

Design of Adaptive FIR Filter using Distributed Arithmetic

Saladhi Alekhya

M.Tech (VLSI DESIGN),
Department of ECE,
Pydah Kaushik College of Engineering,
Visakhapatnam, India.

A.Vivek Babu

Assistant Professor,
Department of ECE,
Pydah Kaushik College of Engineering,
Visakhapatnam, India.

Abstract:

In this manuscript, an unusual adaptive FIR filter using distributed arithmetic (DA) for area efficient design is implemented. DA is bit-serial computational action and uses parallel look-up table (LUTs) apprise and equivalent implementation of filtering and weight-update operations to appliance high throughput filter rates irrespective of the filter length. The full adder based conditional signed carry save accumulation for DA-based inner product computation is swapped and design by using 10 transistor full adder based carry save accumulation of shift accumulation, with the intention of the proposed design, it can reduce the area complexity and power consumption. The least-mean-square (LMS) algorithm adaptation is functioned to update the weight and abate the mean square error between the assessed and chosen output. The weight increment block based adder/subtractor cells is exchanged by carry save adder in order to reduce area difficulty. It comprises of multiplexors, smaller LUT, and practically half the number of adders contrasted to the present DA-based design.

Index Terms: Adaptive Filter, Distributed Arithmetic (DA), Finite Impulse Response (FIR), Least Mean Square (LMS) Algorithm, Lookup table (LUT).

I. INTRODUCTION

Adaptive filters find extensive use in many signal processing applications such as channel equalization, echo cancellation, noise cancellation [1]. The finite impulse response (FIR) filters whose weights are updated by the famous Widrow-Hoff least mean square (LMS) algorithm is the most popularly used adaptive filter not only due to its simplicity but also

due to its satisfactory convergence performance [5]. The direct form configuration on the onward path of the FIR filter results in a long critical path due to an inner product computation to obtain a filter output. Consequently, it is required to reduce the critical path of the structure if the input signal has high sampling rate. By reducing the critical path of the structure, thereby, the critical path could not exceed the sampling period.

Distributed arithmetic (DA) is so named because it performed arithmetic operation. DA is bit serial computation in nature and it eliminates the need for hardware multipliers and is capable of implementing large order filters with very high throughput. A lot of study has been done to implement the DA based adaptive FIR filter for area efficient design, the multiplier-less distributed arithmetic (DA) based technique has achieved plenteous popularity for its high throughput, but it results are increased in cost-effective, area and time efficient computing structures [8]. DA based hardware efficient adaptive FIR filter inner product has been suggested by Allred et al. [2] using two separate lookup tables (LUTs) Filtering lookup table and Auxiliary lookup table for filtering and weight updating module. Later, Guo and DeBrunner [3], [4] have improved the design structure in [2] by using only one lookup table instead of two LUTs for both filter and weight updating module. On the other hand, the design process in [2], [3], [4] and [8] require more cycles for lookup table (LUT) update for each new sample, hence it do not support high sampling rate. Meher and Park have improved the design with low adaptation delay for high speed DA based adaptive filter [6]. In a recent paper, Meher and Park proposed a new DA based adaptive filter

architecture for low power, low area and high throughput with very low adaptation delay [7].

This brief proposes an adaptive FIR filter using distributed arithmetic for area efficient design. High Throughput is achieved by using a parallel lookup table update and equivalent implementation of filtering and weight-updating operations. The conditional signed carry saved accumulation for DA-based inner product computation is designed by using 10 transistor full adder based carry saved accumulation of shift accumulation. The use of the proposed design helps to reduce the area complexity and power consumption.

In the next section, a brief study of the least mean square (LMS) adaptive algorithm, followed by the description of the proposed DA based technique filter in Section III. The structure of the proposed adaptive filter and description of the proposed DA based adaptive FIR filter in Section IV. Results and Conclusions are given in Section V and VI.

II. Review of LMS Adaptive Algorithm

The LMS algorithm computes a filter output and an error value that is equal to the difference between the current filter output and the desired response for every clock cycle. In every training cycle, the estimated error is then used to update the filter weights. The weights of LMS adaptive filter during the n th iteration is updated according to the following equations [6]:

$$w(n+1) = w(n) + \mu \cdot e(n) \cdot x(n) \quad (1a)$$

Where

$$e(n) = d(n) - y(n) \quad (1b)$$

$$y(n) = w^q(n) \cdot x(n) \quad (1c)$$

The input vector $x(n)$ and the weight vector $w(n)$ at the n th training iteration are respected given by

$$x(n) = [x(n), x(n-1), \dots, x(n-N+1)]^T \quad (2a)$$

$$w(n) = [w_0(n), w_1(n), \dots, w_{N-1}(n)]^T \quad (2b)$$

$d(n)$ is the desired response, and $y(n)$ is the filter output of the iteration. $e(n)$ denotes the error value generated during the iteration, which is used to update the weights, is the convergence factor, and N is the filter length.

