CHAPTER -5
CONSTANT RADIX-8 BOOTH MULTIPLIER AND EFFICIENT LIFTING 2-D DWT ARCHITECTURE

5.1 INTRODUCTION

In a typical DSP algorithm, several multiplication operations are performed. Fixed-width multipliers are used to implement these algorithms in VLSI systems to avoid infinite word-length growth. 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 (i) direct post-truncation of 2*n*-bit output referred to *post-truncated* multiplier, and (ii) 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 [Kidambi *et.al* (1996)]. Several error compensation approaches have been suggested in the literature to reduce the error magnitude of truncated multiplier at the cost few overhead logic [Hung *et.al* (2006), Jou *et.al* (2003), Li *et.al* (2011)]. 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 the same amount logic resource as the full-width multiplier, whereas the error compensated truncated multiplier designs involves less logic resources than the post-truncated multiplier design but introduces more truncation error than post-truncated multiplier. Although, the existing error compensated truncated multipliers 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 2*n*-bit multiplier output are truncated. The modern synthesis tools are power full enough to optimize the design during the synthesis by removing the redundant logic operations when any signal is left unused in the design. However, it is observed that the synthesis tools not able to remove all the redundant logic operations resulted when *n* LSBs of 2*n*-bit multiplier output of full-width multiplier design are post-truncated. A specific design approach is necessary to remove all the redundant logics.

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 are variables, whereas *constant multiplier* has one operand is a constant and the other one is 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 designs developed in the literature 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 look-up tables (LUT). 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 [Hartley *et.al* (1996), White *et.al* (1989)]. In general, the fixed-constant multiplier is designed for a fixed value, and its complexity depends on the value of the fixed-constant. Therefore, fixed-constant multiplier needs a 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 design for a specific algorithm and value of the constants can be changed conveniently according to the application requirement whereas separate structures need to design 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 useful for specific application.

Lifting DWT structure involves constant multipliers. But, the lifting DWT algorithm cannot be expressed in the form of inner-product computation [Mohanty *et. al* (2015b)]. 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 based on generic constant multiplier. A typical lifting 2-D DWT structure involves $4N$ on-chip memory words 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. Consequently, lifting 2-D DWT structures offers less ADP for higher block sizes. But, multiplier complexity of block-based lifting 2-D DWT structure increases proportionately with the block-size. Since, lifting 2-D DWT algorithm does not have any redundant computation; there is no scope to reduce multiplier complexity of the block structure without compromising on the throughput rate. However, it is observed that many multipliers of block lifting 2-D DWT structures [Mohanty and Meher (2011), Mohanty *et. al* (2012), Tian *et. al* (2011), Zhang *et. al* (2012)] share a common input operand. Common input multiplier operand is an interesting feature of block lifting 2-D DWT structure and could be exploited in a hardware design using higher radix Booth multipliers which is currently missing in the literature. A group of multipliers with a common multiplying operand can select their partial product terms from a common set using Booth encoding scheme. A significant amount of ADP and power could be saved if the advantage of common multiplier input operand feature block-based lifting 2-D DWT algorithm is taken through a suitable design method. To demonstrate the effectiveness of the proposed scheme, in this Chapter a logic optimized design is proposed for implementation of radix-8 generic constant fixed-width multiplier. The proposed radix-8 generic constant fixed-width multiplier is subsequently used to obtain an area-delay efficient lifting 2-D DWT structure. However, the proposed radix-8 fixed-width generic constant multiplier design is suitable for other DSP algorithms involving constant multiplication as well.

The remaining Chapter is organized as follows: The proposed constant fixed-width radix-8 Booth multiplier is presented in Section 5.2 and its hardware-time complexities are discussed in

Section 5.3. The proposed lifting 2-D DWT structure is presented in Section 5.4. Hardware and time complexity of 2-D DWT structure is discussed in Section 5.5. Conclusion is presented in Section 5.6.

5.2 PROPOSED RADIX-8 FIXED-WIDTH BOOTH MULTIPLIER DESIGN

5.2.1 GENERIC CONSTANT BOOTH MULTIPLIER

The generic radix-4 Booth multiplier design [Kunag *et.al* (2009)] and the radix-8 Booth multiplier design proposed in Section 4.4 easily can be configured to have a generic-constant multiplier for constant multiplication. However, some interesting feature is observed in radix-4 and radix-8 Booth multiplier design. The internal structure of PPG unit of radix-4 and radix-8 multiplier designs is shown in Figure 5.1. As shown in this figure, the PPG unit of radix-4 Booth multiplier design does not involve any adder as the partial product terms {A, 2A} are generated from the A using one shifter only, where A is the multiplicand operand. But, in case of radix-8 Booth multiplier design the PPG unit involves one adder to generate the partial product set {A, 2A, 3A, 4A}. When A is constant, the partial product terms of both radix-4 and radix-8 Booth multiplication can be pre-computed and send to the partial product selection (PPS) unit directly as the input without using PPG unit. Removing PPG unit from the radix-4 multiplier design does not offer any saving in logic resource or critical-path delay since the shifter is implemented through hard-wiring. However, removing the PPG unit from the radix-8 multiplier design offers a saving in logic resource and critical-path delay. As shown in Figure 5.1(b), one (n+2)-bit adder is saved and the critical-path is reduced by one (n+2)-bit adder delay on removal of PPG unit, where n is the input operand word length. Therefore, a generic radix-8 Booth multiplier design when configured for constant multiplication then one (n+2)-bit adder and the critical-path delay is saved in the hardware design. But, generic radix-4 Booth multiplier design does not offer any such saving when configured for constant multiplication. This is an important feature of radix-8 multiplier design and could be an advantage when the multiplier design is configured for constant multiplication.

