Design and Realization of Look-up-Table, FIR Digital Filter

Dr.D.R.V.A.Sharath Kumar, Mr.J.Nageswara Reddy, Mr.T.Sathya Narayana,

***Abstract*— 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. By operand and inner-product decompositions, respectively, we have designed the conventional LUT-multiplier-based and DA-based structures for FIR filter of equivalent throughput, where the LUT-multiplier-based design involves nearly the same memory and the same number of adders, and less number of input register at the cost of slightly higher adder-widths than the other. Moreover, we present two new approaches to LUT-based multiplication, which could be used to reduce the memory size to half of the conventional LUT-based multiplication. Besides, we present a modified transposed form FIR filter, where a single segmented memory-core with only one pair of decoders are used to minimize the combinational area. The proposed LUT-based FIR filter is found to involve nearly half the memory-space and *I/N*times the complexity of decoders and input-registers, at the cost of marginal increase in the width of the adders, and additional *~(4N+W)*AND-OR-INVERT gates and *~(2N+W)*NOR gates. We have synthesized the DA-based design and LUT- multiplier based design of 16-tap FIR filters by Synopsys Design Compiler using TSMC 90 nm library, and find that the proposed LUT-multiplier- based design involves nearly 15% less area than the DA-based design for the same throughput and lower latency of implementation.**

***Index Terms*— Digital signal processing (DSP) chip, distributed arithmetic, FIR filter, LUT-based computing, memory-based computing, VLSI**

I. INTRODUCTION

FINITE-IMPULSE response (FIR) digital filter is widely used as a basic tool in various signal processing and image processing applications [1]. 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 [2], [3]. 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 challengingtask. Several attempts have, therefore, been made and continuedto develop low-complexity dedicated VLSI systems for thesefilters [4]–[7].

Manuscript received November 07, 2014.

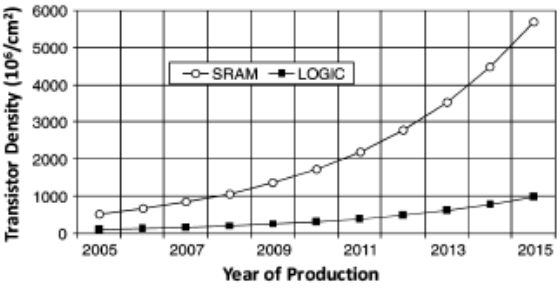


Fig. 1.Trend of transistor density in logic elements and SRAM.

As the scaling in silicon devices has progressed over the last four decades, semiconductor memory has become cheaper, faster and more power-efficient. According to the projections of the international technology roadmap for semiconductors (ITRS) [8], embedded memories will continue to have dominating presence in the system-on-chip (SoC), which may exceed 90% of total SoC content. It has also been found that the transistor packing density of SRAM is not only high, but also increasing much faster than the transistor density of logic devices (Fig. 1). According to the requirement of different application environments, memory technology has been advanced in a wide and diverse manner. Radiation hardened memories for space applications, wide temperature memories for automotive, high reliability memories for biomedical instrumentation, low power memories for consumer products, and high-speed memories for multimedia applications are under continued development process to take care of the special needs [9], [10]. Interestingly also, the concept of memory, only as a standalone subsystem in a general purpose machine is no longer valid, since embedded memories are integrated as part within the processor chip to derive much higher bandwidth between a processing unit and a memory macro with much lower power- consumption [11]. To achieve overall enhancement in performance of computing systems and to minimize the bandwidth requirement, access-delay and power dissipation, either the processor has been moved to memory or the memory has been moved to processor in order to place the computing-logic and memory elements at closest proximity to each other [12]. In addition to that, memory elements have also been used either as a complete arithmetic circuit or a part of that in various application specific platforms.

In this paper, we use the phrase "*memory-based structures*" or "*memory-based systems*" for those systems where

memory elements like RAM or ROM is used either as a part or whole of an arithmetic unit [13]. Memory-based structures are more regular compared with the multiply-accumulate structures; and have 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. Several architectures have been reported in the literature for memory-based implementation of discrete sinusoidal transforms and digital filters for DSP applications [13]–[28]. There are two basic variants of memory-based techniques. One of them is based on distributed arithmetic (DA) for inner product computation [14]– [20], [22]–[24] and the other is based on the computation of multiplication by look-up-table (LUT) [24]–[28]. 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 -point vector with any possible -point bit-vector. If the inner-products are implemented in a straight-forward way, the memory-size of LUT-multiplier based 76 implementation increases exponentially with the wordlengthof input values, while that of the DA-based approach increasesexponentially with the inner-product-length. Attemptshave been made to reduce the memory-space in DA-based architecturesusing offset binary coding (OBC) [14], [29], andgroup distributed technique [20]. A decomposition scheme issuggested in a recent paper [22] for reducing the memory-size ofDA-based implementation of FIR filter. But, it is observed thatthe reduction of memory-size achieved by such decompositions is accompanied by increase in latency as well as the number ofadders and latches.Significant work has been done on efficient DA-basedcomputation of sinusoidal transforms and filters. Various algorithm-architecture co-designs have also been reported forefficient LUT-multiplier-based implementation of sinusoidaltransforms [24]–[28]. In an early paper, Lee *et al.* [13] hadintroduced a memory-based structure for the LUT-multiplier-based implementation of FIR filter. But, we do not findany further work to improve the efficiency of LUT-multiplier-based implementation of FIR filter. In this paper, we aimat presenting two new approaches for designing the LUT forLUT-multiplier-based implementation, where the memory-size
is reduced to nearly half of the conventional approach. Besides,we find that instead of direct-form realization, transposed formrealization of FIR filter is more efficient for the LUT-multiplier-based implementation. In the transposed form, a singlesegmented-memory core could be used instead of separatememory modules for individual multiplications in order toavoid the use of individual decoders for each of those separatemodules.

The remainder of the paper is organized as follows. In Section II, we have presented the proposed design of LUT for memory-based multiplication. In Section III, we have described

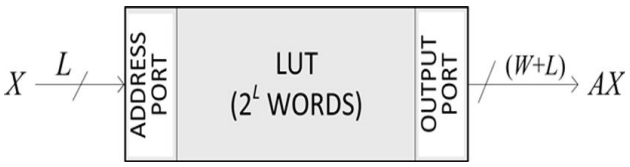


Fig. 2.Conventional Memory-Based Multiplier.

TABLE I
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.

the proposed structure for LUT-multiplier-based implementation of FIR filter, and a DA-based filter of the same throughput rate is derived in Section IV. The area and time-complexity of the LUT-multiplier-based designs and DA-based designs of FIR filter are evaluated and compared in Section V. Conclusions are presented in Section VI.

II. LUT DESIGN FOR MEMORY-BASED MULTIPLICATION

The basic principle of memory-based multiplication is depicted in Fig. 2. Let A be a fixed coefficient and X be an input word to be multiplied withA . If we assume X to be an unsigned binary number of word-length L , there can be 2^L possible values of X, and accordingly, there can be 2^L possible values of product C=A.X . Therefore, for the conventional implementation of memory-based multiplication [24], a memory unit of words 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 $0 \le X_i \le 2^L-1$, is stored at the memory location whose address is the same as the binary value of , such that if – L bit binary value of is used as address for the memory-unit, then the corresponding product value is read-out from the memory. Although 2^Lpossible values of X correspond to 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 [30].