In the case of filter designs, the feedback error $e(n)$ becomes available after certain number of cycles, called the "adaptation delay". The pipelined architectures therefore use the delayed error $e(n-m)$ for updating the current weight instead of the most recent error, where is the adaptation delay. The weight update equation of such delayed LMS adaptive filter is given by

$$w(n+1) = w(n) + \mu \cdot e(n-m) \cdot x(n-m) \quad (3)$$

III. Proposed DA-Based Approach for Inner Product Computation

In each cycle, the LMS adaptive filter needs to perform an inner-product computation which contributes to the most of the critical path. Let the inner product computation of (1c) be given by

$$y = \sum_{k=0}^{N-1} r_k \cdot s_k \quad (4)$$

Where r_k and s_k for $0 \leq k \leq N-1$ form the N -point vectors corresponding to the current weights and most recent input respectively. Let us assume be the bit width of the weight, every component of the vector weight may be expressed in 2's complement representation

$$r_k = -r_{k0} + \sum_{l=1}^{L-1} r_{kl} \cdot 2^{-l} \quad (5)$$

Where W_{kl} denotes the L th bit of r_k Substituting (5), we can write (4) in an expanded form

$$y = -\sum_{k=0}^{N-1} s_k \cdot r_{k0} + \sum_{k=0}^{N-1} s_k \cdot [\sum_{l=1}^{L-1} r_{kl} \cdot 2^{-l}] \quad (6)$$

To convert the sum-of-product form of (4) into a distributed form, the order of summations over the indices and in (6) can be interchanged to have

$$y = -\sum_{k=0}^{N-1} s_k \cdot r_{k0} + \sum_{l=1}^{L-1} 2^{-l} \cdot [\sum_{k=0}^{N-1} s_k \cdot r_{kl}] \quad (7)$$

And the inner product given by (7) can be computed as

$$y = [\sum_{i=1}^{L-1} 2^{-i} \cdot y_i] - y_0 \cdot y_l = \sum_{k=0}^{N-1} s_k \cdot r_{kl} \quad (8)$$

Meanwhile any element of the N-point bit sequence $\{r_k$ for $0 \leq k \leq N-1$ can either be 1 or 0, the partial sum Y_l for $l = 0, 1, \dots, L-1$ can have 2^n possible values. If all the possible values sum are precomputed and stored in a LUT, the partial sum can be read out from the LUT using the bit sequence $\{ \}$ as address bits for computing the inner product.