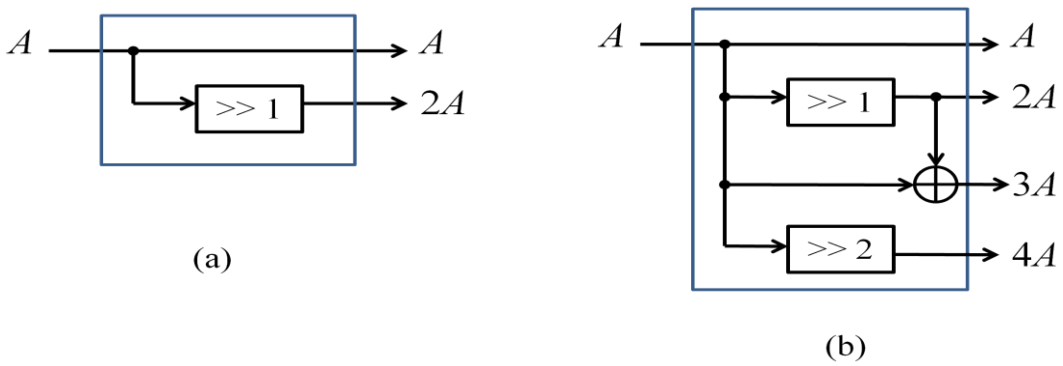
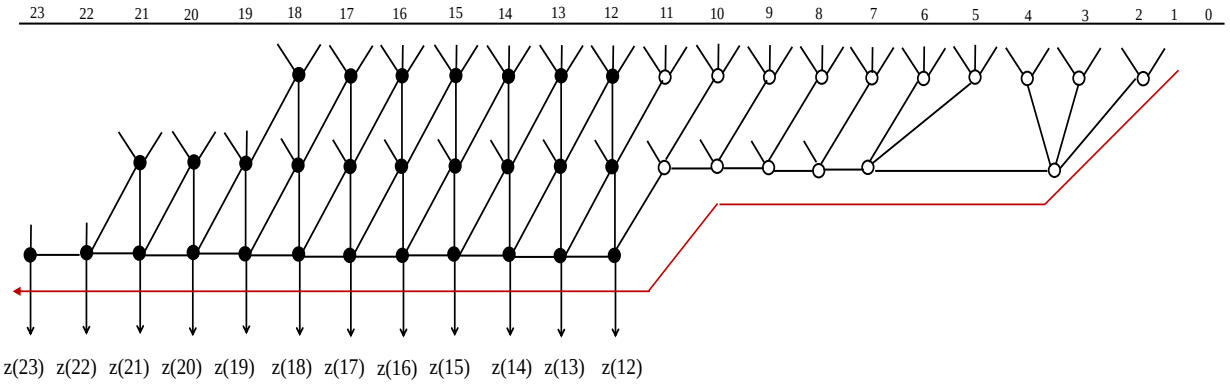


Figure 5.1: Partial product generation unit. (a) Radix-4 Booth multiplier. (b) Radix-8 Booth multiplier.

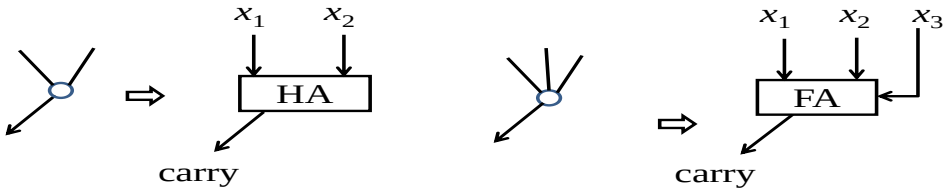
5.2.2 FIXED-WIDTH RADIX-8 BOOTH MULTIPLIER DESIGN

The radix-8 multiplier design proposed in Section 4.4 is referred as full-width radix-8 Booth multiplier since it produces 2*n*-bit output. A fixed-width multiplier produces *n*-bit output for *n*-bit input operand. In a straight-forward manner, the *n* LSBs of 2*n*-bit multiplier output (full-width) are truncated to obtain *n*-bit (fixed-width) multiplier output. The adder unit of full-width multiplier involves few redundant logic operations when *n* LSBs of 2*n*-bit multiplier output are truncated. The modern synthesis tools are power full enough to optimize the design during the synthesis by removing the redundant logic operations when any signal is left unused in the design. However, it is observed that the synthesis tools not able to remove all the redundant logic operations resulted when *n* LSBs of 2*n*-bit multiplier output of full-width multiplier design are truncated. The adder unit of full-width radix-8 Booth multiplier design is optimized to remove the redundant logic operations resulted due to post-truncation of *n* LSBs of 2*n*-bit multiplier output. Since the sum bit corresponding to *n* least significant columns (LSCs) of the PPA of Figure 4.13 are truncated in the final 2*n*-bit output, the partial products of *n* LSCs are accumulated by carry generator while the partial products of remaining *n* most significant columns (MSCs) are accumulated using bit adders. The bit-flow diagram of Figure 4.13 (a) 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-with radix-8 multiplier is shown in Figure 5.2 (a) for *n*=12. Replacing the adder unit of radix-8 multiplier design of Figure 4.10 with the optimized adder unit design of Figure 5.2 (a), an optimized radix-8 fixed-width

multiplier design is obtained. The proposed radix-8 fixed-with 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.



(a)



(b)

Figure 5.2: (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.

5.2.3 HARDWARE-TIME COMPLEXITIES OF RADIX-4 AND RADIX-8 CONSTANT/FIXED-WITH BOOTH MULTIPLIER DESIGNS

The radix-4 generic constant full-width and fixed-width (post-truncated) multiplier designs are derived from the radix-4 generic full-width multiplier design of [Kuang *et.al* (2009)], whereas the radix-8 generic constant full-width multiplier design and generic constant fixed-width (post-truncated) multiplier designs are derived from the generic full-width multiplier structure of [Galal *et. al* (2013)] 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 [Kuang

et.al (2009)]. But, the derived radix-8 generic constant full-width multiplier design and post-truncated multiplier design involve one less 14-bit adder than those required by the generic full-width multiplier design of [Galal *et. al* (2013)]. The proposed radix-8 generic constant full-width multiplier design is identical with design of Figure 4.11 without the PPG unit. The proposed fixed-width multiplier design is identical with the design of Figure 4.11 except that the adder unit uses the logic optimized adder unit of Figure 5.2 (a).

Table 5.1: Comparison of Hardware and Time Complexities of Constant/Fixed-width Radix-4 and radix-8 Booth multipliers for 12-bit

Multiplier Design	OR/NOR/AND /NAND gate	XOR/ XNOR gate	Critical Path
Radix-4 GC/post-truncated [*Kuang et. al* (2009)]	840	295	$9T_x+21T_{OR}+3T_{NAND}+T_{NOR}+20T_{AND}$
Radix-8 GC /post-truncated [*Galal et. al* (2013)]	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 post-truncated	734	244	$3T_x+3T_{NOR}+21T_{AND}+T_{NAND}+16T_{OR}$

Legend: GC: Generic constant.

