Circuits Syst Signal Process (2017) 36:1129–1149
DOI 10.1007/s00034-016-0349-9

Efficient Design for Radix-8 Booth Multiplier and Its Application in Lifting 2-D DWT

Basant Kumar Mohanty[1] · **Abhishek Choubey**[1]

Received: 21 February 2015 / Revised: 29 May 2016 / Accepted: 31 May 2016 /
Published online: 18 June 2016

Abstract In this paper, we present a regular partial product array (PPA) for radix-8 Booth multiplication by removing the extra row with a small overhead complexity. A radix-8 multiplier design is proposed based on the regular PPA which offers a saving of 10.7 % area-delay product (ADP) over the existing radix-8 multiplier design. The n lower-order bits of $2n$ bit output of full-width multiplier are truncated to have a fixed-width multiplier with low truncation error, where n is the operand bit-width. Few redundant logic operations are created in the adder unit when n lower-order bits of $2n$-bit multiplier output are truncated. A specific design is necessary as the modern synthesis tools partially remove these redundant logics. We present an optimized adder unit design after removing redundant logic for post-truncated fixed-width radix-8 Booth multiplier. Comparison result shows that the proposed post-truncated fixed-width multiplier design offers nearly 20.7 % ADP and 18.3 % power saving over the existing radix-8 design optimized by the Synopsys Design Compiler when $2n$-bit output is post-truncated to n-bit. More often, multipliers are used for multiplication of constant. The value of the constant may be fixed or could be changed during run-time by the user. The multiplier that multiplies fixed constant is referred to *fixed-constant multiplier* and that multiplies constant which changes during run-time is referred to *generic-constant multiplier*. Both radix-4 and radix-8 Booth multiplier designs easily can be configured for a generic-constant multiplier. However, radix-8 multiplier design offers to save some area and delay when configured for constant multiplication, while the radix-4 multiplier design does not have this feature. We find

✉ Abhishek Choubey
abhishekchoubey84@gmail.com

Basant Kumar Mohanty
bk.mohanti@juet.ac.in

[1] Department of Electronics and Communication Engineering, Jaypee University of Engineering and Technology, AB-Road, Raghogarh, Guna, Madhya Pradesh 473226, India

that the proposed 12-bit full-width and fixed-width radix-8 generic-constant multiplier designs, respectively, involve 19.4 and 24.7 % less ADP than the existing radix-4 full-width and post-truncated multiplier designs configured for constant multiplication. The existing block-based lifting 2-D DWT structure is synthesized using the proposed radix-8 generic-constant fixed-width multiplier design to demonstrate the effectiveness of proposed multiplier designs. We find that the existing lifting 2-D DWT structure of block size 16 and word length 12 offers 19.3 % ADP saving and 11.5 % power saving when the constant multipliers are implemented using the proposed radix-8 multiplier design instead of the existing radix-4 multiplier design.

Keywords Booth multiplier · Lifting scheme · Discrete wavelet transforms · VLSI architecture

1 Introduction

The multiplier is a widely used component for digital signal processing (DSP) applications such as sinusoidal transforms, fast Fourier transform (FFT), discrete wavelet transform, finite impulse response (FIR) filters and adaptive filters [2,9]. More often, these algorithms are implemented in dedicated very-large-scale integration (VLSI) systems for real-time and low-power applications. Multiplier is the most resource-consuming component in the arithmetic unit and contributes significantly to the chip area and critical path. Several algorithms have been suggested in the literature for efficient implementation of multipliers in hardware. Among these, Booth multiplication algorithm is the most widely used. There are several variants of Booth algorithms that have been proposed in the literature to reduce the partial products. The higher-radix Booth multiplication algorithms reduce the number of partial product rows of the partial product array (PPA) which helps to reduce the adder unit complexity, but higher-radix Booth algorithms increase the complexity of partial product generator (PPG) and partial product selector (PPS). Consequently, the area-delay efficiency of higher-radix Booth multiplier designs is not better than the radix-4 Booth multiplier design. Therefore, radix-4 Booth multiplication algorithm is mostly preferred to design multipliers for various DSP algorithms.

Few attempts have been made to reduce the complexity of radix-8 Booth multiplier design [1,3]. A decoder-based design has been proposed to reduce the complexity of PPS unit of radix-8 Booth multiplier design [3]. The radix-8 multiplier design of [3] uses $\{w + 1\}$ number of partial product rows, where $w = \lceil n/3 \rceil$; n is the multiplier operand bit-width. Ideally, the PPA should have w rows when radix-8 Booth algorithm is used. The extra row is included in the PPA of radix-8 multiplier design to add the sign bit corresponding to the wth partial product row. The extra partial product row involves one extra adder and adder stage which increases the logic complexity and delay of the adder unit. We observe that the extra partial product row can be avoided and a regular PPA of w rows can be obtained in the radix-8 Booth multiplication with a small overhead complexity. The regular PPA helps to reduce the multiplier complexity and delay. In this paper, we present the necessary logic expressions to produce a regular PPA for radix-8 Booth multiplication. We have shown that the ADP

of radix-8 multiplier design is reduced by 10.7 % than the ADP of the existing radix-8 multiplier design of [3].

Several multiplication operations are performed in DSP algorithm. Fixed-width multipliers are used to implement these algorithms in VLSI systems to avoid infinite growth of bit-width. A fixed-width multiplier produces output of the same bit-width as its input operand. The full-width multiplier design is approximated to have the fixed-width multiplier design. The full-width multiplier design can be approximated by (1) direct post-truncation of $2n$-bit output referred to *post-truncated* multiplier, and (2) truncation of PPA referred to *truncated* multiplier. The truncated multiplier involves nearly half of the partial products of those used by full-width multiplier. Therefore, the truncated multiplier involves nearly half of the logic resource of those required by full-width multiplier, but it introduces a large amount of truncation error at the multiplier output due to truncation of carry bits corresponding to n least significant columns (LSC) of full-width multiplier [7]. Several error-compensation approaches have been suggested in the literature to reduce the error magnitude of truncated multiplier at the cost few overhead logic [5,6,10]. The complexity of the error-compensation circuit decides the magnitude of error retained in the multiplier output. On the other hand, the post-truncated fixed-width multiplier introduces minimum error at the multiplier output compared to the error-compensated truncated multiplier and involves the same amount of logic resource as the full-width multiplier. In general, the post-truncation method provides the most accurate fixed-width multiplier and involves almost same amount logic resource as the full-width multiplier, whereas the error-compensated truncated multiplier designs involve less logic resources than the post-truncated multiplier design, but introduce more truncation error than post-truncated multiplier. Although, the existing error-compensated truncated multipliers are suitable to design fixed-point hardware systems for different signal-to-noise ratio (SNR) conditions, the post-truncated multipliers are commonly preferred to design fixed-point hardware systems when SNR is critical. The adder unit of full-width multiplier involves few redundant logic operations when n LSBs of $2n$-bit multiplier output are truncated. The modern synthesis tools are powerful enough to optimize the design by removing the redundant logic operations when any signal is left unused in the design. However, we find that the synthesis tools not able to remove all the redundant logic operations resulted when n LSBs of $2n$-bit multiplier output of full-width multiplier design are truncated. A specific design approach is necessary to remove all the redundant logics. In this paper, we present an optimized adder unit design for post-truncated radix-8 multiplier. Using the optimized adder unit, the proposed fixed-width radix-8 Booth multiplier design involves nearly 20.7 % less ADP and consumes 18.3 % less power than the direct post-truncated radix-8 multiplier design of [3] optimized by Synopsys Design Compiler.

In various DSP algorithms, multiplication operations are performed on variables and constants. Multipliers are named as *generic* or *constant* based on the multiplier operand data types, i.e., variable or constant. A *generic multiplier* has both the operands as variables, whereas *constant multiplier* has one operand as a constant and the other one as a variable. In case of constant multiplier, the constant value may be fixed or changed by the user during run-time. The constant multiplier which has fixed constant value is referred to *fixed-constant multiplier*, and the multiplier which receives the

constant from the user is referred to *generic-constant multiplier*. Most of the dedicated hardware structures developed for processing of digital signals use these three types of multipliers. Both generic multiplier and generic-constant multiplier can be realized using Booth multiplication algorithm whereas fixed-constant multipliers are realized using shift-add method in canonic signed digit (CSD) representation or lookup tables. Multiple constant multiplication (MCM), common sub-expression elimination (CSE) and distributed arithmetic (DA) methods are used to design efficient hardware structures for inner-product computation involving fixed-constant multiplier [4,16]. In general, the fixed-constant multiplier is designed for a fixed value, and its complexity depends on the value of fixed constant. Therefore, fixed-constant multiplier needs redesign for different constant values. On the other hand, generic multiplier structure can be configured to obtain the generic-constant multiplier. The generic-constant multiplier has all the features of generic multiplier, except that it receives one operand from the user. Therefore, the generic-constant multiplier can be reused for different constant values, and its complexity is independent of constant value unlike the fixed-constant multiplier. Using generic-constant multiplier, one single computing structure needs to be designed for a specific algorithm and value of the constants can be changed conveniently to meet the requirement of different applications, whereas separate structures need to be designed for different constant values of the same algorithm using fixed-constant multiplier. Therefore, structures based on generic-constant multiplier are suitable for general applications, whereas structures based on fixed-constant multiplier are application specific.

