CHAPTER -4

EFFICIENT RADIX-8 BOOTH MULTIPLIER DESIGN

4.1 INTRODUCTION

Multiplication is the most common arithmetic operation performed in digital signal processing (DSP) algorithms. Typical DSP algorithms such as sinusoidal transforms, fast Fourier transform (FFT), discrete wavelet transform, and finite impulse response (FIR) filters and adaptive filters are more often implemented in dedicated VLSI systems for real-time and low-power applications. Multiplier is the most resource consuming component in the arithmetic unit and contribute significantly to the chip-area and critical-path. Both signed and unsigned multiplications are encountered in DSP algorithms. Unsigned multiplication is performed in straight forward manner like decimal multiplication which involves only shift-add operations. But, signed multiplication involves addition as well subtraction operation apart from shifting operation. Booth (1951) has proposed an algorithm which involves both +ve and –ve partial product. The +ve and –ve partial products involve addition and subtraction to calculate the multiplication result. Although subtraction is performed similar to addition operation but subtraction operation involves sign extension in 2's complement arithmetic which increases the complexity of the adder unit. Baugh and Woolly (1973) have proposed an algorithm for sign multiplication without using sign extension. However, Baugh and Woolly algorithm generates partial-product rows for each multiplier bits and the partial-product array involves '*w*' rows, where *w* is the bit-width of multiplier operand. The Booth algorithm is presented in modified where different encoding schemes have been proposed to reduce the size of the partial product array of sign multiplication. Amongst these, radix-4 Booth algorithm is popularly used for implementation of sign multiplication in hardware design. There are other variants of Booth algorithms dealing with higher radix have been proposed in the literature to reduce the partial products.

The higher radix Booth multiplication algorithms reduces the number of partial product rows of the partial product array (PPA) which helps to reduce the adder unit complexity, but higher radix Booth encoding increases the complexity of partial product generator (PPG) and partial product

selector (PPS). Consequently, the area-delay efficiency of higher radix Booth multiplier designs appears not better than the radix-4 Booth multiplier design. Therefore, radix-4 Booth multiplication algorithm mostly preferred to design multipliers for various DSP algorithms. Few attempts have been made to reduce the complexity of radix-8 Booth multiplier design [Besli and Deshmukh (2002), Galal *et al*. (2013)]. The radix-8 multiplier design of [Galal *et al*. (2013)] 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 w-th 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. The extra partial product row can be avoided with a regular PPA for radix-8 Booth multiplication. However, re-sizing the PPA requires extra combinational logic resources. Therefore, the adder unit of radix-8 Booth multiplier needs to be designed carefully to make the multiplier design more hardware efficient.

The selection of radix-4 algorithm is the obvious choice to find an efficient structure multiplier, since radix-4 Booth algorithm offers an efficient hardware design over radix-8 or higher radix Booth algorithm [Galal *et al*. (2013)]. However, some interesting result is found in this work. Radix-8 Booth algorithm offers a better hardware design over the radix-4 Booth algorithm when w is an integer multiple of 3.This is an important feature of radix-8 Booth multiplier. In this Chapter, we present an elaborate discussion on the radix-4 and radix-8. Necessary Boolean logic is developed to produce a regular PPA and optimized logic design for PPS of design radix-8 Booth multiplication.

The remaining sections of this Chapter are organized as follows: Review of radix-4 Booth algorithm is discussed in Section 4.2 Mathematical formulation of radix-8 Booth multiplication is presented in Section 4.3. The proposed radix-8 Booth multiplier design is presented in Section 4.4. Hardware and time complexities are discussed in Section 4.5. Conclusion is presented in Section 4.6.

4.2. REVIEW OF RADIX-4 BOOTH ALGORITHM

A brief review of radix-4 Booth algorithm is presented in this section for clarity in presentation. Consider two *n*-bit signed numbers *A* and *B* are multiplied using Booth algorithm, where *A* is multiplicand and *B* is multiplier. *A* and *B* are represented in 2's complement number system as

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \qquad 4.1$$

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \qquad 4.2$$

where, a_{n-1} and b_{n-1} are sign bits.

Booth algorithm decomposes an *n* bit multiplier into ($w=\lceil n/k \rceil$) groups of *k*-bits each using radix-2^k encoding (for radix-4 k=2). Each group of 2 bits and one overlapping bit forms one 3-bit digit, where the i^{th} digit $d_i = \{ b_{2i+1}\, b_{2i}\, b_{2i-1} \}$, b_{2i-1} is the overlapping bit belongs to (i-1)-th digit d_{i-1} and it is zero for i =0. Decimal value of d_i is obtain using the relation