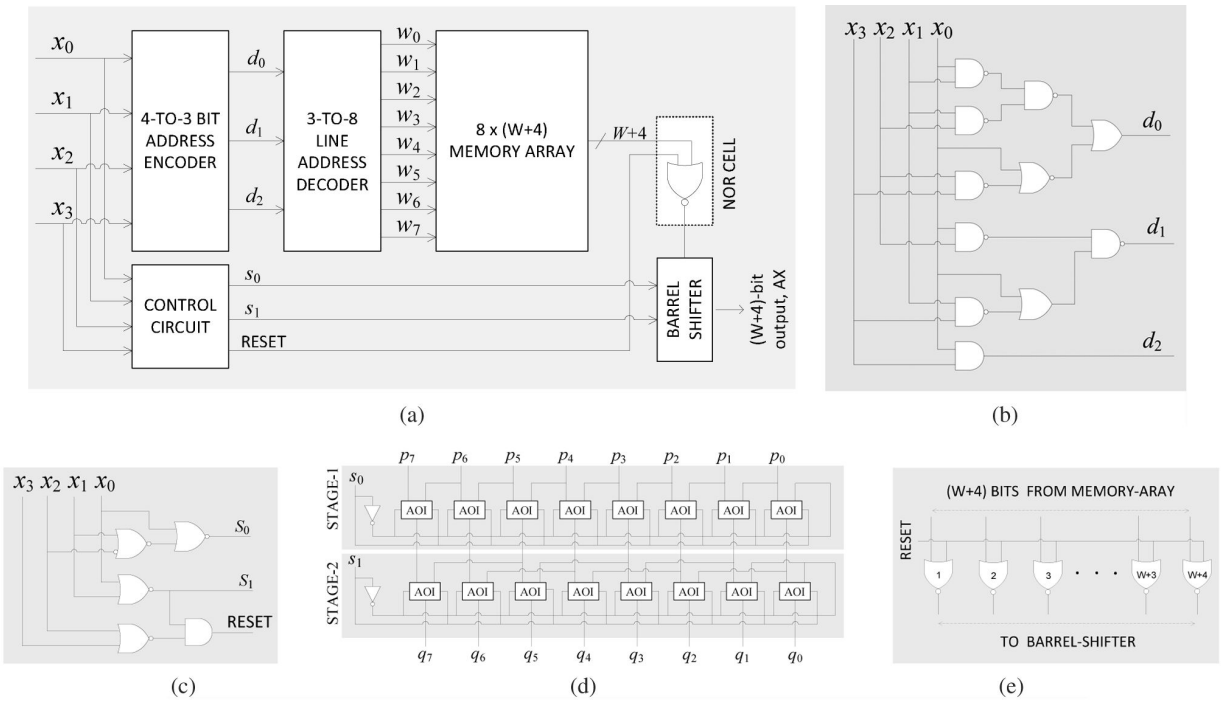


Fig. 3. Proposed LUT design for multiplication of W -bit fixed coefficient, A and 4-bit input operand, X = x_3 x_2 x_1 x_0. The proposed LUT-based multiplier. (b) The 4-to-3 bits input encoder. (c) Control circuit. (d) Two-stage logarithmic barrel-shifter for W = 4. (e) Structure of the NOR-cell.

One of the possible product words is zero, while all the rest $(2^L/2)$-1 are even multiples of 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 P_i as for i=0,1,2,3,4,5,6,7 . The even multiples 2A, 4Aand, 8A are 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=(0 0 0 0) corresponds to (A.x)=0 , which can be obtained by resetting the LUT output. For an input multiplicand of word-size A 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 -bit coefficient is designed by the following strategy:

• A memory-unit of $(2^L/2)$ 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 is used to derive all the even multiples of A .
• The L-bit input word is mappedto (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 .

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

The proposed LUT-based multiplier for input word-size is shown in Fig. 3. 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_o s_1) for the barrel-shifter, and the RESET signal for the NOR-cell.

The 4-to-3 bit input encoder is shown in Fig. 3(b). It receives a four-bit input word (x_3 x_2 x_1 x_o) and maps that onto the three-bit address word(d_2 d_1 d_0) , according to the logical relations

$$d_0 = \overline{(x_0 \cdot x_1) \cdot (x_1 \cdot x_2) \cdot (x_0 + (x_2 \cdot x_3))} \quad (1a)$$

$$d_1 = \overline{(x_0 \cdot x_2) \cdot (x_0 + (x_1 \cdot x_3))} \quad (1b)$$

$$d_2 = x_0 \cdot x_3 \quad (1c)$$

The pre-computed values of Ax(2i+1) are stored as P_i for i=0,1,2,3….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,{wi,0≤i≤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 1 location to left when the input operand X is one of the values {(0010),(0110),(1010),(1110)}. Two left-shifts are required if is either (0 1 0 0) or (1 1 0 0). Only when theinput word X= (1000), three shifts are required. For allother possible input operands, no shifts are required. Since the maximum number of left-shifts required on the stored-wordis three, a two-stage logarithmic barrel-shifter is adequate to perform the necessary left-shift operations.

The number of shifts required to be performed on the outputof the LUT and the control-bits and for different valuesof are shown Table I. The control circuit [shown in Fig. 3(c)]accordingly generates the control-bits given by

$$s_0 = \overline{x_0 + \overline{(x_1 + \overline{x_2})}} \quad (2a)$$

$$s_1 = \overline{(x_0 + x_1)} \quad (2b)$$

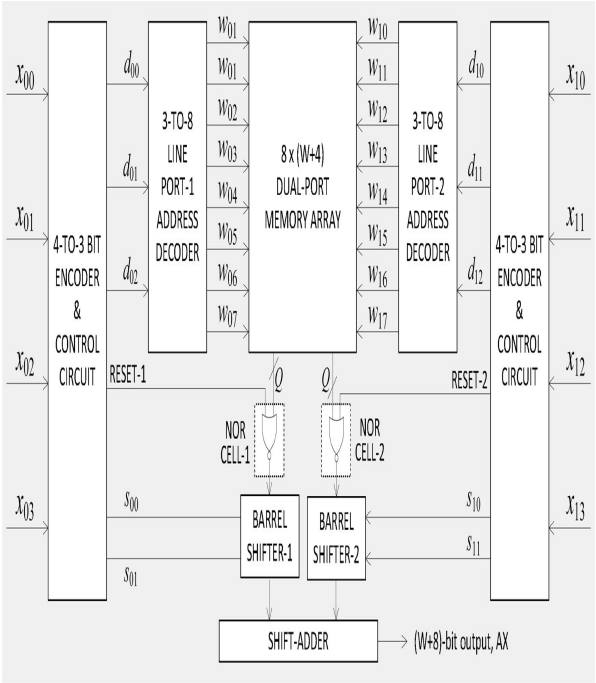


Fig. 4.Memory-based multiplier using dual-port memory-array. Q=(W+4)

A logarithmic barrel-shifter for W=L=4 is shown in Fig. 3(d). It consists of two stages of 2-to-1 line bit-level multiplexors with inverted output, where each of the two stages involves number of 2-input AND-OR-INVERT (AOI) gates. The control- bits(s_0,s_O) and (s_1,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=(0 0 0 0) corresponds to multiplication by X=0 which results in the product value A.X=0 . Therefore, when the input operand word X= (0 0 0 0), the output of the LUT is required to be reset. The reset function is not implemented by a NOR-cell consisting of (W+4) NOR gates as shown in Fig. 3(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 NOR gates of NOR cell are fed with bits of LUT output in parallel. When , the control circuit in Fig. 3(c), generates an active-high RESET according to the logic expression:

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

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 [31], [32]. 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.