The existing radix-4 and radix-8 Booth multiplier designs easily can be configured to have a generic-constant multiplier for constant multiplication. However, we find some interesting features of radix-8 Booth multiplier design. The PPG unit of radix-4 multiplier does not involve any adder, as the partial product terms $\{X, 2X\}$ are generated from the X using one shifter only, whereas the PPG unit of radix-8 Booth multiplier involves one adder to generate the partial product set $\{X, 2X, 3X, 4X\}$, where X is the multiplicand operand. Therefore, the PPG unit of radix-8 multiplier contributes some combinational logic to the multiplier area, and delay to the critical path, whereas the PPG unit of radix-4 multiplier does not contribute any combinational logic to the multiplier area and delay to the critical path. Removal of PPG unit can affect the ADP of the radix-8 multiplier design, whereas removal of PPG unit has almost no effect on the ADP of radix-4 multiplier design. This is an important feature of radix-8 multiplier design and could be an advantage when the generic multiplier design is configured for constant multiplication. We have shown that the proposed full-width and fixed-width radix-8 generic-constant multiplier designs involve 19.4 and 24.7 % less ADP than that of the state-of-the-art full-width and post-truncated radix-4 multiplier designs of [8] when configured for constant multiplications.

Lifting DWT structure involves constant multipliers. However, the lifting DWT algorithm cannot be expressed in the form of inner-product computation [11]. Therefore, the constant multipliers of lifting DWT cannot be implemented using MCM, CSE or DA approaches. Most of the existing hardware structures of lifting DWT proposed in the literature are generic-constant multiplier based. The lifting 2-D DWT structure involves $4N$ on-chip memory to perform computations of first-level decomposition, where N is the input image width. Interestingly, on-chip memory complexity

of 2-D DWT is independent of block size. Since the complexity of on-chip memory does not increase with increase in input block size, the 2-D DWT structures have less ADP for higher block sizes. But, the multiplier complexity of block-based 2-D DWT structure increases proportionately with the block size. Since, no redundant computation is available in lifting 2-D DWT, there is no scope to reduce multiplier complexity without compromising on the throughput rate. However, we observe that many multipliers of block-lifting 2-D DWT structures [12,13,15,17] share a common input operand. A group of multipliers with a common multiplying operand can select their partial product terms from a common set using Booth encoding scheme. Post-truncated 12-bit radix-4 Booth multipliers are generally consider to design 2-D DWT structures for various image and video processing applications. A significant amount of ADP and power could be saved when the block-based lifting 2-D DWT structure is implemented using the proposed radix-8 multiplier design instead of the most preferred radix-4 Booth multiplier design. To demonstrate the effectiveness of the proposed scheme, in this paper, we have considered the proposed radix-8 generic-constant fixed-width multiplier design for implementation of post-truncated multiplier of lifting 2-D DWT. However, the proposed radix-8 fixed-width generic-constant multiplier design is suitable for other DSP structures involving constant multiplication. We find that the existing lifting 2-D DWT structure of block size 16 and word length 12 offers 19.3 % ADP saving and 11.5 % power saving when the constant multipliers are implemented using the proposed radix-8 multiplier design instead of the existing radix-4 multiplier design.

The rest of the paper is organized as follows: Regular partial product array for radix-8 Booth multiplication is presented in Sect. 2. The proposed radix-8 full-width and fixed-width multiplier design is presented in Sect. 3. The hardware and time complexities of proposed radix-8 generic-constant multiplier are discussed in Sect. 4. Area-delay complexities of lifting 2-D DWT structure using proposed radix-8 generic-constant multiplier are presented in Sect. 5. Conclusions are drawn in Sect. 6.

2 Regular Partial Product Array for Radix-8 Booth multiplier

Consider two n-bit signed numbers X and Y are multiplied using Booth algorithm, where X is multiplicand and Y is multiplier. X and Y are represented in 2's complement number system as

$$X = -x_{n-1}2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i \tag{1}$$

$$Y = -y_{n-1}2^{n-1} + \sum_{i=0}^{n-2} y_i 2^i \tag{2}$$

where x_{n-1} and y_{n-1} are sign bits.

Booth algorithm decomposes an n bit multiplier into $w = \lceil n/3 \rceil$ groups of three bits each using radix-8 encoding. Each group of three bits and one overlapping bit forms one 4-bit digit, where the ith digit d_i is defined as $d_i = \{y_{3i+2}y_{3i+1}y_{3i}y_{3i-1}\}$, y_{3i-1}

is the overlapping bit belongs to $(i-1)$th digit d_{i-1}, and $d_{-1}=0$. The decimal value of d_i is obtained using the relation:

$$d_i = -4y_{3i+2} + 2y_{3i+1} + y_{3i} + y_{3i-1} \tag{3}$$

Substituting (3) in (2), we have

$$Y = \sum_{i=0}^{w-1} d_i 2^{3i} \tag{4}$$

Substituting (3) and (4) in the product of X and Y, we have

$$Z = X \sum_{i=0}^{w-1} (-4y_{3i+2} + 2y_{3i+1} + y_{3i} + y_{3i-1}) 2^{3i} \tag{5}$$

The d_i takes decimal value from the integer set $\{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$, and the product term $(X.d_i)$ forms a partial product. The possible partial products of radix-8 Booth multiplication are $\{-4X, -3X, -2X, -X, 0, X, 2X, 3X, 4X\}$. The radix-8 Booth encoding scheme for generating partial product bits (p_{ij}, for $0 \le i \le w-1, 0 \le j \le n-1$) is given in Table 1, where the partial product terms $\{2X, 4X\}$ are obtained by left-shifting X by one bit and two bits, respectively, and $3X$ is obtained by adding X and $2X$. The negative partial product terms $\{-4X, -3X, -2X, -X\}$ are obtained by taking the 2's complement of the corresponding positive partial product values $\{4X, 3X, 2X, X\}$. The 2's complement of $\{4X, 3X, 2X, X\}$ is obtained in two steps. In the first step, the 1's complement of $\{4X, 3X, 2X, X\}$ is obtained. In the second step, the sign bit (n_i) is added with the corresponding 1's complement partial product term to obtain the required 2's complement of positive partial product terms. The step 1 of the 2's complement operation is implemented in the Booth selector, while the step 2 is implemented in the partial product accumulation stage. The partial product generator produces partial product terms $\{X, 2X, 3X, 4X\}$ for the Booth selector which selects one value from the set $\{X, 2X, 3X, 4X\}$ and produces the partial product term in normal or complemented form. The Booth selector uses the control signals $\{a_i, b_i, c_i, d_i\}$ to select the partial product value from the set $\{X, 2X, 3X, 4X\}$ and uses control bit (n_i) to produce the 1's complement of the partial product. The control signal $a_i = 1$, when partial product term $P = \{X, -X\}$, $b_i = 1$, when $P = \{2X, -2X\}$, $c_i = 1$, when $P = \{3X, -3X\}$, $d_i = 1$, when $P = \{4X, -4X\}$, and $n_i = 1$, when $P = \{-X, -2X, -3X, -4X\}$.

