Batchu Jeevanarani, Thota Sreenivas / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 5, September- October 2012, pp.1003-1009

Memory-Based Realization of FIR Digital Filter by Look-Up-Table Optimization

Batchu Jeevanarani and Thota Sreenivas
Department of ECE, Sri Vasavi Engg College,
Tadepalligudem, West Godavari (DT), Andhra Pradesh, India

Abstract—

Finite impulse response (FIR) digital filter is widely used in signal processing and image processing applications. Distributed arithmetic (DA)-based computation is popular for its potential for efficient memory-based implementation of finite impulse response (FIR) filter where the filter outputs are computed as inner-product of input-sample vectors and filter-coefficient vector. In this paper, however ,we show that the look-up-table(LUT)-multiplier-based approach, where the memory elements store all the possible values of products of the filter coefficients could be an area-efficient alternative to DA-based design of FIR filter with the same throughput of implementation.

***Keywords*—Distributed arithmetic, FIR filter, throughput, Memory-based implementation, look-up-table, LUT-multiplier-based approach.**

I. INTRODUCTION

Finite-impulse-response (FIR) filters are basic processing elements in applications such as video signal processing and audio signal processing. The order of an FIR filter primarily determines the width of the transition-band, such that the higher the filter order, the sharper is the transition between a pass-band and adjacent stop-band. Many applications in digital communication (channel equalization, frequency channelization), speech processing (adaptive noise cancelation), seismic signal processing (noise elimination), and several other areas of signal processing require large order FIR filters. Since the number of multiply-accumulate (MAC) operations required per filter output increases linearly with the filter order, real-time implementation of these filters of large orders is a challenging task.

Along with the progressive device scaling, semiconductor memory has become cheaper, faster, and more power-efficient. Moreover, according to the projections of the international technology roadmap for semiconductors, embedded memories will have dominating presence in the system- on-chips (SoCs), which may exceed 90%, of the total Soc content. It has also been found that the transistor packing density of memory components are not only higher but also increasing much faster than those of logic components. Apart from that, memory-based computing structures are more regular than the multiply–accumulate structures and offer many other advantages, e.g., greater Potential for high-throughput and reduced-latency implementation, (since the memory-access-time is much shorter than the usual multiplication-time) and are expected to have less dynamic power consumption due to less switching activities for memory-read operations compared to the conventional multipliers .Memory-based structures are well-suited for many digital signal processing (DSP) algorithms, which involve multiplication with a fixed set of coefficients.

There are two basic variants of memory-based techniques. one of them is based on distributed arithmetic (DA)for inner- product computation and the other is based on the computation of multiplication by look-up-table (LUT).In the LUT-multiplier-based approach, multiplications of input values with a fixed-coefficient are performed by an LUT consisting of all possible pre-computed product values corresponding to all possible values of input multiplicand, while in the DA-based approach, an LUT is used to store all possible values of inner-products of a fixed N –point vector with any possible N –point bit-vector. If the inner-products are implemented in a straight-forward way, the memory-size of LUT-multiplier-based implementation increases exponentially with the word-length of input values, while that of the DA-based approach increases exponentially with the inner-product-length. Attempts have been made to reduce the memory-space in DA-based architectures using offset binary coding (OBC) and group distributed technique. A decomposition scheme is used for reducing the memory-size of DA-based implementation of FIR filter. But, it is observed that the reduction of memory-size achieved by such decompositions is accompanied by increase in latency as well as the number of adders and latches.

II. LUT DESIGN FOR MEMORY BASED MULTIPLICATION

The basic principle of memory-based multiplication is depicted in Fig. 1. Let A be a fixed coefficient and X be an input word to be multiplied with A.

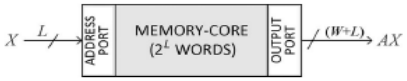

Fig: 1. Conventional LUT-based multiplier

If we assume X to be an unsigned binary number of word-length L, there can be 2^Lpossible values of X, and accordingly, there can be 2^Lpossible values of product C=A.X. Therefore, for the conventional implementation of memory-based multiplication, a memory unit of 2^Lwords is required to be used as look-up-table consisting of pre-computed product values corresponding to all possible values of X. The product-word (A.X_i),for $0 \leq X_i \leq 2^L-1$, is stored at the memory location whose address is the Same as the binary value of X_i, such that if L-bit binary value of $X_i$is used as address for the memory-unit, then the corresponding Product value is read-out from the memory.

