LUT Optimization for Memory-Based Computation

Pramod Kumar Meher, *Senior Member, IEEE*

***Abstract*—Recently, we have proposed the antisymmetric product coding (APC) and odd-multiple-storage (OMS) techniques for lookup-table (LUT) design for memory-based multipliers to be used in digital signal processing applications. Each of these techniques results in the reduction of the LUT size by a factor of two. In this brief, we present a different form of APC and a modified OMS scheme, in order to combine them for efficient memory-based multiplication. The proposed combined approach provides a reduction in LUT size to one-fourth of the conventional LUT. We have also suggested a simple technique for selective sign reversal to be used in the proposed design. It is shown that the proposed LUT design for small input sizes can be used for efficient implementation of high-precision multiplication by input operand decomposition. It is found that the proposed LUT-based multiplier involves comparable area and time complexity for a word size of 8 bits, but for higher word sizes, it involves significantly less area and less multiplication time than the canonical-signed-digit (CSD)-based multipliers. For 16- and 32-bit word sizes, respectively, it offers more than 30% and 50% of saving in area–delay product over the corresponding CSD multipliers.**

***Index Terms*—Digital signal processing (DSP) chip, lookup-table (LUT)-based computing, memory-based computing, very large scale integration (VLSI).**

I. Introduction

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 [1], 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 is 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 low-latency implementation and less dynamic power consumption. Memory-based computing is well suited for many digital signal processing (DSP) algorithms, which involve multiplication with a fixed set of coefficients.

A conventional lookup-table (LUT)-based multiplier is shown in Fig. 1, where A is a fixed coefficient, and X is an input word to be multiplied with A. Assuming X to be a positive 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 $C = A \cdot X$. Therefore, for memory-based multiplication, an LUT of 2^L words, consisting of precomputed product values corresponding to all possible values of X, is conventionally used. The product word $A \cdot X_i$ is stored at the location X_i for $0 \leq X_i \leq 2^L - 1$, such that if an L-bit binary value of X_i is used as the address for the LUT, then the corresponding product value $A \cdot X_i$ is available as its output.

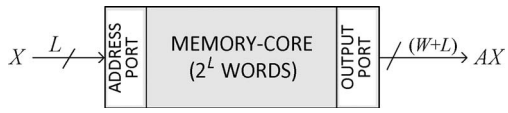

Fig. 1. Conventional LUT-based multiplier.

Several architectures have been reported in the literature for memory-based implementation of DSP algorithms involving orthogonal transforms and digital filters [2]–[8]. However, we do not find any significant work on LUT optimization for memory-based multiplication. Recently, we have presented a new approach to LUT design, where only the odd multiples of the fixed coefficient are required to be stored [9], which we have referred to as the *odd-multiple-storage* (OMS) scheme in this brief. In addition, we have shown that, by the *antisymmetric product coding* (APC) approach, the LUT size can also be reduced to half, where the product words are recoded as antisymmetric pairs [10].

The APC approach, although providing a reduction in LUT size by a factor of two, incorporates substantial overhead of area and time to perform the two's complement operation of LUT output for sign modification and that of the input operand for input mapping. However, we find that when the APC approach is combined with the OMS technique, the two's complement operations could be very much simplified since the input address and LUT output could always be transformed into odd integers.[1] However, the OMS technique in [9] cannot be combined with the APC scheme in [10], since the APC words generated according to [10] are odd numbers. Moreover, the OMS scheme in [9] does not provide an efficient implementation when combined with the APC technique. In this brief, we therefore present a different form of APC and combined that with a modified form of the OMS scheme for efficient memory-based multiplication.

In the next section, we have discussed the modified APC and the combined OMS–APC approach. The implementation of combined OMS–APC scheme is described in Section III, and the design of the LUT-based multiplier for high input precision is discussed in Section IV. The synthesis results of the proposed multiplier and canonical-signed-digit (CSD)-based multipliers, along with the conclusion, are presented in Section V.

Manuscript received June 25, 2009; revised November 14, 2009. Current version published April 21, 2010. This paper was recommended by Associate Editor V. Gaudet.

The author is with the Department of Communication Systems, Institute for Infocomm Research, Singapore 138632 (e-mail: pkmeher@i2r.a-star.edu.sg).

Digital Object Identifier 10.1109/TCSII.2010.2043467

[1]We have shown that it is simpler to perform the two's complement operation of the binary equivalent of an odd integer compared to that of any general integer.

