International Journal of Computational Intelligence and Informatics, Vol. 3: No. 2, July - September 2013

Memory Based FIR Filter Design on FPGA using Distributed Arithmetic and OBC Coding Technique

M Mahalingam
Electronics and Communication Engineering
Kumaraguru College of Technology
Coimbatore, India
ammahalingam@gmail.com

S Govindaraju
Electronics and Communication Engineering
Kumaraguru College of Technology
Coimbatore, India
govindaraju.s.ece@kct.ac.in

***Abstract*-This paper provides an efficient implementation of FIR filter without using multipliers. Area complexity in an algorithm of finite impulse response (FIR) filter is mainly caused by multipliers. Among the multiplierless implementation of FIR filter, Distributed Arithmetic (DA) is most efficient technique. In Distributed Arithmetic inner products are precomputed and stored in Look Up Table (LUT), than this precomputed values are added and shifted with number of times equal to the precision of input samples. If filter order increases than Look Up Table size also increases in its basic structure, makes it inefficient for many applications. In order to eliminate exponential growth of LUT with the order of filter we use memory partitioning (slicing) technique. We presented 16-tap FIR filter, with different size of memory partitioning of LUT and combine with OBC Coding. Implementation and synthesis result shows drastic improvement in performance in terms of speed as well as saving in area, with more number of slices.**

Keywords-Finite Impulse Response, multiplierless, distributed arithmetic, Field Programmable Gate Array, OBC Coding

I. INTRODUCTION

Finite Impulse Response (FIR) filters have played a central role in digital signal processing because of its advantages. FIR filter can implement linear-phase filtering. This means that the filter has no phase shift across the frequency band. Alternately, the phase can be corrected independently of the amplitude and it can be used to correct frequency –response errors in a loudspeaker to a finer degree of precision than using Infinite Impulse Response filters. FIR has desirable numeric properties. In practice, all DSP filters must be implemented using finite-precision arithmetic, that is, a limited number of bits. The use of finite-precision arithmetic in IIR filters can cause significant problems due to the use of feedback, but FIR filters without feedback can usually be implemented using fewer bits, and the designer has fewer practical problems to solve related to non-ideal arithmetic, they can be implemented using fractional arithmetic. Unlike IIR filters, it is always possible to implement a FIR filter using coefficients with magnitude of less than 1.0. (The overall gain of the FIR filter can be adjusted at its output, if desired.) .

This is an important consideration when using fixed-point DSP's, because it makes the implementation much simpler. In general FIR filter is characterized by

$$y = \sum_{n=1}^{k} \mathrm{A_n X_n} \quad (1)$$

Equation (1) shows that, large number of multiplication involved in implementing FIR filter. Multiplier caused large delay and area in VLSI implementation. So most of the researcher doing research in multiplierless implementation. In multipliered FIR filters area is reduced by means of sharing of multipliers or by manipulating the coefficients so we can reduce the number of multiplication. Distributed Arithmetic Technique and Constant Coefficient Multiplier comes under multiplierless implementation. Computation Sharing Differential Coefficient (CSDC) method, which can be used to obtain low-complexity multiplierless implementation of finite-impulse response (FIR) filters[1], this method is applicable to signal processing tasks involving multiplications with a set of constants. Look Up Table optimization for memory-based computation which gives idea about reduction of memory requirement based on antisymmetric product coding (APC) and Odd Multiple Storage (OMS) techniques [2]. Partial LUT Size Analysis in Distributed Arithmetic FIR Filters on FPGAs which can be used to reduce memory requirement by means of partial LUT slicing concept[3] Moreover, the OMS scheme in [2] does not provide an efficient implementation when combined with the APC technique. In this brief, we therefore present a slicing concept combined with an OBC coding for efficient memory based multiplication. This paper is organized as follows.

The review of basic Distributed Arithmetic Technique is given in Section II and in Section III architecture of Distributed Arithmetic Technique is presented, and also implementation steps based on memory partitioning is given. Section VI gives implementation of distributed arithmetic architecture using offset binary coding. Section V gives DA architecture implementation combine with LUT slicing and offset binary coding and Section VI gives area utilization and performance of the proposed DA architecture. Its comparison with previous work is also presented. At the last conclusions are given in section VII.