$$d_i = -2b_{2i+1} + b_{2i} + b_{2i-1} \qquad 4.3$$

Substituting (4.3) in (4.2) we have

$$B = \sum_{i=0}^{w-1} d_i 2^{2i} \qquad 4.4$$

Substituting (4.3) and (4.4) in the product of *A* and *B*, we have

$$Z = A\sum_{i=0}^{w-1} (-2b_{2i+1} + b_{2i} + b_{2i-1})2^{2i} \qquad 4.5$$

Rewriting (4.5) as

$$Z = A\sum_{i=0}^{w-1} F(d_i)2^{2i} \qquad 4.6$$

The function $F(.)$ represents the selection operation and the bit-vector $\{d_i\}$ represents the control bits used for selection of partial product values from a set. Radix-4 Booth encoding scheme is given in Table 4.1. The block diagram of radix-4 Booth multiplication algorithm of (4.6) is shown in Figure 4.1. The partial product generation (PPG) unit produces 8 partial product values $\{0, A, A, 2A, -2A, -A, -A, 0\}$ for the partial product selection (PPS) unit. The PPS unit consists of ($w=\lceil n/2 \rceil$) number of 8:1 multiplexors (MUXes), where n is multiplier input operand bit-width. Each of these 'w' MUXes receives the set of 8 partial product values from the PPG and selects one partial product values according to the Booth encoded digit (d_i) which is used for selection logic. The i-th MUX (for $0 \leq i \leq w$-1) of the PPS unit receives the bits $\{d_i=b_{2i+1}, b_{2i}, b_{2i-1}\}$ at the selection input to select the partial-product value from the set of 8 values available at its input. The w MUXes of the PPS unit select w different partial product values from the partial product set using w different bit-vectors $\{d_i=b_{2i+1},b_{2i},b_{2i-1}\}$. Therefore, the PPS unit produces w partial-product values corresponding to w Booth encoded digits of operand B from w MUXes in parallel. Partial product values generated by PPS unit are sent to the adder-unit (AU) for accumulation to produce the multiplier result (Z).

Table 4.1: Radix-4 Booth Encoding

b_{2i+1}	b_{2i}	b_{2i-1}	Partial Product values
0	0	0	0
0	0	1	$+A$
0	1	0	$+A$
0	1	1	$2A$
1	0	0	$-2A$
1	0	1	$-A$
1	1	0	$-A$
1	1	1	0

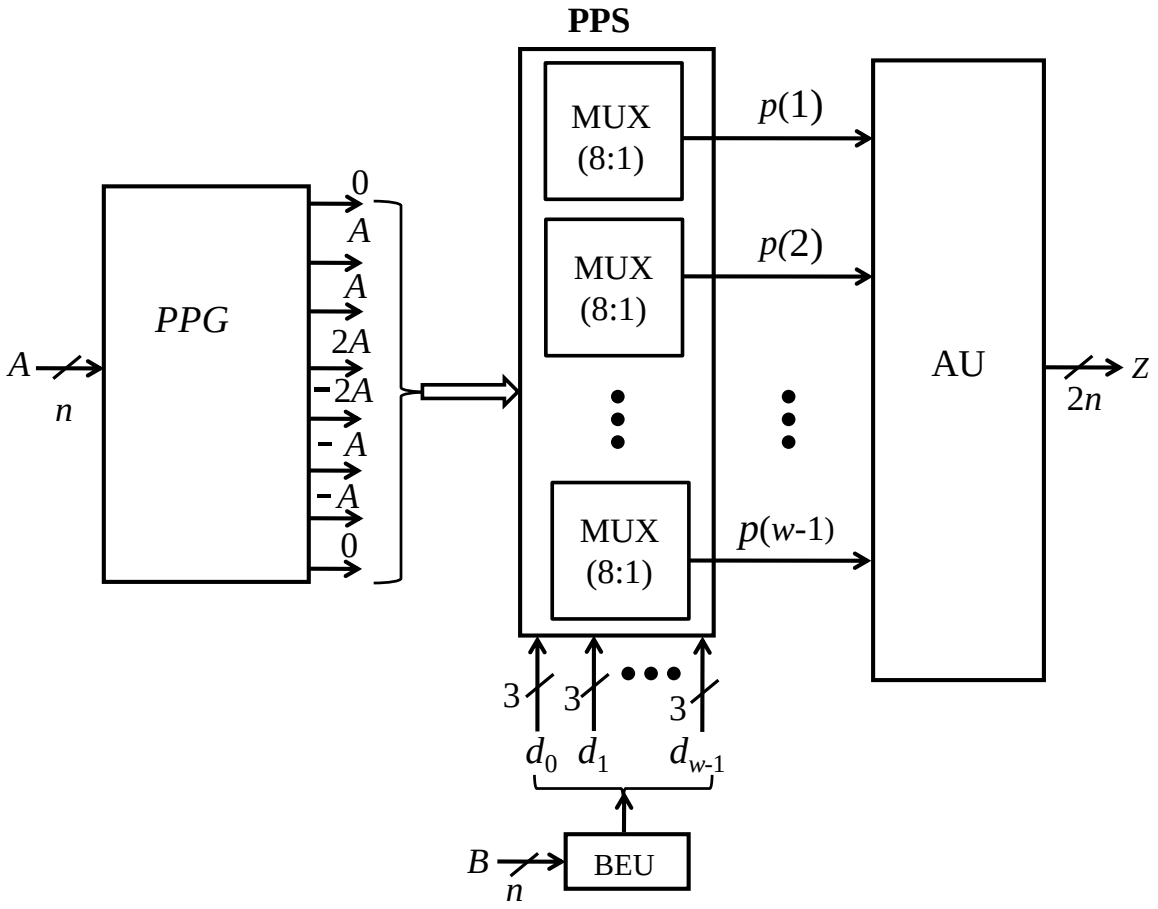

Figure 4.1: Block diagram of radix-4 Booth multiplication, where $d_i = \{b_{2i+1}, b_{2i}, b_{2i-1}\}$.