To compare the area of the proposed LUT-multiplier and the existing LUT-multiplier, we have synthesized the multipliers for L=4 for different coefficient width by Synopsys Design Compiler [33] using TSMC 90 nm library and listed in Table II.

TABLE II
COMPLEXITIES OF LUT-BASED MULTIPLIERS FOR L=4

coefficient width W	conventional design	proposed design		saving of area
		overhead area	total area	
8	700.7	215.2	538.4	23.16 %
16	1112.7	349.3	858.7	22.83 %
24	1527.6	476.3	1172.0	23.28 %
32	1939.7	599.8	1488.1	23.28 %

The estimated areas are in sq.μm.

Both the designs have nearly the same data arrival time, but the proposed LUT design is found to offer a saving of nearly 23% of area over the conventional design. The saving of area in the proposed LUT design resulting from lower storage and less decoder complexity is reduced mainly due to the overhead of barrel-shifter and NOR cells (indicated in Table II).

Multiplication of an 8-bit input with a -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 barrelshifters as shown in Fig. 4. 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. In the next sub-section, we present two other optimization schemes which has been proposed recently for reduction of storage size of LUT-multipliers[34].

B. Other Optimizations of Look-Up-Table for Memory-Based Multiplication

Assuming the input X to be a signed number in sign-magnitude form, we can write the product C=A.X, as

$$C = s_X * A \cdot |X| = s * |A| \cdot |X| \quad (4a)$$

Where |A| and |X| are, respectively, the magnitude-parts of and ; denotes bit concatenation operation at the MSB of |X|; and

$S=s_A$ XOR s_X (4b)

TABLE III
PRODUCT WORDS AND STORED WORDS FOR DIFFERENT |X| VALUES

modulus, $\|X\|$ $x_3\ x_2x_1x_0$	product values	OPC representation	stored words	
			2's comp.	sign-mag.
0 0 0 0	0	$(8A-8A)$	$8A$	$8\|A\|$
0 0 0 1	A	$(8A-7A)$	$7A$	$7\|A\|$
0 0 1 0	$2A$	$(8A-6A)$	$6A$	$6\|A\|$
0 0 1 1	$3A$	$(8A-5A)$	$5A$	$5\|A\|$
0 1 0 0	$4A$	$(8A-4A)$	$4A$	$4\|A\|$
0 1 0 1	$5A$	$(8A-3A)$	$3A$	$3\|A\|$
0 1 1 0	$6A$	$(8A-2A)$	$2A$	$2\|A\|$
0 1 1 1	$7A$	$(8A-A)$	A	$\|A\|$
1 0 0 0*	$8A$	$(8A-0)$	0	0
1 0 0 1	$9A$	$(8A+A)$	--	--
1 0 1 0	10	$(8A+2A)$	--	--
1 0 1 1	$11A$	$(8A+3A)$	--	--
1 1 0 0	$12A$	$(8A+4A)$	--	--
1 1 0 1	$13A$	$(8A+5A)$	--	--
1 1 1 0	$14A$	$(8A+6A)$	--	--
1 1 1 1	$15A$	$(8A+7A)$	--	--

* Instead of storing a '0' for $(x_3x_2x_1x_0) = (1\ 0\ 0\ 0)$, the LUT output is RESET.

For s_A and s_X, being the sign-bits of A and X , respectively.

Since |X| is an (L-1)-bit binary number, all possible values of the product |A|.|X| could be stored as $2^{(L-1)}$ LUT words, while the sign-bit could be derived by an XOR operation according to (4b). The product words for positive values of $X=x_4x_3x_2x_1x_0$ for LUT-based multiplication (for) is shown in the second column of Table III. The product words corresponding to the negative values ofX can be obtained by sign-modification of the product words stored for the same value of $X=x_4x_3x_2x_1x_0$. It requires only one additional XOR gate to determine the sign of product word according to (4b). Therefore, instead of 32 product words only 16 values of |A|.|X| for all possible values of $|X|=x_4x_3x_2x_1x_0$ are required to be stored. Note that the sign-exclusion technique can also be applied for 2's complement representation of .

It may be observed in the first column of Table III that, the address word |X| on the ith row is 2's complement of that on (16+2+i)th row for $2\le i\le 8$. Besides, the sum of product values on these two rows is. Let the product values on the ith and (16+2+i)th rows be$u$ and$v$, respectively. Since one can write $u=[(u+v)/2-(v-u)/2]$and$v=[(u+v)/2-(v-u)/2]$, for $(u+v)=16A$, we can have

$$u = 8A - \left[\frac{v-u}{2}\right] \quad \text{and} \quad v = 8A + \left[\frac{v-u}{2}\right] \qquad (5)$$

The product values on the second column of Table III, therefore, could be re-written (in a form as given in the third column of the Table) according to (5), which we have referred here as *output product coding* (OPC). It can be seen that the product words on the third column of this Table have a negative mirror symmetry. This behavior of OPC can be used for further reduction of LUT size as shown in the fourth column of the Table, where instead of storing u and v at the ith and(16+2+i) th rows, respectively, $[(u+v)/2]$is stored at ith row, for $2\le i\le 8$. The desired product can be obtained by adding or subtracting the stored $[(v-u)/2]$ value to or from 8A whenx_3 is 1 or 0, respectively. For the first and the eighth row, we can store 8A and 0, respectively, in LUT. But, instead of storing 0 for $(x_3x_2x_1x_0)=(1000)$, it would be more convenient to reset the LUT output by a RESET signal given by

$$\text{RESET} = \overline{(x_3x_2x_1x_0)} \qquad (6)$$

OPC can be used for sign-magnitude as well as 2's complement representation of the fixed coefficient, A. As shown in the last column of Table III (for sign-magnitude form ofA), all the stored values are positive, and the sign of product word, can be determined according (4b). Using the OPC approach, instead of storing 16 words after sign-bit elimination for$L=5$, one can store only eightwords as shown in Table III. In general, for L –bit input, $(2^L)/8$only words are required to be stored in the LUT using sign-bit exclusion and OPC along with the odd-multiple storage scheme discussed in the previous sub- section. Note that OPC scheme is similar to the OBC [14], [29] for DA-based implementations, and sign-bit exclusion scheme for LUT-multiplier can be applied in case ofDA, as well. Therefore, for performance comparison between DA- based and LUT-multiplier based implementations of FIR filter (in Section V), we restrict to the LUT-optimization by the odd-multiple storage scheme only.