TABLE1 LUT WORDS AND PRODUCT VALUES FOR INPUT WORD LENGTH L=*4*

address $d_2d_1d_0$	word symbol	stored value	input $x_3x_2x_1x_0$	product value	# of shifts	control s_1 s_0
0 0 0	$P0$	A	0 0 0 1	A	0	0 0
			0 0 1 0	$2^1 \times A$	1	0 1
			0 1 0 0	$2^2 \times A$	2	1 0
			1 0 0 0	$2^3 \times A$	3	1 1
0 0 1	$P1$	$3A$	0 0 1 1	$3A$	0	0 0
			0 1 1 0	$2^1 \times 3A$	1	0 1
			1 1 0 0	$2^2 \times 3A$	2	1 0
0 1 0	$P2$	$5A$	0 1 0 1	$5A$	0	0 0
			1 0 1 0	$2^1 \times 5A$	1	0 1
0 1 1	$P3$	$7A$	0 1 1 1	$7A$	0	0 0
			1 1 1 0	$2^1 \times 7A$	1	0 1
1 0 0	$P4$	$9A$	1 0 0 1	$9A$	0	0 0
1 0 1	$P5$	$11A$	1 0 1 1	$11A$	0	0 0
1 1 0	$P6$	$13A$	1 1 0 1	$13A$	0	0 0
1 1 1	$P7$	$15A$	1 1 1 1	$15A$	0	0 0

s_0 and s_1 are control bits of the logarithmic barrel-shifter.

Although 2^L possible values of X correspond to 2^L possible Values of C=A.X, recently we have shown that only $(2^L/2)$ words corresponding to the odd multiples of A may only be Stored in the LUT .One of the possible product words is Zero, while all the rest $(2^L/2)-1$ are even multiples of A which could be derived by left-shift operations of one of the odd multiples of A. We illustrate this in Table I for L=4. At eight memory locations, eight odd multiples A×(2i+1) are stored as $P_i$for i=0,1,2,…,7.The even multiples 2A,4A and 8Aare derived by left-shift operations of A . Similarly, 6A and 12A are derived by left-shifting 3A, while 10A and 14A are derived by left-shifting 5A and 7A, respectively. The address X= (0000) corresponds to (A.X) =0, which can be obtained by resetting the LUT output.For an input multiplicand of word-size L similarly, only $(2^L/2)$ odd multiple values need to be stored in the memory-core of the LUT, while the other $(2^L/2-1)$ non-zero values could be derived by left-shift operations of the stored values. Based on the above, an LUT for the multiplication of an L -bit input with W-bit coefficient is designed by the following strategy:

- A memory-unit of $(2^L/L)$ words of (W+L)-bit width is used to store all the odd multiples of A.
- A barrel-shifter for producing a maximum of (L–1) left Shifts are used to derive all the even Multiples of A.
- The L-bit input word is mapped to (L–1) -bit LUT- Address by an encoder.
- The control-bits for the barrel-shifter are derived by a control-circuit to perform the necessary shifts of the LUT output. Besides, a RESET signal is generated by the same control circuit to reset the LUT output when X=0.

III.Proposed LUT-Based Multiplier for 4-Bit Input

The proposed LUT-based multiplier for input word-size L=4 is shown in Fig. 2. It consists of a memory-array of eight words of (W+4) -bit width and a 3-to-8 line address decoder, along with a NOR-cell, a barrel-shifter, a 4-to-3 bit encoder to map the 4-bit input operand to 3-bit LUT-address, and a control circuit for generating the control-word (s_0s_1) for the barrel-shifter, and the RESET signal for the NOR-cell.

The 4-to-3 bit input encoder is shown in Fig. 2(b). It receives a four-bit input word (x3x2x1x0) and maps that onto the three-bit address word (d2d1d0) according to the logical relations

$$d_0 = \overline{\overline{(x_0 \cdot x_1)} \cdot \overline{(x_1 \cdot x_2)} \cdot (x_0 + \overline{(x_2 \cdot x_3)}}$$
$$d_1 = \overline{\overline{(x_0 \cdot x_2)} \cdot (x_0 + \overline{(x_1 \cdot x_3)})}$$
$$d_2 = x_0 \cdot x_3$$

The pre-computed values of A×(2i+1) are stored as P_i for i=0,1,2,…,7 at 8 consecutive locations of the memory-array as specified in Table I in bit-inverted form. The decoder takes the 3-bit address from the input encoder, and generates 8 word-select signals, $\{w_i, \text{for } 0 \leq i \leq 7\}$, to select the referenced-word from the memory-array. The output of the memory-array is either AX or its sub-multiple in Bit-inverted form depending on the value of X. From Table I, we find that the LUT output is required to be shifted through 1location to left when the input operand is one of the values{(0010), (0110), (1010), (1110)}. Two left-shifts are required if X is either (0 1 0 0) or (1 1 0 0). Only when the input word X= (1000), three shifts are required. For all other possible input operands, no shifts are required. Since the maximum

number of left-shifts required on the stored-word is three, a two-stage logarithmic barrel-shifter is adequate to perform the necessary left-shift operations.