II. DISTRIBUTED ARITHMETIC

Distributed arithmetic is an efficient procedure for computing inner products between a fixed and a variable data vector. The basic principle is owed to Croisier et al. (Patent), and Peled and Liu have independently presented a similar method. Distributed Arithmetic is bit-serial in nature. It can therefore appear to be slow. When the number of elements in a vector is nearly the same as the word size, than DA is quite fast. Area savings from using DA can be up to 80% in DSP hardware designs.

Consider general FIR filter equation from (1), Where the coefficients An, n= 1, 2, 3 . . . m are fixed. A two's-complement representation is used for the data components which are scaled so that |xn |≤ 1. Let xn be an N-bit scaled two's complement number. In other words,

$$X_n = -X_{n0} + \sum_{m=1}^{N-1} X_{nm} 2^{-m} \tag{2}$$

The inner product can be rewritten

$$y = \sum_{n=1}^{k} A_n \left[-X_{n0} + \sum_{m=1}^{N-1} X_{nm}\, 2^{-m}\right] \tag{3}$$

Where $X_{nm}$is the *m*th bit in ***X****n*, By interchanging order of two summations we get

$$y = -\sum_{n=1}^{k} A_n X_{n0} + \sum_{m=1}^{N-1}\left[\sum_{n=1}^{k} A_n X_{nm}\right] 2^{-m} \tag{4}$$

Which can be written as

$$y = -F_0(X_{10}, X_{20}, \ldots., X_{K0}) \quad + \sum_{m=1}^{N-1} F_m\,(X_{1m}, X_{2m}, \ldots., X_{km}) 2^{-m} \tag{5}$$

From (5) we can observe that the inner product take one of the possible 2k values given that X€{0,1}, and this 2k values are correspond to all possible sum combinations of filter coefficient. These values are precomputed computed and stored in memories, addressed by Xnm thus; the multiplier required for MAC algorithm of FIR filter is eliminated by means of LUT access and summations. Analysis shows that; the direct implementation of filter from (1); the number of MAC units increases with increase in the filter order so this will cause more delay and area consumption, whereas in DA architecture based hardware in critical path is decoupled from the order of filter. Hence this architecture is most preferred one for implementing algorithm in Field Programmable Gate Array.

III. DISTRIBUTED ARTHIMETIC ARCHITECTURE

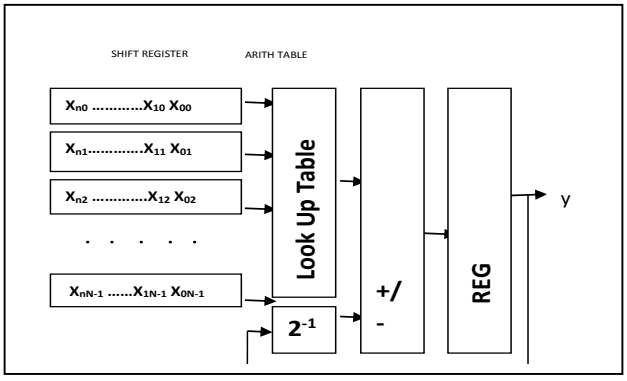

Figure 1. Basic architecture of distributed arithmetic

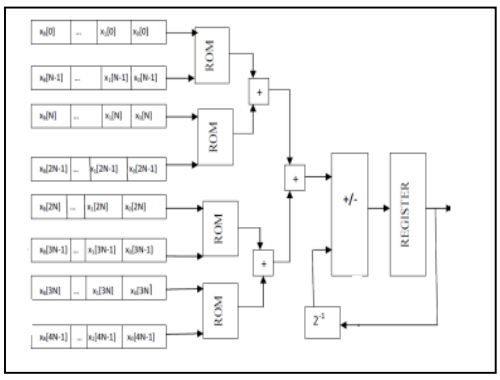

Figure 2. DA Architecture with Partition 4

In basic architecture of Distributed Arithmetic (Figure 1), the size of Look up Table increases rapidly with the order of filter.