Since the partial product rows of Booth-encoded multiplication are represented in 2's complement arithmetic and each partial product row is left shifted by 3-bit positions with respect to its previous row, sign extension of partial product rows is required up to $(2n-1)$ bit position of $2n$ bit product to obtain the correct multiplication result. Extension of sign bit up to $(2n-1)$ bit position increases adder unit complexity of Booth multiplier significantly. By inserting appropriate guard bits in each partial product row, the extension of sign bit up to $(2n-1)$ bit position can be avoided. With

Table 1 Partial product values of radix-8 Booth-encoded multiplication

$y_{3i+2}y_{3i+1}y_{3i}y_{3i-1}$	Operation	a_i	b_i	c_i	d_i	n_i	pp_{ij}
0000	0	0	0	0	0	0	0
0001	$+X$	1	0	0	0	0	x_j
0010	$+X$	1	0	0	0	0	x_j
0011	$+2X$	0	1	0	0	0	x_{j-1}
0100	$+2X$	0	1	0	0	0	x_{j-1}
0101	$+3X$	0	0	1	0	0	$x_j + x_{j-1}$
0110	$+3X$	0	0	1	0	0	$x_j + x_{j-1}$
0111	$+4X$	0	0	0	1	0	x_{j-2}
1000	$-4X$	0	0	0	1	1	$\overline{x_{j-2}}$
1001	$-3X$	0	0	1	0	1	$\overline{x_j + x_{j-1}}$
1010	$-3X$	0	0	1	0	1	$\overline{x_j + x_{j-1}}$
1011	$-2X$	0	1	0	0	1	$\overline{x_{j-1}}$
1100	$-2X$	0	1	0	0	1	$\overline{x_{j-1}}$
1101	$-X$	1	0	0	0	1	$\overline{x_j}$
1110	$-X$	1	0	0	0	1	$\overline{x_j}$
1111	0	0	0	0	0	0	0

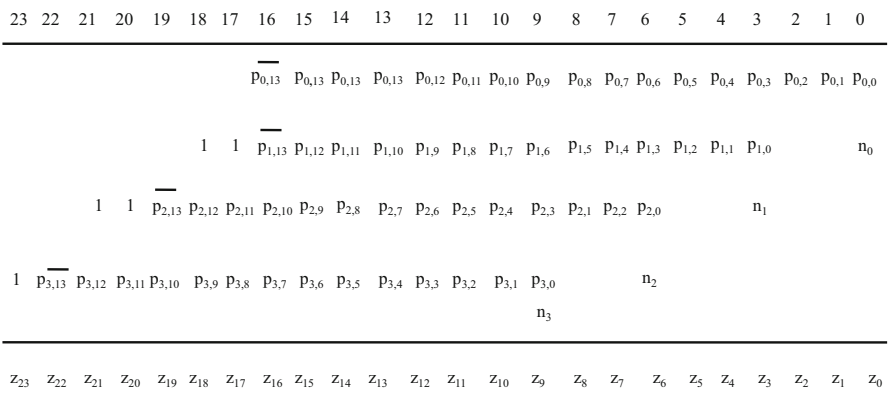

Fig. 1 Partial product array of radix-8 Booth multiplier for 12-bit

the guard bits, the partial product array (PPA) of radix-8 Booth multiplication is shown in Fig. 1 for bit-width $w = 12$.

As mentioned earlier, the control bit (n_i) is added to each row of the PPA to convert 1's complement number into 2's complement number. Due to addition of control bit n_i to the partial product rows, the size of the PPA is increased by one extra row. The PPA accumulator requires an extra adder and an extra stage to accumulate the extra partial product row. The existing radix-8 Booth multipliers structures [1,3] are based on the PPA shown in Fig. 1.

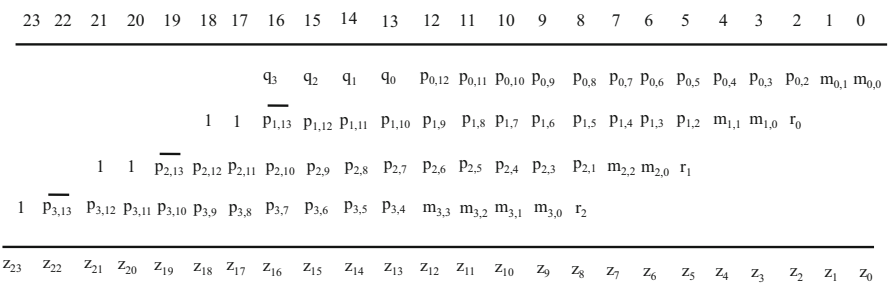

Fig. 2 Regular partial product array of radix-8 Booth multiplier for 12-bit

Kuang et al. [8] have suggested a scheme to obtain a regular partial product array for Radix-4 Booth multiplication without increasing partial product rows. We have extended the approach given in [8] for radix-8 Booth multiplication and modified the PPA of Fig. 1 to obtain a regular PPA. The regular PPA of 12-bit radix-8 Booth multiplier is shown in Fig. 2. As shown in Fig. 2, the two LSBs ($p_{i,1}$, $p_{i,0}$) of $(i+1)$th row of PPA are modified into ($m_{i,1}$, $m_{i,0}$) by absorbing the control bit n_i, and a carry bit (r_i) is generated which appended to the right of the $(i+2)$th row LSB of PPA. In case of wth row of PPA, four LSBs ($p_{w-1,3}$, $p_{w-1,2}$, $p_{w-1,1}$, $p_{w-1,0}$) are modified into ($m_{w-1,3}$, $m_{w-1,2}$, $m_{w-1,1}$, $m_{w-1,0}$) to absorb the control bit (n_{w-1}) and produce the carry bit (r_{w-1}, not shown in Fig. 2). The sign extension bits of first row of PPA are modified into (q_3, q_2, q_1, q_0) to absorb the carry bit (r_{w-1}). The modified bits ($m_{i,1}, m_{i,0}$) and carry bit (r_i) are generated using the following logic expressions:

$$m_{i0} = p_{i_0} \oplus n_i \tag{6}$$

$$m_{i1} = p_{i,1}\overline{p_{i,0}} + p_{i,1}\overline{n_i} + \overline{p_{i,1}}p_{i,0}n_i \tag{7}$$

$$r_i = p_{i,1}p_{i,0}n_i \qquad 0 \le i \le n/3-2 \tag{8}$$

$m_{3,2}, m_{3,1}, r_3, q_3, q_2, q_1, q_0$ are generated using the logic expressions:

$$m_{32} = p_{i,2}\overline{p_{i,1}} + p_{i,2}\overline{p_{i,o}} + p_{i,2}\overline{n_3} + \overline{p_{i,2}}p_{i,1}p_{i,0}n_3 \tag{9}$$

$$m_{33} = p_{i,3}\overline{p_{i,2}} + p_{i,2}\overline{p_{i,0}} + p_{i,3}\overline{n_3} + \overline{p_{i,3}}p_{i,2}p_{i,1}p_{i,0}n_3 \tag{10}$$

$$r_3 = p_{3,1}p_{3,2}p_{3,1}p_{3,0}n_3 \tag{11}$$

$$q_0 = p_{0,13} \odot r_3 \tag{12}$$

$$q_3 = \overline{p_{0,13}\overline{r_3}} \tag{13}$$

$$q_2, q_1 = \overline{q_3} \tag{14}$$

$$a_i = \overline{(y_{3i-1} \odot y_{3i})(\overline{y_{3i+1} \odot y_{3i+2}})} \tag{15}$$

$$b_i = \overline{(\overline{y_{3i-1} \odot y_{3i}})(y_{3i} \odot y_{3i+1})} \tag{16}$$

$$c_i = \overline{(y_{3i-1} \odot y_{3i})(y_{3i+1} \odot y_{3i+2})} \tag{17}$$

$$d_i = \overline{(\overline{y_{3i-1} \odot y_{3i}})(\overline{y_{3i+1} \odot y_{3i+2}})(y_{3i+1} \odot y_{3i+2})} \tag{18}$$

where $\oplus$ and $\odot$ denote the XOR and XNOR operations, respectively.

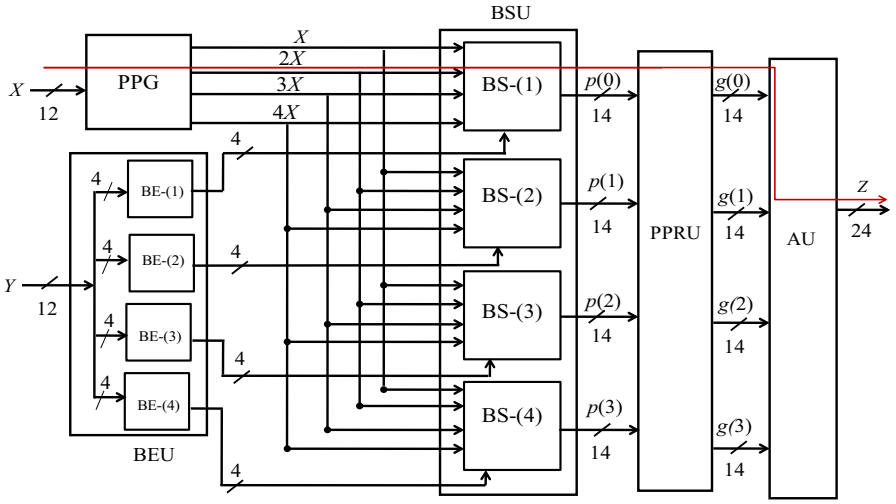

Fig. 3 Block diagram of proposed radix-8 Booth multiplier structure for 12-bit. The *red-colored arrow* shows the critical path (Color figure online)