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

We derive here the proposed structure for memory-based realization of an -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

$$y(n) = h(0)\cdot x(n) + h(1)\cdot x(n-1) + h(2) \cdot x(n-2) + \cdots h(N-1)\cdot x(n-N+1) \qquad (7)$$

where $h(n)$, for $n=0,1,2,\ldots\ldots N-1$, represent the filter coefficients, while $x(n-1)$, for , represent recent input samples, and 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 (7) alternatively as

$$y(n) = \text{sign}(0)\cdot|h(0)|\cdot x(n) + \text{sign}(1) \cdot|h(1)|\cdot x(n-1) + \cdots + \text{sign}(N-1) \cdot|h(N-1)|\cdot x(n-N+1) \qquad (8)$$

Where h(n)=sign(n).|h(n)| , for n=0,1……..N-1,|h(n)| denotes the absolute value of h(n) and sign(n) =±1is the sign-factor, which could be absorbed with the additions of the corresponding term. Equation (8), then may be written in a recursive
Form

$$y(n) = \text{sign}(0)\cdot|h(0)|\cdot x(n) + \text{sign}(1)\cdot \mathbf{D}(|h(1)| \cdot x(n) + \text{sign}(2)\cdot \mathbf{D}(|h(2)|\cdot x(n) + \cdots + \text{sign}(N-1)\cdot \mathbf{D}(|h(N-1)|\cdot x(n))\cdots)) \qquad (9)$$

A. Memory-Based FIR Filter Using Conventional LUT

The recursive computation of FIR filter output according to (9) is represented by a transposed form data-flow graph (DFG) shown in Fig. 5. It consists of N multiplication nodes (M) and (N-1) add-subtract (AS) nodes. The function of these nodes is depicted in Fig. 5(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 -tap FIR filter for inputwordlength L=8, as shown in Fig. 6, is derived accordingly from the DFG of Fig. 5. 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 are fed to all the LUT-multipliers in parallel as a pair of 4-bit addresses and. The structure of the LUT-multiplier is shown in Fig. 6(b). It consists of a dual-port memory unit of size (consisting of 16 words of -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 -bit output. The shift operation in the shift-add cells is hardwired with the adders, so that no additional shifters are required. The outputs of the multipliers are 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 sub tractor depending on whether the corresponding filter weight 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. 5. The FIR filter structure of Fig. 6 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 cycles.

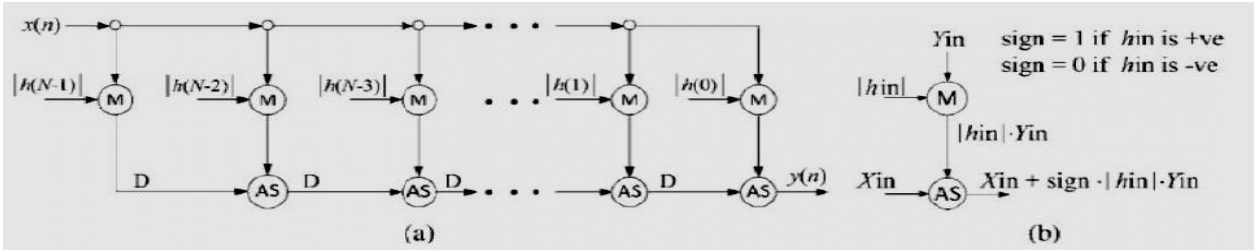


Fig. 5. Modified transposed form data-flow graph (DFG) of an -tap FIR filter for LUT-multiplier-based implementation. (a) The DFG. (b) Function of eachmultiplication node (M) and add-subtract node (AS) of the DFG.

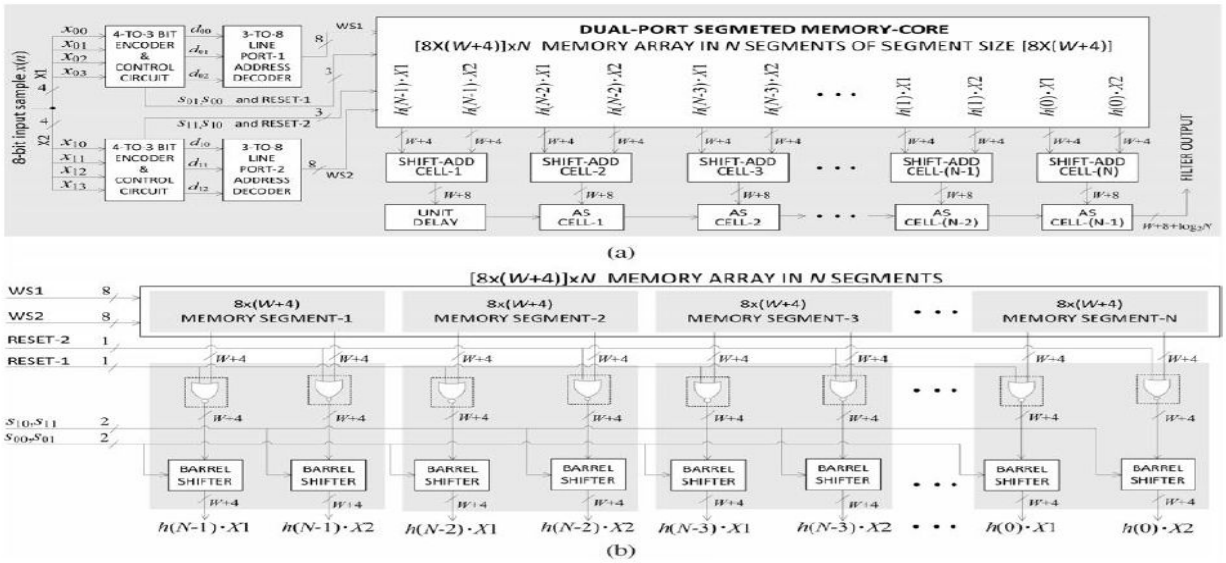


Fig. 7. (a) Structure of Nth order FIR filter using proposed LUT-multiplier. (b) The dual-port segmented memory core for the Nth order FIR filter. We derive here the proposed structure for memory-based real-ization of an N-tap FIR filter, and discuss the design of

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) /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 NLUT-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 singlememory module, the hardware complexity of 2(N–1) decoder circuits (used in Fig. 5) could be eliminated.

