T.-F. Lin, C.-J. J. Chien, and X.-T. Hsu, “A novel common-subexpression-elimination method for synthesizing fixed-point FIR filters,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 51, no. 11, pp. 2215–2221, Sep. 2004

[7] O. Gustafsson, “Lower bounds for constant multiplication problems,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 54, no. 11, pp. 974–978, Nov. 2007.

[8] Y. Voronenko and M. Puschel, “Multiplierless multiple constant multiplication,” ACM Trans. Algorithms, vol. 3, no. 2, pp. 1–38, May 2007.

[9] D. Shi and Y. J. Yu, “Design of linear phase FIR filters with high probability of achieving minimum number of adders,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 58, no. 1, pp. 126–136, Jan. 2011.

[10] R. Huang, C.-H. H. Chang, M. Faust, N. Lotze, and Y. Manoli, “Signextension avoidance and word-length optimization by positive-offset representation for FIR filter design,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 58, no. 12, pp. 916–920, Oct. 2011.

[11] P. K. Meher, “New approach to look-up-table design and memory-based realization of FIR digital filter,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 57, no. 3, pp. 592–603, Mar. 2010.

[12] P. K. Meher, S. Candrasekaran, and A. Amira, "FPGA realization of FIR filters by efficient and flexible systolization using distributed arithmetic," IEEE Trans. Signal Process., vol. 56, no. 7, pp. 3009–3017, Jul. 2008.
[13] S. Hwang, G. Han, S. Kang, and J.-S. Kim, "New distributed arithmetic algorithm for low-power FIR filter implementation," IEEE Signal Process. Lett., vol. 11, no. 5, pp. 463–466, May 2004.
[14] H.-J. Ko and S.-F. Hsiao, "Design and application of faithfully rounded and truncated multipliers with combined deletion, reduction, truncation, and rounding," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 58, no. 5, pp. 304–308, May 2011.
[15] H. Samueli, "An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficient," IEEE Trans. Circuits Syst., vol. 36, no. 7, pp. 1044–1047, Jul. 1989.
[16] Y. C. Lin and S. Parker, "Discrete coefficient FIR digital filter design based upon an LMS criteria," IEEE Trans. Circuits Syst., vol. 30, no. 10, pp. 723–739, Oct. 1983
[17] A. Blad and O. Gustafsson, "Integer linear programming-based bit-level optimization for high-speed FIR filter architecture," Circuits Syst. Signal Process., vol. 29, no. 1, pp. 81–101, Feb. 2010.
[18] F. Xu, C. H. Chang, and C. C. Jong, "Design of low-complexity FIR filters based on signed-powers-of-two coefficients with reusable common subexpressions," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 26, no. 10, pp. 1898–1907, Oct. 2007.