Distributed Arithmetic based Low-Power LMS Adaptive FIR Filter Design

Wasim Maroofi
Research Scholar
NIIST, Bhopal
Madhya Pradesh, India

Lalit Jain
Assistant Professor
NIIST, Bhopal
Madhya Pradesh, India

ABSTRACT
Adaptive filtering forms a significant class of DSP algorithms employed in several hand held mobile devices for applications like echo cancellation, signal de-noising, and channel equalization. This paper presents a different pipelined architecture for low-power implementation of Adaptive filter based on distributed arithmetic (DA). The traditional adder-based shift accumulation for Distributed Arithmetic based computation of inner-product is swapped by conditional signed carry-save accumulation. A fast bit clock is employed only for carry-save accumulation which results in reduction of power consumption in the proposed design, while use of a much slower bit clock is used for rest of the operations. It contains the smaller Look-Up Table (LUT), same quantity of multiplexers and almost half the number of adders in comparison to the existing Distributed Arithmetic-based design. By changing the inner block, a reduction in power consumption is aimed at. So the previous DA-based adaptive filter in average for filter lengths N=4 and N=16 have been implemented.

General Terms
Distributed Arithmetic, LMS Adaptive Filter, VHDL

Keywords
Distributed Arithmetic, DA, Adaptive Filter, LMS algorithm, Carry save adder, Noise Cancellation, Digital Signal Processing

1. INTRODUCTION
In the last few decades, the field of digital signal processing, and particularly adaptive signal processing, has developed immensely due to the increasing availability of technology for the implementation of the emerging algorithms. These algorithms have been applied to a widespread range of problems which include noise and acoustic echo canceling, channel equalization, signal prediction, adaptive arrays, wireless channel estimation, radar guidance systems as well as many others [1]. An adaptive algorithm is employed for estimating a time varying signal. There are many adaptive algorithms at our disposal like the Recursive Least Square (RLS), Kalman filter, etc., but the most frequently used is the Least Mean Square (LMS) algorithm. LMS is a simple yet powerful algorithm which can be implemented to take advantage of the Lattice FPGA architecture. The tapped-delay line finite impulse response (FIR) filter wherein the weights are updated by the well-known Widrow–Hoff LMS algorithm [2] is the most popularly used adaptive filter also because of its satisfactory convergence performance [3]. The direct form configuration on the forward path of the FIR filter consequently leads to a long critical path due to an inner-product computation to obtain a filter output. Hence, when the input signal is having a high sampling rate, it becomes crucial to reduce the critical path of the structure so that it could not surpass the sampling period [4]. Distributed Arithmetic based technique comprises of a structure which is devoid of multiplier that increases the throughput. Such DA-based structures have been proposed by Allred et al [5].

2. FIR FILTER
Finite Impulse Response (FIR) filters are one of the key building blocks of numerous signal processing applications in communication systems. Channel equalization, interference cancellation and matched filtering are some of the variety of FIR filter applications. Hence, the programmable and reconfigurable FIR filter architectures are required for next generation communication systems which consume less power, have low complexity and satisfy high speed operation requirements. The major holdup in FIR filter implementation is the implementation of the multiplier coefficients, which are conventionally implemented using add/sub/shift operations.

Digital filters are the essential units for digital signal processing systems. Traditionally, digital filters are realized in Digital Signal Processor (DSP), but DSP-based solution cannot meet the high speed requirements in some applications because of its structure which is sequential in nature. Nowadays, Field Programmable Gate Array (FPGA) technology is extensively used in digital signal processing area because FPGA-based solution can accomplish high speeds owing to its parallel structure and configurable logic, thus providing great amount of flexibility and high reliability throughout design process and later maintenance. A number of architectures have been reported in the literature for memory-based implementation of DSP algorithms which involve orthogonal transforms and digital filters [6].

The FIR filter is implemented serially using a multiplier and an adder with a feedback which is illustrated in the high level schematic in Fig. 1.

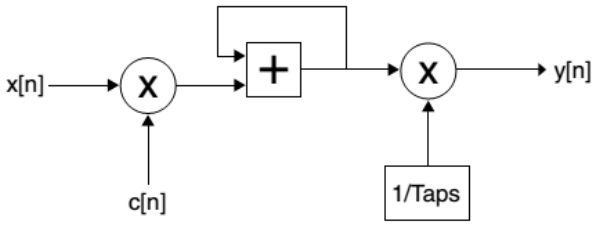

Fig.1: FIR Filter

The LMS algorithm updates the coefficient in an iterative manner and feeds it to the FIR filter. The FIR filter then uses this coefficient c(n) in addition to the input reference signal x(n) to generate the output response y(n). The output y(n) is then subtracted from the desired signal d(n) to generate an error signal e(n), which is ultimately utilized by the LMS algorithm to compute the next set of coefficients.