TABLE I
APC WORDS FOR DIFFERENT INPUT VALUES FOR $L = 5$

Input, X	product values	Input, X	product values	address $x'_3x'_2x'_1x'_0$	APC words
0 0 0 0 1	A	1 1 1 1 1	$31A$	1 1 1 1	$15A$
0 0 0 1 0	$2A$	1 1 1 1 0	$30A$	1 1 1 0	$14A$
0 0 0 1 1	$3A$	1 1 1 0 1	$29A$	1 1 0 1	$13A$
0 0 1 0 0	$4A$	1 1 1 0 0	$28A$	1 1 0 0	$12A$
0 0 1 0 1	$5A$	1 1 0 1 1	$27A$	1 0 1 1	$11A$
0 0 1 1 0	$6A$	1 1 0 1 0	$26A$	1 0 1 0	$10A$
0 0 1 1 1	$7A$	1 1 0 0 1	$25A$	1 0 0 1	$9A$
0 1 0 0 0	$8A$	1 1 0 0 0	$24A$	1 0 0 0	$8A$
0 1 0 0 1	$9A$	1 0 1 1 1	$23A$	0 1 1 1	$7A$
0 1 0 1 0	$10A$	1 0 1 1 0	$22A$	0 1 1 0	$6A$
0 1 0 1 1	$11A$	1 0 1 0 1	$21A$	0 1 0 1	$5A$
0 1 1 0 0	$12A$	1 0 1 0 0	$20A$	0 1 0 0	$4A$
0 1 1 0 1	$13A$	1 0 0 1 1	$19A$	0 0 1 1	$3A$
0 1 1 1 0	$14A$	1 0 0 1 0	$18A$	0 0 1 0	$2A$
0 1 1 1 1	$15A$	1 0 0 0 1	$17A$	0 0 0 1	A
1 0 0 0 0	$16A$	1 0 0 0 0	$16A$	0 0 0 0	0

For $X = (0\ 0\ 0\ 0\ 0)$, the encoded word to be stored is $16A$.

II. PROPOSED LUT OPTIMIZATIONS FOR MEMORY-BASED MULTIPLICATION

We discuss here the proposed APC technique and its further optimization by combining it with a modified form of OMS.

A. APC for LUT Optimization

For simplicity of presentation, we assume both X and A to be positive integers.[2] The product words for different values of X for $L = 5$ are shown in Table I. It may be observed in this table that the input word X on the first column of each row is the two's complement of that on the third column of the same row. In addition, the sum of product values corresponding to these two input values on the same row is $32A$. Let the product values on the second and fourth columns of a row 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 $(u+v) = 32A$, we can have

$$u = 16A - \left[\frac{v-u}{2}\right] \quad v = 16A + \left[\frac{v-u}{2}\right]. \tag{1}$$

The product values on the second and fourth columns of Table I therefore have a *negative mirror symmetry*. This behavior of the product words can be used to reduce the LUT size, where, instead of storing u and v, only $[(v-u)/2]$ is stored for a pair of input on a given row. The 4-bit LUT addresses and corresponding coded words are listed on the fifth and sixth columns of the table, respectively. Since the representation of the product is derived from the antisymmetric behavior of the products, we can name it as *antisymmetric product code*. The 4-bit address $X' = (x'_3x'_2x'_1x'_0)$ of the APC word is given by

$$X' = \begin{cases} X_L, & \text{if } x_4 = 1 \\ X'_L, & \text{if } x_4 = 0 \end{cases} \tag{2}$$

where $X_L = (x_3x_2x_1x_0)$ is the four less significant bits of X, and X'_L is the two's complement of X_L. The desired product could be obtained by adding or subtracting the stored value $(v-u)$ to or from the fixed value $16A$ when x_4 is 1 or 0, respectively, i.e.,

$$\text{Product word} = 16A + (\text{sign value}) \times (\text{APC word}) \tag{3}$$

where sign value $= 1$ for $x_4 = 1$ and sign value $= -1$ for $x_4 = 0$. The product value for $X = (10000)$ corresponds to APC value "zero," which could be derived by resetting the LUT output, instead of storing that in the LUT.

[2]It could, however, be easily extended for signed values of A and X in sign-magnitude form or two's complement form.