The hardware complexity and time-complexity of proposed and existing multiplier designs are estimated for input word length n=12. The hardware complexity is estimated in terms of gate counts while the critical-path delay is estimated in terms of gate delay. The estimated values are listed in Table 5.1 for comparison. As shown in Table 5.1, the proposed radix-8 generic constant full-width multiples design involves marginally less number of gates than those of [Galal *et. al* (2013)] and CPD less by ($2T_X+T_{OR}$). But, the proposed radix-8 generic constant fix-width (post-truncated) multiples design involves 71 less gates than those of [Galal *et. al* (2013)], and the CPD less by ($5T_X+3T_{OR}$). The proposed radix-8 generic constant full-width multiplier design involves 88 less gates and the CPD by ($3T_X+3T_{OR}+2T_{AND}-2T_{NOR}$) compared to those of radix-4 generic constant full-width multiplier design of [Kuang *et.al* (2009)]. The proposed radix-8 generic constant fixed-width (post-truncated) multiplier design involves 157 less gates and CPD

less by ($6T_X$+$7T_{OR}$-$2T_{NOR}$-T_{AND}) as compared to the radix-4 generic constant post-truncated multiplier design of [Kuang *et.al* (2009)] and introduces the same amount of truncation error to the multiplier output as the radix-4 generic post-truncated multiplier design of [Kuang *et.al* (2009)].

5.2.4 COMPARISON OF SYNTHESIS RESULT

The 2*n* bit output of existing full-width radix-4 multiplier design of [Kuang *et.al* (2009)] and the existing full-width radix-8 multiplier design of [Galal *et. al* (2013)] are truncated by discarding *n* 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 [Galal *et. al* (2013)] and the existing radix-4 generic constant full-width and direct post-truncated multiplier designs of [Kuang *et.al* (2009)] are coded in VHDL for *n* =12. All these designs are synthesized in Synopsys Design Compiler using SAED 90nm CMOS library. The area, delay and power estimates of these designs reported by Design Compiler are listed in Table 5.2 and Table 5.3 for comparison. As shown in Table 5.2, the radix-8 generic constant full-width multiplier design of [Galal *et. al* (2013)] involves 4.5% less area and marginally less delay compared to those of radix-4 generic constant full-width multiplier design of [Kuang *et.al* (2009)]. The proposed radix-8 generic constant full-width multiplier design involves 10% less area and 10.4% less CPD than those of radix-4 generic constant full-width multiplier design of [Kuang *et.al* (2009)].

Table 5.2: Comparison of synthesis result of full-width constant Booth multiplier designs

Multiplier design	Area (μm^2)	DAT(ns)	ADP ($\mu m^2 \times ms$)	Power (mW)
Radix-4 [Kuang *et. al* (2009)]	7344.22	16.11	118315.38	39.9
Radix-8 [Galal *et. al* (2013)]	6999.35	15.20	106390.12	37.6
Proposed radix-8	6597.30	14.42	95133.06	32.8

Legend: GC: Generic constant

Table 5.3: Comparison of synthesis result of generic constant fixed-width (post-truncated) Booth multiplier designs

Multiplier design	Area (μm^2)	DAT (ns)	ADP ($\mu m^2 \times ms$)	Power (mW)
Radix-4 [Kuang *et. al* (2009)]	6904.24	14.10	97349.78	32.7
Radix-8 [Galal *et. al* (2013)]	6607.90	13.97	92312.36	31.6
Proposed radix-8	5562.36	13.17	73256.28	23.6

As mentioned earlier, the Design Compiler optimizes the design when certain output signals are not assigned. In Table 5.3, the multiplier design of [Kuang *et.al* (2009)] and [Galal *et. al* (2013)] represents the direct post-truncated design while the proposed design represents a logic optimized radix-8 fix-width multiplier design. From Table 5.2 and 5.3, one can find that the radix-4 generic constant fixed-width (direct post-truncated) design of [Kuang *et.al* (2009)] involves nearly 5.9% less area and marginally less delay than the corresponding constant full-width multiplier design of [Kuang *et.al* (2009)]. Similarly, the radix-8 generic constant fixed-width (direct post-truncated) design of [Galal *et. al* (2013)] involves nearly 5.6% less area and marginally less delay than the corresponding constant full-width multiplier design of [Galal *et. al* (2013)]. The proposed generic constant fixed-width multiplier design involves 15.8% less area and 5.7% less delay than the radix-8 generic constant fixed-with (direct post-truncated) multiplier design of [Galal *et. al* (2013)]. It involves 19.4% less area and 6.6% less delay than the radix-4 generic constant fixed-width (direct post-truncated) multiplier design of [Galal *et. al* (2013)]. 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 fixed-width (direct post-truncated) radix-8 multiplier design of [Galal *et. al* (2013)]. It involves 24.7% less ADP and consumes 27.8% less power than the generic constant fixed-width (direct post-truncated) radix-4 multiplier design of [Kuang *et.al* (2009)]. 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 fixed-width (direct post-truncated) multiplier design of [Galal *et. al* (2013)] and radix-4 generic constant fixed-width (direct post-truncated) multiplier design of [Kuang *et.al* (2009)].

It is well known that radix-4 Booth multiplier design mostly preferred over the other higher radix Booth multipliers designs as it requires relatively less area, delay and power than others radix multipliers. But, an optimized radix-8 generic constant Booth multiplier design involves less ADP and consumes less power than the radix-4 generic constant Booth multiplier for selected input operand bit-width. As shown in this Section that radix-8 Booth multiplier design involves less ADP than the radix-4 multiplier design for 12-bit implementation. Therefore, radix-8 multiplier should be preferred over the radix-4 multiplier for implementation of 12-bit constant multiplication. The proposed radix-8 generic constant fixed-width multiplier design is chosen over the existing radix-4 generic constant Booth multiplier design of [Kuang *et.al* (2009)] to develop area-delay efficient fixed-point parallel architectures of lifting 2-D DWT. This is discussed in the following Section.

5.3 PROPOSED LIFTING STRUCTURE OF 2-D DWT

Block 2-D DWT structure efficiently utilizes on-chip memory and frame-memory. But, the adder and multiplier complexity of block structure increases proportionately with the block-size. Multiplier consumes more combinational logic than the adders. Therefore, optimization of multiplier complexity can helps to improve the hardware efficiency of block structures. Interestingly, the lifting based 2-D DWT structure involves constant multiplication. The lifting 2-D DWT structure of block-size '*S*' involves (4.5*S*) multipliers [Mohanty *et. al* (2012)]. One operand of these (4.5*S*) multipliers is one of the four lifting constants $\{\alpha, \beta, \gamma, \delta\}$ or two scaling constants $\{k^2, 1/k^2\}$ and they are distributed into six groups. Multipliers of a particular group share a common input operand. The multiplication of a group of variables with a common operand can be expressed in terms of *scalar-vector multiplication* operation. An *N*-point scalar-vector multiplier involves *N* multiplication, where *N* is the vector size. When these multiplications are implemented in parallel using *N* Booth multipliers then there is scope to share the partial product generation unit of *N* multipliers. To take advantage this feature in hardware design the lifting computation is expressed in modified form using scalar-vector multiplication.