4.2.1 MODIFIED RADIX-4 BOOTH MULTIPLICATION ALGORITHM

As shown in Table 4.1, partial-product values for $b_{2i+1} = 1$ are anti-symmetric of partial-product values corresponding to $b_{2i+1} = 0$. Negative partial products $\{-A, -2A\}$ can be obtained from positive partial product values $\{A, 2A\}$ by using a sign control. This type of encoding is referred to *anti-symmetric product coding* (APC). Using APC, the required partial-product values of radix-4 Booth encoded multiplication are obtained from a set of 4 partial-product values $\{0, A, A, 2A\}$. The selection function of (4.6) is modified using APC and expressed as

$$Z = \sum_{i=0}^{w-1} [(b_{2i+1} \oplus (F(a_i))]2^{2i} \qquad 4.7$$

where, $a_i = (b_{2i}, b_{2i-1}), F(a_i) = \{0, A, A, 2A\}$

A block-based structure of radix-4 Booth multiplication using APC of (4.7) is shown in Figure 4.2. The block diagram of Figure 4.2 is identical with the block diagram of Figure 4.1 except that the PPG unit and PPS unit have half the size in case of APC-based radix-4 Booth multiplication than the direct radix-4 Booth algorithm. However, the APC-based structure requires an extra logic unit to introduce the sign inversion by 1's complement. The 1's complement logic unit is comprised of w number of word-level XOR cells. The sign bit (b_{2i+1}, for $0 \le i \le w$-1) of each Booth-encoded digit d_i is added to with the corresponding partial-product value obtained from 1's complement unit in the AU to obtain the 2's complement representation of the negative partial product value. The overhead logic complexity of APC algorithm (in terms of 1's complement logic and extra bit-adders required by the AU) is marginal compared to the amount of combinational logic saved from the PPG unit and PPS unit.

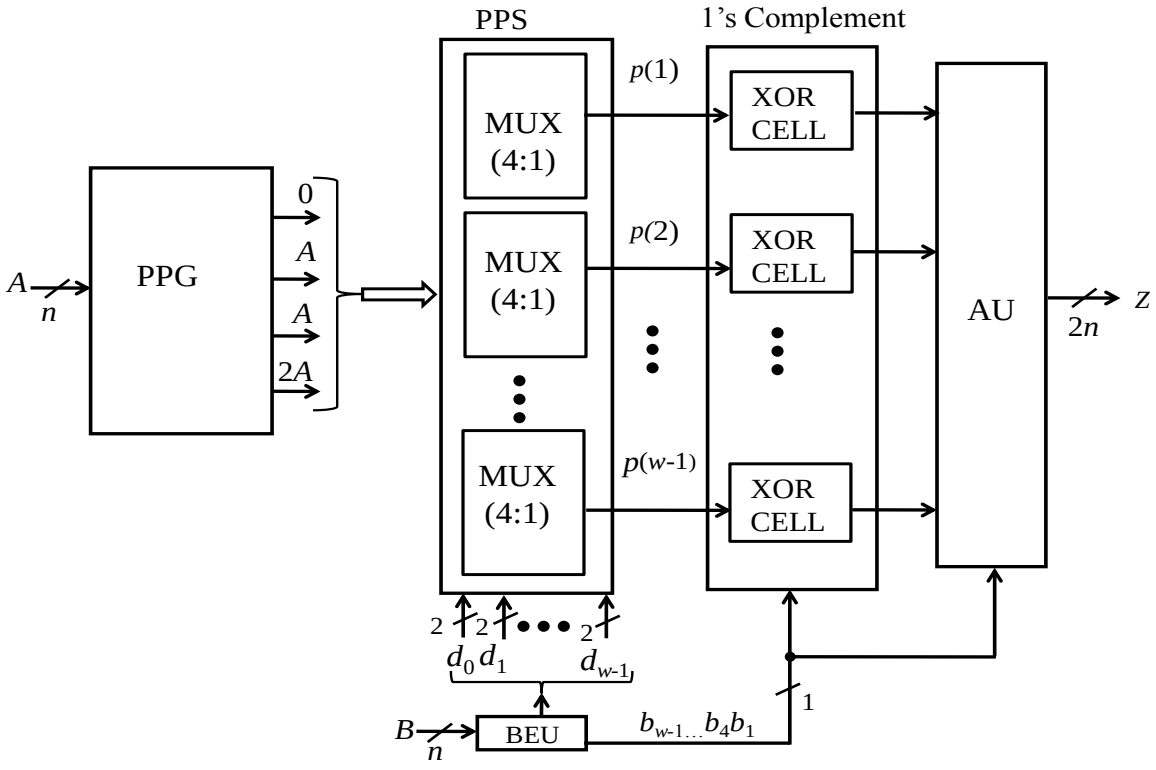

Figure 4.2: Block diagram of for radix-4 Booth multiplication based on anti-symmetric product coding.