3. ADAPTIVE FIR FILTER

An adaptive filter may be understood as a self-modifying digital filter that adjusts its coefficients in order to minimize an error function. This error function, which is also denoted as the cost function, is a distance measurement between the reference (desired signal) and the output response of the adaptive filter. The sheer ability of an adaptive filter to operate satisfactorily in an indefinite environment along with tracking of time variations of input statistics make the adaptive filter a powerful device for signal processing and control applications [7]. They can automatically adapt (self-optimize) in the face of varying situations and changing system requirements. They can be trained to perform specific filtering and decision-making tasks in accordance with some updating equations. Moreover, an adaptive filter, due to its real-time self-adjusting characteristic, is occasionally expected to track the optimum behavior of a slowly varying environment. Due to its simplicity and efficacy, the most widely employed adaptive filter structure is by far the transversal filter (or tapped-delay line) associated to standard finite-duration impulse response (FIR) filters. Filter structure significantly influences the computational complexity of a given adaptive filter algorithm and the overall speed of the adaptation process. The basic configuration of an adaptive filter, operating in the discrete-time domain k, is shown in Figure 2. In such a scheme, the input signal is denoted by x(k), the reference or desired signal d(k) (that usually includes some noise component), y(k) is the output of the adaptive filter, and the error signal is defined as e(k) = d(k)− y(k).

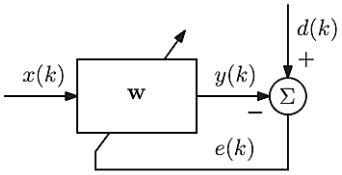

Fig.2: Block diagram of adaptive filter

The error signal is used by the adaptation algorithm for updating the adaptive filter coefficient vector w(k) according to some performance criterion. In broader sense, the entire adaptation process targets at minimizing some metric of the error signal, driving the adaptive filter output signal to approximate the reference signal.

4. LMS ADAPTIVE FILTER (EXISTING DESIGN)

The Least Mean Square (LMS) algorithm was introduced by Widrow and Hoff in 1959. The LMS algorithm has established itself as the mainstay of adaptive signal processing for two major reasons:

- Simplicity of implementation and a computational efficiency that is linear in the number of adjustable parameters.
- Robust performance

It is an adaptive algorithm, which makes use of a gradient-based method of steepest decent. LMS algorithm uses the estimates of the gradient vector from the data that becomes available. LMS incorporates an iterative procedure that makes successive corrections to the weight vector in the negative direction of the gradient vector which eventually leads to the minimum mean square error (MSE). The gradient is the del operator (partial derivative) and is applied to find the divergence of a function, which represents the error with respect to the nth coefficient in this case. The LMS algorithm approaches towards the minimum of a function so as to minimize error by taking the negative gradient of the function.

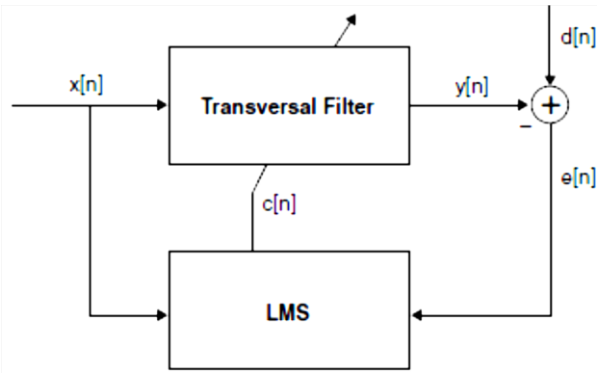

Fig.3: LMS Implementation Using FIR Filter

The desired signal d(n) is tracked by adjusting the filter coefficients c(n). The input reference signal x(n) is a known signal that serves as an input to the FIR filter. The difference between d(n) and y(n) is the error e(n). The error e(n) is then given to the LMS algorithm to compute the filter coefficients c(n+1) which minimizes the error in an iterative manner. The LMS equation for computing the FIR coefficients is as follows:

$$c(n+1) = c(n) + \mu \,.\, e(n) \,.\, x(n) \quad (1)$$

where $e(n) = d(n) - y(n)$ (2)

$$y(n) = c^{qT}(n) \,.\, x(n) \quad (3)$$

The convergence time of the LMS algorithm depends on the step size μ.