TABLE II
OMS-BASED DESIGN OF THE LUT OF APC WORDS FOR $L = 5$

input X' $x'_3x'_2x'_1x'_0$	product value	# of shifts	shifted input, X''	stored APC word	address $d_3d_2d_1d_0$
0 0 0 1	A	0	0 0 0 1	$P0 = A$	0 0 0 0
0 0 1 0	$2 \times A$	1			
0 1 0 0	$4 \times A$	2			
1 0 0 0	$8 \times A$	3			
0 0 1 1	$3A$	0	0 0 1 1	$P1 = 3A$	0 0 0 1
0 1 1 0	$2 \times 3A$	1			
1 1 0 0	$4 \times 3A$	2			
0 1 0 1	$5A$	0	0 1 0 1	$P2 = 5A$	0 0 1 0
1 0 1 0	$2 \times 5A$	1			
0 1 1 1	$7A$	0	0 1 1 1	$P3 = 7A$	0 0 1 1
1 1 1 0	$2 \times 7A$	1			
1 0 0 1	$9A$	0	1 0 0 1	$P4 = 9A$	0 1 0 0
1 0 1 1	$11A$	0	1 0 1 1	$P5 = 11A$	0 1 0 1
1 1 0 1	$13A$	0	1 1 0 1	$P6 = 13A$	0 1 1 0
1 1 1 1	$15A$	0	1 1 1 1	$P7 = 15A$	0 1 1 1

B. Modified OMS for LUT Optimization

It is shown in [9] that, for the multiplication of any binary word X of size L, with a fixed coefficient A, instead of storing all the 2^L possible values of $C = A \cdot X$, only $(2^L/2)$ words corresponding to the odd multiples of A may be stored in the LUT, while all the even multiples of A could be derived by left-shift operations of one of those odd multiples. Based on the above assumptions, the LUT for the multiplication of an L-bit input with a W-bit coefficient could be designed by the following strategy.

1) A memory unit of $[(2^L/2) + 1]$ words of $(W + L)$-bit width is used to store the product values, where the first $(2^L/2)$ words are odd multiples of A, and the last word is zero.
2) A barrel shifter for producing a maximum of $(L-1)$ left shifts is used to derive all the even multiples of A.
3) The L-bit input word is mapped to the $(L-1)$-bit address of the LUT by an address encoder, and control bits for the barrel shifter are derived by a control circuit.

In Table II, we have shown that, at eight memory locations, the eight odd multiples, $A \times (2i+1)$ are stored as P_i, for $i = 0, 1, 2, \ldots, 7$. The even multiples $2A$, $4A$, and $8A$ 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. A barrel shifter for producing a maximum of three left shifts could be used to derive all the even multiples of A.

As required by (3), the word to be stored for $X = (00000)$ is not 0 but $16A$, which we can obtain from A by four left shifts

TABLE III
PRODUCTS AND ENCODED WORDS FOR $X = (00000)$ AND (10000)

input X $x_4x_3x_2x_1x_0$	product values	encoded word	stored values	# of shifts	address $d_3d_2d_1d_0$
1 0 0 0 0	$16A$	0	– – –	– –	– – –
0 0 0 0 0	0	$16A$	$2A$	3	1 0 0 0

using a barrel shifter. However, if $16A$ is not derived from A, only a maximum of three left shifts is required to obtain all other even multiples of A. A maximum of three bit shifts can be implemented by a two-stage logarithmic barrel shifter, but the implementation of four shifts requires a three-stage barrel shifter. Therefore, it would be a more efficient strategy to store $2A$ for input $X = (00000)$, so that the product $16A$ can be derived by three arithmetic left shifts.

The product values and encoded words for input words $X = (00000)$ and (10000) are separately shown in Table III. For $X = (00000)$, the desired encoded word $16A$ is derived by 3-bit left shifts of $2A$ [stored at address (1000)]. For $X = (10000)$, the APC word "0" is derived by resetting the LUT output, by an active-high RESET signal given by

$$\text{RESET} = (\overline{x_0 + x_1 + x_2 + x_3}) \cdot x_4. \tag{4}$$

It may be seen from Tables II and III that the 5-bit input word X can be mapped into a 4-bit LUT address $(d_3d_2d_1d_0)$, by a simple set of mapping relations

$$d_i = x''_{i+1}, \qquad \text{for } i = 0, 1, 2 \quad \text{and} \quad d_3 = \overline{x''_0} \tag{5}$$

where $X'' = (x''_3x''_2x''_1x''_0)$ is generated by shifting-out all the leading zeros of X' by an arithmetic right shift followed by address mapping, i.e.,

$$X'' = \begin{cases} Y_L, & \text{if } x_4 = 1 \\ Y'_L, & \text{if } x_4 = 0 \end{cases} \tag{6}$$

where Y_L and Y'_L are derived by circularly shifting-out all the leading zeros of X_L and X'_L, respectively.[3]

III. IMPLEMENTATION OF THE LUT-BASED MULTIPLIER USING THE PROPOSED LUT OPTIMIZATION SCHEME

In this section, we discuss the implementation of the LUT-based multiplier using the proposed scheme, where the LUT is optimized by a combination of the proposed APC scheme and a modified OMS technique.