To avoid this drawback the main idea followed by this research is slicing of Look Up Table into desired number. By this technique we reduce the size of memory, with small increase in area requirement due to adders. For example if filter order is 16 then 65536 memory locations are needed for implementing filter without slicing concept. With LUT slicing by a factor four reduces the memory locations to 64. This sliced Look Up Table architecture on FPGA have Registers, sliced Look Up Table units and the accumulator unit.

In Look up Table slicing concept large Look Up Table is divided into small size Look Up Table, then partial product is calculated by means of adding the result of small size

I. Register

Input samples Xn of data width N stored in input register (Figure 3). Input samples are given in parallel form so we need to convert these parallel samples into serial form in order to get address for LUT. these parallel formed input samples are converted into serial form advanced to right for every clock, so as we create an address of Look Up Table.

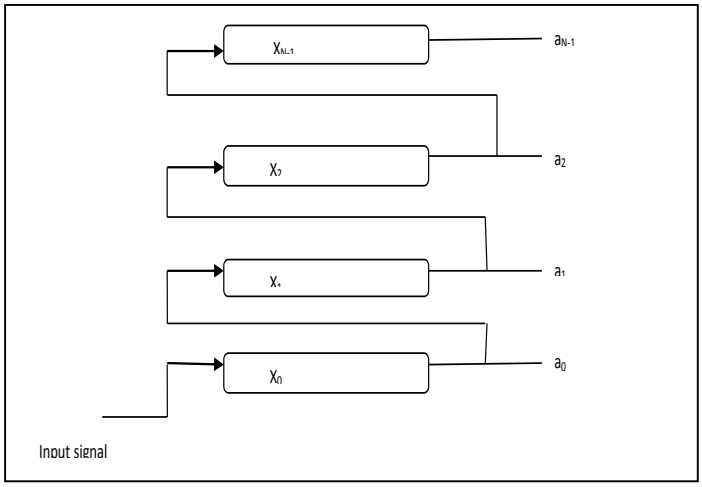

Figure 3. Register

II. LUT Slicing

Distributed Arithmetic is efficient only when filter order is low. If filter order is high then LUT size will increases rapidly, for example for 16-tab filter 65536 locations are needed for implementation. This reduces the performance of the system. So, we need to reduce the memory requirement to acceptable level in order to increase the system performance. Structure of LUT without slicing is shown in Figure 4. To reduce the size of LUT we subdivided the large LUT into a number of small size LUTs, called LUT partitioning. Each partitioned LUT operates on a different set of filter coefficient. LUT with partitioning is shown in Figure 5.

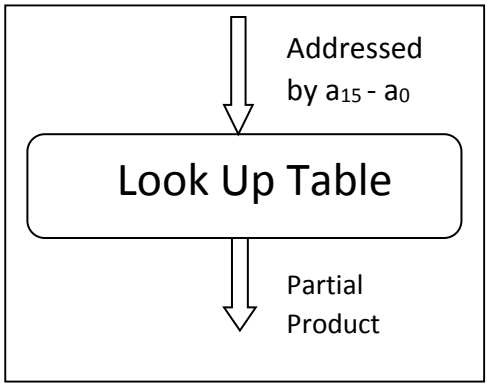

Figure 4. LUT Structure

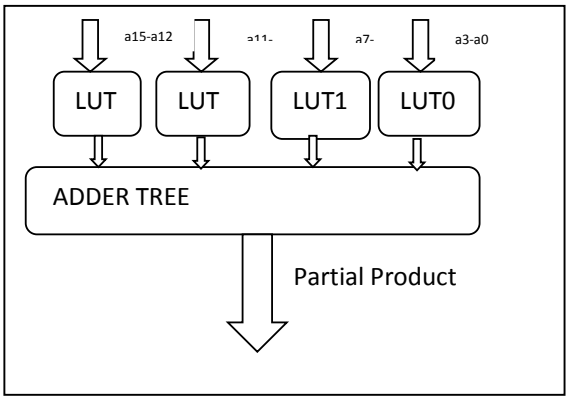

Figure 5. LUT with partitioning

In this work analysis of 16-tab FIR filter is carried out on various size of partitioning. Details of LUT with partitioning by a factor 4 are shown in Figure 5. Partial product term can be calculated by adding the output of all slices by adder tree. Further, by taking the number of accumulation and shift operation, final output is calculated.