The input vector x(n) and the weight vector c(n) at the *n*th training iteration are respectively given by:

$$x(n) = [x(n), x(n-1), \ldots, x(n-N+1)]^T \quad (4)$$

$$c(n) = [c_0(n), c_1(n), \ldots, c_{N-1}(n)]^T \quad (5)$$

d(n) is the desired response,

y(n) is the filter output of the nth iteration.

e(n) denotes the error computed during the nth iteration, used for updating the weights,

μ is the convergence factor, and

N is the filter length.

In pipelined architecture, the feedback error e(n) becomes available only after a definite number of cycles, called the adaptation delay. So, the pipelined architectures make use of delayed error e(n − m) in order to update the current weight instead of the most recent error, where 'm' is the adaptation delay. The weight-update equation of such delayed LMS adaptive filter is given by:

$$c(n+1) = c(n) + \mu \cdot e(n-m) \cdot x(n-m). \quad (6)$$

5. DISTRIBUTED ARITHMETIC (DA)

Distributed arithmetic (DA) is an efficient multiplication-free technique for calculating inner products first introduced by Croisier, et al. [8] and Zohar [9] and further developed by

Peled and Liu [10] more than three decades ago. The multiplication operation is substituted by a mechanism that generates partial products before summing the products together. The basic difference between distributed arithmetic and standard multiplication is the way in which the partial products are generated and added together. Since its introduction, distributed arithmetic has been extensively implemented in many digital signal processing applications, including but not limited to digital filtering, discrete cosine transform, discrete Fourier transform.

6. DISTRIBUTED ARITHMETIC BASED APPROACH (PROPOSED DESIGN)

The LMS adaptive filter involves performing of an inner-product computation during each cycle, which contributes to the most of the critical path. Let the inner product of eqn. (3) be given by:

$$y = \sum_{k=0}^{N-1} c_k \, . x_k \qquad (7)$$

where c_k and x_k for $0 \le k \le N-1$ are the N-point vectors corresponding to the current weights and most recent N − 1 input, respectively. If L is assumed to be the bit width of the weight, then each component of the weight vector may be expressed in two's complement representation as follows:

$$c_k = -c_{k0} + \sum_{l=1}^{L-1} c_{kl} \, . 2^{-l} \qquad (8)$$

where c_{kl} denotes the lth bit of c_k.

Substituting (8), we can write (7) in an expanded form as follows:

$$y = -\sum_{k=0}^{N-1} x_k \, . c_{k0} + \sum_{k=0}^{N-1} x_k \, . [\sum_{k=0}^{N-1} x_k \, . c_{kl}] \qquad (9)$$

Now for converting the sum-of-products form of (7) into a distributed form, the order of summations over the indices *k* and *l* in (6) may be switched to get:

$$y = -\sum_{k=0}^{N-1} x_k \, . c_{k0} + \sum_{l=1}^{L-1} 2^{-l} \, . [\sum_{k=0}^{N-1} x_k \, . c_{kl}] \qquad (10)$$

and the inner product given by (10) can be computed as:

$$y = [\sum_{l=1}^{L-1} 2^{-l} \, . y_l] - y_0 \text{ , where } y_l = \sum_{k=0}^{N-1} c_{kl} \, . x_k \qquad (11)$$