A. Implementation of the LUT Multiplier Using APC for $L = 5$

The structure and function of the LUT-based multiplier for $L = 5$ using the APC technique is shown in Fig. 2. It consists of a four-input LUT of 16 words to store the APC values of product words as given in the sixth column of Table I, except on the last row, where $2A$ is stored for input $X = (00000)$ instead of storing a "0" for input $X = (10000)$. Besides, it consists of an address-mapping circuit and an add/subtract circuit. The address-mapping circuit generates the desired address $(x'_3x'_2x'_1x'_0)$ according to (2). A straightforward implementation of address mapping can be done by multiplexing X_L and

[3]Note that, in the case of $X = (00000)$ and $X = (10000)$, the circular shifting does not make any difference to X_L and X'_L.

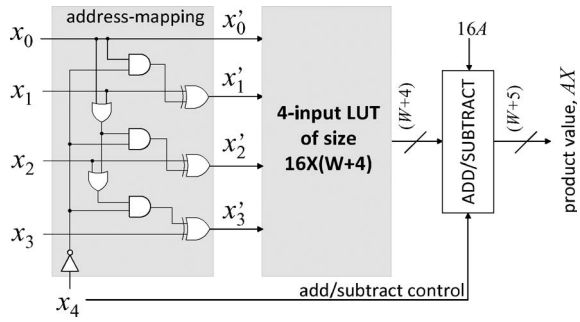

Fig. 2. LUT-based multiplier for $L = 5$ using the APC technique.

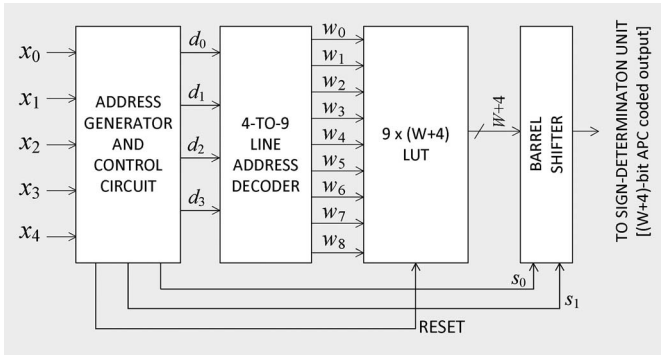

Fig. 3. Proposed APC–OMS combined LUT design for the multiplication of W-bit fixed coefficient A with 5-bit input X.

X'_L using x_4 as the control bit. The address-mapping circuit, however, can be optimized to be realized by three XOR gates, three AND gates, two OR gates, and a NOT gate, as shown in Fig. 2. Note that the RESET can be generated by a control circuit (not shown in this figure) according to (4). The output of the LUT is added with or subtracted from $16A$, for $x_4 = 1$ or 0, respectively, according to (3) by the add/subtract cell. Hence, x_4 is used as the control for the add/subtract cell.

B. Implementation of the Optimized LUT Using Modified OMS

The proposed APC–OMS combined design of the LUT for $L = 5$ and for any coefficient width W is shown in Fig. 3. It consists of an LUT of nine words of $(W + 4)$-bit width, a four-to-nine-line address decoder, a barrel shifter, an address-generation circuit, and a control circuit for generating the RESET signal and control word (s_1s_0) for the barrel shifter.