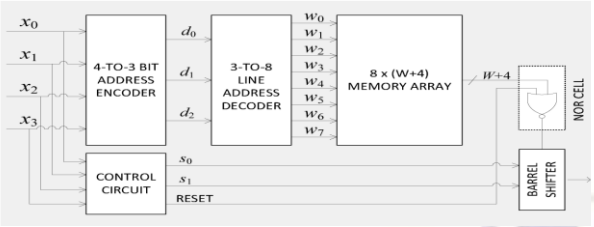

Fig: 2. (a). Proposed LUT design based 4-bit multiplier

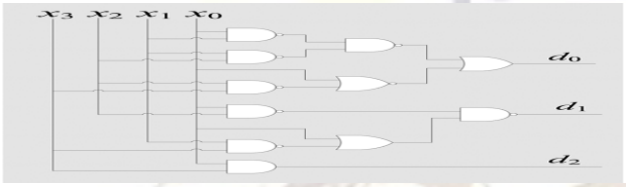

Fig: 2. (b). The 4-to-3 bits input encoder.

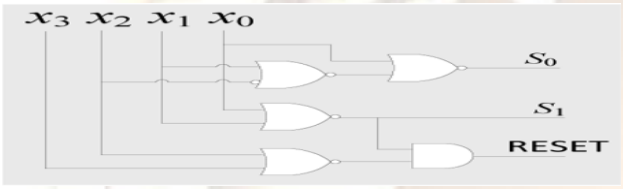

Fig: 2. (c). Control circuit

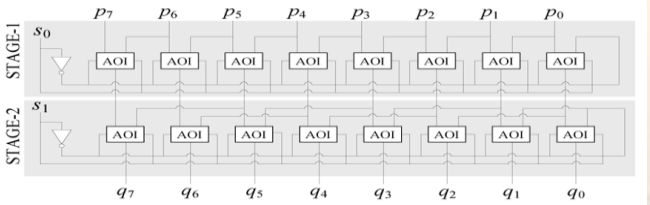

Fig: 2. (d). Two-stage logarithmic barrel-shifter for W=4

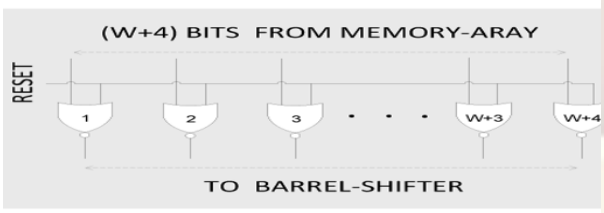

Fig: 2. (e). Structure of the NOR-cell

The number of shifts required to be performed on the output of the LUT and the control-bits S0 and S1 for different values of X are shown Table I. The control circuit [shown in Fig. 2(c)] accordingly generates the control-bits given by

$$s_0 = \overline{x_0 + \overline{(x_1 + \overline{x_2})}}$$
$$s_1 = \overline{(x_0 + x_1)}$$

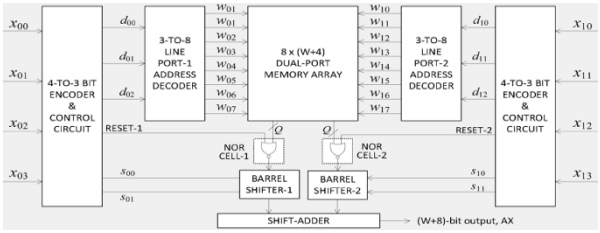

Fig: 3. Memory-based multiplier using dual-port Memory-array. Q= (W+4).

A logarithmic barrel-shifter for W=L=4 is shown in Fig. 2(d). It consists of two stages of 2-to-1 line bit-level multiplexers with inverted output, where each of the two stages involves (W+4) number of 2-input AND-OR-INVERT (AOI) gates .The control-bits $(s_0, \bar{s}_0)$ and $(s_1, \bar{s}_1)$ are fed to the AOI gates of stage-1 and stage-2 of the barrel-shifter, respectively. Since each stage of the AOI gates perform inverted multiplexing, after two stages of inverted multiplexing, outputs with desired number of shifts are produced by the barrel-shifter in (the usual) un-inverted form.