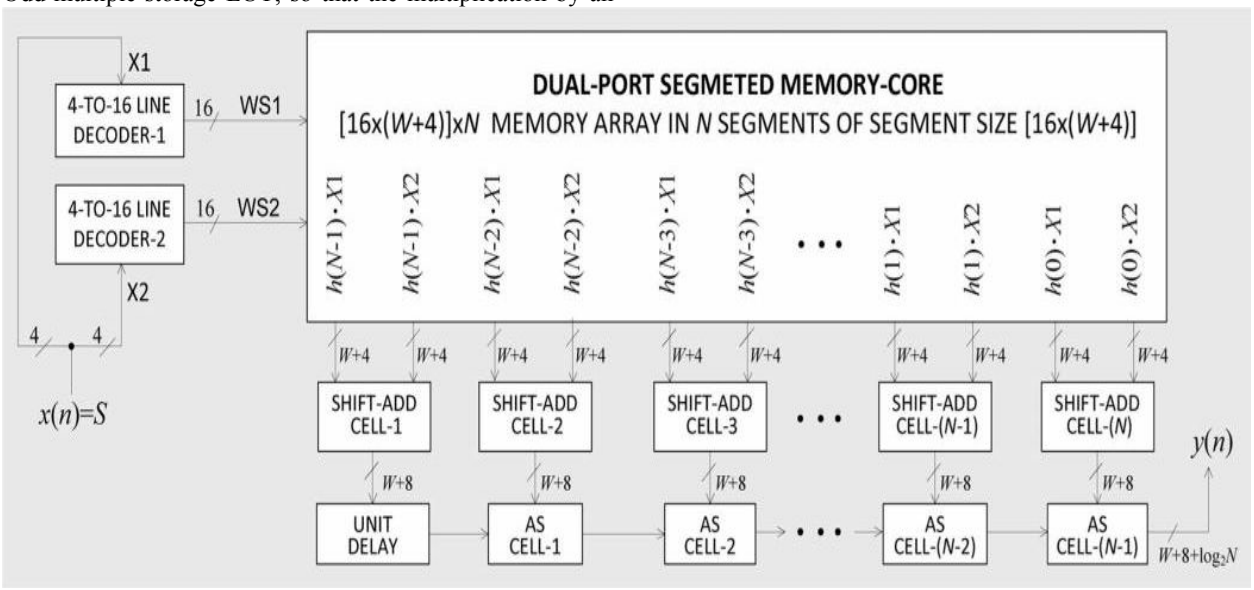
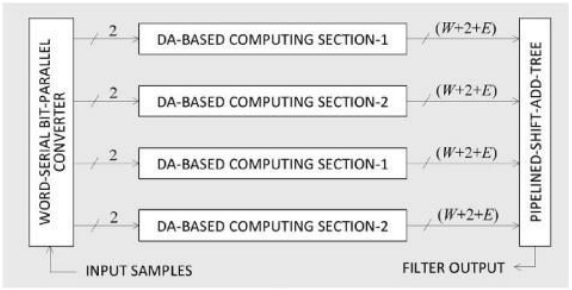


Fig. 8.LUT-multiplier-based structure of an -tap FIR filter by transposed form realization using segmented memory-core.

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

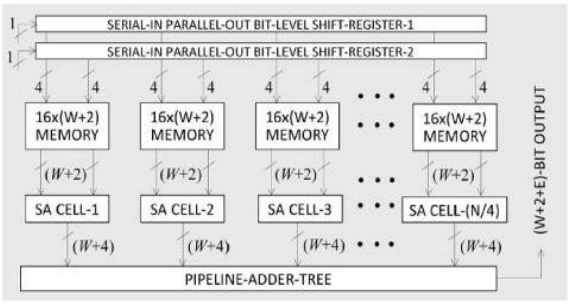
consists of a pair of 4-to-3 bit encoders and control circuits and a pair of 3-to-8 line decoders to generate the necessary control signals and word select signals for the dual-port memory core. The dual-port memory core (shown in Fig. 6(b)) consists of [8× (W+4)] ×N array of bit-level memory-elements 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≤ i ≤ 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((s01, s00)) and(s11, s10)) 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



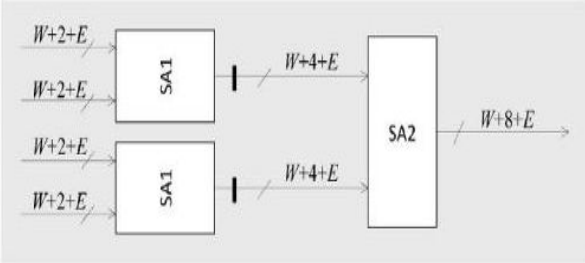
(a)

pairs of output ((hi) . X1, h(i).X2) for 0≤ i≤ 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.



(b)



IV. 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. To derive the DA-based design, we follow here the data decomposition scheme for DA-based implementation presented in [22]. Throughput scalable 2-dimensional (2-D) architecture for the DA-based implementation of FIR filter based on a flexible decomposition scheme is proposed and area-delay performance is presented in [22]. It is found there that the DA-based FIR filter structure results in minimum area and minimum area-delay product for address-length 4. In Fig.9, we have shown a modified form of the 2-D structure of FIR filter presented in [22] for input word-size L = 8. The systolic pipelined adder in the structure of [22] is replaced by pipelined-adder-tree and pipelined-shift-add-tree in Fig.9 to reduce the number of latches and latency. 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.9(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.

The memory module, in each cycle, is thus 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.9(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 are added together by a pipelined-adder-tree to produce the output of a DA-based section. The output of 4 DA-based sections are added by pipelined shift-add-tree consisting of three adders in two pipelined stages (shown in Fig.10). 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)

9cycles, 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. 9. The outputs of all the 8-bit filter sections are shift-added in a pipeline shift-add-tree to derive the filter outputs as discussed in Section III-C for the LUT-multiplier-based implementation. The structure for L = 8k would have the same throughput of one output per cycle with a latency of (N + $\log_2$ N + $\log_2$ k + 2) cycles.

V. COMPLEXITY CONSIDERATIONS

We discuss the estimation of hardware and time com-plexities of DA-based as well as the LUT-multiplier-based implementation of FIR filter using conventional and the pro-posed LUT designs. Based on the estimated complexities, we compare here the performances of all these implementations.

A. Complexity of Memory-Based Implementation using Con-ventional LUT-Multiplier

The structure for conventional LUT-multiplier-based imple-mentation of an N-th order FIR filter (Fig.6) for 8-bit input and W -bit filter-coefficients requires N dual-port memory modules each consisting of a memory array of size [16 × (W + 4)], a (W + 4)-bit adder, and a (W + 4)-bit latch , (N −1) AS cells and a delay cell. Each AS cell consists of one adder/subtractor followed by a latch. The width of these adders grows from (W + 8) bits to (W + 8 + $\log_2$ N) bits across the pipeline. Accordingly, the number of bit-latches also grows across the pipeline. The critical-path of this structure is max{T_M^{CL} , T_{SA}^{O}}, where T_M^{CL} is the access-time of the memory modules of size [16 ×(W +4)] and T_{SA}^{O} is the time required by the last adder in the systolic pipeline that produces the final output. The actual value of the minimum clock period not only will depend on the coefficient word-length W and filter order N but also on how the memory module and the adders are implemented. When, the input samples are (k × 8)-bit wide, k such parallel sections of 8-bit filters with (k − 1) adders in a pipeline shift-add-tree are required. The latency for the structure for 8-bit input is (N + 2) cycles and offers a minimum sampling period of one sample per cycle. For (k × 8)-bit input samples, the structure yields the same throughput per cycle with a latency is (N + $\log_2$ k + 2) cycles since the pipeline shift-add-tree involves $\log_2$ k cycles. The conventional LUT-multiplier-based design using the segmented memory core (Fig.8) involves only one

pair of 4-to-16 lines decoders, instead of N such pairs of decoders of Fig.6. It has the same throughput rate and the same latency as the filter of Fig.6.