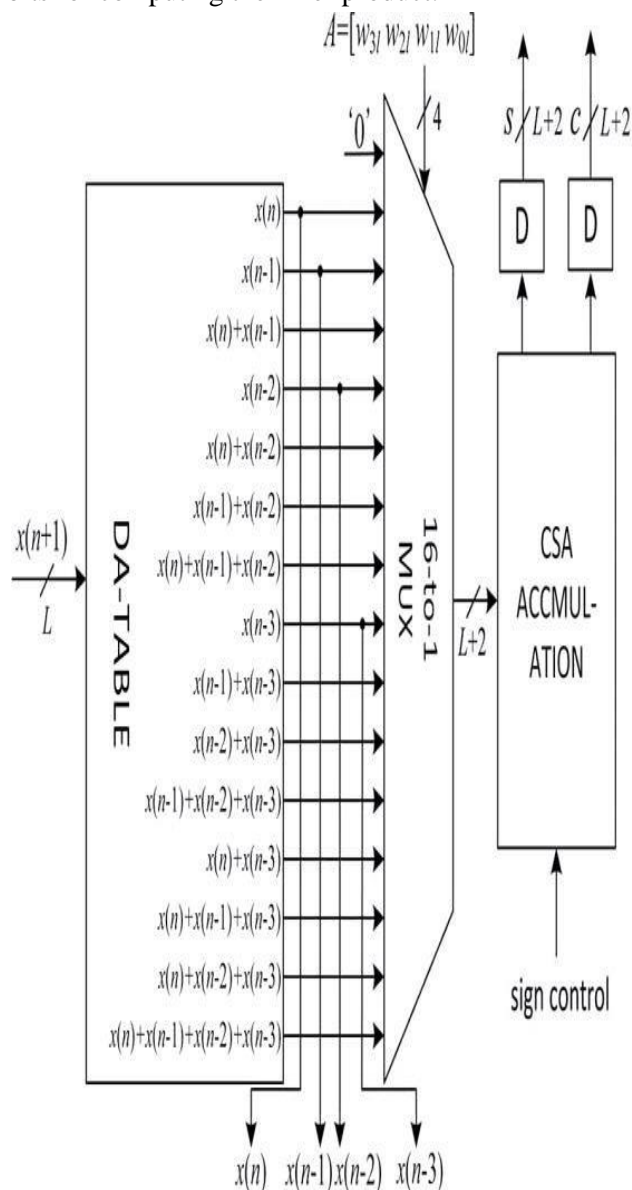


Figure 1: DA-based implementation of four point inner product

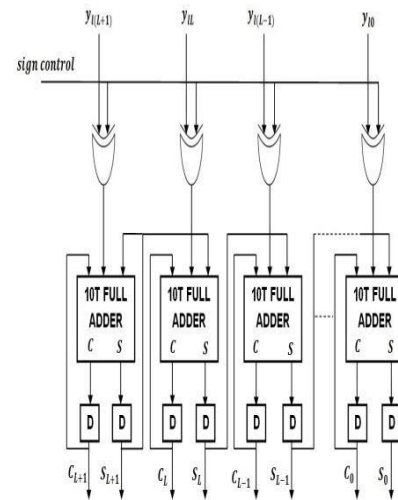


Fig 2: Carry save implementation of shift accumulation

The inner product of (8) can therefore be calculated in cycles of carry save implementation of shift accumulation, followed by LUT-read operations corresponding to number of bit slices $\{r_{kl}\}$ for $0 \leq l \leq L-1$, as shown in Fig. 1. Since the carry save implementation of shift accumulation in Fig. 2

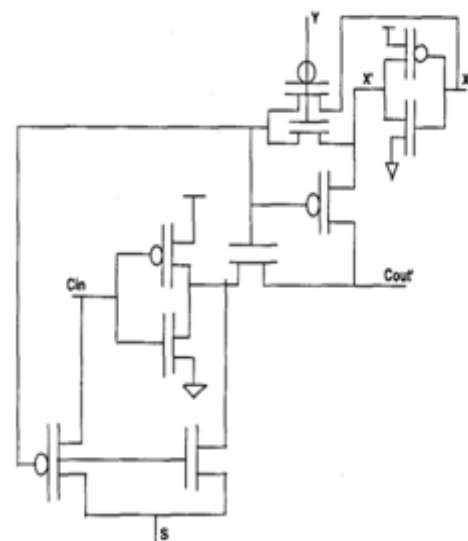


Figure 3: 10T 1-Bit Full Adder

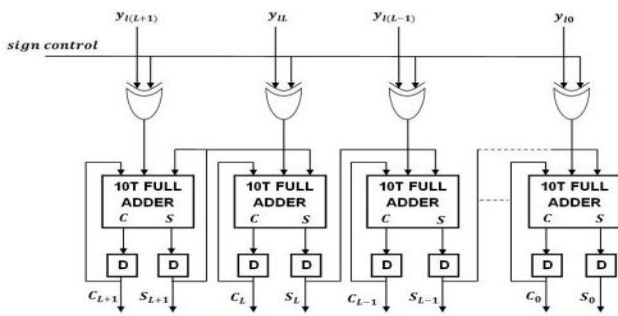


Figure 4: Carry save adder accumulation

The carry save implementation of shift accumulation based full adder is design by using 10 transistor one bit-full adder [9] as shown in Fig. 4. The bit slices of vector are fed one after the next in the LSB to the MSB order to the carry save accumulator. Finally, the sum and carry output of the carry save accumulator is obtained after clock cycle are required to be added by a final adder. The content of the LUT location can be expressed as

$$c_k = \sum_{j=0}^{N-1} x_j \cdot k_j \quad (9)$$

Where k_j is the $(j+1)$ th bit of the N -bit binary representation of integer for can be precomputed and stored in RAM based LUT of words. However, instead of storing words in LUT, we store words in a DA table of registers

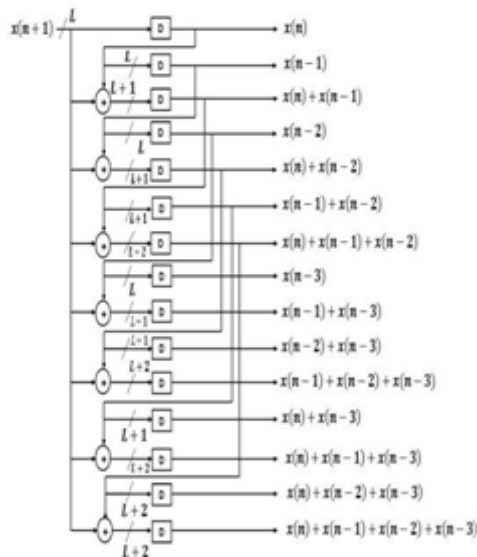


Fig 5: Distributed arithmetic table

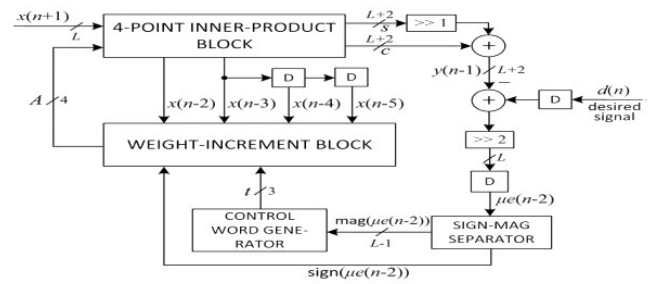


Figure 6: Proposed structure of DA-based LMS adaptive filter length

DA table for $N=4$ is shown in Fig. 5. DA table contains only 15 registers to store the precalculated sums of input words. In DA table, seven new values of are computed by seven adders in parallel.