The input X= (0000) corresponds to multiplication by X=0 which results in the product value A.X=0 .Therefore, when the input operand word X= (0000), the output of the LUT is required to be reset. The reset function is implemented by a NOR-cell consisting of (W+ 4) NOR gates as shown in Fig. 2(e) using an active-high RESET. The RESET bit is fed as one of the inputs of all those NOR gates, and the other input lines of (W+4) NOR gates of NOR cell are fed with (W+4) bits of LUT output in parallel. When X= (0000), the control circuit in Fig. 2(c), generates an active-high RESET according to the logic expression:

$$\text{RESET} = \overline{(x_0 + x_1)} \cdot \overline{(x_2 + x_3)}$$

When RESET=1, the outputs of all the NOR gates become 0, so that the barrel-shifter is fed with (W+4) number of zeros. When RESET=0, the outputs of all the NOR gates become the complement of the LUT output-bits. Note that, keeping this in view, the product values are stored in the LUT in bit-inverted form. Reset function can be implemented by an array of 2-input AND gates in a straight-forward way, but the implementation of reset by the NOR-cell is preferable since the NOR gates have Simpler CMOS implementation compared with the AND gates. Moreover, instead of using a separate NOR-cell, the NOR gates could be integrated with memory-array if the LUT is implemented by a ROM .The NOR cells, therefore, could be eliminated by using a ROM of 9 words,

where the 9th word is zero and RESET is used as its word-select signal. Multiplication of an 8-bit input with a W-bit fixed coefficient can be performed through a pair of multiplications using a Dual-port memory of 8 words (or two single-port memory units) along with a pair of decoders, encoders, NOR cells and barrel-shifters as shown in Fig. 3. The shift-adder performs left-shift operation of the output of the barrel-shifter corresponding to more significant half of input by four bit-locations, and adds that to the output of the other barrel-shifter.

IV. MEMORY-BASED STRUCTURES FOR FIR FILTER USING LOOK-UP-TABLE MULTIPLIERS

We derive here the proposed structure for memory-based realization of an N -tap FIR filter, and discuss the design of memory cell to be used as LUT in the structure. The input-output relationship of an N -tap FIR filter in time-domain is given by

$$y(n) = h(0)\cdot x(n) + h(1)\cdot x(n-1) + h(2) \cdot x(n-2) + \cdots h(N-1)\cdot x(n-N+1)$$

Where h(n), for n=0,1,...,N–1, represent the filter coefficients, while x(n–i) , for i=0,1,…. ,N–1, represent N recent input samples, and y(n) represents the current output sample. Memory-based multipliers can be implemented for signed as well as unsigned operands. In case of signed operands, the input words and the stored product values need to be in two's complement representation. Since the stored product values require sign-extension in case of two's complement representation during shift-add operations, the LUT-based multiplication could have a simpler implementation when the multiplicands are unsigned numbers. Besides, without loss of generality, we can assume the input samples{x(n)} to be unsigned numbers and the filter coefficients {h(n)} to be signed numbers, in general. Keeping this in view, we write above equation alternatively as

$$y(n) = \text{sign}(0)\cdot|h(0)|\cdot x(n) + \text{sign}(1) \cdot|h(1)|\cdot x(n-1) + \cdots + \text{sign}(N- \cdot|h(N-1)|\cdot x(n-N+1)$$

Where $h(n) = \text{sign}(n)\cdot|h(n)|$, for $n = 0,1,\ldots,N-1$, $|h($ denotes the absolute value of h(n) and sign(n) = ±1 is the sign-factor, which could be absorbed with the additions of the corresponding term. The above equation, then may be written in a recursive form

$$y(n) = \text{sign}(0)\cdot|h(0)|\cdot x(n) + \text{sign}(1)\cdot \mathbf{D}(|h(1)| \cdot x(n) + \text{sign}(2)\cdot \mathbf{D}(|h(2)|\cdot x(n) + \cdots + \text{sign}(N-1)\cdot \mathbf{D}(|h(N-1)|\cdot x(n))\cdots))$$

D stands for the delay operator, such that D x(n–i)=x(n–i–1),for i=1,2,…,N–1.

A. Memory-Based FIR Filter Using Conventional LUT

The recursive computation of FIR filter output is represented by a transposed form data-flow graph (DFG) shown in Fig. 4. It consists of N multiplication nodes (M) and (N–1) add-subtract (AS) nodes.