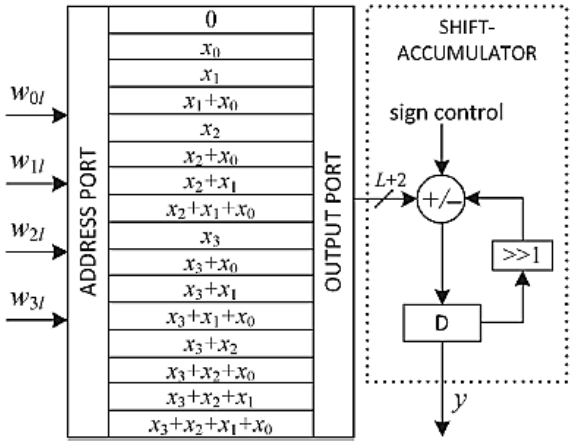

Fig.4: Traditional implementation of DA-based four-point inner product

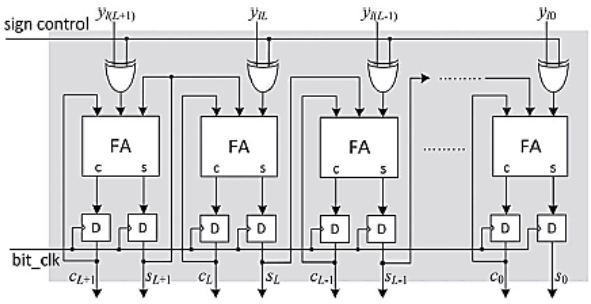

Fig.5: Carry-save implementation of shift accumulation

Since any element of the N-point bit sequence $\{c_{kl}$ for $0 \le k \le N-1\}$ can either be zero or one, the partial sum y_l for l = 0, 1,…, L − 1 can have 2^N possible values. If all 2^N possible values of y_l are pre-computed and stored in a LUT, the partial sums y_l can be read out from the LUT using the bit sequence $\{c_{kl}\}$ as address bits for computation of the inner product.

Hence, the inner product of eqn. (11) can be calculated in L cycles of shift accumulation, which is then followed by LUT-read operations corresponding to L number of bit slices $\{c_{kl}\}$ for $0 \le l \le L-1$, as shown in Fig. 4. As the shift accumulation in Fig. 4 encompasses substantial critical path, it is performed using carry-save accumulator, as shown in Fig. 5. The bit slices of vector *c* are fed one after the other in the LSB to the MSB order to the carry-save accumulator. However, the negative (two's complement) of the LUT output is required to be accumulated in case of MSB slices. So, the entire bits of LUT output are passed through XOR gates with a sign-control input which is set to '1' only when the MSB slice appears as address. The XOR gates thus produce the one's complement of the LUT output corresponding to the MSB slice but do not affect the output for other bit slices. Finally, the sum and carry words that are acquired after L clock cycles are essential to be added by a final adder which has been excluded from the figure, and the input carry of the final adder is needed to be set to '1' to account for the two's complement operation of the LUT output corresponding to the MSB slice. The content of the k_{th} LUT location can be expressed as:

$$c_k = \sum_{j=0}^{N-1} x_j . k_j \qquad (12)$$

where *kj* is the (j + 1)th bit of N-bit binary representation of integer k for $0 \le k \le 2^N - 1$. Note that c_k for $0 \le k \le 2^N - 1$ can be pre-computed and stored in RAM-based LUT of 2^N words. However, instead of storing 2^N words in LUT, we store (2^N − 1) words in a DA table of $2^N - 1$ registers. An example of such a DA table for N = 4 is shown in Fig. 6. It contains only 15 registers for storing the pre-computed sums of input words. Seven adders in parallel compute the new values of c_k.

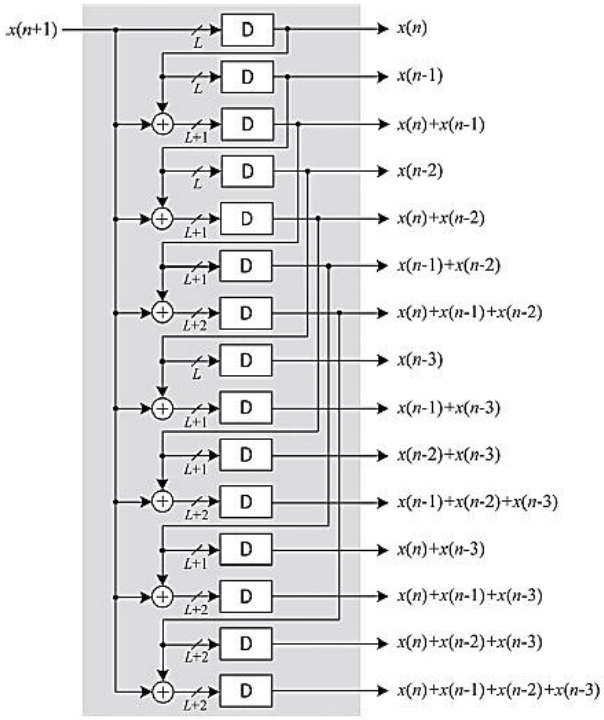

Fig.6: DA Table for generation of possible sums of input samples

The computation of adaptive filters of large orders must be decomposed into small adaptive filtering blocks since DA-based implementation of inner product of long vectors requires a very large LUT [5].