IV. Proposed Structure of Adaptive FIR Filter

A straight-forward DA-based implementation of inner product requires LUT of very large size. For that reason, the computation of the inner products of large orders needs to be decomposed [4] into small adaptive filtering blocks as shown in Fig. 6 and large order adaptive filters shown in Fig. 7.

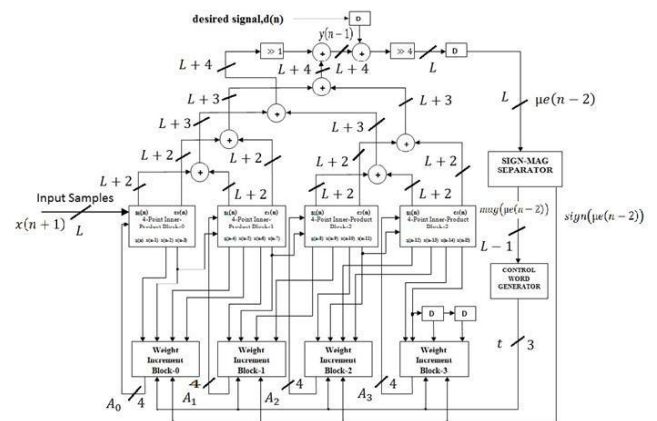


Figure 7: Proposed structure of DA-based LMS adaptive filter of length $N=16$

The structure of DA-based adaptive filter of length $N=4$ comprises of a four-point inner-product block and a weight-increment block along with additional circuits for the computation of error value () and control word for the barrel shifters. The four-point inner-product block [shown in Fig. 1] contains a DA

table consisting of an array of 15 registers [shown in Fig. 5] which stores the partial inner products for and a 16:1 multiplexor to select the content of one of those registers from the DA table. Bit slices of weights $A = \{ \}$ for are fed to the MUX as control in LSB –to- MSB order, and the output of the MUX is fed to the carry save accumulator using 10T full adder as shown in Fig. 4. After bit cycles, the carry save accumulator shift accumulates all the partial inner products and generates a sum and carry output word of size $(+ 2)$ 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 $e(n)$.

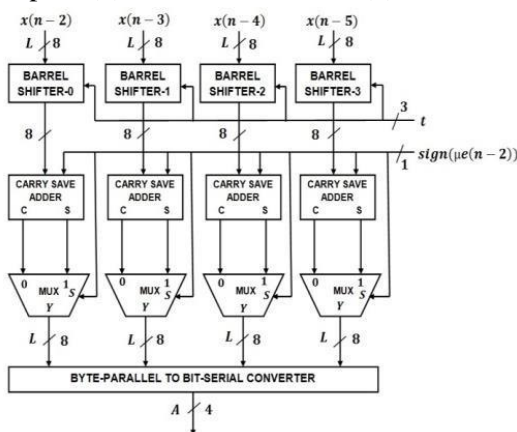


Figure 8: Structure of the weight-increment block for N=4

```

if r6=1 then t= "000";
else if r5 = 1 then t= "001";
else if r4 = 1 then t= "010";
else if r3 = 1 then t= "011";
else if r2 = 1 then t= "100";
else if r1 = 1 then t= "101";
else if r0 = 1 then t= "110";

```

$r = abs(\mu e(n - 2))$
 r_i : *ith* bit of 7 – bit word *r*

Figure 9: Logic used for generation of control word t for the barrel shifter for L=8

As in the case in [4], all the bits of the error except the most significant bit (MSB) one are ignored (8th Bit). The remaining bits are magnitude of the error, the

magnitude of the computed error is decoded to generate the control word for the barrel shifter. The logic used for the generation of control word for the barrel shifter is shown in Fig. 9. The number of shifts in that case is increased by locations accordingly to reduce the hardware complexity. The weight Increment unit [shown in Fig. 8] for comprises of 4 barrel shifters and four carry save adder cells. The barrel shifter shifts the different input values for $= 0,1,2,\dots$, by appropriate number of locations. The barrel shifter yields the desired increments are fed to the carry save adder with the sign bit from the error value. The sign bit of the error is used as the control for the 2:1 MUX to select any one of the sum or carry output from the Carry save adder. The output of the MUX is fed to the Byte- parallel to Bit-serial converter to convert 8 bit data into 1 bit data. The output waveform of DA-based adaptive FIR filter (N=16) as shown in Fig. 10.