5.3.1 BLOCK FORMULATION OF LIFTING DWT USING SCALAR-VECTOR MULTIPLICATION

Suppose a block of N samples $\mathbf{x} = \{x(n), x(n\text{-}1), \ldots, x(n\text{-}N\text{+}1)\}$ are transformed using lifting DWT and calculates one block of low-pass sub-band ($\mathbf{u}_l$) and high-pass sub-band ($\mathbf{u}_h$) of size ($N/2$) each. The lifting computation of a block of N-point samples is expressed in scalar-vector form as:

$$\mathbf{r}_1(n) = \mathbf{x}_2(n) + \alpha(\mathbf{x}_1(n) + \mathbf{x}_3(n)) \quad 5.1$$

$$\mathbf{r}_2(n) = \mathbf{x}_3(n) + \beta(\mathbf{r}_1(n) + \mathbf{r}_1(n-1)) \quad 5.2$$

$$\mathbf{u}_h(n) = \mathbf{r}_1(n-1) + \gamma(\mathbf{r}_2(n) + \mathbf{r}_2(n-1)) \quad 5.3$$

$$\mathbf{u}_l(n) = \mathbf{r}_2(n-1) + \delta(\mathbf{u}_h(n) + \mathbf{u}_h(n-1)) \quad 5.4$$

where,

$$\mathbf{x}_1(n) = [x(2n), x(2n-2), \ldots, x(2n-N+2)]^T \quad 5.5$$

$$\mathbf{x}_2(n) = [x(2n-1), x(2n-3), \ldots, x(2n-N+1)]^T \quad 5.6$$

$$\mathbf{x}_3(n) = [x(2n-2), x(2n-4), \ldots, x(2n-N)]^T \quad 5.7$$

$$\mathbf{r}_1(n) = [r_1(n), r_1(n-1), \ldots, r_1(n-(\tfrac{N}{2})+1)]^T \quad 5.8$$

$$\mathbf{r}_2(n) = [r_2(n), r_2(n-1), \ldots, r_2(n-(\tfrac{N}{2})+1)]^T \quad 5.9$$

$$\mathbf{u}_l(n) = [u_l(n), u_l(n-1), \ldots, u_l(n-(\tfrac{N}{2})+1)]^T \quad 5.10$$

$$\mathbf{u}_h(n) = [u_h(n), u_h(n-1), \ldots, u_h(n-(\tfrac{N}{2})+1)]^T \quad 5.11$$

5.3.2 BLOCK LIFTING STRUCTURE FOR 1-D DWT

A full parallel structure for computation of *N*-point lifting DWT is derived using the (5.1)-(5.4) and shown in Figure 5.3. Input-vectors $\{\mathbf{x}_1(n), \mathbf{x}_2(n), \mathbf{x}_3(n)\}$ each of size $(N/2)$ are derived from the *N*-point vector $\{\mathbf{x}(n)\}$ according to (5.5)-(5.7). As shown in Figure 5.3, the full-parallel structure uses 4 scalar-vector multiplier units (SVMUs) and 8 adder units (AUs) of size $(N/2)$ each.

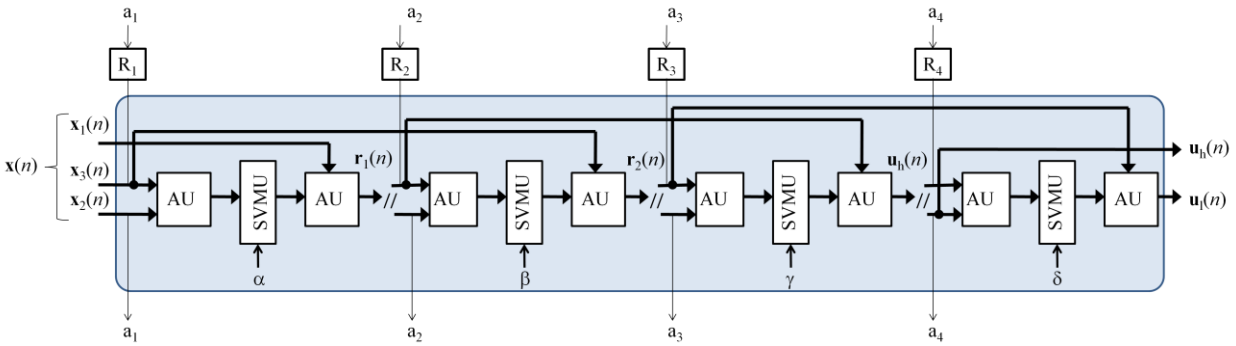


Figure 5.3: Full-parallel structure for computation of *N*-point lifting DWT. Symbol {//} represent the data-overlapping