3 Proposed Radix-8 Full-Width and Fixed-Width Multiplier Designs

The block diagram of radix-8 Booth multiplier is shown in Fig. 3. The partial product unit generates the partial product set $\{4X, 3X, 2X, X\}$ form input operand X using one adder and two shifters. The Booth encoder unit (BEU) consists of w Booth encoders (BEs) where each BE produces four control signals. The $(i+1)$th BE (for $0 \leq i \leq w-1$) produces control signals $\{a_i, b_i, c_i, d_i\}$ from the digit $d_i = \{y_{3i+2}y_{3i+1}y_{3i}y_{3i-1}\}$ using the logic expressions of (15)–(18). The internal structure of $(i+1)$th BE is shown in Fig. 4a which is almost identical to the BE structure of [3]. The Booth selector unit (BSU) is comprised of w Booth selectors (BSs) corresponding to w rows of PPA. The $(i+1)$th BS receives control bits $\{a_i, b_i, c_i, d_i\}$ from $(i+1)$th BE and selects one partial product from the set $\{4X, 3X, 2X, X\}$. The BS structure suggested in [3] is modified to include the resent circuit to produce zero partial product term when $d_i = \{0000, 1111\}$. The modified BS structure is shown in Fig. 4b. The PPA rows produced by BSU are further modified by the partial product resize unit (PPRU) to produce regular PPA. The PPRU is implemented using the logic expressions of (6)–(14). The regular partial product rows are shift-added in the adder unit (AU) to produce the $2n$-product bits. The AU is implemented using Wallace tree adder. Bit-flow diagram of the AU is shown in Fig. 5a for $n = 12$, and its node operation is shown in Fig. 5b.

3.1 Radix-8 Fixed-Width Booth Multiplier Design

The adder unit of proposed full-width radix-8 Booth multiplier design is optimized to remove the redundant logic operations resulted due to post-truncation of n LSBs of $2n$-bit multiplier output. Since the sum bit corresponding to n LSCs of the PPA of Fig. 2 is truncated in the final $2n$-bit output, the partial products of n LSCs are accumulated

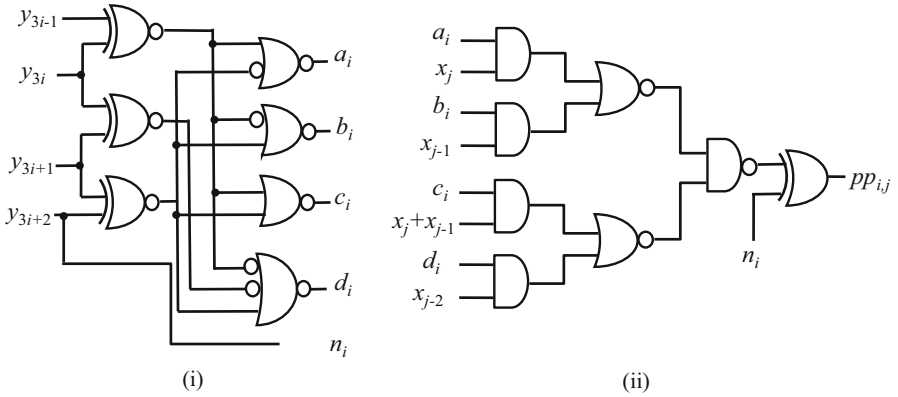

Fig. 4 Internal structure of Booth encoder (BE) and Booth selector (BS)

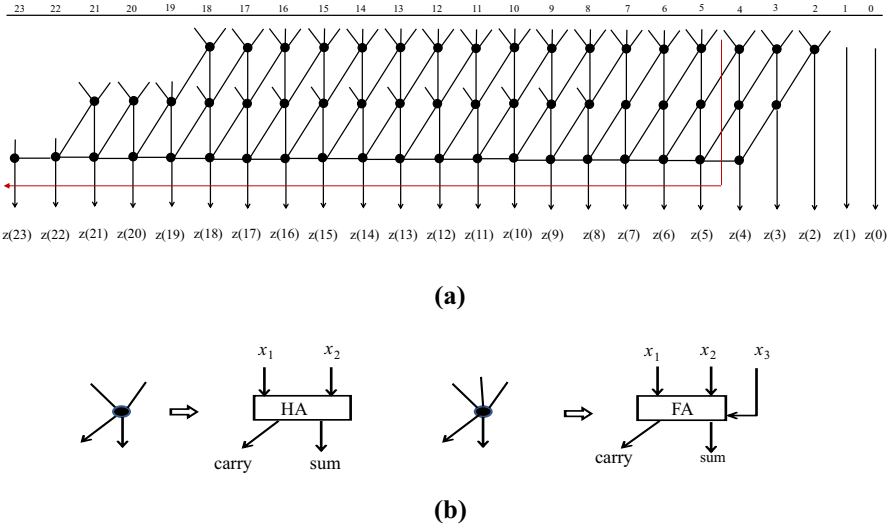

Fig. 5 **a** Bit-flow diagram of adder unit of radix-8 Booth multiplier for 12-bit. Critical path is shown in *red-colored arrow*. **b** Node operation (Color figure online)

by carry generator while the partial products of remaining n most significant columns (MSCs) are accumulated using bit adders. The bit-flow diagram of Fig. 5a is modified to obtain the bit-flow diagram of the adder unit of post-truncated radix-8 Booth multiplier. The bit-flow diagram of adder unit of post-truncated fixed-width radix-8 multiplier is shown in Fig. 6a for $n = 12$ and its node operation is depicted in Fig. 6b. The proposed radix-8 multiplier design is optimized for fixed-width multiplication by replacing the AU with by the modified AU design of Fig. 6a. The proposed fixed-width multiplier design involves less logic resources than those of conventional post-truncated radix-8 Booth multiplier design and introduces the same amount of truncation error as the other.

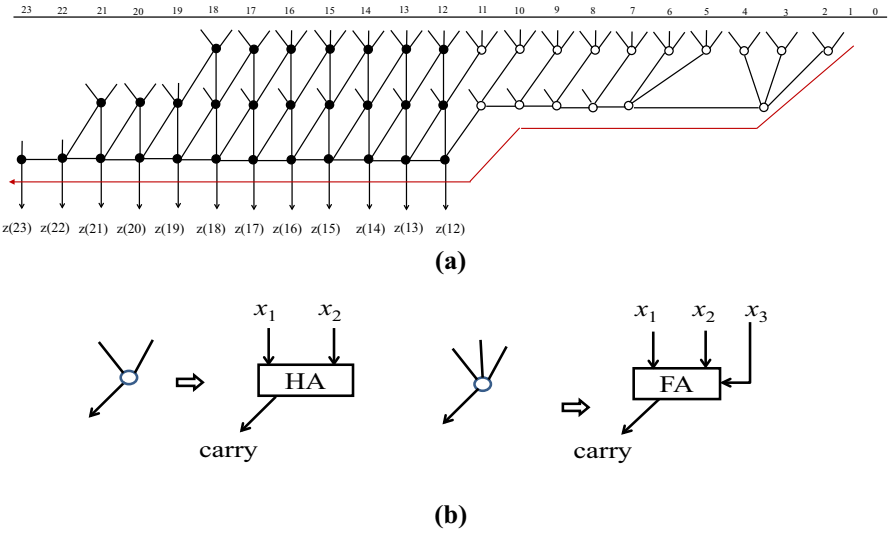

Fig. 6 **a** Bit-flow diagram of adder unit of post-truncated Radix-8 Booth multiplier for 12-bit. Critical path is shown in the *red- colored arrow*. **b** Node operation (Color figure online)

3.1.1 Hardware Complexity

The proposed radix-8 multiplier structure has one PPG unit, one BEU, one BSU, one PPRU and one AU. The hardware complexity of the proposed radix-8 multiplier is estimated for $n = 12$. The PPG unit consists of one 12-bit ripple carry adder (RCA), where the 12-bit RCA is implemented using 11 full-adders (FAs) and one half-adder (HA). The BEU consists of four BEs where each BE further comprised of three XNOR gates and five NOR gates. Similarly, the BSU consists of four BSs, and each BS further comprised of four AND gates, two NOR gates, one NAND gate and one XOR gate. The PPRU involves 46 logic gates. The adder unit involves 42 FAs and 14 HAs. Therefore, the proposed radix-8 multiplier design involves 53 FAs, 15 HAs, 192 XOR/XNOR gates, 336 NOR gates and 240 AND/NAND gates, Assuming each FA is implemented using two XOR gates and three AND/OR gates, and HA is implemented using one XOR gate and one AND gate, the proposed radix-8 multiplier involves 313 XOR/XNOR gates and 774 AND/NOR/OR gates.