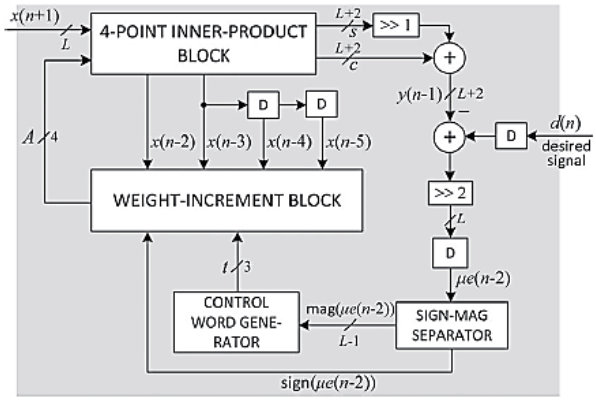

Fig. 7: Proposed structure of DA-based LMS adaptive filter

The proposed structure of DA-based adaptive filter of length N = 4 is shown in Fig. 7. It contains a four-point inner-product block and a weight-increment block in addition to additional circuits for the computation of error value e(n) and control word ***t*** for the barrel shifters. The four-point inner-product block [shown in Fig. 8] includes a DA table consisting of an array of 15 registers which stores the partial inner products y_l for $0 < l \leq 15$ and a 16 : 1 multiplexer to select the content of one of those registers. Bit slices of weights $A = \{w_{3l}\ w_{2l}\ w_{1l}\ w_{0l}\}$ for $0 \leq l \leq L - 1$ are fed to the MUX as control in LSB-to- MSB order, and the output of the MUX is given to the carry-save accumulator (shown in Fig. 4). After L bit cycles, the carry-save accumulator shift accumulates all the partial inner products and generates a sum word and a carry word of size (L + 2) bit each. The carry and sum words are shifted - added with an input carry "1" to generate filter output which is subsequently subtracted from the desired output d(n) to obtain the error e(n). As is the case in [5], all the bits of the error except the most significant one are ignored, such that multiplication of input x_k by the error is implemented by a right shift through the number of locations given by the number of leading zeros in the magnitude of the error. The magnitude of the computed error is decoded to generate the control word ***t*** for the barrel shifter. The logic used for the generation of control word ***t*** to be used for the barrel shifter is shown in Fig. 10. The convergence factor μ is usually taken to be O(1/N). Convergence factor has been taken as μ = 1/N. However, one can take μ as $2^{-i/N}$, where i is a small integer. The number of shifts ***t*** in that case is increased by i, and the input to the barrel shifters is pre-shifted by i locations accordingly to reduce the hardware complexity. The weight-increment unit [shown in Fig. 9] for N = 4 consists of four barrel shifters and four adder/subtractor cells. The barrel shifter shifts the different input values x_k for k = 0, 1, ..., N − 1 by suitable number of locations (determined by the location of the most significant one in the estimated error). The barrel shifter produces the desired increments that are to be added with or subtracted from the current weights. The sign bit of the error is used as the control for adder/subtractor cells such that, when sign bit is zero or one, the barrel-shifter output is respectively added with or subtracted from the content of the corresponding current value in the weight register.