a_3	a_2	a_1	a_0	Data
0	0	0	0	0
0	0	0	1	A_0
0	0	1	0	A_1
0	0	1	1	A_0+A_1
0	1	0	0	A_2
0	1	0	1	A_2+A_0
0	1	1	0	A_2+A_1
0	1	1	1	$A_2+A_1+A_0$
1	0	0	0	A_3
1	0	0	1	A_3+A_0
1	0	1	0	A_3+A_1
1	0	1	1	$A_3+A_1+A_0$
1	1	0	0	A_3+A_2
1	1	0	1	$A_3+A_2+A_0$
1	1	1	0	$A_3+A_2+A_1$
1	1	1	1	$A_3+A_2+A_1+A_0$

Figure 6. 2^4 Word LUT of Data

III. Accumulator and Shifter Unit

This phase consists of an accumulator and shifter. The partial product is generated by adding outputs of LUTs. Partial product generated by LUTs is added and shifted in every iteration. Number of iteration is equal to the input precision.

IV. Control Unit

Control unit used to control circuit components behaviour and the whole circuit behaviour. In Distributed Arithmetic control unit is counter whose upper limit depends basically on the input precision and defines the circuit throughput. Compare with other methods, an advantage of distributed Arithmetic is that the throughput in DA-based architectures is independent of the order of the filter.

IV. DISTRIBUTED ARITHMETIC USING OFFSET BINARY CODING

The memory size may be reduced from LUT slicing concept. Again memory size may be halved to 1/2(2n) with the help of LUT slicing combine with offset binary coding. in order to understand how memory size will reduced by offset binary coding, we interpret the input data as being cast not in a (0.1) straight binary code, but instead as being cast in (-1, 1) offset binary code. if we represent input as

$$Xn=1/2\ [\ Xn-(-Xn)\ ] \tag{6}$$

-Xn can be represent as

$$-\mathrm{Xn} = \overline{-\mathrm{b}_{\mathrm{n}(\mathrm{N}-1)}}2^{(\mathrm{N}-1)} + \sum_{\mathrm{n}=1}^{\mathrm{N}-2}\overline{-\mathrm{b}_{\mathrm{nm}}}2^{\mathrm{n}} + 1 \tag{7}$$

Where the overscore symbol indicates the negation of bit. From (6) and (7)

$$\mathrm{X_n} = 1/2\left[-\left(\mathrm{b}_{\mathrm{n}(\mathrm{N}-1)} - \overline{\mathrm{b}_{\mathrm{n}(\mathrm{N}-1)}}\right)2^{(\mathrm{N}-1)} + \sum_{\mathrm{m}=1}^{\mathrm{N}-1}\left(\mathrm{b}_{\mathrm{nm}} - \overline{\mathrm{b}_{\mathrm{nm}}}\right)2^{\mathrm{n}} - 1\right] \tag{8}$$

For simplification

$$\mathrm{c}_{\mathrm{nm}} = \mathrm{b}_{\mathrm{nm}} - \overline{\mathrm{b}_{\mathrm{nm}}} \qquad \mathrm{n} \neq \mathrm{N}-1 \tag{9}$$

$$c_{n(N-1)} = b_{n(N-1)} - \overline{b_{n(N-1)}} \quad (10)$$

From (1) and (8)

$$y = {}^{1}/_{2} \sum_{n=1}^{k} A_n \left[-\left(b_{n(N-1)} - \overline{b_{n(N-1)}}\right) 2^{(N-1)} + \sum_{m=1}^{N-1} \left(b_{nm} - \overline{b_{nm}}\right) 2^n - 1 \right] \quad (11)$$

$$y = \sum_{n=1}^{k} Q(b_n) + Q(0) \quad (12)$$

$$Q(b_n) = \sum_{n=1}^{k} \frac{A_n}{2} c_{nm} \quad 2^n \; n \neq N-1 \quad and \quad Q(0) = -\sum_{n=1}^{k} \frac{A_n}{2} \quad (13)$$

$$Q(b_{N-1}) = -\sum_{n=1}^{k} \frac{A_n}{2} c_{n(N-1)} \quad 2^{(N-1)} \qquad n = N-1 \quad (14)$$