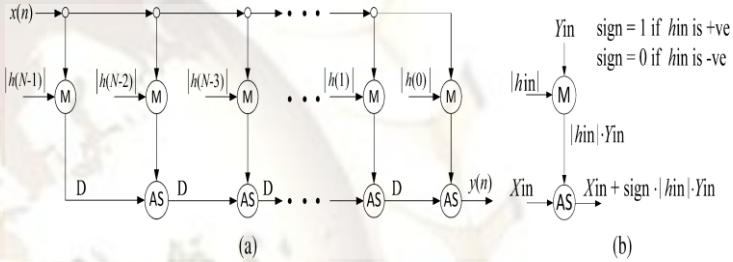

Fig: 4. Modified transposed form data flow graph (DFG) of an N-tap FIR filter for LUT-multiplier-based implementation. (a)The DFG. (b)Function of each multiplication node (M) and add-subtract node (AS) of the DFG.

The function of these nodes is depicted in Fig. 4(b). Each multiplication node performs the multiplication of an input sample value with the absolute value of a filter coefficient. The AS node adds or subtracts its input from top with or from that of its input from the left when the corresponding filter coefficient is positive or negative, respectively. It may be noted here that each of the multiplication nodes of the DFG performs multiplications of input samples with a fixed positive number. This feature can be utilized to implement the multiplications by an LUT that stores the results of multiplications of all possible input values with the multiplying coefficient of a node as unsigned numbers. The multiplication of an L–bit unsigned input with W -bit magnitude part of fixed filter-weight, to be performed by each of the multiplication-nodes of the DFG, can be implemented conventionally by a dual-port memory unit consisting of $(2^L/2)$ words of (W+L)- bit width. Each of the (N–1) AS nodes of the DFG along with a neighboring delay element can be mapped to an add-subtract (AS) cell.

A fully pipelined structure for N -tap FIR filter for input word length L=8, as shown in Fig. 5, is derived accordingly from the DFG of Fig. 4. It consists of N memory-units for conventional LUT-based multiplication, along with (N–1) AS cells and

a delay register. During each cycle, all the 8 bits of current input sample x (n) are fed to all the LUT-multipliers in parallel as a pair of 4-bit addresses X1 and X2. The structure of the LUT-multiplier is shown in Fig. 5(b). It consists of a dual-port memory unit of size [16× (W+4)] (consisting of 16 words of (W+4)-bit width) and a shift-add (SA) cell. The SA cell shifts its right-input to left by four bit-locations and adds the shifted value with its other input to produce a (W+8)-bit output. The shift operation in the shift-add cells is hardwired with the adders, so that no additional shifters are required.

The output of the multipliers is fed to the pipeline of AS cells in parallel. Each AS cell performs exactly the same function as that of the AS node of the DFG. It consists of either an adder or a subtractor depending on whether the corresponding filter weight h (n) is positive or negative, respectively. Besides, each of the SA cells consists of a pipeline latch corresponding to the delays in the DFG of Fig. 4. The FIR filter structure of Fig. 5 takes one input sample in each clock cycle, and produces one filter output in each cycle. The first filter output is obtained after a latency of three cycles (1 cycle each for memory output, the SA cell and the last AS cell). But, the first (N–1) outputs are not correct because they do not contain the contributions of all the filter coefficients. The correct output of this structure would thus be available after a latency of (N+2) cycles.

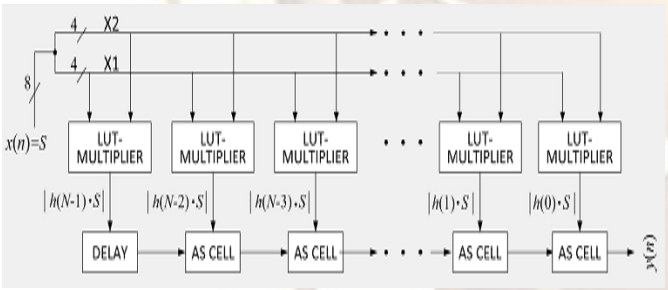

Fig: 5.(a) Conventional LUT-multiplier-based structure of an N-tap transposed form FIR filter for input-width L=8.