The proposed fixed-width radix-8 multiplier structure is identical to the proposed radix-8 multiplier structure except the AU design. As shown in Fig. 6a, the AU of proposed fixed-width radix-8 multiplier involves 23 FAs, 6 HAs, 13 full-carry generators (FCGs) and three half-carry generators (HCGs), where each HCG and FCG is implemented using one AND gate and three AND/OR gates, respectively. Therefore, the proposed fixed-width radix-8 multiplier structure involves 47 FAs, 15 HAs, 192 XOR/XNOR gates, 336 NOR gates and 240 AND/NAND gates, In terms of gate count, the proposed fixed-width radix-8 multiplier involves 261 XOR/XNOR gates and 801 AND/NOR/OR gates.

We have estimated hardware complexity of the existing radix-8 multiplier design of [3] for $n = 12$ for comparison. The radix-8 multiplier structure of [3] consists of one PPG unit, one BEU, one BSU and one AU. The logic complexity of PPG unit and BEU is same as those of the proposed structure. The BSU of multiplier structure of [3] involves one less XOR gate than those of the proposed structure. However, the logic complexity of AU of [3] is higher than the logic complexity of AU of proposed structure since the AU of [3] involves one extra adder stage than those required by the AU of proposed structure. The AU of [3] involves 44 FAs and 32 HAs. Therefore, the radix-8 multiplier structure of [3] involves 55 FAs, 33 HAs, 192 XOR/XNOR gates, 336 NOR gates and 240 AND gates. In terms of gate count, the radix-8 multiplier of [3] involves 335 XOR/XNOR gates and 791 AND/NOR/OR gates.

3.1.2 Critical Path Delay

The critical path of the proposed multiplier structure is shown in Fig. 3. The CPD is $T = \{T_{\mathrm{PPG}} + T_{\mathrm{BS}} + T_{\mathrm{PPRU}} + T_{\mathrm{AU}}\}$, where T_{PPG}, T_{BS}, T_{PR}, and T_{AU}, are, respectively, the delay of PPG unit, BSU, PPRU, and AU. The delay of 12-bit RCA is the T_{PPG}, and it is given as $T_{\mathrm{PPG}} = 23T_{\mathrm{X}}$, T_{BS} is estimated as $\{T_{\mathrm{AND}} + T_{\mathrm{NOR}} + T_{\mathrm{NAND}} + T_{\mathrm{X}}\}$, and the T_{PR} is estimated as $\{3T_{\mathrm{AND}} + T_{\mathrm{OR}}\}$, where T_{X}, T_{AND}, T_{NAND}, T_{OR}, and T_{NOR}, are, respectively, the delay of 2-input XOR gate, 2-input AND gate, 2-input NAND gate, 2-input OR gate, and 2-input NOR gate. T_{AU} is estimated from the bit-flow diagram of Fig. 5a and found to be $T_{\mathrm{AU}} = 4T_{\mathrm{X}} + 18T_{\mathrm{AND}} + 18T_{\mathrm{OR}}$. The delay of AU of Fig. 6a is estimated as $28T_{\mathrm{X}} + 18T_{\mathrm{AND}} + 18T_{\mathrm{OR}}$. Therefore, the proposed radix-8 full-width multiplier has CPD $T = 28T_{\mathrm{X}} + 21T_{\mathrm{AND}} + 19T_{\mathrm{OR}} + T_{\mathrm{NOR}} + T_{\mathrm{NAND}}$, and the proposed fixed-width multiplier has CPD $T = 25T_{\mathrm{X}} + 21T_{\mathrm{AND}} + 16T_{\mathrm{OR}} + T_{\mathrm{NOR}} + T_{\mathrm{NAND}}$. The CPD of radix-8 multiplier [3] is $T = \{T_{\mathrm{PPG}} + T_{\mathrm{BS}} + T_{\mathrm{AU}}\}$. The delay of PPG unit and BSU is same as the proposed radix-8 multiplier structure, but the delay of AU unit is $T_{\mathrm{AU}} = 7T_{\mathrm{X}} + 19T_{\mathrm{X}} + 19T_{\mathrm{OR}}$. Therefore, CPD of radix-8 multiplier structure of [3] is $T = 30T_{\mathrm{X}} + 20T_{\mathrm{AND}} + 19T_{\mathrm{OR}} + T_{\mathrm{NOR}} + T_{\mathrm{NAND}}$.

3.1.3 Performance Comparison

The theoretical estimates of hardware and time complexities of proposed multiplier structures and existing multiplier structure of [3] are listed in Table 2 for $n = 12$. As shown in Table 2, the proposed full-width multiplier structure involves 22 less XOR/XNOR gates, 17 less AND/OR/NOR gates, and CPD less by $(2T_{\mathrm{X}} + T_{\mathrm{AND}} + T_{\mathrm{OR}})$. The proposed fixed-width multiplier structure involves 74 less XOR gates, 37 more AND/OR/NOR gates, and CPD less by $(5T_{\mathrm{X}} + 5T_{\mathrm{OR}} + 2T_{\mathrm{NAND}} - T_{\mathrm{OR}})$ compared to those of the existing multiplier structure of [3] and introduces the same amount of truncation error to the multiplier output as the existing structure does when multiplier output is post-truncated. Therefore, the proposed fixed-width radix-8 multiplier structure involves significantly less area and delay compared to the existing radix-8 multiplier structure [3] and introduces the same amount of truncation error to the multiplier output.

Table 2 Comparison of hardware and time complexities of proposed radix-8 Booth multiplier and existing radix-8 Booth multiplier

Multiplier design	OR/NOR/AND/NAND gate	XOR/XNOR gate	Critical path
Radix-8 fixed-width/post-truncated [3]	774	335	$30T_X + T_{NOR} + 20T_{AND} + T_{NAND} + 19T_{OR}$
Proposed radix-8 full-width	791	313	$28T_X + T_{NOR} + 21T_{AND} + T_{NAND} + 18T_{OR}$
Proposed radix-8 fixed-width	801	261	$25T_X + T_{NOR} + 21T_{AND} + T_{NAND} + 16T_{OR}$

T_X : 2-input XOR/XNOR-gate delay, T_{OR} : 2-input OR-gate delay, T_{NOR}: 2-input OR-gate delay, T_{AND} : 2-input AND-gate delay, T_{NAND} : 2-input NAND-gate delay

Table 3 Comparison of synthesis results of radix-8 multiplier

Multiplier design	Area (μ sqm)	DAT (ns)	ADP (μ sqm s)	Power (mW)
Radix-8 full-width [3]	8447.84	21.23	179347.02	49.8
Proposed radix-8 full-width	7790.98	20.56	160182.54	43.1
Radix-8 fixed-width with direct truncation [3]	7406.87	20.71	153396.27	35.4
Proposed radix-8 fixed-width	6709.47	18.13	121642.69	28.9

3.1.4 Comparison of Synthesis Results

Output ($2n$ bits) of existing radix-8 multiplier design of [3] is truncated by discarding $n = 12$ lower-order bits to have a direct post-truncated fixed-width constant multiplier. The proposed full-width and the fixed-width multiplier designs and the existing radix-8 multiplier full-width and direct post-truncated multiplier designs of [3] are coded in VHDL for $n = 12$. All these designs are synthesized in Synopsys Design Compiler using SAED 90-nm CMOS library. The area, delay and power estimates of these designs reported by Design Compiler are listed in Table 3 for comparison. As shown in Table 3, the proposed radix-8 multiplier structure involves slightly less area and less delay than the existing radix-8 multiplier structure of [3]. This is mainly the saving in XOR gates in the proposed design. The proposed radix-8 multiplier structure involves 10.7 % less area-delay product (ADP) and consumes 13.4 % less power than the existing multiplier structure of [3].

The Design Compiler optimizes the design when certain output signals are not assigned. Therefore, the direct post-truncated design of [3] involves nearly 12.3 % less area and marginally less delay than the corresponding full-width multiplier [3]. The proposed fixed-width multiplier design involves 9.4 % less area and 12.5 % less delay than the direct post-truncated multiplier design of [3] and introduces the same amount of truncation error to the multiplier output. The proposed fixed-width radix-8 Booth multiplier design involves nearly 20.7 % less ADP and consumes 18.3 % less power than the direct post-truncated radix-8 multiplier design of [3].

4 Radix-4 and Radix-8 Generic-Constant Booth Multiplier Designs

Both radix-4 and radix-8 Booth multipliers can be configured to have generic-constant multiplier for constant multiplication. It is emphasized here that while configuring radix-4 and radix-8 Booth multipliers for constant multiplication, they do not sacrifice their generic feature unlike fixed-constant multipliers. The selection of radix-4 generic multiplier structure is the obvious choice to find an efficient structure for generic-constant multiplier, since radix-4 generic Booth multiplier structure has better area-delay efficiency over the radix-8 or higher generic multiplier structures [14]. However, we observe that when radix-4 and radix-8 Booth multiplier designs are approximated for constant multiplication, then the radix-8 Booth multiplier design becomes more area-delay-efficient than the constant radix-4 Booth multiplier design. The hardware and time complexities of proposed radix-8 Booth multiplier design and the existing radix-4 Booth multiplier design of [8] are discussed in this section when both the design are configured for constant multiplication.