B. Complexity of Memory-Based Implementation using Pro-posed LUT-Multiplier

Like the conventional LUT-multiplier-based structure, the proposed LUT-multiplier-based structure also involves N shift-add cells, (N −1) AS cells, one delay cell. It differs only in the LUT implementation. It requires a single segmented

module of [8 × (W + 4)] × N bit-size, a pair of 4-to-3 bit encoders and control-circuits, a pair of 3-to-8 lines decoders, 2N NOR cells and equal number of barrel shifters. Each NOR cell consists of (W +4) NOR gates and each of the two stages of a barrel-shifter consists of (W + 4) AOI gates. The critical-path of the structure amounts to $\max\{T_M^{PL}, T_{SA}^{O}\}$, where
$T_M^{PL} = T_M^{CORE} + T_{NOR} + 2T_{AOI}$ is the delay of the proposed LUT-multiplier and T_M^{CORE} is the access-time of the LUT core of depth 8, while T_{NOR} and T_{AOI} are the propagation delays of a NOR gate and an AOI gate, respectively. T_{SA}^{O} is the time required by the last adder in the systolic pipeline that produces the final output, which is the same as that defined for the conventional LUT-multiplier-based design. It has the same throughput rate of one output per cycle and the same number of cycles of latency as the conventional LUT-multiplier-based design.

C. Complexity DA-Based Implementation of FIR Filter

The DA-based FIR filter for 8-bit input size has four DA-based sections, one word-serial bit-parallel convertor, and a pipelined shift-add-tree. The word-serial bit-parallel converter consists of an 8-bit register. The pipelined shift-add-tree con-sists of three adders (two adders of $(W + 2 + \log_2 N)$-bit width in the first pipeline-stage and one adder of $(W + 4 + \log_2 N)$ width in the second stage). Each section of the DA-based filter requires two bit-level SIPOSRs, (N/4) memory modules of size [16 × (W + 2)] each, (N/4) SA cells (each consisting

module of [8 × (W + 4)] × N bit-size, a pair of 4-to-3 bit encoders and control-circuits, a pair of 3-to-8 lines decoders, 2N NOR cells and equal number of barrel shifters. Each NOR cell consists of (W +4) NOR gates and each of the two stages of a barrel-shifter consists of (W + 4) AOI gates. The critical-path of the structure amounts to $\max\{T_M^{PL}, T_{SA}^{O}\}$, where
$T_M^{PL} = T_M^{CORE} + T_{NOR} + 2T_{AOI}$ is the delay of the proposed LUT-multiplier and T_M^{CORE} is the access-time of the LUT core of depth 8, while T_{NOR} and T_{AOI} are the propagation delays of a NOR gate and an AOI gate, respectively. T_{SA}^{O} is the time required by the last adder in the systolic pipeline that produces the final output, which is the same as that defined for the conventional LUT-multiplier-based design. It has the same throughput rate of one output per cycle and the same number of cycles of latency as the conventional LUT-multiplier-based design

TABLE IV
HARDWARE AND TIME COMPLEXITIES OF PROPOSED AND CONVENTIONAL LUT-MULTIPLIER-BASED FIR FILTER OF ORDER N AND THE DA-BASED FILTER OF THE SAME THROUGHPUT PER CYCLE. (WORD-LENGTH OF COEFFICIENTS = W AND WORD-LENGTH OF INPUT SAMPLES L = 8 × k).

designs	conventional LUT-multiplier-based design*	DA-based design	proposed LUT-multiplier-based design**
memory	k # [16 × (W + 4) × N]-bit memory	Nk # [16 × (W + 2)]-bit memory	k # [8 × (W + 4) × N]-bit memory
decoders	2k # 4 : 16 decoders	2Nk # 4 : 16 decoders	2k # 3 : 8 decoders
adders	Nk # (W + 4)-bit adders k(N − 1) # (W + E + 7)-bit adders (k − 1) # (W + E + E + 7)-bit adders	Nk # (W + 2)-bit adders k(N − 1) # (W + 5)-bit adders (k − 1) # (W + E + E + 7)-bit adders	Nk # (W + 4)-bit adders k(N − 1) # (W + E + 7)-bit adders (k − 1) # (W + E + E + 7)-bit adders
registers[a]	k # 8-bit input-register (W + 8k + E) bit output-register	k # [(N + 1) × 8]-bit input-register (W + 8k + E) bit output-register	k # 8-bit input-register (W + 8k + E) bit output-register
latches[a]	kN(4W + E + 23)+ (k − 1)(W + E + 8) + k(E − 1)	k[2W (N − 1) + 10N − 2E − 8]+ (k − 1)(W + E + 8) + k(E − 1)	kN(4W + E + 23)+ (k − 1)(W + E + 8) + k(E − 1)
critical-path	$\max\{T_M^{CL}, T_{SA}^{O}\}$	$\max\{T_M^{DA}, T_{SA}^{O}\}$	$\max\{T_M^{PL}, T_{SA}^{O}\}$
latency	(N + E + 2) cycles	(N + E + E + 2) cycles	(N + E + 2) cycles
throughput	1 per cycle	1 per cycle	1 per cycle

*It corresponds to that of Fig.8 of conventional LUT-multiplier-based design using a single segmented memory module. ** Proposed LUT-multiplier-based design involves a pair of 4-to-3 bit encoders and control-circuits, 2N(W + 4) NOR gates and 4N(W + 4) AOI gates along with those listed above.

Represents the average bit-width of adders rounded to the nearest integers. [a] Indicate bit-level registers and latches. E = $\log_2$ N and E = $\log_2$ k.

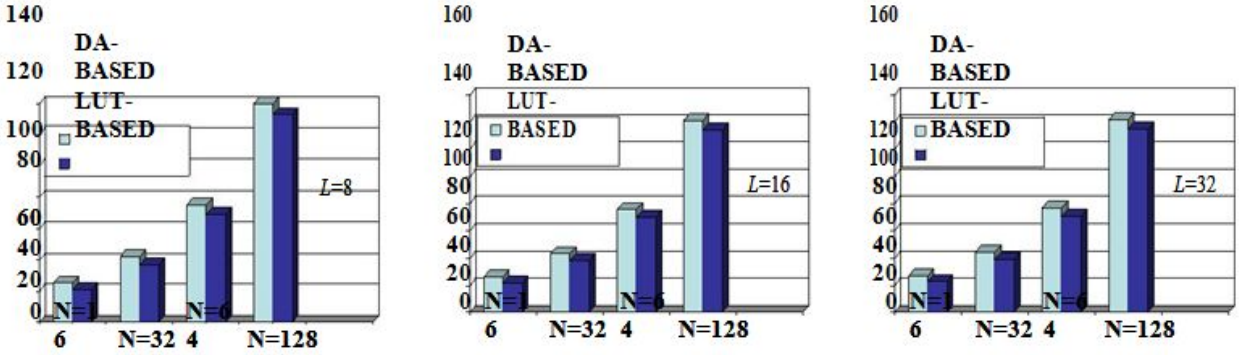


Fig. 11.Latency chart of the DA-based and LUT-multiplier-based FIR filter for different filter orders.