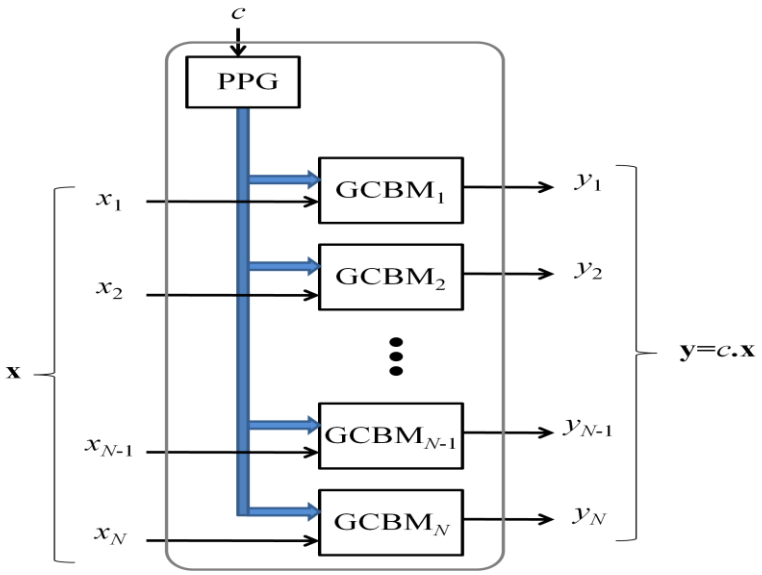


Figure 5.4: Structure of *N*-point scalar-vector multiplier unit

A generalized structure of N-point scalar-vector multiplier unit (SVMU) is shown in Figure 5.4 using generic constant Booth multiplier (GCBM). The SVMU receives an N-point input-vector $\{\mathbf{x}\}$ and produce the N-point output-vector $\{\mathbf{y}\}$ which is the scaled version of $\{\mathbf{x}\}$. Therefore, each (N/2)-point SVMU of the full-parallel lifting DWT structure comprises of (N/2) GCBMs, and each (N/2)-point AU is comprised of (N/2) 2-input adders. According to (5.8)-(5.11), the pair of data-vectors $\{\mathbf{r}_1(n), \mathbf{r}_1(n\text{-}1)\}$, $\{\mathbf{r}_2(n), \mathbf{r}_2(n\text{-}1)\}$, $\{\mathbf{u}_h(n), \mathbf{u}_h(n\text{-}1)\}$ are overlapped by $\{(N/2)\text{-}1\}$ samples with one non-overlapping sample. Therefore, data-vectors $\{\mathbf{r}_1(n\text{-}1), \mathbf{r}_2(n\text{-}1), \mathbf{u}_h(n\text{-}1)\}$ are obtained from $\{\mathbf{r}_1(n), \mathbf{r}_2(n), \mathbf{u}_h(n)\}$ through data-overlapping operation denoted by symbol {//} in Figure 5.3. The non-overlapping sample of $\{\mathbf{r}_1(n\text{-}1), \mathbf{r}_2(n\text{-}1), \mathbf{u}_h(n\text{-}1)\}$ are obtained from three registers named $\{R_2, R_3, R_4\}$. The register R_1 holds the input sample $\{x(n\text{-}N)\}$ of data-vector $\{\mathbf{x}_3(n)\}$ which has $\{(N/2)\text{-}1\}$ overlapped samples with $\{\mathbf{x}_2(n)\}$. The operation is implemented in hardware through hard-wiring. The full-parallel structure of Figure 5.3 easily can be scaled for different block sizes by scaling the SVMU and AU and this structure is used as building block to develop structures for lifting 2-D DWT.

5.3.3 PROPOSED LIFTING 2-D DWT STRUCTURE USING DATA-BLOCKS {(P/2)×2}

Recently, Mohanty *et. al* (2012) have suggested a block-based lifting 2-D DWT structure, where a block of {(P/2)×2} samples are processed in each clock cycles and produces one block of (P/4) components of four sub-bands (**A**, **B**, **C**, **D**) of one-level lifting 2-D DWT. This structure is designed using radix-4 Booth multiplier. This structure is considered as a reference design and introduces the necessary modification into the design by replacing radix-4 Booth multiplier with the proposed radix-8 GCBM to its advantage. The modified structure is shown in Figure 5.5 which is identical to the original structure of [Mohanty *et. al* (2012)], except that each multiplier of the 2-D DWT structure of [Mohanty *et. al* (2012)] is implemented using the proposed radix-8 GCBM. 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 lifting constants$\{\alpha,\beta,\gamma,\delta\}$, and fed these vectors to the structure as an external input from the user. As shown in Figure 5.5, the input-vectors $\{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \mathbf{g}_4\}$ are propagated across the processing elements (PEs) of row and column processors as multipliers of these PEs use lifting

constants $\{\alpha, \beta, \gamma, \delta\}$. The modified structure of 1-D array of row-processor using radix-8 generic-constant multiplier is shown in Figure 5.6. The modified 1-D array uses four functional (FUs) corresponding to four lifting constants $\{\alpha, \beta, \gamma, \delta\}$. The modified structure of FU is shown is Figure 5.7, which is comprised of one radix-8 GCBM and two adders. The radix-8 GCBM receives partial-product values $\{ g, 2g, 3g, 4g \}$ through the input-vector '**g**', where '**g**' takes one of the four lifting constants $\{\alpha, \beta, \gamma, \delta\}$. The modified structure of low-pass/high-pass block (2-D array) is shown in Figure 5.8, where each SVMU and AU of size (P/4). The 1-D arrays of the row-processor and the low-pass/high/pass blocks of column processor share the partial product terms $\{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \mathbf{g}_4\}$ of four lifting constants $\{\alpha, \beta, \gamma, \delta\}$. These partial product terms are pre-computed and buffered in a separate PPG unit. The partial product terms $\{\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, \mathbf{g}_4\}$ are transmitted to each 1-D array of row-processor and low-pass/high-pass blocks of column processor through wiring to share the partial product terms.