Note that memory size will be reduced from 2^n to 2^{n-1}

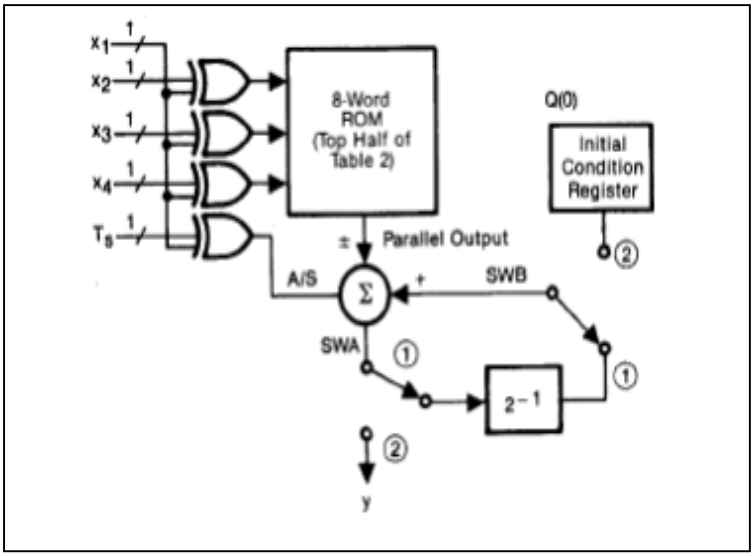

Figure 7. DA Architecture with offset binary coding

V. DISTRIBUTED ARCHITECTURE COMBINE WITH LUT SLICING AND OFFSET BINARY CODING

Input Code b_{1n}	b_{2n}	b_{3n}	b_{4n}	8-Word Memory Contents,Q
0	0	0	0	$-1/2(A_1+A_2+A_3+A_4)$
0	0	0	1	$-1/2(A_1+A_2+A_3-A_4)$
0	0	1	0	$-1/2(A_1+A_2-A_3+A_4)$
0	0	1	1	$-1/2(A_1+A_2-A_3-A_4)$
0	1	0	0	$-1/2(A_1-A_2+A_3+A_4)$
0	1	0	1	$-1/2(A_1-A_2+A_3-A_4)$
0	1	1	0	$-1/2(A_1-A_2-A_3+A_4)$
0	1	1	1	$-1/2(A_1-A_2-A_3-A_4)$
1	0	0	0	$1/2(A_1-A_2-A_3-A_4)$
1	0	0	1	$1/2(A_1-A_2-A_3+A_4)$
1	0	1	0	$1/2(A_1-A_2+A_3-A_4)$
1	0	1	1	$1/2(A_1-A_2+A_3+A_4)$
1	1	0	0	$1/2(A_1+A_2-A_3-A_4)$
1	1	0	1	$1/2(A_1+A_2-A_3+A_4)$
1	1	1	0	$1/2(A_1+A_2+A_3-A_4)$
1	1	1	1	$1/2(A_1+A_2+A_3+A_4)$

If LUT is sliced into two then output is given by

$$y = \sum_{n=1}^{k/2} A_n X_n + \sum_{n=\frac{k}{2}+1}^{k} A_n X_n \quad (15)$$

From (14) and (15)

$$y = \sum_{n=1}^{k/2} Qb_n 2^{-n} + 2^{-(N-1)}Q(0) \; + \sum_{n=\frac{k}{2}+1}^{k} Qb_n 2^{-n} + 2^{-(N-1)}Q(0) \quad (16)$$

From above equations (5), (15) and (16) we can conclude that for k-tab filter memory requirements are 2k,2(2k/2) , 2(2(k/2)-1) respectively. For example if filter order is 8 than memory requirements are 256(without slicing), 32(sliced by 2), 16(slicing by 2 combine with OBC) locations respectively. Architecture combine with LUT slicing and offset binary coding is efficient in terms of memory as well as delay so proposed architecture is efficient compare to existing methods.