Suppose the operand (X) received by the PPG unit of Booth multiplier design is a constant, then the partial product terms generated by the PPG become constant. These constant partial product terms can be pre-computed and directly fed to the PSU, and the PPG unit can be removed from the generic-constant multiplier design. The PPG unit of radix-4 multiplier does not involve any adder as the partial product terms $\{X, 2X\}$ are generated from the X using one shifter only, whereas the PPG unit of radix-8 Booth multiplier involves one adder. Therefore, the PPG unit of radix-8 multiplier contributes some combinational logic to the area complexity and delay to the critical path, whereas the PPG unit of radix-4 multiplier does not contribute any combinational logic to the area complexity and delay to the critical path. Removal of PPG unit can affect the ADP of the radix-8 multiplier design, whereas removal of PPG unit has almost no effect on the ADP of radix-4 multiplier design. This is an important feature of radix-8 multiplier design and becomes an advantage when the generic multiplier design is configured for constant multiplication. The PPG unit of the proposed radix-8 multiplier design (shown in Fig. 3) and the existing radix-8 multiplier design of [3] are moved to optimize these designs for constant multiplication. We have considered the radix-4 Booth multiplier design of [8] which is the best among the existing radix-4 multiplier designs to study the area-delay efficiency of generic-constant radix-8 and generic-constant radix-4 Booth multiplier designs.

4.1 Hardware–Time Complexities of Radix-4 and Radix-8 Generic-Constant Booth Multiplier Designs

The radix-4 generic-constant full-width and post-truncated multiplier designs are derived from the radix-4 generic full-width multiplier design of [8], whereas the radix-8 generic-constant full-width multiplier design and generic-constant post-truncated multiplier design are derived from the generic full-width multiplier structure of [3] for comparison. The derived radix-4 generic- constant full-width and post-truncated multiplier designs are identical to the generic full-width multiplier design of [8], whereas

Table 4 Comparison of hardware and time complexities of Booth multiplier for 12-bit

Multiplier design	OR/NOR/ AND/NAND gate	XOR/XNOR gate	Critical Path
Radix-4 GC/post-truncated [8]	840	295	$9T_X + 21T_{OR} + 3T_{NAND} + T_{NOR} + 20T_{AND}$
Radix-8 GC /post-truncated [3]	740	312	$8T_X + 3T_{NOR} + 21T_{AND} + T_{NAND} + 19T_{OR}$
Proposed radix-8 GC	757	290	$6T_X + 3T_{NOR} + 21T_{AND} + T_{NAND} + 18T_{OR}$
Proposed radix-8 GC fixed-width	734	244	$3T_X + 3T_{NOR} + 21T_{AND} + T_{NAND} + 16T_{OR}$

GC Generic-constant

the derived radix-8 generic-constant full-width multiplier design and post-truncated multiplier design involve one less 12-bit adder than those required by the generic full-width multiplier design of [3]. The proposed radix-8 generic-constant full-width and fixed-width multiplier designs are obtained from the proposed generic full-width multiplier design (shown in Fig. 3) and the proposed generic fixed-width multiplier design by removing the PPG unit.

The hardware complexity and time complexity are estimated for $n = 12$. The estimated values are listed in Table 4 for comparison. As shown in Table 4, the radix-8 generic-constant full-width or the post-truncated multiplier design of [3] involves 17 more XOR gates, 100 less AND/OR/NOR gates and less CPD by $(T_X + 2T_{OR} + 2T_{AND} - 2T_{NOR})$ delay compared to those of radix-4 generic-constant full-width or post-truncated multiplier design of [8]. The proposed radix-8 generic-constant full-width multiplier design involves 5 less XOR gates, 87 less AND/OR/NOR gates and less CPD by $(3T_X+3T_{OR}+2T_{AND}-2T_{NOR})$ delay compared to those of radix-4 generic-constant full-width multiplier design of [8]. The proposed radix-8 generic-constant fixed-width multiplier design involves 51 less XOR gates and 106 less AND/OR/NOR gates and CPD less by $(6T_X + 7T_{OR} - 2T_{NOR} - T_{AND})$ delay as compared to the radix-4 generic- constant post-truncated multiplier design of [8] and introduces the same amount of truncation error to the multiplier output as the radix-4 generic post-truncated multiplier design of [8].

4.2 Comparison of Synthesis Result

Output ($2n$ bits) of existing radix-4 multiplier design of [8] and the existing radix-8 multiplier design are truncated by discarding $n = 12$ lower-order bits to have a direct post-truncated fixed-width constant multiplier. The proposed full-width and fixed-width generic-constant multiplier designs, the existing radix-8 generic-constant full-width and direct post-truncated multiplier designs of [3] and the existing radix-4 generic-constant full-width and direct post-truncated multiplier designs of [8] are coded in VHDL for $n = 12$. All these designs are synthesized in Synopsys Design Compiler using SAED 90-nm CMOS library. The area, delay and power estimates of

Table 5 Comparison of synthesis result of generic-constant full-width Booth multiplier structures

Multiplier design	Area (μ sqm)	DAT (ns)	ADP (μ sqm s)	Power (mW)
Radix-4 GC [8]	7334.22	16.11	118154.28	39.9
Radix-8 GC [3]	6999.35	15.20	106390.12	37.6
Proposed radix-8 GC	6597.53	14.42	95136.38	32.8

GC Generic-constant

Table 6 Comparison of synthesis result of generic-constant fixed-width Booth multiplier structures

Multiplier design	Area (μ sqm)	DAT (ns)	ADP (μ sqm.s)	Power(mW)
Radix-4 GC post-truncated [8]	6904.24	14.10	97346.4	32.7
Radix-8 GC post-truncated [3]	6607.90	13.97	92312.36	31.3
Proposed radix-8 GC fixed-width	5562.36	13.17	73256.93	23.6

these designs reported by Design Compiler are listed in Tables 5 and 6 for comparison. As shown in Table 5, the radix-8 generic-constant full-width multiplier design of [3] involves 4.5 % less area and marginally less delay compared to those of radix-4 generic-constant full-width multiplier design of [8]. The proposed radix-8 generic-constant full-width multiplier design involves 10 % less area and 10.4 % less delay than those of radix-4 generic-constant full-width multiplier design of [8].

As mentioned earlier, the Design Compiler optimizes the design when certain output signals are not assigned. As shown in Table 6, the radix-4 generic-constant direct post-truncated design of [8] involves nearly 5.9 % less area and marginally less delay than the corresponding constant full-width multiplier design of [8]. Similarly, the radix-8 generic-constant direct post-truncated design of [3] involves nearly 5.6 % less area and marginally less delay than the corresponding constant full-width multiplier design of [3]. The proposed generic-constant fixed-width multiplier design involves 15.8 % less area and 5.7 % less delay than the radix-8 generic-constant direct post-truncated multiplier design of [3]. It involves 19.4 % less area and 6.6 % less delay than the radix-4 generic-constant direct post-truncated multiplier design of [3]. The proposed radix-8 generic-constant fixed-width multiplier design involves nearly 20.6 % less ADP and consumes 24.6 % less power than the generic- constant direct post-truncated radix-8 multiplier design of [3]. It involves 24.7 % less ADP and consumes 27.8 % less power than the generic-constant direct post-truncated radix-4 multiplier design of [8]. It is interesting to note that the proposed radix-8 generic-constant fixed-width multiplier design introduces the same amount of truncation error to the multiplier output as the radix-8 generic-constant post-truncated multiplier design of [3] and radix-4 generic-constant post-truncated multiplier design of [8].

It is well known that radix-4 Booth multiplier design mostly preferred over the other higher-radix Booth multiplier designs as requires relatively less area, delay and power than others radix multipliers. Therefore, radix-4 generic Booth multiplier design mostly preferred over the radix-8 generic multiplier designs. However, we find

an interesting property of radix-8 multiplier design. An optimized radix-8 generic-constant Booth multiplier design involves less ADP and consumes less power than the radix-4 generic-constant Booth multiplier. Therefore, radix-8 multiplier should be preferred over the radix-4 multiplier for constant multiplication. Due to less ADP, we have chosen the proposed radix-8 generic-constant fixed-width multiplier design over the existing radix-4 generic-constant Booth multiplier design to develop area-delay-efficient fixed-point architectures for computation of 2-D DWT. We have considered the existing multiplier-based 2-D DWT structure of [13] and modified this structure to take advantage of radix-8 generic-constant multiplier.