One can observe form Table 4.1 that in the partial-product set {0, *A*, *A*, 2*A*}, two values are identical (symmetric). Using symmetric, the four product values {0, *A*, *A*, 2*A*} can be produced from two values {*A*, 2*A*}. The *zero* partial product value can be obtained using some reset circuit. The selection function of (4.7) is further modified and expressed as:

$$Z = \sum_{i=0}^{w-1} [s_{3i+2}(s_{3i+1} \oplus (F(\mathbf{v}_i)) + s_{3i+1}]2^{2i} \qquad 4.8$$

where $v_i = (s_{3i})$, $F(\mathbf{v}_i) = \{A, 2A\}$

The encoding signals $(s_{3i}, s_{3i+1}, s_{3i+2,})$ are derived from the digit $d_i = (b_{2i+1}b_{2i}b_{2i-1})$ using the Boolean expressions

$$s_{3i} = b_{2i} \odot b_{2i-1} \qquad 4.9(a)$$

$$s_{3i+1} = b_{2i+1} \qquad 4.9(b)$$

$$s_{3i+2} = (\bar{b}_{2i+1}b_{2i-1} + b_{2i}\bar{b}_{2i+1} + b_{2i+1}\bar{b}_{2i}) \qquad 4.9(c)$$

Table 4.2: Control signal for symmetric and anti-symmetric product coding for radix-4 Booth multiplication

s_{3i+2}	s_{3i+1}	s_{3i}	PP
0	×	×	0
1	0	0	+*A*
1	0	0	+*A*
1	0	1	+2*A*
1	1	1	-2*A*
1	1	0	-*A*
1	1	0	-*A*
0	1	×	0

Table 4.2 lists 8 possible partial product values corresponding to 3-bit control signals $\{s_{3i+2}, s_{3i+1}, s_{3i}\}$. The control bits **(**$s_{3i}$) fed to the control input of 2:1 MUX to select one on partial product value from set $\{A, 2A\}$. The control bit (s_{3i+1}) is used for introducing sign inversion to the partial-product value as required by APC and the control bit (s_{3i+2}) is used to reset the partial product value for obtaining zero partial-product value. The block diagram of SAPC-based radix-4 Booth multiplication algorithm of using (4.8) and (4.9) is shown in Figure 4.3. As shown in this figure the block diagram of SAPC-based multiplier involves an extra reset circuit than the block diagram of APC-based multiplier (shown in Figure 4.2). The reset logic unit is comprised of w number of word-level AND cells. Besides, block diagram of SAPC-based multiplication involves a separate encoder unit (EU) to generate the require control signals $\{s_{i=}s_{3i+2}, s_{3i+1}, s_{3i}\}$ from digit $\{d_i=b_{2i+1}, b_{2i}, b_{2i-1}\}$. However, the size of PPS unit of SAPC-based multiplication is almost half the size of the PPS unit of APC-based multiplication and one-fourth of the size of the PPS unit of multiplication using direct radix-4 Booth algorithm. The overhead complexity due to reset and sign control circuit is marginal compared to the amount of combinational logic saved from PPS unit using SAPC algorithm.

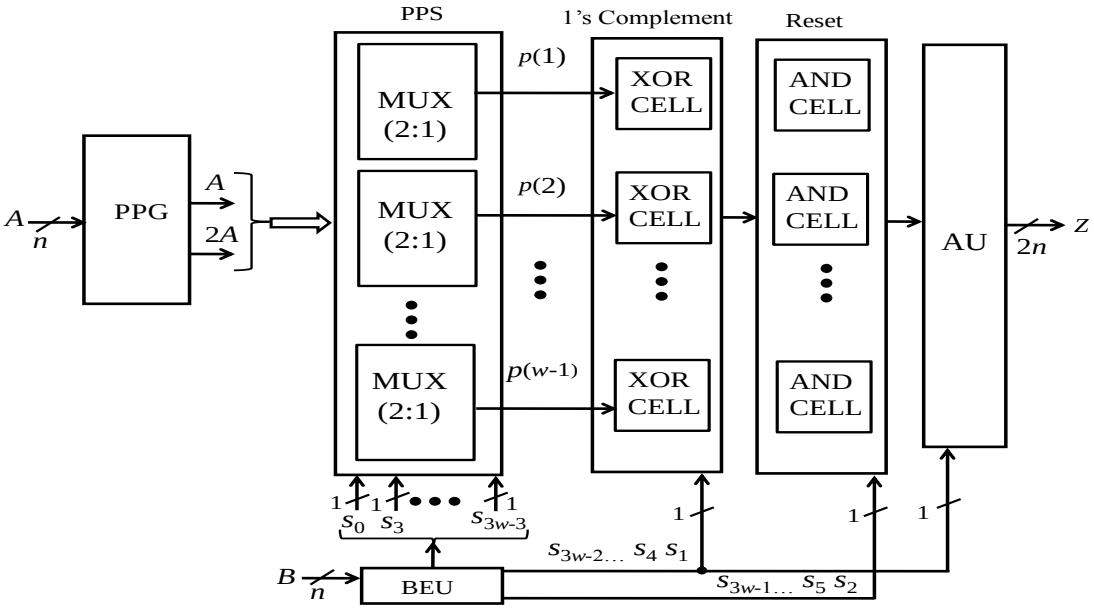

Figure 4.3: Block diagram Radix-4 Booth multiplication based on symmetric and anti-symmetric product coding.