VI. Implementation Result And Performance Analysis

Xilinx Integrated Environment (ISE) is used for synthesis and implementation of a design. FIR filter is designed and implemented with fixed filter coefficient. In order to evaluate performance of the proposed scheme first filter is designed using MATLAB tool from this collect the filter coefficient and then coefficients are truncated and scaled with 8 bits of precision. The Magnitude and phase response of the design filter in MATLAB is shown in Figure 9.

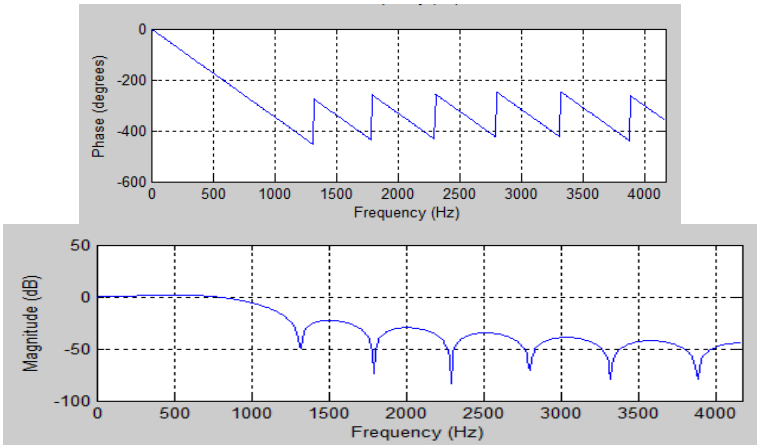