C. Complexity DA-Based Implementation of FIR Filter

The DA-based FIR filter for 8-bit input size has four DA-based sections, one word-serial bit-parallel convertor, and a pipelined shift-add-tree. The word-serial bit-parallel converter consists of an 8-bit register. The pipelined shift-add-tree con-sists of three adders (two adders of (W +2+$\log_2$N)-bit width in the first pipeline-stage and one adder of (W + 4 + $\log_2$ N) width in the second stage). Each section of the DA-based filter requires two bit-level SIPOSRs, (N/4) memory modules of size [16 × (W + 2)] each, (N/4) SA cells (each consisting

of a (W + 2) bit adder followed by (W + 4) bit latch).

The pipelined-adder-tree consists of (N/4 − 1) adders in $\log_2$ N − 2 stages, where the width of the adders and latches increase by one bit after every stage. The expression of critical-path of the DA-based structure is max{T_M^{DA} , T_{SA}^{O}}, where T_M^{DA} is the access-time of the memory modules of size [16 × (W + 2)] and T_{SA}^{O} is the time required by the last adder in the pipeline shift-add-tree that produces the final output. The latency for this structure is (N + $\log_2$ N + 2) cycles and has the same throughput of one output per cycle as that of the LUT-multiplier-based structures. For input word-size L = 8k, it would require k parallel filter sections and a pipeline shift-add-tree as in the case of other designs. The latency for L = 8k becomes (N +$\log_2$ N +$\log_2$ k+2) cycles due to the additional $\log_2$ k cycles involved in the shift-add-tree.

D. Comparative Performances

In Table IV, we have listed the hardware- and time-complexities of DA-based design and LUT-multiplier-based designs. All the structures have the same throughput per cycle. The actual throughput per unit time would, therefore, be higher for the structure with smaller clock period. The duration of minimum clock periods, however, depend on the word-length W , filter order N and the way the adders and the LUT are implemented. The second term in the expressions of critical-path increases with the filter order and coefficient word-length in all the structures, while the first term in the critical-path.

TABLE V
LATENCIES OF LUT-MULTIPLIER-BASED AND DA-BASED DESIGNS FOR DIFFERENT FILTER ORDER N AND INPUT WORD-SIZE L.

N	LUT-Based Design			DA-Based Design		
	L = 8	L = 16	L = 32	L = 8	L = 16	L = 32
8	13	14	15	16	17	18
16	21	22	23	25	26	27
32	37	38	39	42	43	44
64	69	70	71	75	76	77
128	133	134	135	140	141	142

expression is memory access time. The memory access time would, however, be smaller than the output addition-time in many practical situations, and the critical path for all the structures in that case would be equal to the time involved in the output adder T_{SA}^{O}, and accordingly all the structures would have the same throughput rate. The latency of DA-based design is found to be $\log_2$ N cycles more than the LUT-multiplier-based designs. The latencies of DA-based and LUT-multiplier-based structures for different filter orders and different input word-size are listed in Table V, and

plotted in Fig. 11. All the structures have nearly the same complexity of pipeline latches and output register, but the DA-based design has N times higher complexity of input registers. All the structures involve the same number of adders but the LUT-multiplier-based designs have higher adder complexity, since they use adders of slightly higher widths. The LUT-multiplier-based designs involve only one pair of address decoders, while the DA-based design involves N pairs of address decoders. The proposed LUT-multiplier-based design involves half the memory complexity of the conventional LUT-multiplier-based design and the DA-based design at the cost of one pair of 4-to-3 bit encoders and control-circuits, 2N(W +4) NOR gates and 4N(W + 4) AOI gates. The complexities of conventional DA-based FIR filter for unit throughput rate are list in Table VI. It involves less number of adders and latches than the proposed DA-based design, but its memory size as well as the address decoder complexity increases exponentially with filter order, which becomes too large for large filter orders. Moreover, the bit-width of its adders are larger than the proposed DA-based design, and memory access time of conventional DA-based design becomes too high for large values of N due to high decoder complexity. We have synthesized the DA-based design and LUT-multiplier-based designs of 16-tap FIR filters for 8-bit and 16-bit input word-size by Synopsys Design Compiler using TSMC 90nm library. The area complexities of different designs thus obtained from the synthesis result are presented in TableVII. The proposed LUT multiplier-based design is found to involve nearly 15% less area than the DA-based design for the same throughput of computation.

VI. CONCLUSIONS

New approaches to LUT-based-multiplication are suggested to reduce the LUT-size over that of conventional design. By odd-multiple-storage scheme, for address-length 4, the LUT size is reduced to half by using a two-stage logarithmic barrel-shifter and (W + 4) number of NOR gates, where W is the word-length of the fixed multiplying coefficients.

TABLE VI
HARDWARE AND TIME COMPLEXITIES OFCONVENTIONAL DA-BASED FILTER FOR UNIT THROUGHPUT PER CYCLE. (WORD-LENGTH OF COEFFICIENTS = W AND WORD-LENGTH OF INPUT SAMPLES L = 8 × k.)

components	complexities
memory	$8k[(W + E - 1)2^{N/2}]$-bit memory
decoders	$8k \# (N/2) : (2^{N/2})$ decoders
adders	$8k \# (W + E - 1)$-bit adders $[8k - 1] \# (W + E)$-bit adders
registers[a]	$k \# [(N + 1) \times 8]$-bit input-register $(W + 8k + E)$ bit output-register
latches[a]	$(W + E)(3L + \log_2 L - 1) - 2(\log_2 L + 1)$
critical-path	$\max\{T_M^{DA}, T_{SA}^{O}\}$
latency	$(N + \log_2 L + 2)$ cycles
throughput	1 per cycle

Represents the average bit-width of adders rounded to the next integers. [a] Indicate bit-level registers and latches. $E = \log_2 N$.

TABLE VII
AREA COMPLEXITY OF DA-BASED AND LUT-MULTIPLIER-BASED FIR

FILTERS FOR N = 16 AND W = 8.

FIR filter design	input operand width L	area (sq.µm)
DA-based	8	39029.56
	16	78895.96
conventional LUT-multiplier-based	8	40845.77
	16	82519.92
proposed LUT-multiplier-based	8	33880.80
	16	68558.92

Three memory-based structures having one throughput per cycle are designed further 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. The conventional LUT-multiplier-based filter has nearly the same memory requirement and the same number of adders, and less number of input registers than the DA-based design at the cost of higher adder-widths. The LUT-multiplier-based filter involve N times less number of decoders than the DA-based design. The proposed LUT-multiplier-based design involves half the memory than the DA-based and conventional LUT-based designs at the cost of ~ 4N W AOI gates and nearly ~ 2N W NAND/NOR gates. The LUT-multiplier-based design of FIR filter therefore could be more efficient than the DA-based approach in terms of area-complexity for a given throughput and lower latency of im-plementation. From the synthesis result obtained by Synposis Design Compiler using TSMC 90 nm library, we find that the proposed LUT multiplier-based design involves nearly 15% less area than the DA-based design for the implementation of a 16-tap FIR filter having the same throughput per cycle. The proposed LUT-multiplier could be used for memory-based implementation of cyclic and linear convolutions, sinusoidal transforms, and inner-product computation. The performance of memory-based structures, with different adder and memory implementations could be studies in future for different DSP applications. The implementation of DA-based design could be improved further by sign-bit exclusion and OBC techniques, and so also the LUT-multiplier could be improved further by similar techniques. Further work is required to be carried out to find other possibilities of LUT-optimization with different address sizes for efficient memory-based

multiplication.