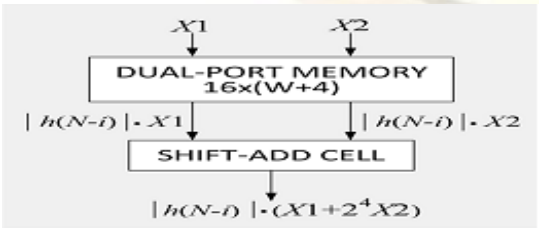

Fig: 5. (b) Structure of each LUT-multiplier

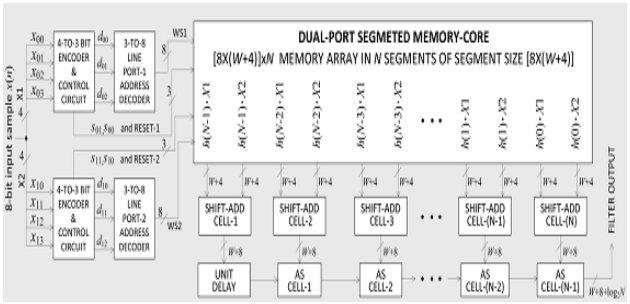

Fig: 6. (a). Structure of Nth order FIR filter using proposed LUT-multiplier.

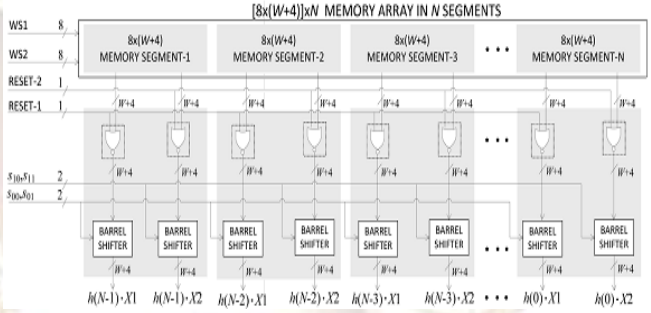

Fig: 6. (b).The dual-port segmented memory core for the Nth order FIR filter.

B. Memory-Based FIR Filter Using Proposed LUT Design

The memory-based structure of FIR filter (for 8-bit inputs) using the proposed LUT design is shown in Fig.6. It differs from that of the conventional memory-based structure of FIR Filter of Fig. 5 in two design aspects.

1) The conventional LUT-multiplier is replaced by proposed Odd-multiple-storage LUT, so that the multiplication by an L-bit word could be implemented by $(2^L/2)/2$ words in the LUT in a dual-port memory, as described in previous.

2) Since the same pair of address words X1and X2 are used by all the N LUT-multipliers in Fig. 5, only one memory module with N segments could be used instead of N modules. If all the multiplications are implemented by a single memory module, the hardware complexity of 2(N–1) decoder circuits (used in Fig. 5) could be eliminated.

As shown in Fig. 6, the proposed structure of FIR filter consists of a single memory-module, and an array of N shift-add (SA) cells, (N–1) AS cells and a delay register. The structure of memory module of Fig. 6(a) is similar to that of Fig. 3. Like the structure of Fig. 3, it consists of a pair of 4-to-3 bit encoders and control circuits and a pair of 3-to-8 line decoders to generate the necessary control signals and word select signals for the dual-port memory core. The dual-port memory core (shown in

Fig. 6(b)) consists of [8× (W+4)] ×N array of bit-level memory-elements arranged in 8 rows of [(W+4)] -bit width. Each row of this memory consists of N segments, where each segment is (W+4) -bit wide. Each segment of this dual-port core is of size 8×(W+4), such that the ith memory segment stores the 8 words corresponding to the multiplication of any 4-bit input with the filter weight h(i) for $0 \leq i \leq N-1$. During each cycle, a pair of 4-bit sub-words X1and X2 are derived from the recent-most input sample x(n) and fed to the pair of 4-to-3 bit encoders and control circuits, which produce two sets of word-select signals (WS1 and WS2), a pair of control signals((s_{01}, s_{00})) and(s_{11}, s_{10})) two reset signals. All these signals are fed to the dual-port memory-core as shown in Fig. 6. N segments of the memory-core then produce N pairs of corresponding output, those are fed subsequently to the N pairs of barrel-shifters through the 2N NOR cells. The array of N pairs of barrel-shifters thus produce N pairs of output ((hi). X1, h(i).X2) for $0 \leq i \leq N-1$. Since the same pair of address words X1 and X2 are used by all the N LUT-multipliers in Fig. 5, only one memory module with N segments could be used instead of N independent memory modules.