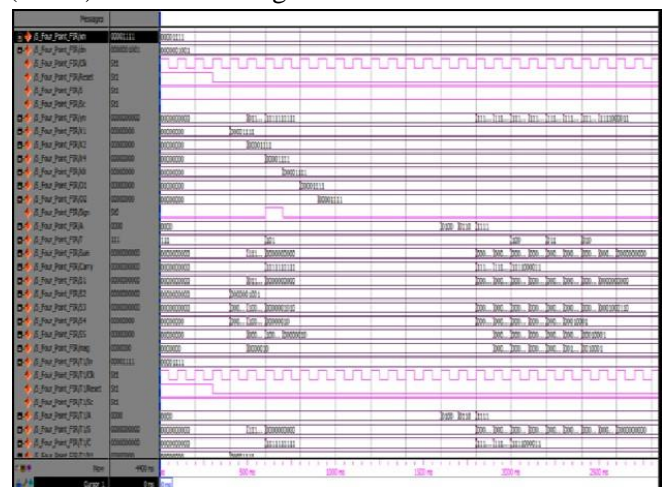


Figure 10: DA-based LMS adaptive FIR filter of length N=16

V. RESULTS AND DISCUSSIONS

In this chapter, the expected outputs of the proposed design are presented. All the individual units i.e., Distributed Arithmetic table, Carry Save Accumulation adder, Four Point Inner Product block and Weight increment block were implemented using Xilinx and their results (RTL schematics and Test bench waveforms) were observed.

XILINX SIMULATIONS

These individual units were used to design a small order LMS based Adaptive filter using DA, which was also implemented in the Xilinx and the corresponding test bench waveform was observed. The filter weights as well as the error value (difference between filter output and the desired value) was updated for each cycle.

1. DA TABLE:

It consists of 15 registers to store pre computed sums of input words.