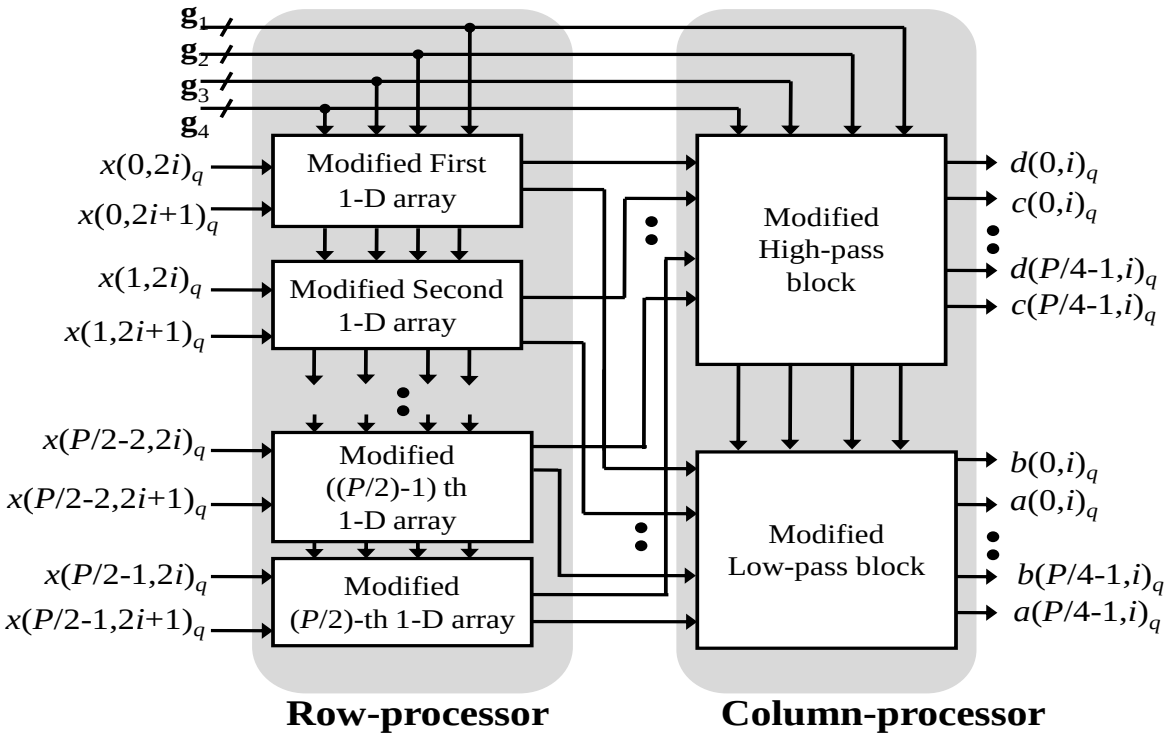


Figure 5.5: Modified lifting 2-D DWT structure, where $x(k, i)q = x(Pq + k, i)$, $a(j, i)_q = a(Pq/4 + j, i)$, $b(j, i)_q = b(Pq/4 + j, i)$, $c(j, i)_q = c(Pq/4 + j, i)$, and $d(j, i)_q = d(Pq/4 + j, i)$, where $0 \le q \le Q - 1$, $0 \le i \le (N/2) - 1$, $0 \le k \le (P/2) - 1$, $0 \le j \le (P/2) - 1$

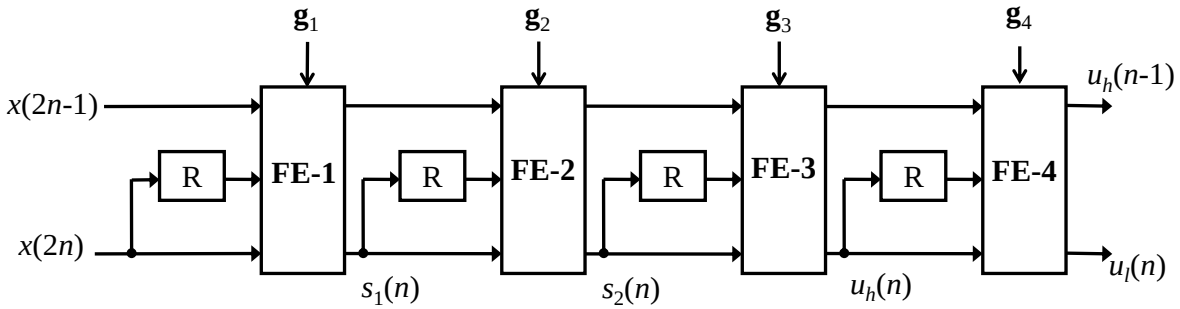


Figure 5.6: Modified structure of 1-D array

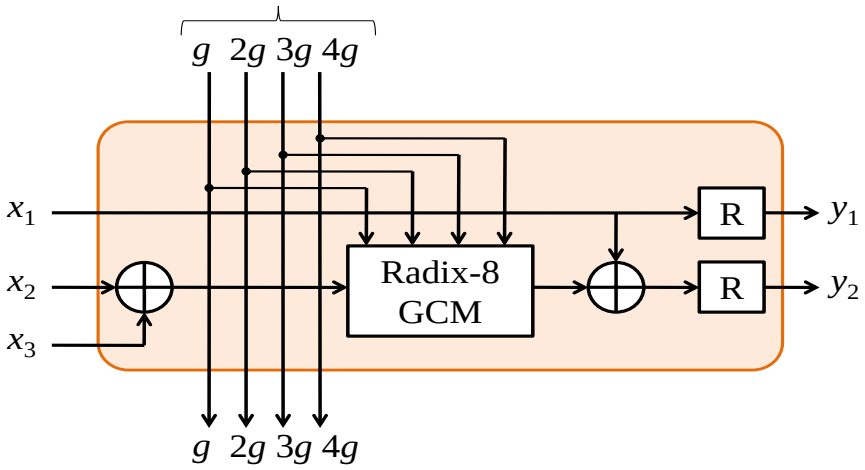


Figure 5.7: Internal structure of modified functional element (FE)