C. DA-BASED IMPLEMENTATION OF FIR FILTER

In this Section, we present a DA-based implementation of FIR filter that has the same throughput rate as that of the LUT multiplier-based structures. In each cycle, one 8-bit input sample is fed to the word-serial bit-parallel converter of the structure, out of which a pair of consecutive bits are transferred to each of its four DA-based computing sections. The structure of each DA-based section is shown in Fig. 7(b). It consists of a pair of serial-in parallel-out bit-level shift-registers (SIPOSRs),(N/4) memory modules of size [16× (W+2)], (N/4) shift-add (SA) cells and a pipelined shift-adder-tree.

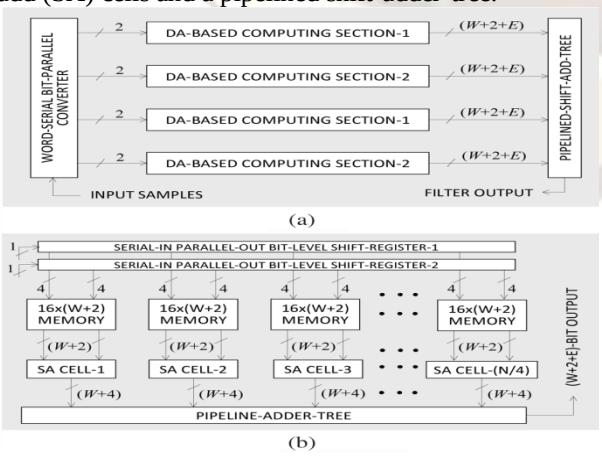

Fig: 7.DA-based structure for FIR filter. (a) The DA-based FIR filter structure (b) Structure of each section of the filter. $E=\log_2 N$

The memory module, in each cycle, is fed with a pair of 4-bit words at the pair of address-ports. The left address-port receives 4 bits from SIPOSR-1 while the right address-port receives 4 bits from SIPOSR-2. The bits available at right address port are the next significant bits corresponding to the bits available at its left address-port (Fig. 7(b)). According to the pair of 4-bit addresses a pair of (W+2) -bit words are read-out from each memory module, and fed to an SA cell. The SA cell shifts the right-input by one position to left and adds that with the left-input to produce a (W+4) -bit output. The output of the SA cells is added together by a pipelined-adder-tree to produce the output of a DA-based section. The output of 4 DA-based sections is added by pipelined shift-add-tree consisting of three adders in two pipelined stages (shown in Fig. 8). The pair of shift-adders (SA1) in stage-1 shift their lower input to left by two-bit positions and add with their upper input, while the shift-adder (SA2) in stage-2 shifts the lower input by four-bit positions and adds that to the upper input to produce a $(W+8+\log_2 N)$-bit output. The structure takes N cycles to fill in the SIPOSRs, one cycle for memory access and one cycle to produce the output of the shift-add cell in DA-based computing sections, $(\log_2 N-2)$ cycles in the pipelined-adder-tree and two cycles at pipelined shift-add-tree. The latency for this structure is therefore $(N+\log_2 N+2)$ cycles, and it has the same throughput of one output per cycle, as that of the LUT-multiplier-based structures. When the input word-length is a multiple of 8, such as L=8k, (where k is any integer in general), the DA-based filter could also be implemented by k parallel sections where each section is an 8-bit filter identical to one of the structures of Fig. 7. The outputs of all the 8-bit filter sections are shift-added in a pipeline shift-add-tree to derive the filter outputs, for the LUT-multiplier-based implementation. The structure for L=8 would have the same throughput of one output per cycle with a latency of $(N+\log_2 N+\log_2 k+2)$ cycles.