5 Comparison of Area-Delay Complexity of Lifting 2-D DWT Structure Using Radix-4 and Radix-8 Booth Multipliers

Block-based 2-D DWT structure efficiently utilizes on-chip memory and frame memory. However, the adder and multiplier complexity increases proportionately with the block size. Multiplier consumes more combinational logic than the adders. Interestingly, the lifting-based 2-D DWT structure involves constant multiplication. The lifting 2-D DWT structure of block size P involves $4.5P$ multipliers [13]. One operand of these $4.5P$ multipliers is one of the four lifting constants $\{\alpha, \beta, \gamma, \delta\}$ or two scaling constants $\{k^2, 1/k^2\}$. These multipliers are distributed into six groups. Multipliers of a particular group share a common input operand. This feature can be exploited in a hardware structure to reduce the logic complexity using the proposed radix-8 generic-constant multiplier. We have considered the recently proposed multiplier-based lifting 2-D DWT structure of [13] and made the necessary modification in the design for using radix-8 generic-constant multiplier. Note that, by replacing the generic multiplier with generic-constant multiplier, there is no change in behavior of the lifting 2-D DWT structure of [13].

The modified structure of lifting 2-D DWT is shown in Fig. 7 which is identical to the structure of [13], except that each multiplier of the 2-D DWT structure of [13] is implemented using the proposed radix-8 generic-constant multiplier. Each vector $\{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \mathbf{g}_4\}$ represents four partial product values corresponding to radix-8 Booth multiplication of a particular constant value, where $\mathbf{g}_1 = \{\alpha, 2\alpha, 3\alpha, 4\alpha\}$, $\mathbf{g}_2 = \{\beta, 2\beta, 3\beta, 4\beta\}$, $\mathbf{g}_3 = \{\gamma, 2\gamma, 3\gamma, 4\gamma\}$, and $\mathbf{g}_4 = \{\delta, 2\delta, 3\delta, 4\delta\}$. These vectors $\{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \mathbf{g}_4\}$ are pre-computed using values of lifting constants $\{\alpha, \beta, \gamma, \delta\}$, and fed to the structure as an external input from the user. As shown in Fig. 7, the input vectors $\{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \mathbf{g}_4\}$ are propagated across the functional elements (FEs) of row and column processors as multipliers of these FEs use lifting constants $\{\alpha, \beta, \gamma, \delta\}$. The modified structures of 1-D array of row processor and high-pass/low-pass block of column processor using radix-8 generic-constant multiplier are shown in Figs. 8 and 9, respectively. The modified 1-D array uses four FEs corresponding to four lifting constants $\{\alpha, \beta, \gamma, \delta\}$. The modified structure of FE is shown is Fig. 10, which is comprised of one radix-8 generic-constant multiplier (GCM) and two adders. The radix-8 GCM receives partial product values $\{g, 2g, 3g, 4g\}$ through the input vector '**g**', where 'g' is one of the four lifting constant. All the FEs of 1-D arrays of the row processor and the low-pass/high-pass blocks of column processor correspond-

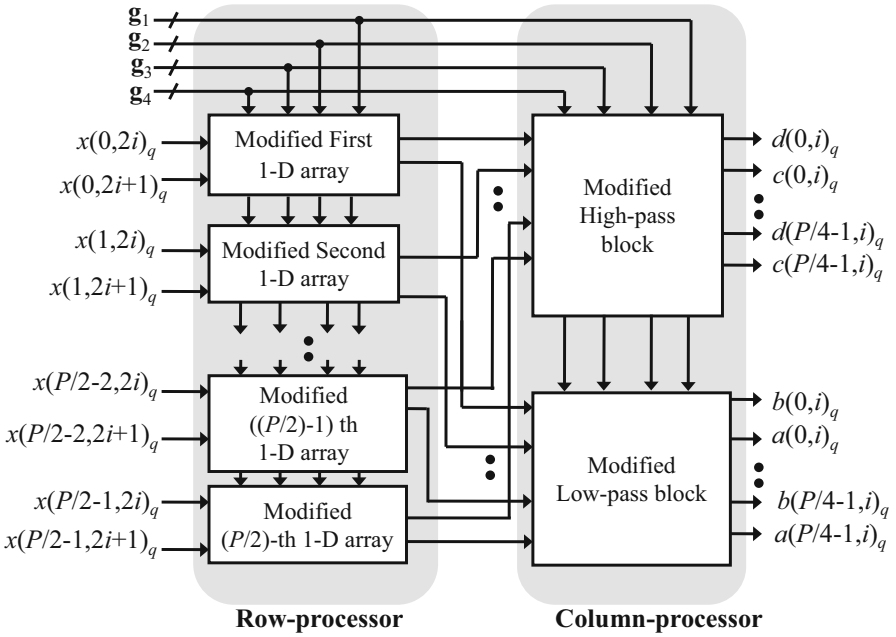

Fig. 7 Modified lifting 2-D DWT structure, where $x(k,i)_q = x(P_q + k, i), a(j,i)_q = a(P_q/4 + j, i), b(j,i)_q = b(P_q/4 + j, i), c(j,i)_q = c(P_q/4 + j, i)$, and $d(j,i)_q = d(P_q/4 + j, i)$, where $0 \le q \le Q-1, 0 \le i \le (N/2)-1, 0 \le k \le (P/2)-1, 0 \le j \le (P/2)-1$

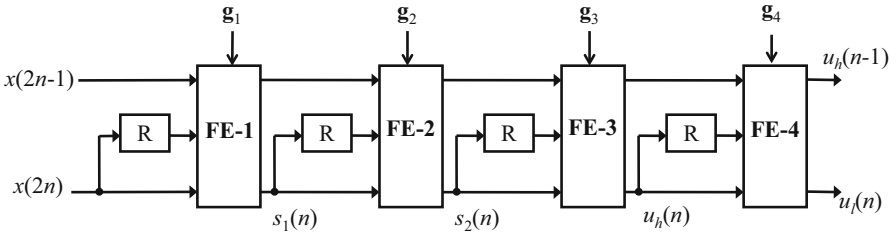

Fig. 8 Modified structure of 1-D array

ing to a particular lifting constant share the partial product terms. Therefore, input vectors $\{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \mathbf{g}_4\}$ are transmitted through each 1-D array of row processor and low-pass/high-pass blocks of column processor for sharing the partial product terms.

The proposed structure and the structure of [13] are identical and involve the same amount of on-chip memory, frame memory, multipliers and adders for a given image size and block size. However, these two structures have different combinational logic complexity since they use radix-8 and radix-4 generic-constant Booth multipliers. Note that radix-4 generic Booth multiplier and generic-constant Booth multiplier have the identical structures since the PPG unit does not contribute any combinational logic to the multiplier design. We have coded the proposed 2-D DWT structure and the structure of [13] using radix-4 generic-constant Booth multiplier design of [8] in VHDL for

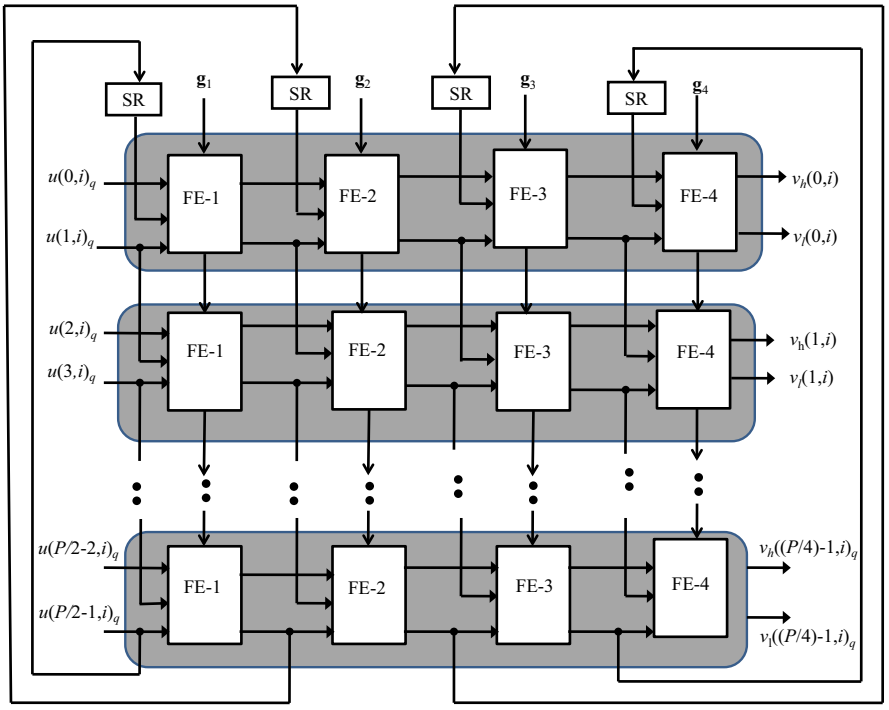

Fig. 9 Modified structure of high-pass block/low-pass block, where $u(k_1, i)_q = u_l(P_q + k_1, i)/u_h(P_q + k_1, i)$; $v_h(k_2, i)_q = b(P_q/4 + k_2, i)/d(P_q/4 + k_2, i)$; $v_l(k_2, i)_q = a(P_q/4 + k_2, i)/c(P_q/4 + k_2, i)$; $0 \le i \le (N/2) - 1$; $0 \le q \le Q - 1$; $0 \le k_1 \le (P/2) - 1$; $0 \le k_2 \le (P/4) - 1$; $N = PQ$