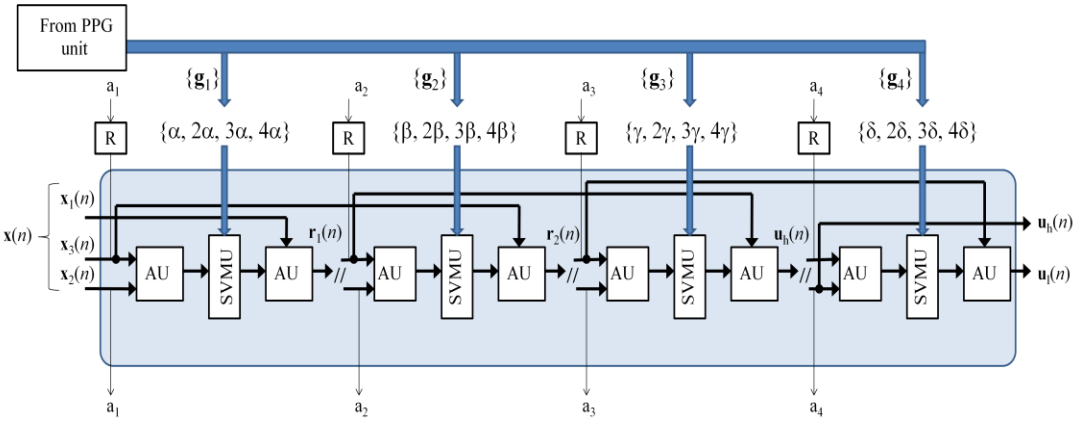


Figure 5.8: Modified structure of high pass/low pass block

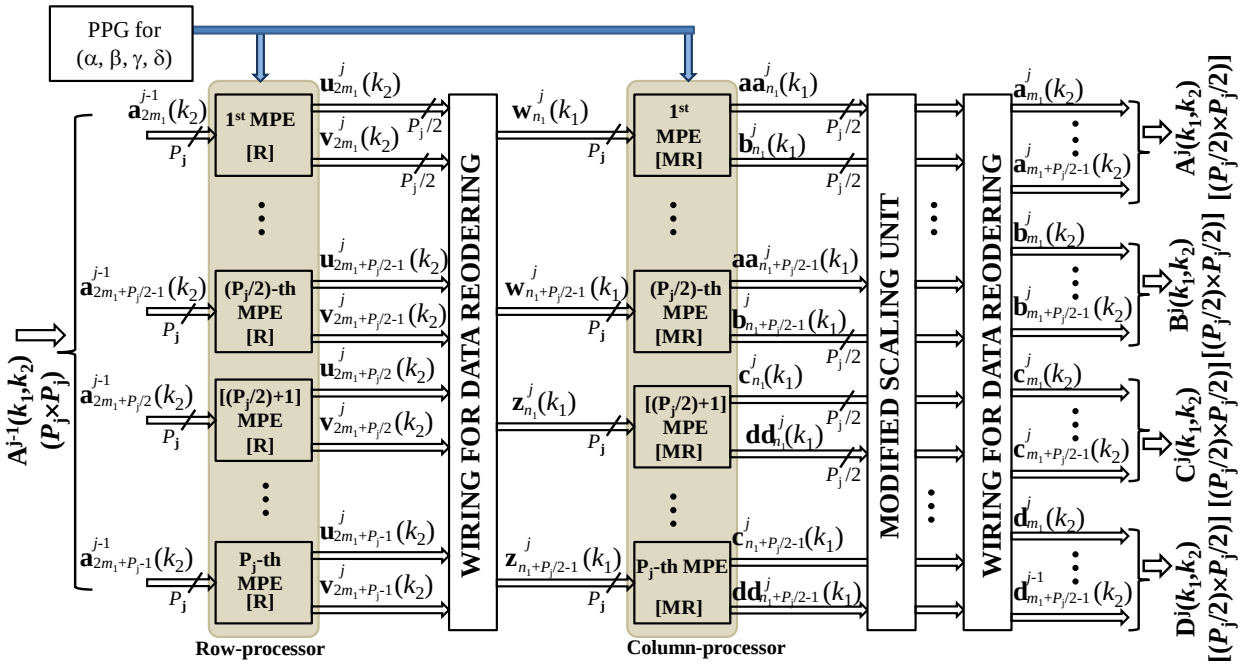


Figure 5.9: Modified structure of the processing unit for computation of *j*-th level lifting 2-D DWT

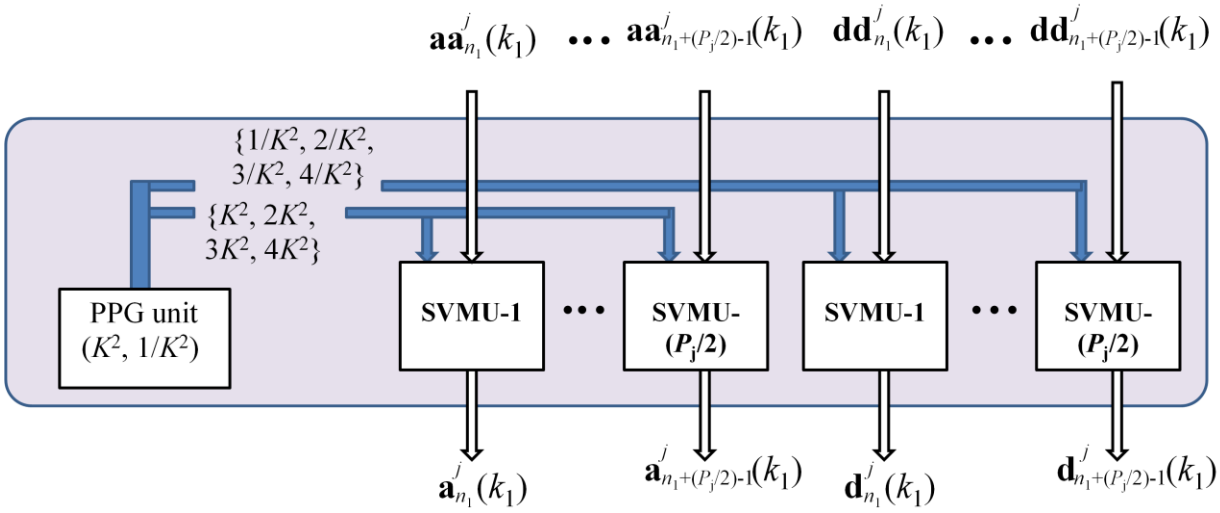


Figure 5.10: Modified structure of the scaling unit

5.3.4 PROPOSED LIFTING 2-D DWT STRUCTURE USING DATA-BLOCKS (*P*×*P*)

In Section 3.4.1, a generic design of the *j*-th PU is presented for multi-level lifting computation of 2-D DWT. The *j*-th PU receives ($P_j \times P_j$) size data-blocks of (*j*-1)-th level low-low sub-band $\mathbf{A}^{j-1}(k_1,k_2)$ from the (*j*-1)-th PU and computes the data-blocks $\mathbf{A}^{j}(k_1,k_2), \mathbf{B}^{j}(k_1,k_2), \mathbf{C}^{j}(k_1,k_2)$ $\mathbf{D}^{j}(k_1,k_2)$ of size (P_j/2)×(P_j/2) corresponding to four sub-bands [$\mathbf{A}^j$, $\boldsymbol{B}^j$, $\boldsymbol{C}^j$, $\mathbf{D}^j$] of *j*-th level DWT. This structure is designed using radix-4 generic Booth multiplier. This structure is considered as a reference design and introduces the necessary modification into the design by replacing radix-4 generic Booth multiplier with the proposed radix-8 GCBM to take its advantage. The modified structure is shown in Figure 5.9 which is identical to the original structure of Figure 3.7 except that each multiplier of the structure is implemented using the proposed radix-8 GCBM. The partial product terms {$\mathbf{g}_1$, $\mathbf{g}_2$, $\mathbf{g}_3$, $\mathbf{g}_4$} corresponding four lifting constants $\{\alpha,\beta,\gamma,\delta\}$ are pre-computed and buffered in a separate PPG unit. From the PPG unit, the partial product terms {$\mathbf{g}_1$, $\mathbf{g}_2$, $\mathbf{g}_3$, $\mathbf{g}_4$} are transmitted to each modified processing element (MPE) of row-processor and column processor through wiring. The structure of the modified processing element is identical to the modified 2-D array of Figure 5.8. Each SVMU and AU of MPE of sizes (P_j/2). The modified structure of the scaling unit is shown in Figure 5.10.