The precomputed values of $A \times (2i + 1)$ are stored as P_i, for $i = 0, 1, 2, \ldots, 7$, at the eight consecutive locations of the memory array, as specified in Table II, while $2A$ is stored for input $X = (00000)$ at LUT address "1000," as specified in Table III. The decoder takes the 4-bit address from the address generator and generates nine word-select signals, i.e., $\{w_i, \text{for } 0 \le i \le 8\}$, to select the referenced word from the LUT. The 4-to-9-line decoder is a simple modification of 3-to-8-line decoder, as shown in Fig. 4(a). The control bits s_0 and s_1 to be used by the barrel shifter to produce the desired number of shifts of the LUT output are generated by the control circuit, according to the relations

$$s_0 = \overline{x_0 + \overline{(x_1 + \overline{x_2})}} \tag{7a}$$

$$s_1 = \overline{(x_0 + x_1)}. \tag{7b}$$

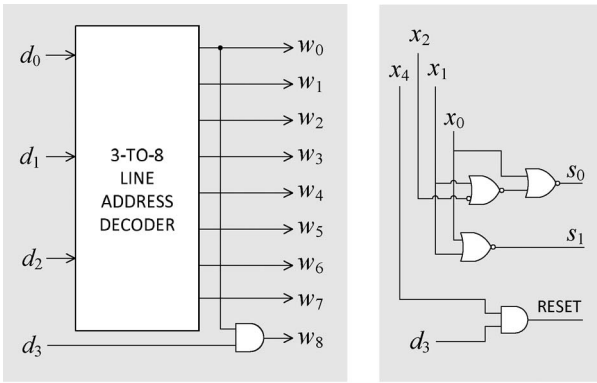

Fig. 4. (a) Four-to-nine-line address-decoder. (b) Control circuit for generation of s_0, s_1, and RESET.

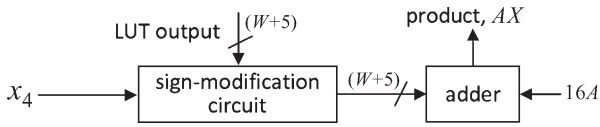

Fig. 5. Modification of the add/subtract cell in Fig. 2 for the two's complement representation of product words.

Note that $(s_1 s_0)$ is a 2-bit binary equivalent of the required number of shifts specified in Tables II and III. The RESET signal given by (4) can alternatively be generated as (d_3 AND x_4). The control circuit to generate the control word and RESET is shown in Fig. 4(b). The address-generator circuit receives the 5-bit input operand X and maps that onto the 4-bit address word $(d_3 d_2 d_1 d_0)$, according to (5) and (6). A simplified address generator is presented later in this section.

C. Optimized LUT Design for Signed and Unsigned Operands

The APC–OMS combined optimization of the LUT can also be performed for signed values of A and X. When both operands are in sign-magnitude form, the multiples of magnitude of the fixed coefficient are to be stored in the LUT, and the sign of the product could be obtained by the XOR operation of sign bits of both multiplicands. When both operands are in two's complement forms, a two's complement operation of the output of the LUT is required to be performed for $x_4 = 1$. There is no need to add the fixed value $16A$ in this case, because the product values are naturally in antisymmetric form. The add/subtract circuit is not required in Fig. 2, instead of that a circuit is required to perform the two's complement operation of the LUT output. For the multiplication of unsigned input X with signed, as well as unsigned, coefficient A, the products could be stored in two's complement representation, and the add/subtract circuit in Fig. 2 could be modified as shown in Fig. 5. A straightforward implementation of sign-modification circuit involves multiplexing of the LUT output and its two's complement. To reduce the area–time complexity over such straightforward implementation, we discuss here a simple design for sign modification of the LUT output.

Note that, except the last word, all other words in the LUT are odd multiples of A. The fixed coefficient could be even or odd, but if we assume A to be an odd number, then the all the stored product words (except the last one) would be odd. If the stored value P is an odd number, it can be expressed as

$$P = P_{D-1}\, P_{D-2}\, \cdots\, P_1\, 1 \tag{8}$$

and its two's complement is given by

$$P' = P'_{D-1}\, P'_{D-2}\, \cdots\, P'_1\, 1 \tag{9}$$

where P'_i is the one's complement of P_i for $1 \le i \le D-1$, and $D = W + L - 1$ is the width of the stored words. If we store the two's complement of all the product values and change the sign of the LUT output for $x_4 = 1$, then the sign of the last LUT word need not be changed. Based on (9), we can therefore have a simple sign-modification circuit [shown in Fig. 6(a)] when A is an odd integer. However, the fixed coefficient A could be even as well. When A is a nonzero even integer, we can express it as $A' \times 2^l$, where $1 \le l \le D-1$ is an integer, and A' is an odd integer. Instead of storing multiples of A, we can store multiples of A' in the LUT, and the LUT output can be left shifted by l bits by a hardwired shifter. Similarly, using (5) and (6), we can have an address-generation circuit as shown in Fig. 6(b), since all the shifted-address Y_L (except the last one) is an odd integer.