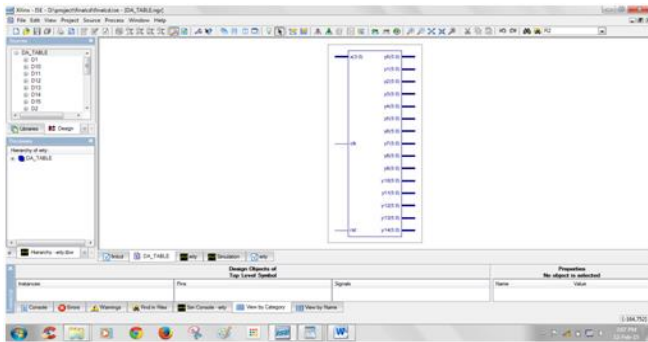


Figure 6.1 Block diagram of DA Table

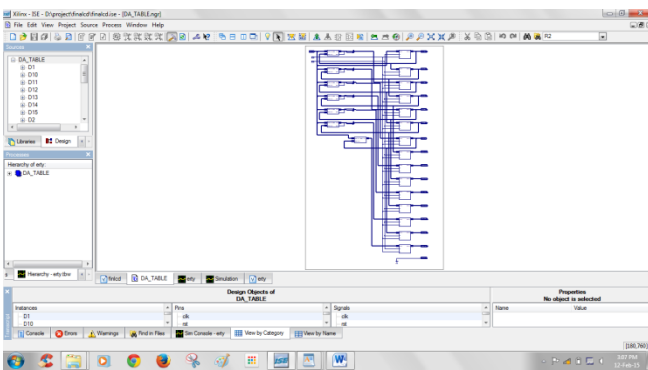


Figure 6.2 RTL Schematic of DA Table

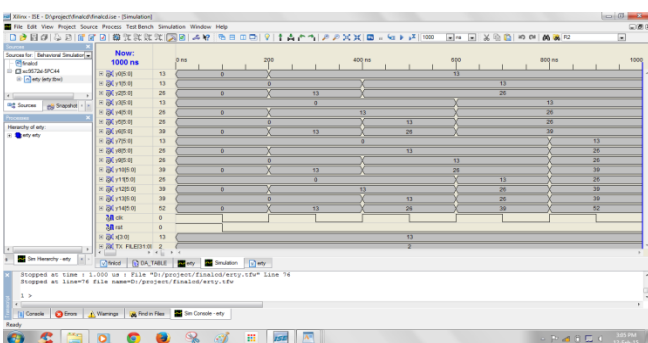


Figure 6.3 Test Bench waveform of DA Table

2. CARRY SAVE ACCUMULATION ADDER

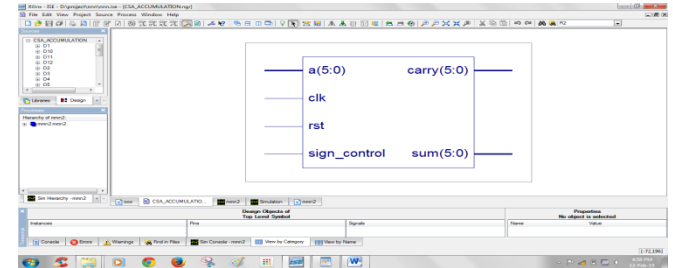


Figure 6.4 Block diagram of Carry Save Accumulation adder

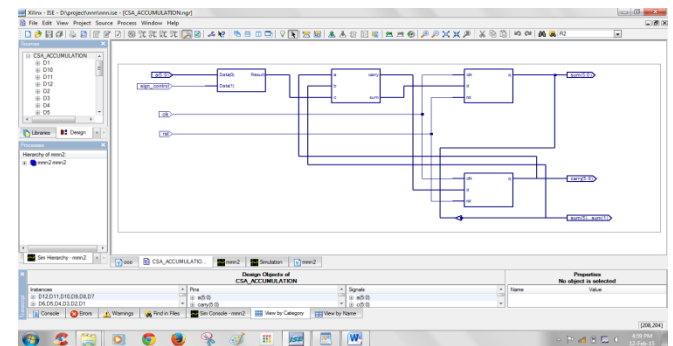


Figure 6.5 RTL schematic of Carry Save Accumulation adder

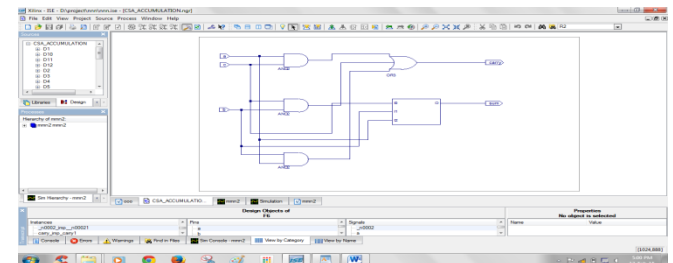


Figure 6.6 RTL schematic of Ex-OR gate

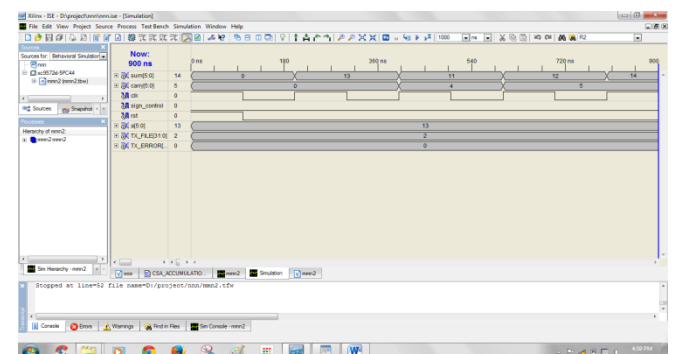


Figure 6.7 Test bench waveform of Carry Save Accumulation adder

3. FOUR POINT INNER PRODUCT BLOCK:

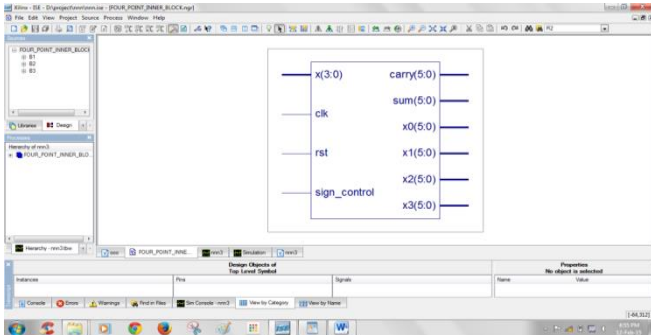


Figure 6.8 Block diagram of Four Point Inner Product Block

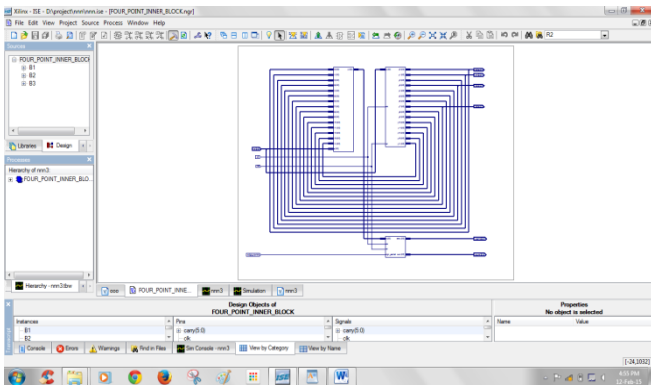


Figure 6.9 RTL schematic of Four Point Inner Product Block

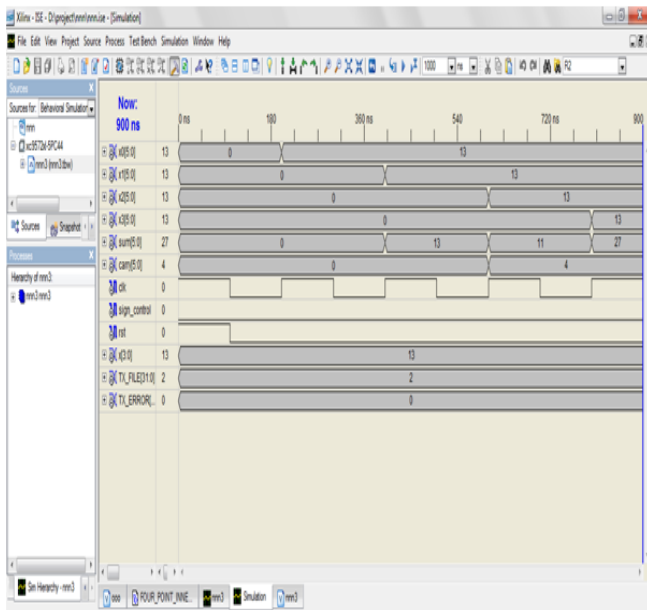


Figure 6.10 Test bench waveform of Four Point Inner Product Block

3. FOUR BIT LMS BASED ADAPTIVE FIR FILTER:

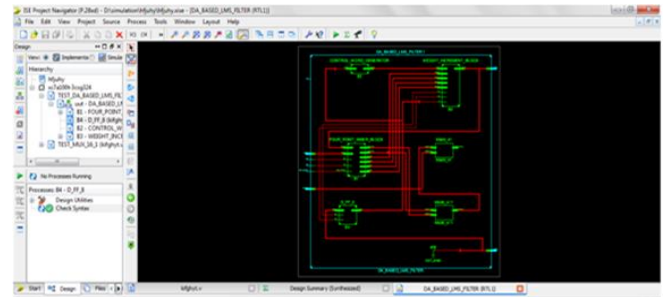


Figure 6.11 RTL schematic of four bit LMS based Adaptive FIR Filter

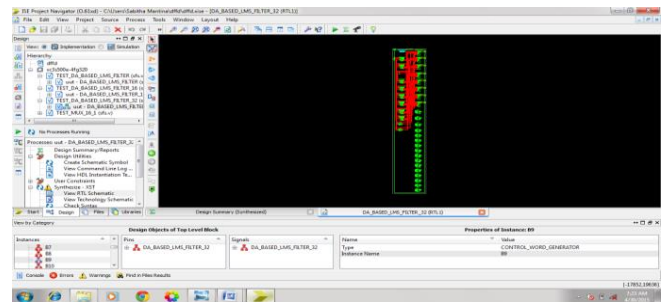


Figure 6.12 RTL schematic of 32-bit LMS based Adaptive FIR Filter

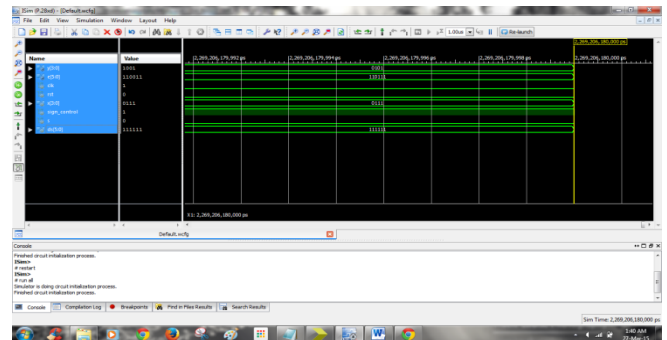


Figure 6.13 Test bench waveform-1 of 4-bit adaptive filter