REFERENCES

[1] J. G. Proakis and D. G. Melonakos, Digital Signal Processing: Princi-ples, Algorithms and Applications. Upper Saddle River, NJ: Prentice-Hall, 1996.

[2] G. Mirchandani, R. L. Zinser, Jr., and J. B. Evans, "A new adaptive noise cancellation scheme in the presence of crosstalk [speech signals]," IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing, vol. 39, no. 10, pp. 681–694, Oct. 1995.

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

[4] K. K. Parhi, VLSI Digital Signal Procesing Systems: Design and Implementation. New York: John Wiley & Sons, Inc, 1999.

[5] H. H. Kha, H. D. Tuan, B.-N. Vo, and T. Q. Nguyen, "Symmetric orthog-onal complex-valued filter bank design by semidefinite programming,"
IEEE Transactions on Signal Processing, vol. 55, no. 9, pp. 4405–4414, Sept. 2007.

[6] H. H. Dam, A. Cantoni, K. L. Teo, and S. Nordholm, "FIR variable digital filter with signed power-of-two coefficients," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 54, no. 6, pp. 1348– 1357, June 2007.

[7] R. Mahesh and A. P. Vinod, "A new common subexpression elimination algorithm for realizing low-complexity higher order digital filters," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 2, pp. 217–229, Feb. 2008.

[8] International Technology Roadmap for Semiconductors. [Online]. Available: http://public.itrs.net/

[9] B. Prince, "Trends in scaled and nanotechnology memories," in Proc. IEEE 2004 Conference on Custom Integrated Circuits, Nov. 2005, p. 7.

[10] K. Itoh, S. Kimura, and T. Sakata, "VLSI memory technology: Current status and future trends," in Proc. 25th European Solid-State Circuits Conference,ESSCIRC '99, Sept. 1999, pp. 3–10.

[11] T. Furuyama, "Trends and challenges of large scale embedded memo-ries," in Proc. IEEE 2004 Conference on Custom Integrated Circuits, Oct. 2004, pp. 449–456.

[12] D. G. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, and R. Mcken-zie, "Computational RAM: implementing processors in memory," IEEE Trans. Design & Test of Computers, vol. 16, no. 1, pp. 32–41, Jan. 1999.

[13] H.-R. Lee, C.-W. Jen, and C.-M. Liu, "On the design automation of the memory-based VLSI architectures for FIR filters," IEEE Trans. Consumer Electronics, vol. 39, no. 3, pp. 619–629, Aug. 1993.

[14] S. A. White, "Applications of the distributed arithmetic to digital signal processing: A tutorial review," IEEE ASSP Magazine, vol. 6, no. 3, pp. 5–19, July 1989.

[15] M. Mehendale, S. D. Sherlekar, and G. Venkatesh, "Area-delay tradeoff in distributed arithmetic based implementation of FIR filters," in Proc. Tenth International Conference on VLSI Design, Jan. 1997, pp. 124–129.

[16] H. Yoo and D. V. Anderson, "Hardware-efficient distributed arithmetic architecture for high-order digital filters," in Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP '05), vol. 5, Mar. 2005, pp. v/125–v/128.

[17] 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 and Systems I: Regular Papers, vol. 52, no. 7, pp. 1327–1337, July 2005.

[18] S.-S. Jeng, H.-C. Lin, and S.-M. Chang, "FPGA implementation of FIR filter using M-bit parallel distributed arithmetic," in Proc. 2006 IEEE Intl. Symp. on Circuits and Systems, ISCAS 2006, May 2006, p. 4.

[19] Y. H. Chan and W. C. Siu, "On the realization of discrete cosine transform using the distributed arithmetic," IEEE Trans. Circuits Syst. I: Fundam. Theory Appl., vol. 39, no. 9, pp. 705–712, Sept. 1992.

[20] H.-C. Chen, J.-I. Guo, T.-S. Chang, and C.-W. Jen, "A memory-efficient realization of cyclic convolution and its application to discrete cosine transform," IEEE Trans. Circuits Syst for Video Technol., vol. 15, no. 3, pp. 445–453, Mar. 2005.

[21] M. Z. Zhang and V. K. Asari, "A fully pipelined multiplierlessarchi-tecture for 2D convolution with quadrant symmetric kernels," in Proc. IEEE Asia Pacific Conference on Circuits and Systems, APCCAS 2006, Dec. 2006, pp. 1559–1562.

[22] P. K. Meher, S. Chandrasekaran, and A. Amira, "FPGA realization of FIR filters by efficient and flexible systolization using distributed arithmetic," IEEE Trans. Signal Processing, vol. 56, no. 7, pp. 3009–3017, July 2008.

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

[24] J.-I. Guo, C.-M. Liu, and C.-W. Jen, "The efficient memory-based VLSI array design for DFT and DCT," IEEE Trans. Circuits and Syst. II: Analog and Digital Signal Process., vol. 39, no. 10, pp. 723–733, Oct. 1992.

[25] D. F. Chiper, "A systolic array algorithm for an efficient unified memory-based implementation of the inverse discrete cosine transform," in IEEE Conf. Image Processing, Oct. 1999, pp. 764–768.

[26] D. F. Chiper, M. N. S. Swamy, M. O. Ahmad, and T. Stouraitis, "Systolic algorithms and a memory-based design approach for a unified architecture for the computation of DCT/DST/IDCT/IDST," IEEE Trans. Circuits Syst-I: Regular Papers, vol. 52, no. 6, pp. 1125–1137, June 2005.

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

[28] P. K. Meher, J. C. Patra, and M. N. S. Swamy, "High-throughput memory-based architecture for DHT using a new convolutional formu-lation," IEEE Trans. Circuits Syst. II: Express Briefs, vol. 54, no. 7, pp. 606–610, July 2007.

[29] J. P. Choi, S.-C. Shin, and J.-G. Chung, "Efficient ROM size reduction for distributed arithmetic," in Proc. IEEE International Symp on Circuits and Syst., 2000. ISCAS, vol. 2, May 2000, pp. 61–64.

[30] P. K. Meher, "New approach to LUT implementation and accumulation for memory-based multiplication," in Proc. 2009 IEEE International Symposium on Circuits and Systems, ISCAS '09, May 2009, pp. 453– 456.

[31] A. K. Sharma, Advanced Semiconductor Memories : Architectures, Designs, and Applications. IEEE Press, Piscataway, NJ and Wiley-Interscience, Hoboken, NJ, 2003.

[32] E. John, "Semiconductor memory circuits," in Digital Design and Fabrication, V. G. Oklobdzija, Ed. New York: CRC Press, 2008.

[33] "Synposys, DesignWare. Foundry Libraries, Mountain View, CA." [Online]. Available: http://www.synopsys.com/

[34] P. K. Meher, "New look-up-table optimizations for memory-based mul-tiplication," Submiteed to The International Symposium on Integrated Circuits, (ISIC '09) ' to be held in December 2009.