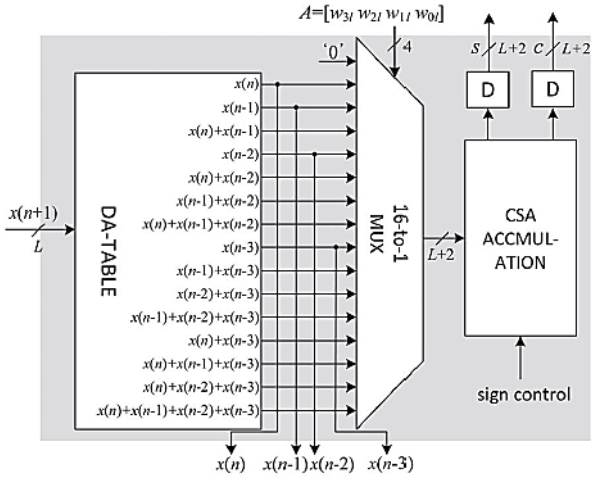

Fig. 8: Structure of four-point inner-product block

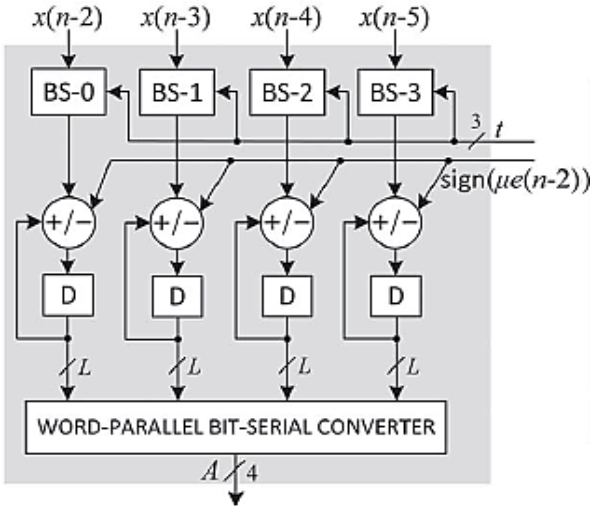

Fig. 9: Structure of weight-increment block for N=4

if r_6 = 1 then t = “000”;
else if r_5 = 1 then t = “001”;
else if r_4 = 1 then t = “010”;
else if r_3 = 1 then t = “011”;
else if r_2 = 1 then t = “100”;
else if r_1 = 1 then t = “101”;
else if r_0 = 1 then t = “110”;
else then t = “111”;

Fig. 10: Logic used for generation of control word *t* for the barrel shifter for L=8

7. RESULTS

Table 1: Result Comparison of Existing and Proposed Work with respect to Area, Delay and Power

DESIGN	Gate Count	Delay (nS)	Power (mW)
EXISTING	**31,184**	**13.704**	**183**
PROPOSED	**31,500**	**13.704**	**91**

Here it can be seen that the power consumption has reduced to just below half of that in the existing design. This has resulted because of a reduced switching activity of the design based on carry-save adder. An efficient pipelined architecture for low-power, and low delay implementation of DA-based adaptive filter. A carry-save accumulation scheme of signed partial inner products for the computation of filter output has been implemented. From the synthesis results, it was found that the proposed design consumes less power over our previous DA-based FIR adaptive filter. In future, work can be implemented on digital communication, signal processing application, digital radio receivers, software radio receivers and echo cancellation.

8. CONCLUSION

This paper presented the implementation of carry-save accumulation scheme of signed partial inner products for the computation of filter output. It is well implemented for Adaptive Filtering applications. From the synthesis results, it was found that the proposed design consumes less power over our previous DA-based FIR adaptive filter.

9. ACKNOWLEDGMENTS

We would like to extend sincere gratitude towards our mentor Prof. Puran Gour, HOD (E&C Dept., NIIST, Bhopal), Prof. Tahseenul Hasan, Asst. Professor (ETC Dept., ACET, Nagpur) and Mr. Rehan Maroofi, who have been there for constant guidance and provided support to achieve success in our endeavor.

10. REFERENCES

[1] Apolinário Jr, José A., and Sergio L. Netto. "Introduction to Adaptive Filters." In QRD-RLS Adaptive Filtering, pp. 1-27. Springer US, 2009.

[2] B. Widrow and S. D. Stearns, Adaptive signal processing. Prentice Hall, Englewood Cliffs, NJ, 1985. .

[3] S. Haykin and B. Widrow, Least-mean-square adaptive filters. Wiley-Interscience, Hoboken, NJ, 2003.

[4] Park, Sang Yoon, and Pramod Kumar Meher. "Low-power, high-throughput, and low-area adaptive FIR filter based on distributed arithmetic." Circuits and Systems II: Express Briefs, IEEE Transactions on 60, no. 6 (2013): 346-350.

[5] D. J. Allred, H. Yoo, V. Krishnan, W. Huang, and D. V. Anderson, “LMS adaptive filters using distributed arithmetic for high throughput,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 52, no. 7, pp. 1327–1337, Jul. 2005.

[6] P. K. Meher, ‘LUT Optimization for Memory-Based Computation,’ IEEE Trans on Circuits & Systems-II, pp.285-289, April 2010.

[7] Haykin, Simon S. Adaptive filter theory. Pearson Education India, pp.18, 1996.

[8] A. Croisier, D. Esteban, M. Levilion, and V. Rizo, “Digital filter for PCM encoded signals US Patent 3, 777, 130,” 1973.

[9] S. Zohar, “New Hardware Realizations of Nonrecursive Digital Filters,” IEEE Transactions on Computers, vol. C-22, no. 4, pp. 328–338, 1973.

[10] A. Peled and B. Liu, “A New Hardware Realization of Digital Filters,” IEEE Transactions on ASSP, vol. 22, no. 6, pp. 456–462, 1974.