5.4 HARDWARE COMPLEXITY OF LIFTING 2-D DWT STRUCTURE

5.4.1 COMPARISON OF PROPOSED 2-D DWT STRUCTURE FOR DATA BLOCKS {(*P*/2)×2}

The proposed structure for block-size {(*P*/2)×2} and the structure of [Mohanty *et. al* (2012)] are identical and they involve the same amount of on-chip, frame-memory, multipliers and adders for a given image-size and block size. However, these two structures have different combinational logic complexity since the proposed structure uses radix-8 generic constant Booth multiplier and the structure of [Mohanty *et. al* (2012)] uses radix-4 generic constant Booth multiplier. To estimate the area saving offered by the proposed radix-8 generic constant Booth multiplier when used in 2-D DWT structure, the proposed 2-D DWT structure of block size {(*P*/2)×2} and the structure of [Mohanty *et. al* (2012)] and coded in VHDL for block-sizes (*S*=8,

16) and image-size (512×512). The radix-4 generic constant Booth multiplier design of [Kuang *et.al* (2009)] is considered for the structure of [Mohanty *et. al* (2012)]. The core of the proposed lifting 2-D DWT structure and the structure of [Mohanty *et. al* (2012)] is considered for synthesis which comprising of arithmetic unit and on-chip memory. Assumed 8-bit input pixel, and 12-bit width for intermediate and output signals. Both the designs are synthesized in Synopsys Design Compiler using SAED 90 nm CMOS library. The area, minimum cycle period (MCP) and power consumption reported by the Design Compiler are listed in Table 5.4 for comparison. Power consumption of both designs is estimated at common clock of frequency 50 MHz. Also estimated ADP (*ADP= area* × *MCP* × *computation time* in cycles) for both designs, where computation time = N^2/S. As shown in Table 5.4, the proposed structure for block size 8 and 16 involves 10% and 13% less area than the structure of [Mohanty *et. al* (2012)]. This area saving mainly due to the advantage of radix 8 generic constant Booth multiplier over radix 4 generic constant Booth multiplier. The proposed structure for block sizes 8 and 16 involve 15.6% and 19.3% less ADP compared to the structure of [Mohanty *et. al* (2012)] for block size 8 and 16, respectively. For the same block-sizes, the proposed structure dissipates 8.6% and 11.5% less power than that of [Mohanty *et. al* (2012)], respectively.

Table 5.4: Comparison of synthesis results of proposed 2-D DWT structure for data blocks {(*P*/2)×2}

Designs	Block size	MCP (ns)	Area (μm^2)	ADP (μm^2×ms)	Power (mW)
Structure of [Mohanty *et. al* (2012)]	8	23.80	665810.11	519.2	6.82
Structure of [Mohanty *et. al* (2012)]	16	24.19	1194195.12	473.3	14.43
Proposed structure	8	22.20	601865.32	437.8	6.23
Proposed structure	16	22.38	1042259.46	382.2	12.76

5.4.2 COMPARISON OF PROPOSED 2-D DWT STRUCTURE FOR DATA BLOCKS (*P*×*P*)

The proposed 2-D DWT for data blocks (*P*×*P*) is suitable for multi-level decomposition. To study the area saving offered by the proposed radix-8 generic constant Booth multiplier, the proposed 2-D DWT structure for data blocks (*P*×*P*) and the existing multi-level 2-D DWT structure of [Hu and Jong (2013)] are coded in VHDL for block-sizes (16=4×4) and (64=16×16), and image-size (256 × 256). The core comprising of arithmetic unit and on-chip memory is only considered for comparison. Both the designs are synthesized in Synopsys Design Compiler using SAED 90 nm CMOS library. The area, minimum cycle period (MCP) and power consumption reported by the Design Compiler are listed in Table 5.5 for comparison. As shown in Table 5.5, for DWT level *J*=2 and block size 16, the proposed structure has 4% less MCP and 4% less area than those of [Hu and Jong (2013)]. For DWT level *J*=3 and block size 64, the proposed structure has 5% less MCP and 20% less area than those of [Hu and Jong (2013)]. Due to saving in area and critical-path delay, the proposed structure has 8% and 24% less ADP and dissipates 6% and 10% less power than those of [Hu and Jong (2013)]. Note that the structure of [Hu and Jong (2013)] requires 7*N* extra memory words in the input buffer than those required by the proposed structure due to overlapped input data-blocks.

Table 5.5: Synthesis results of the proposed structure using radix-8 Booth multiplier and the existing structures for 9/7 wavelet filters for image size (256 × 256)

Designs	DWT level	Block Size	MCP (ns)	Total area (μm^2)	ADP ($\mu m^2 \times ms$)	Power (mW)
Hu and Jong [(2013)]	2	16	1.76	307727	2.23	12.1
	3	64	1.81	766000	1.42	29.9
Proposed structure	2	(4×4)	1.69	296511	2.05	11.4
	3	(8×8)	1.72	615964	1.08	27.1

5.5 CONCLUSION

In this chapter, a constant post-truncated radix-8 Booth multiplier is presented. Few redundant logic operations are created within the adder unit when n lower-order bits of $2n$-bit multiplier output are truncated. An optimized adder unit design is presented 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 radix-8 Booth multiplier design offers nearly 20.7% less ADP and consumes 18.3% less power over the existing radix-8 design optimized by Synopsys Design Compiler for post-truncation. It is observed that 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. It is shown that the proposed 12-bit fixed-width post-truncated radix-8 generic multiplier design involves 24.7% less ADP than the existing post-truncated radix-4 multiplier designs when both the multiplier designs are configured for constant multiplication. Block lifting 2-D DWT structure involves several multipliers. But, the interesting fact is many of these multipliers share a common input operand. To take advantage of this feature, the proposed radix-8 generic constant multiplier is considered instead of radix-4 Booth multiplier for the block 2-D DWT structure. The block-based lifting 2-D DWT structure proposed in Chapter 3 is synthesized using the proposed radix-8 generic-constant fixed-width multiplier to demonstrate the effectiveness of proposed scheme. ASIC synthesis result shows that the 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. Compared with the existing block lifting 2-D DWT structure of [Hu and Jong (2013)], the proposed block lifting 2-D DWT structure offers a saving in 24% ADP and 10% power for J=3 and block size 64, and higher saving for higher block sizes. However, the block 2-D DWT structure of higher block sizes involves a large number of multiplier which consumes a major part of the arithmetic core area and power. A multiplier less designs may reduce the complexity of the arithmetic unit. A solution to this problem is discussed in Chapter 6.