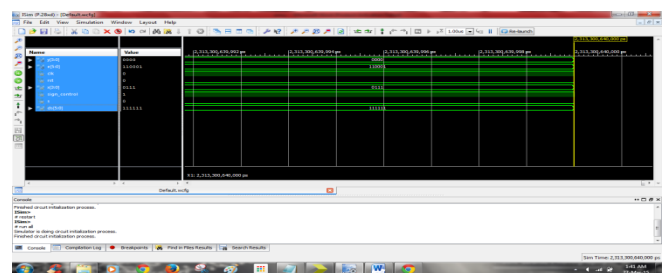


Figure 6.14 Test bench waveform-2 of 4-bit adaptive filter

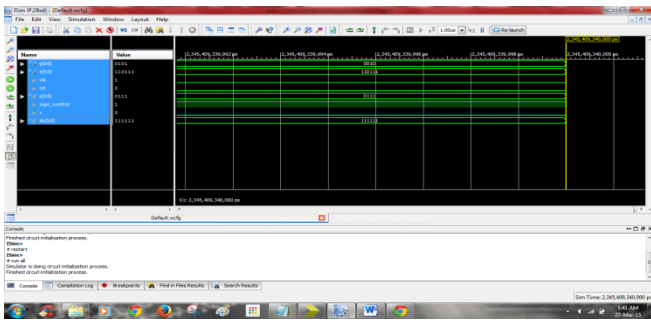


Figure 6.15 Test bench waveform-3 of 4-bit adaptive filter

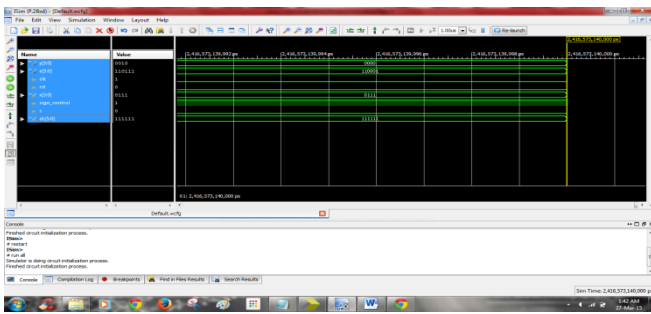


Figure 6.16 Test bench waveform-4 of 4-bit adaptive filter

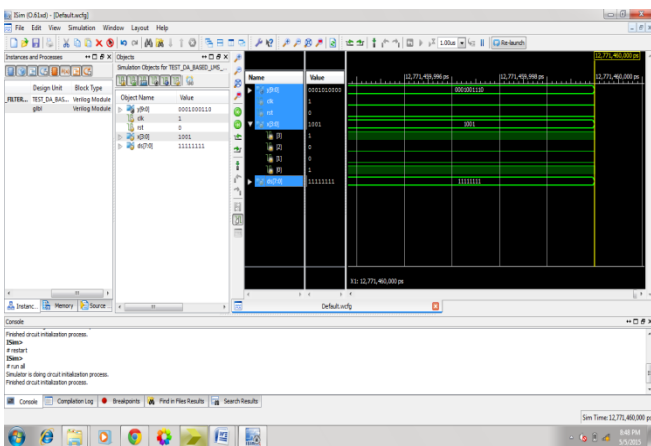


Figure.6.17.test bench waveform of 32-bit Adaptive FIR Filter

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	988	960	102%
Number of Slice Flip Flops	1287	1920	67%
Number of 4 input LUTs	1165	1920	60%
Number of bonded IOBs	36	108	33%
Number of GCCLKs	1	24	4%

Table .6.1.device utilization

From the above table we observe that number of slices used by the proposed system is 988, and number of slice flip-flops are 1287, and number of 4 input LUT's are 1165, 60% is utilized.

VI. Conclusion

In this script, an adaptive FIR filter using distributed arithmetic (DA) for area efficient design is implemented. High throughput is drastically enriched by parallel (LUTs) update and equivalent implementation of filtering and weight-update operations. The proposed carry save accumulation using 10 transistor full adder schemes of signed partial inner products for the computation of the filter output and also modified in weight increment block. By this way it utilizes low area, low power consumption and the throughput of the filter rates increases irrespective of the filter length.