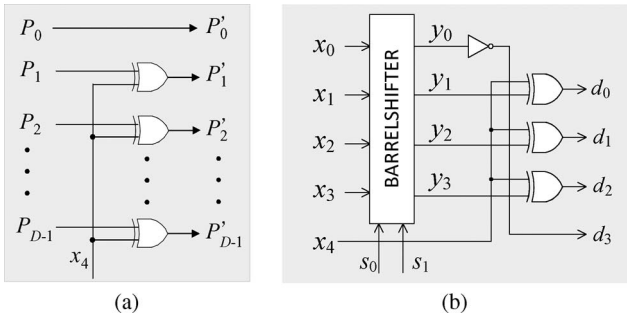

Fig. 6. (a) Optimized implementation of the sign modification of the odd LUT output. (b) Address-generation circuit.

IV. Memory-Based Multiplication by Input Operand Decomposition

Although the memory core of the LUT multiplier is reduced to nearly one-fourth by the proposed optimization technique, it is not efficient for operands of small widths, since it requires an adder to add the offset value. However, it could be used for multiplication with input of large word size by an input decomposition scheme. When the width of the input multiplicand X is large, direct implementation of LUT multiplier involves a very large LUT. Therefore, the input word could be decomposed into a certain number of segments or subwords, and the partial products pertaining to different subwords could be shift added to obtain the desired product as discussed in the following.

Let the input operand X be decomposed into T subwords, i.e., $\{X_1, X_2, \ldots, X_T\}$, where each $X_i = \{x_{(i-1)S} x_{(i-1)S+1}, \ldots, x_{iS-1}\}$ is an S-bit subword, for $1 \le i \le T-1$, and $X_T = \{x_{(T-1)S}, x_{(T-1)S+1}, \ldots, x_{(T-1)S+S'-1}\}$ is the last subword of S' bit, where $S' \le S$, and x_i is the $(i+1)$th bit of X. The

TABLE IV
AREA AND TIME COMPLEXITIES OF LUT MULTIPLIERS FOR DIFFERENT WORD LENGTHS

Word Size	Addition Scheme	CSD-Based Multiplier			Proposed LUT-Based Multiplier					ADP Saving
		Area	Delay	ADP	LUT-Area	Total Area	LUT-Time	Total-Time	ADP	
8-bit	Wallace-Tree	700.66	2.65	1856.7	– – –	– – –	– – –	– – –	– – –	– – –
	Ripple-Carry	712.65	3.09	2202.1	340.80	654.09	0.97	2.97	1942.7	11.78 %
16-bit	Wallace-Tree	3270.45	5.05	16515.8	1011.83	2679.16	0.98	5.13	11118.5	32.68 %
	Ripple-Carry	3293.74	6.03	19861.3	1011.83	2679.16	0.98	5.90	13181.5	33.63 %
32-bit	Wallace-Tree	14380.13	11.15	160338.4	3052.43	9171.39	1.02	8.99	73096.0	54.41 %
	Ripple-Carry	14382.95	12.22	175759.6	3052.43	9171.39	1.02	10.34	85477.3	51.37 %

For word-size $L = 8$, the proposed LUT-based multiplier results in two LUT outputs, those are added by a ripple-carry adder; and Wallace-tree reduction is not required in this case. Areas are measured in um^2 and delays are measured in nano-seconds. The fixed coefficients in CSD representation are assumed to have maximum non-zero bits.

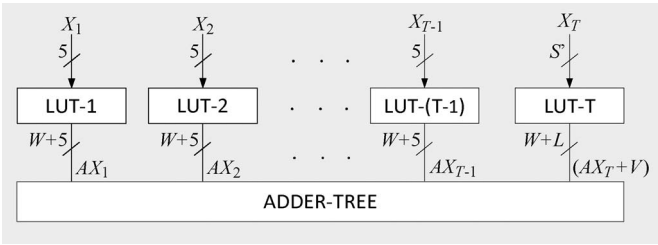

Fig. 7. Proposed LUT-based multiplier for $L = 5(T-1) + S'$.