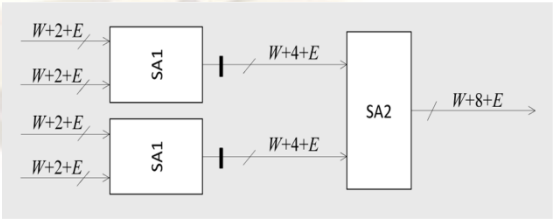

Fig: 8. Pipelined shift-add-tree. $E=\log_2 N$

V. RESULTS

Simulation results of 4-tab FIR filter for input operand width L=8

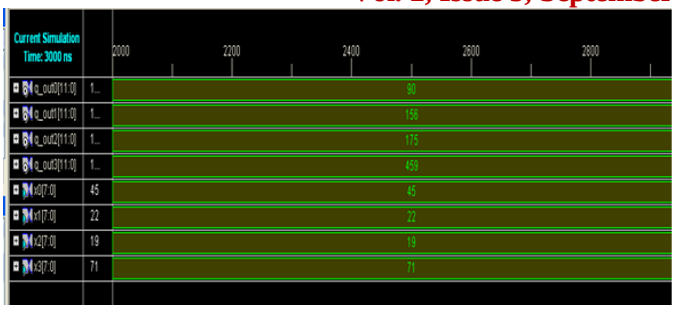

Fig: 9.Conventional LUT-multiplier-based design

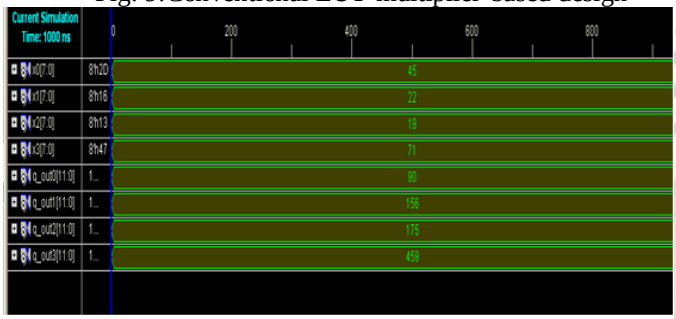

Fig: 10.Proposed LUT-multiplier-based design

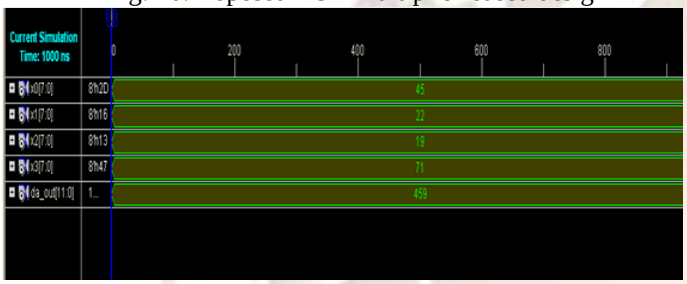

Fig: 11.DA-based design

Synthesis Report:

FIR filter design	No. of slices utilization out of 4656	No. of 4 input LUTS out of 9312	Number of bonded IOBs out of 232	path delay
DA-based design	115 (2%)	206 (2%)	44 (18%)	27.910ns
Conventional LUT-multiplier-based design	51 (1%)	93 (0%)	80 (34%)	27.910ns
Proposed LUT-multiplier-based design	39 (0%)	72 (0%)	80 (34%)	14.835ns

VI. CONCLUSION

A new approach to LUT-based-multiplication is suggested to reduce the LUT-size over that of conventional design. Three memory-based structures having unit throughput rate are designed for the implementation of FIR filter. One of the structures is based on DA principle, and the other two are based on LUT-based multiplier using the conventional and the proposed LUT designs. All the structures are found to have the same or nearly the same cycle periods, which depend on the implementation of adders, the word-length and the filter order. But the LUT-multiplier-based design of FIR filter is more efficient than the DA-based approach in terms of area-complexity for a given throughput and lower latency of implementation.

REFERENCES

[1] Meher, "New look-up-table optimizations For memory-based Multiplication," in Proc.ISIC, Dec.2009, pp.663–666.

[2] International Technology Roadmap for Semiconductors. [Online].

[3] P.K.Meher, "New approach to LUT implementation and Accumulation for memory-Multiplication,"inProc. IEEEISCAS, May2009, pp.453–456.

[4] Meher, 'Memory-based hardware for resource-constrained digital signal processingSystems,'inproc.6thInt.conf.ICI CS, Dec2007, pp.1–4

[5] P. K. Meher, S. Chandrasekaran, 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.

[6] P. K. Meher, J. C. Patra, and M. N. S. Swamy, "High-throughput Memory-based architecture for DHT using a new convolutionl Formulation,"IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 54, no. 7, pp. 606–610, Jul. 2007.

[7] P. K. Meher and M. N. S. Swamy, "New systolic algorithm and Array architecture for prime-length discrete sine transforms," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 54, no. 3, pp. 262–266, Mar. 2007.

[8] P. K. Meher, "Unified systolic-like architecture for DCT and DST Using distributed arithmetic," IEEE Trans. Circuits Syst. I, Reg. Papers, vol.53, no. 5, pp. 2656–2663, Dec. 2006.

[9] H. Yoo and D. V. Anderson, "Hardware-efficient distributed Arithmetic architecture for high-order digital filters," in Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing, (ICASSP'05), Mar. 2005, vol.5

[10] D. Xu and J. Chiu, "Design of a high-order FIR digital filtering and variable gain ranging seismic data acquisition system," in Proc. IEEE Southeastcon'93, Apr. 1993, p. 6.