REFERENCES

- [1] Sang Yoon Park, Member, IEEE, and Pramod Kumar Meher, Senior Member, IEEE "Low-Power, High-Throughput, and Low-Area Adaptive FIR Filter Based on Distributed Arithmetic" IEEE Transactions On Circuits And Systems—II: Express Briefs, Vol. 60, No. 6 June 2013.
- [2] 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.
- [3] R. Guo and L. S. DeBrunner, "Two high-performance adaptive filter implementation schemes using distributed arithmetic," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 58, no. 9, pp. 600–604, Sep. 2011.
- [4] R. Guo and L. S. DeBrunner, "A novel adaptive filter implementation scheme using distributed arithmetic," in Proc. Asilomar Conf. Signals, Syst., Comput., , pp. 160–164, Nov. 2011.



[5] S. Haykin and B. Widrow “Least-Mean-Square Adaptive Filters” Hoboken, NJ, USA: Wiley, 2003.

[6] P. K. Meher and S. Y. Park, “High-throughput pipelined realization of adaptive FIR filter based on distributed arithmetic,” in VLSI Symp. Tech. Dig., Oct. 2011, pp. 428–433.

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

[8] S. A. White “Applications of the distributed arithmetic to digital signal processing: A tutorial review,” IEEE ASSP Mag., vol. 6, no. 3, pp. 4–19, Jul. 1989.

[9] Hanan A.Mahmoud and Magdy.A. Bayoumi, “ A 10-Transistor Low Power High Speed Full Adder Cell”.