product word $C = A \cdot X$ can be written as the sum of partial products as

$$C = \sum_{i=1}^{T} 2^{S(i-1)} \cdot C_i \tag{10a}$$

$$C_i = A \cdot X_i, \qquad \text{for } 1 \le i \le T. \tag{10b}$$

A generalized structure for parallel implementation of LUT multiplication for input size $L = 5 \times (T-1) + S'$ is shown in Fig. 7, where $S' < 5$. The input multiplicand X is decomposed into $(T-1)$ more significant subwords $X_1, X_2, \ldots, X_{T-1}$ and less significant S'-bit subword X_T. The partial products $C_i = A.X_i$, for $1 \le i \le (T-1)$, are obtained from $(T-1)$ LUT multipliers optimized by APC and OMS. The Tth LUT multiplier is a conventional LUT, which is not optimized by APC and OMS, since it is required to store the sum of offset values $V = 16A \times 2^{S'}[(2^{5(T-1)} - 1)/(2^5 - 1)]$ pertaining to all the $(T-1)$ optimized LUTs for partial product generation.[4] The Tth LUT therefore stores the values $(A \cdot X_T + V)$. The sign of all the optimized LUT outputs are modified according to the value of the most significant bit of corresponding subword X_i, for $1 \le i \le T-1$, and all the LUT outputs are added together by an adder tree, as shown in Fig. 7.

V. RESULTS AND DISCUSSION

The proposed LUT multipliers for word size $L = W = 8$, 16, and 32 bits are coded in VHDL and synthesized by Synopsys Design Compiler using the TSMC 90-nm library, where the LUTs are implemented as arrays of constants, and additions are implemented by the Wallace tree and ripple carry array. The CSD-based multipliers having the same addition schemes are also synthesized with the same technology library. The area and delay complexities of the multipliers estimated from the synthesis results are listed in Table IV. It is found that the proposed LUT design involves comparable area and time complexities for a word size of 8 bits, but for higher word sizes, it involves significantly less area and less multiplication time than the CSD-based multiplier. For $L = W = 16$, and 32 bits, respectively, it offers more than 30% and 50% of saving in area–delay product (ADP) over the CSD multiplier.

In this brief, we have shown the possibility of using LUT-based multipliers to implement the constant multiplication for DSP applications. The full advantages of proposed LUT-based design, however, could be derived if the LUTs are implemented as NAND or NOR read-only memories and the arithmetic shifts are implemented by an array barrel shifter using metal–oxide–semiconductor transistors [11]. Further work could still be done to derive OMS–APC-based LUTs for higher input sizes with different forms of decompositions and parallel and pipelined addition schemes for suitable area–delay tradeoffs.

[4]It refers to the sum of the $(T-1)$ terms in geometric progression with the first term $(2^{S'} \times 16A)$ and ratio 2^5.

REFERENCES

[1] International Technology Roadmap for Semiconductors. [Online]. Available: http://public.itrs.net/
[2] J.-I. Guo, C.-M. Liu, and C.-W. Jen, "The efficient memory-based VLSI array design for DFT and DCT," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 39, no. 10, pp. 723–733, Oct. 1992.
[3] 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. Consum. Electron.*, vol. 39, no. 3, pp. 619–629, Aug. 1993.
[4] D. F. Chiper, M. N. S. Swamy, M. O. Ahmad, and T. Stouraitis, "A systolic array architecture for the discrete sine transform," *IEEE Trans. Signal Process.*, vol. 50, no. 9, pp. 2347–2354, Sep. 2002.
[5] 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. Video Technol.*, vol. 15, no. 3, pp. 445–453, Mar. 2005.
[6] 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, Reg. Papers*, vol. 52, no. 6, pp. 1125–1137, Jun. 2005.
[7] P. K. Meher, "Systolic designs for DCT using a low-complexity concurrent convolutional formulation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 9, pp. 1041–1050, Sep. 2006.
[8] P. K. Meher, "Memory-based hardware for resource-constrained digital signal processing systems," in *Proc. 6th Int. Conf. ICICS*, Dec. 2007, pp. 1–4.
[9] P. K. Meher, "New approach to LUT implementation and accumulation for memory-based multiplication," in *Proc. IEEE ISCAS*, May 2009, pp. 453–456.
[10] P. K. Meher, "New look-up-table optimizations for memory-based multiplication," in *Proc. ISIC*, Dec. 2009, pp. 663–666.
[11] A. K. Sharma, *Advanced Semiconductor Memories: Architectures, Designs, and Applications*. Piscataway, NJ: IEEE Press, 2003.