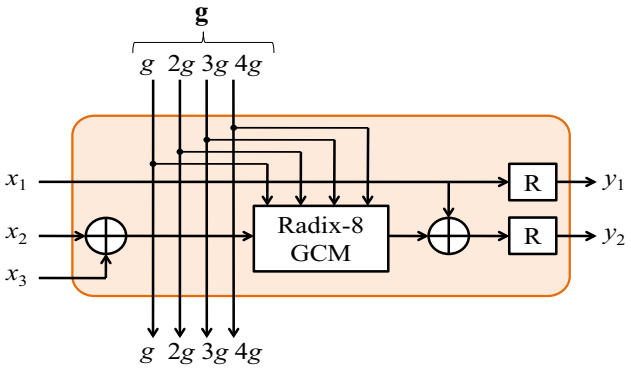

Fig. 10 Internal structure of modified functional element (FE)

block sizes ($P = 8, 16$) and image size (512×512). We have only considered the core of the lifting 2-D DWT of both the structures for comparison which is comprised of arithmetic unit and on-chip memory. We have assumed 8-bit input pixel, 12-bit width for the intermediate and output signals. Both the designs are synthesized in Synopsys Design Compiler using SAED 90-nm CMOS library. The area, data arrival time (DAT) and power reported by the Design Compiler are listed in Table 7 for comparison. Power

Table 7 Comparison of synthesis results of 2-D DWT structure for different block sizes

Designs	DAT (ns)	Area (μ sqm)	ADP (μ sqm s)	Power (mW)
Structure of [13] using radix-4 GC multiplier (P = 8)	23.80	665810.11	519.2	6.82
Structure of [13] using proposed radix-8 GC multiplier (P = 8)	22.20	601865.32	437.8	6.23
Structure of [13] using radix-4 GC multiplier (P = 16)	24.19	1194195.12	473.3	14.43
Structure of [13] using proposed radix-8 GC multiplier (P = 16)	22.38	1042259.46	382.2	12.76

consumption of both designs is estimated at common clock frequency of 50 MHz. We have also estimated ADP ($ADP = area \times DAT \times computation\ time$ in cycles) for both designs, where computation time = N^2/P. The estimated ADP is also listed in Table 7. Compared with the structure of [13], the proposed structure for block sizes 8 and 16 involve 15.6 and 19.3 % less ADP, respectively. For the same block sizes, the proposed structure dissipates 8.6 and 11.5 % less power than that of [13], respectively. Since, on-chip memory is a dominant component of 2-D DWT structure and its complexity is common in both the designs, the area-delay efficiency achieved in the proposed structure is considered to be significant. The proposed structure has higher ADP saving over the existing structure for higher block sizes.

6 Conclusions

In this paper, we present a regular PPA for radix-8 Booth multiplication by removing the extra row with a small overhead complexity. A radix-8 multiplier design is proposed based on the regular PPA which offers a saving of 10.7 % ADP over the existing radix-8 multiplier design. Few redundant logic operations are created in the adder unit when n lower-order bits of $2n$-bit multiplier output are truncated. We present an optimized adder unit design after removing all such redundant logics for post-truncated fixed-width radix-8 Booth multiplier. Comparison result shows that the proposed post-truncated fixed-width multiplier design offers nearly 20.7 % ADP saving and 18.3 % power saving over the existing radix-8 design optimized by Synopsys Design Compiler when the $2n$-bit output is post-truncated to n-bit. Besides, radix-8 multiplier design offers some area and delay saving when configured for constant multiplication, while the radix-4 multiplier design does not have this feature. We have shown that the proposed 12-bit full-width and fixed-width radix-8 generic multiplier designs,

respectively, involve 19.4 and 24.7 % less ADP than the existing radix-4 full-width and post-truncated multiplier designs when these designs are configured for constant multiplication. The existing block-based lifting 2-D DWT structure is synthesized using the proposed radix-8 generic-constant fixed-width multiplier and the existing radix-4 multiplier to demonstrate the effectiveness of proposed multiplier design scheme. We find that the existing lifting 2-D DWT structure of block size 16 and word length 12 offers 19.3 % ADP saving and 11.5 % power saving when the constant multipliers are implemented using the proposed radix-8 multiplier design instead of the existing radix-4 multiplier design.

References

1. N. Besli, R.G. Deshmukh, A novel redundant binary signed-digit (RBSD) Booth's encoding. In *Proceedings of IEEE Southeast Conference*, pp 426–431 (2002)
2. P. Bougas, P. Kalivas, A. Tsirikos, K.Z. Pekmestzi, Pipelined array-based FIR filter folding. IEEE Trans. Circuits Syst. I. **52**(1), 108–118 (2005)
3. S. Galal, O. Shacham, J. S. Brunhaver II, J. Pu, A. Vassiliver, M. Horowitz, FPU Generator for design Space Exploration. *IEEE 21st Symposium on Computer Arithmetic* (2013)
4. R.I. Hartley, Subexpression sharing in filters using canonic signed digit multipliers. IEEE Trans. Circuits Syst. II **43**(10), 677–688 (1996)
5. H.A. Huang, Y.C. Liao, H.C. Chang, A self-compensation fixed-width Booth multiplier and its 128-point FFT applications. *IEEE Int. Symp. Circuits Syst*, 3538–3541 (2006)
6. S.J. Jou, M.H. Tsai, Y.L. Tsao, Low-error reduced-width Booth multipliers for DSP applications. IEEE Trans. Circuits Syst.-I **50**(11), 1470–1474 (2003)
7. S.S. Kidambi, F. El-Guibaly, A. Antoniou, Area-efficient multipliers for digital signal processing applications. IEEE Trans. Circuits Syst. II **43**(2), 90–95 (1996)
8. S.R. Kuang, J.P. Wang, C.Y. Guo, Modified Booth multipliers with a regular partial product array. IEEE Trans. Circuits Syst. II Express Briefs **56**(5), 404–408 (2009)
9. H.Y. Lee, I.C. Park, Balanced binary-tree decomposition for area efficient pipelined FFT processing. IEEE Trans. Circuits Syst. I **54**(4), 889–900 (2007)
10. C.Y. Li, Y.H. Chen, T.Y. Chang, J.N. Chen, A probabilistic estimation bias circuit for fixed-width Booth multiplier and its DCT application. IEEE Trans. Circuits Syst. II Express Briefs **58**(4), 215–219 (2011)
11. B.K. Mohanty, P.K. Meher, M.N.S. Swamy, Low-area and low-power reconfigurable architecture for convolution-based 1-D DWT using 9/7 and 5/3 Filters. 28 *International Conference on VLSI Design*, Bangaluru, 27-533 (2015)
12. B.K. Mohanty, P.K. Meher, Memory-efficient modular VLSI architecture for high-throughput and low-latency implementation of multilevel lifting 2-D DWT. IEEE Trans. Signal Process. **59**(5), 2072–2084 (2011)
13. B.K. Mohanty, A. Mahajan, P.K. Meher, Area and power-efficient architecture for high-throughput implementation of lifting 2-D DWT. IEEE Trans. Circuits Syst. II Express Briefs **59**(7), 434–438 (2012)
14. K.L.S. Swee, L.H. Hiung, Performance comparison review of radix-based multiplier designs, 4th *International conference on intelligent and advanced systems* (ICIAS), pp 854–859 (2012)
15. X. Tian, L. Wu, Y.H. Tan, J.W. Tian, Efficient multi-input/multi-output VLSI architecture for two-dimensional lifting-based discrete wavelet transform. IEEE Trans. Comput. **60**(8), 1207–1211 (2011)
16. S.A. White, Applications of distributed arithmetic to digital signal processing: a tutorial review. IEEE ASSP Mag. **6**, 4–19 (1989)
17. W. Zhang, Z. Jiang, Z. Gao, Y. Liu, An efficient VLSI architectures for lifting-based discrete wavelet transform. IEEE Trans. Circ. Syst. II Express Briefs. **59**(3), 158–162 (2012)