Figure 9. Magnitude and Phase Response of FIR Filter using MATLAB

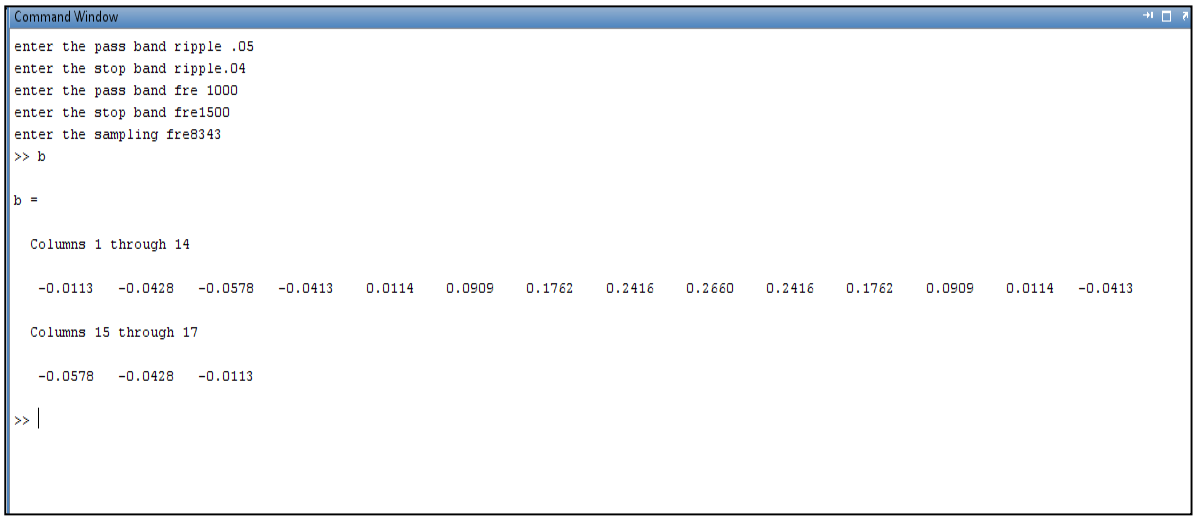

Figure 10. Coefficients calculation using MATLAB

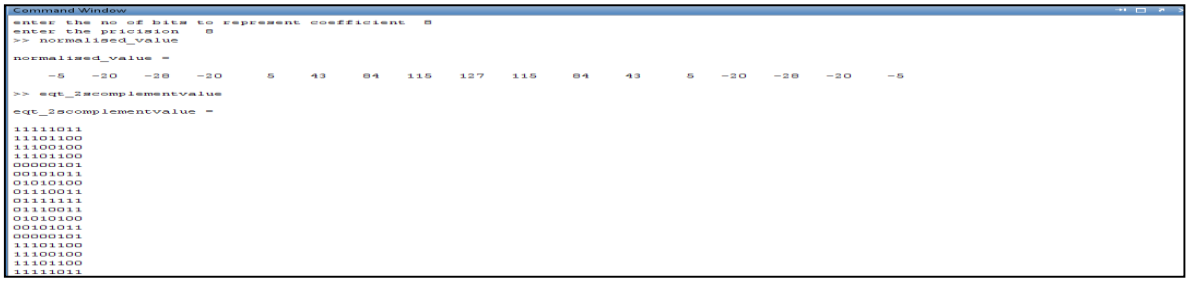

Figure 11. Normalised value of coefficients and its 2's complement equivalent

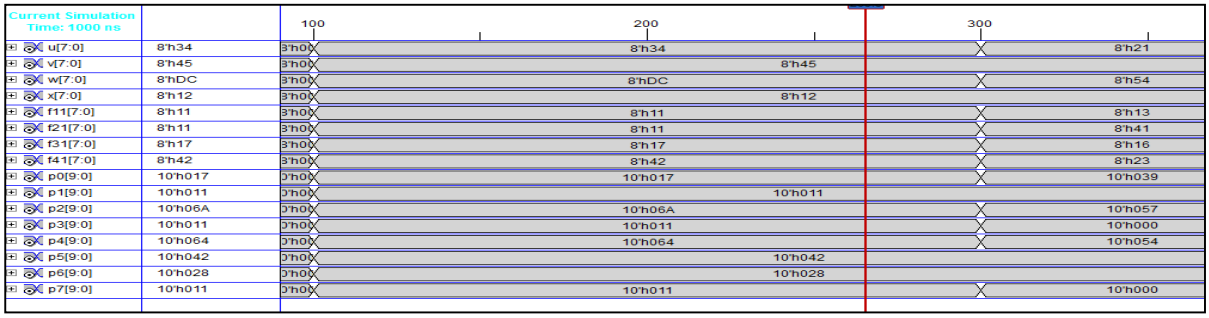

Figure 12. Simulation Result for Partial Product evaluation with address size of 4

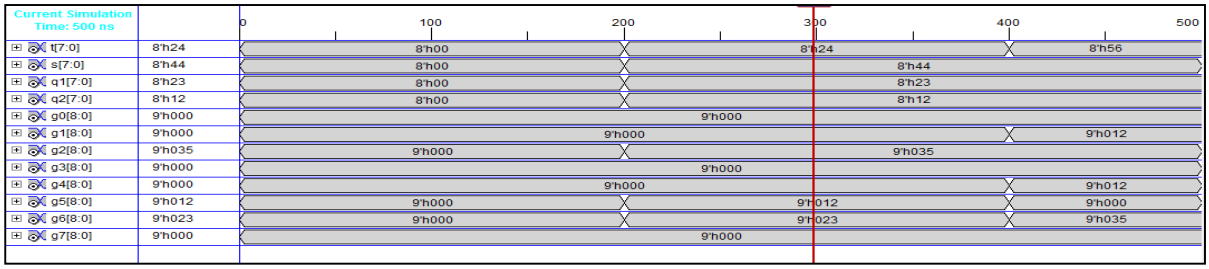

Figure 13. Simulation Result for Partial Product evaluation with address size of 2