4.3 RADIX-8 BOOTH MULTIPLICATION ALOGRITHM

For radix-8 Booth encoding, k=3 is assumed for grouping of multiplier bits. Each group of 3 bits and one overlapping bit forms one digit of size 4 bits, where the i^{th} digit $d_i = \{b_{3i+2}\, b_{3i+1}\, b_{3i}\, b_{3i-1}\}$, b_{3i-1} is the overlapping bit belongs to (i-1)-th digit d_{i-1} and it is zero for i =0. Decimal value of d_i for radix-8 encoding is obtain using the relation

$$d_i = -4b_{3i+2} + 2b_{3i+1} + b_{3i} + b_{3i-1} \qquad 4.10$$

Substituting (4.10) in (4.2) we have

$$B = \sum_{i=0}^{w-1} d_i 2^{3i} \qquad 4.11$$

Substituting (4.10) and (4.11) in the product of A and B, we have

$$Z = A\sum_{i=0}^{w-1} (-4b_{3i+2} + 2b_{3i+1} + b_{3i} + b_{3i-1})2^{3i} \qquad 4.12$$

Rewriting (4.12) as

$$Z = A\sum_{i=0}^{w-1} F(d_i)2^{3i} \qquad 4.13$$

The selection function F(.) selects one partial product value from a set of 16 partial-product values {0, A, A, 2A, 2A, 3A, 3A, 4A, -4A, -3A, -3A, - 2A, -2A, -A,-A, 0} according to the Booth encoding scheme given in Table 4.3. The block diagram of radix-8 Booth multiplication algorithm of (4.13) is shown in Figure 4.4. The PPS unit consists of (w=$\lceil n/3 \rceil$) number of 16:1 multiplexors (MUXes).

Table 4.3: Radix-8 Booth Encoding

b_{3i+2}	b_{3i+1}	b_{3i}	b_{3i-1}	PP
0	0	0	0	0
0	0	0	1	+A
0	0	1	0	+A
0	0	1	1	+2A
0	1	0	0	+2A
0	1	0	1	+3A
0	1	1	0	+3A
0	1	1	1	+4A
1	0	0	0	-4A
1	0	0	1	-3A
1	0	1	0	-3A
1	0	1	1	-2A
1	1	0	0	-2A
1	1	0	1	-A
1	1	1	0	-A
1	1	1	1	0

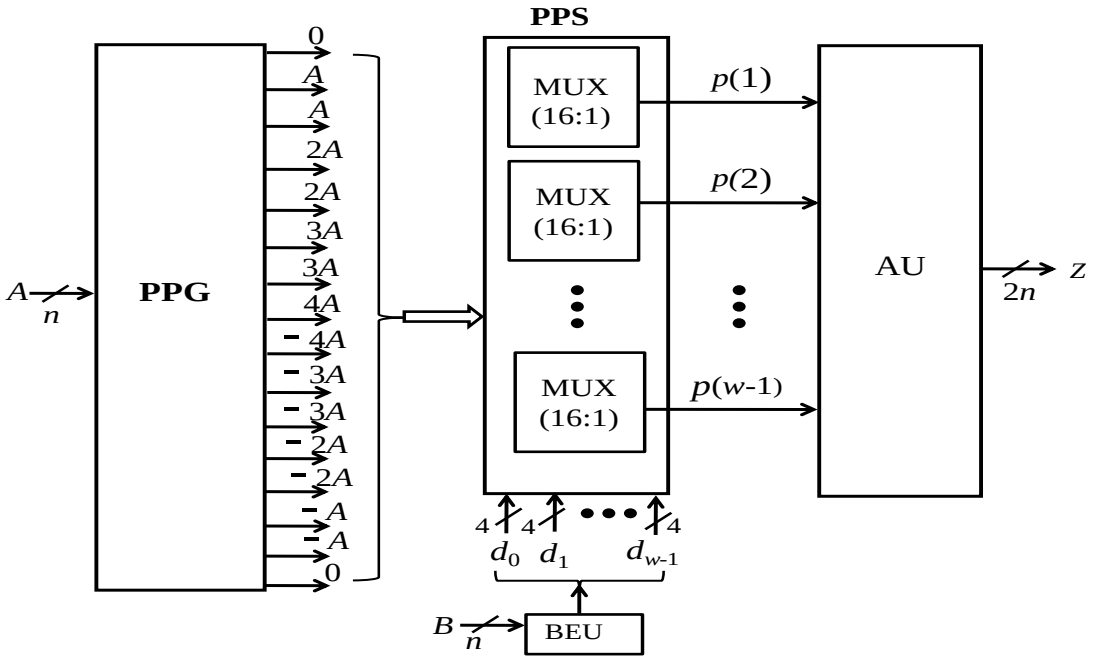

Figure.4.4: Block diagram of radix-8 Booth multiplication, where $d_i = \{b_{3i+2}, b_{3i+1}, b_{3i}, b_{3i-1}\}$.

4.3.1 MODIFIED RADIX-8 BOOTH MULTIPLICATION ALGORITHM

Radix-8 Booth algorithm is also modified. The required 16 partial-product values of radix-8 Booth encoded multiplication are obtained from positive partial product values {0, *A*, *A*, 2*A*, 2*A*, 3*A*, 3*A*, 4*A*,} and the negative partial product values {-*A*, -*A*, -2*A*, -2*A*, -3*A*, -3*A*, -4*A*} are obtained from the positive partial product set using a sign control signal (b_{3i+2}). The selection function of (4.13) is modified using APC and expressed as

$$Z = \sum_{i=0}^{w-1} [(b_{4i+2} \oplus (F(a_i))]2^{3i} \qquad 4.14$$

where, $a_i = (b_{3i+1}, b_{3i}, b_{3i-1})$, $F(a_i) = \{0, A, A, 2A, 2A, 3A, 3A, 4A\}$

Using symmetric coding along with APC four partial products values {*A*, 2*A*, 3*A*, 4*A*} only needed to be generate the required 16 partial product values of radix-8 Booth multiplication. The selection operation of (4.14) is further modified for symmetric coding and expressed as:

$$Z = \sum_{i=0}^{w-1} [s_{4i+3}(s_{4i+2} \oplus (F(\mathbf{v}_i)) + s_{4i+2}]2^{3i} \qquad 4.15$$

where $\mathbf{v}_i = (s_{4i+1}, s_{4i})$, $F(\mathbf{v}_i) = \{A, 2A, 3A, 4A\}$

Control signals $\{s_{4i}, s_{4i+1}, s_{4i+2,} s_{4i+3}\}$ are generated from the digit $d_i = (b_{3i+2} b_{3i+1} b_{3i} b_{3i-1})$ using the following Boolean expressions

$$s_{4i} = b_{3i} \oplus b_{3i\text{-}1} \qquad 4.16(a)$$

$$s_{4i+1} = \bar{b}_{3i+2} b_{3i+1} (b_{3i-1} + b_{3i}) + b_{3i+2} \bar{b}_{3i+1} (b_{3i} + \bar{b}_{3i-1}) \qquad 4.16(b)$$

$$s_{4i+2} = b_{3i+2} \qquad 4.16(c)$$

$$s_{4i+3} = \bar{b}_{3i+2} b_{3i-1} + b_{3i} \bar{b}_{3i-1} + b_{3i+1} \bar{b}_{3i} + b_{3i+2} \bar{b}_{3i+1} \qquad 4.16(d)$$

Table 4.4: Control signals of direct Booth encoding and SAPC encoding

b_{4i+3}	b_{4i+2}	b_{4i+1}	b_{4i}	s_{4i+3}	s_{4i+2}	s_{4i+1}	s_{4i}
0	0	0	0	0	0	×	×
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	0
0	0	1	1	1	0	0	1
0	1	0	0	1	0	0	1
0	1	0	1	1	0	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	1	0
1	0	1	1	1	1	0	1
1	1	0	0	1	1	0	1
1	1	0	1	1	1	0	0
1	1	1	0	1	1	0	0
1	1	1	1	0	1	×	×

Table 4.5: SAPC encoding

s_{4i+3}	s_{4i+2}	s_{4i+1}	s_{4i}	PP
0	0	×	×	0
1	0	0	0	+A
1	0	0	1	+2A
1	0	1	0	+3A
1	0	1	1	+4A
1	1	1	1	-4A
1	1	1	0	-3A
1	1	0	1	-2A
1	1	0	0	-A
0	1	×	×	0

Table 4.4 lists SAPC control signals $\{s_{4i+3,}s_{4i+2,}s_{4i+1}, s_{4i}\}$ corresponding to direct Booth encoding $d_i = (b_{3i+2}b_{3i+1}b_{3i}b_{3i-1})$. Out of four control signals, pair control signals $\mathbf{v}_i=\{s_{4i+1}, s_{4i}\}$ are used by the PPS unit to select the partial-product value where control signals (s_{4i+2}) and (s_{4i+3}) are for sign control and reset the partial product value. The SAPC encoding scheme is given in Table 4.5. The block diagram of SAPC-based radix-8 Booth multiplication is shown in Figure.4.5,

which is identical to the block diagram of SAPC-based radix-4 Booth multiplication except the size of PPS unit and adder unit.