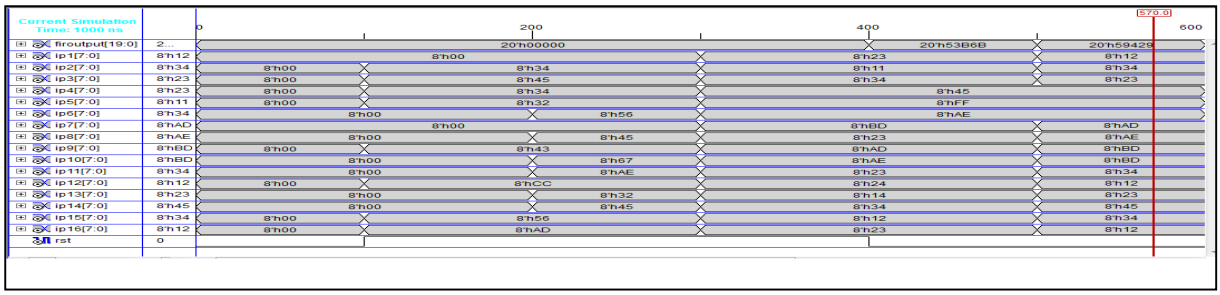

Figure 14. 16-tap FIR filter output

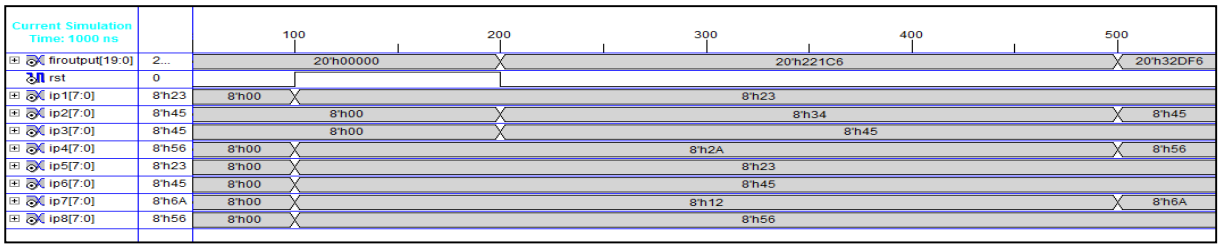

Figure 15. 8-tap FIR filter output

Table I: PERFORMANCE COMPARISONS OF FIR FILTER

Filter order	Parameter	Conventional FIR	FIR Without Partitioning	FIR With Partitioning Size of 4	FIR With Partitioning Size of 2	FIR Combine With Partitioning and OBC Coding
16	Slices	700	558	321	296	**280**
	Delay(ns)	46.02	28.85	24.45	22.67	**20.73**
	Power (mw)	85.48	46.08	37.45	32.45	**28.62**
	Memory(Kb)	-	245048	209622	198632	**193654**
8	Slices	365	282	171	162	**158**
	Delay	28.32	22.47	19.83	18.26	**16.86**
	Power (mw)	52.34	31.03	27.24	25.53	**23.40**
	Memory(Kb)	-	213462	195752	185473	**180214**

VII. CONCLUSION

Distributed Arithmetic Architecture combine with LUT slicing and OBC coding has proved to be an efficient technique for FIR filter implementation. Because of its highly flexible nature of this structure, allow it to use in complete serial to full parallel form and also Distributed arithmetic architecture combine with Look Up Table slicing and OBC coding decreases the memory requirements for FIR implementation compared with distributed arithmetic architecture without LUT slicing. Compared with conventional FIR implementation (multiplier based DA architecture combine with LUT slicing and OBC coding reduces the delay present in a system .

REFERENCES

[1] M. Kumm, K. M¨oller, and P. Zipf, "Partial LUT Size Analysis in Distributed Arithmetic FIR Filters on FPGAs", in Circuits and Systems, IEEE Int. Sym. on (ISCAS), 2013.

[2] Pramod Kumar Meher, SeniorMember,IEEE, "LUT Optimization for Memory Based Computation", IEEE Transactions on circuits and Systems, Vol. 54, No. 4, pp.245-252, April 2010.

[3] S. Mirzaei, R. Kastner, and A. Hosangadi, "Layout aware optimization of high speed fixed coefficient FIR filters for FPGAs", Int. Journal of Reconfigurable Computing, Vol. 3, pp. 1–17, Jan 2010.

[4] S. Mirzaei, A. Hosangadi, and R. Kastner, "FPGA Implementation of High Speed FIR Filters Using Add and Shift Method", IEEE, International Conference on Computer Design (ICCD), pp. 308-313, April 2010.

[5] J. Park, K. Muhammad, and K. Roy, "High-Erformance Fir Filter Design Based On Sharing Multiplication", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 11, No. 2, April 2003, pp.244-253.

[6] M. Yamada, and A. Nishihara, "FIR Digital Filter with CSD Coefficients Implement on FPGA", in Proceedings of IEEE Design Automation Conference, pp. 7-8, 2001.