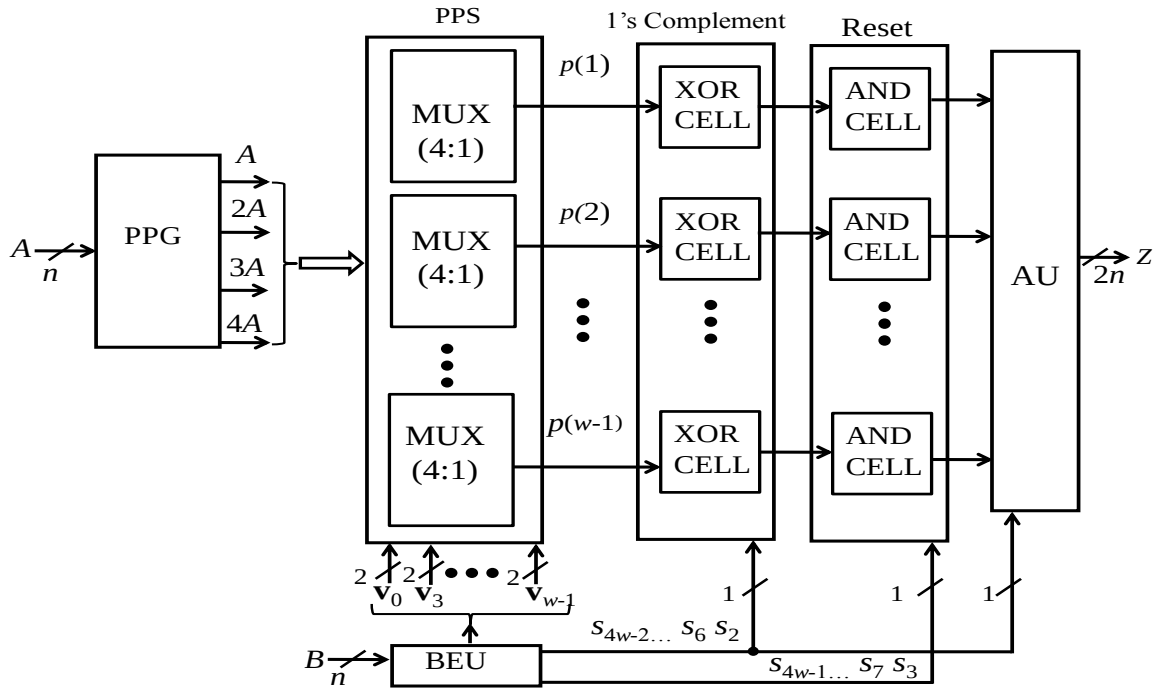

Figure 4.5: Radix-8 generic Booth multiplier structure based on symmetric and anti-symmetric product coding. $\mathbf{v}_i = \{s_{4i+1}, s_{4i}\}$ for $0 \le i \le w\text{-}1$, $w = n/3$.

4.3.2 COMPLEXITY ANALYSIS

Combinational logic complexity and time complexity of radix-4 and radix-8 multiplication algorithm is discussed here to find the scope to optimize the hardware design of radix-8 Booth multiplier. Both the direct and SAPC Booth encoding based radix-4 and radix-8 multiplication algorithms are included for discussion. The estimated values are listed in Table 4.5. The combinational logic complexity is listed in terms of 2:1 MUX (word-level), 2-input XOR-gate (word), 2-input (AND-gate) and ripple carry adder (RCA). The encoder unit complexity is excluded from the discussion since it involves few gates (bit-level) only. As shown in Table 4.5, the radix-4 multiplier based on SAPC involves 7 times less MUXes than those required by the radix-4 Booth multiplier based on direct method and it has marginally higher critical-path delay (CPD). But, the radix-4 multiplier based on direct method involves (n/2) more XOR-gates

(word) and AND-gates (word). Similarly SAPC based radix-8 Booth multiplier involves 4 less adders, 5 times less MUXes against (n/3) more XOR-gates (word) and AND-gates (word) and involves marginally less CPD compared with radix-8 multiplier based on direct Booth algorithm.

Compared with the SAPC-based radix-4 Booth multiplier, the SAPC-based radix-8 multiplier involves (n/6) number of less adders, (n/6) number of less XOR-gates (word) and AND-gates (word), nearly (n/2) number of more MUXes and it has nearly $\{T_A+T_m-T_x-(n/6)T_f\}$ more CPD, where T_A, T_m and T_x are, respectively, adder delay, 2:1 MUX delay and 2-input XOR-gate delay. Since, T_A is significantly higher than T_x and T_f, the CPD of radix-8 multiplier is higher than that of radix-4 multiplier. Hardware complexity and CPD of radix-8 and radix-4 multiplier for different bit-widths are estimated using the generalized expressions of Table 4.6. The estimated values are listed in Table 4.7 for comparison. As shown in Table 4.7 the radix-8 multiplier involves on average 30% less adder, 30% less XOR-gate (word) and 30% less AND-gate (word) than those of radix-4 multiplier, but it involves nearly 50% more MUXes than radix-4 multiplier on average for different bit-widths. It has nearly $(2T_f - T_m - T_x)$ more CPD than radix-4 multiplier for different bit-widths. Since the difference in CPD of radix-4 and radix-8 multipliers is almost independent of bit-widths, the area-delay advantage of radix-8 generic multiplier over the radix-4 multiplier is largely depends on the change in MUX complexity.

Table 4.6: General comparison of hardware and time complexities of radix-4 and radix-8 Booth multiplier

Algorithm	Adder	MUX (2:1)	XOR-gate (word)	AND-gate (word)	CPD
Direct radix-4 Booth multiplication	(n/2)-1	7(n/2)	--	--	$3T_m+T_A+[(n/2)-2]T_f$
SAPC based radix-4 Booth multiplication	(n/2)	n/2	n/2	n/2	$T_m+2T_x+T_n+T_A+[(n/2)-2]T_f$
Direct radix-8 Booth multiplication	(n/3)+4	15(n/3)	--	--	$4T_m+3T_A+[(n/3)-1]T_f$
SAPC based radix-8 Booth multiplication	(n/3)	3(n/3)	3(n/3)	3(n/3)	$2T_m+T_x+2T_n+2T_A+[(n/3)-1]T_f$

Legend: GC: gate count, T_m: 2:1 MUX delay, T_x: 2-input XOR-gate delay, T_n: 2-input AND-gate delay, T_A: adder delay, T_f: full-adder delay

Table 4.7: Comparison of hardware and time complexities of Booth multiplier for different word-length sizes based on SAPC

Multiplier	Bit-width	Adder	MUX (2:1)	XOR-gate (word)	AND-gate (word)	CPD
Radix-4	12	6	6	6	6	$T_m+2T_x+T_n+2T_A+4T_f$
	16	8	8	8	8	$T_m+2T_x+T_n+2T_A+6T_f$
	24	12	12	12	12	$T_m+2T_x+T_n+2T_A+10T_f$
Radix-8	12	4	12	4	4	$2T_m+2T_x+T_n+3T_A+3T_f$
	16	6	18	6	6	$2T_m+2T_x+T_n+3T_A+5T_f$
	24	8	24	8	8	$2T_m+2T_x+T_n+3T_A+7T_f$

The complexity analysis reveals that PPS unit is a significant component of radix-8 Booth multiplier. Although the radix-8 Booth algorithm offers a saving in adder unit complexity but the PPS unit complexity increases much higher amount than the saving. This is one of the causes which make radix-8 Booth multiplier design less efficient than the radix-4 Booth multiplier. Therefore, optimizing the logic design of PPS unit could improve the efficiency of radix-8 multiplier design. Besides, the SAPC based Booth encoding generates an irregular partial product array due to addition of sign control bit to represent the partial product values in 2's complement format. The irregular partial product array increases the adder complexity as well as critical-path delay. Generating a regular partial product array for SAPC-based radix-8 Booth multiplication is another issue which is currently missing in the literature. These issues are addressed in following Section.

4.4 PROPOSED RADIX-8 BOOTH MULTIPLIER DESIGN

4.4.1 MODIFIED 4:1 MULTIPLEXER DESIGN

The 4:1 MUX structure using 2:1 MUX is shown in Figure 4.6. The Boolean expression of 4:1 MUX is represents in (4.17) using decoded control signals. The modified Boolean expression of (4.17) is given as (4.19). The modified logic design of 4:1 MUX is shown in Figure 4.7. The logic complexity of conventional type MUX and the modified design is given in Table 4.8. Note

that the decoder complexity is not excluded from the comparison since the decoder complexity is independent of the input bit-width of the multiplexor. As shown in Table 4.8, the modified logic design involves 2 less AND gate compared to conventional MUX design for bit level implementation. A word level 4:1 MUX based on modified logic design offers a saving in 2*n* gates, where *n* is the input word length. The saving would be higher compared to the bit-level 2:4 decoder complexity which is an overhead for the MUX design.

$$y = [\bar{t}_1(\bar{t}_0 i_0 + t_0 i_1) + t_1(\bar{t}_1 i_3 + t_1 i_4)] \quad 4.17$$

$$y = [(\bar{t}_0 \bar{t}_1 i_0 + t_0 \bar{t}_1 i_1) + (\bar{t}_0 t_1 i_2 + t_0 t_1 i_3)] \quad 4.18$$

$$y = [(ei_0 + fi_1) + (gi_3 + hi_4)] \quad 4.19$$

where (i_0, i_1, i_2, i_3) Input of MUX and (t_0, t_1) control signal.

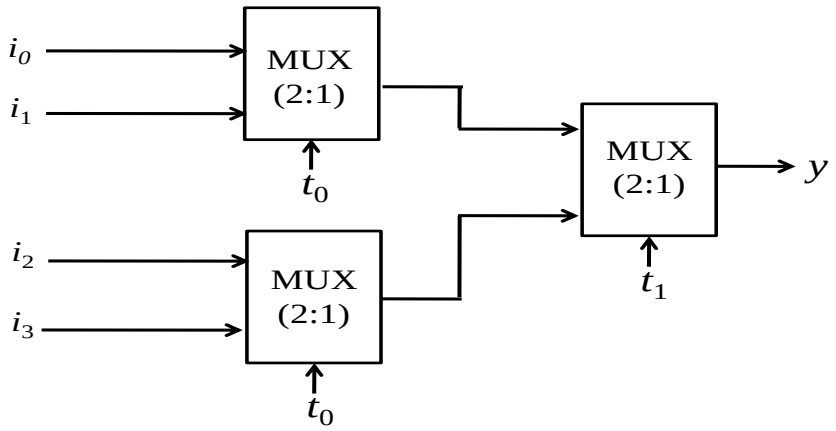

Figure 4.6: Block diagram of 4:1 MUX

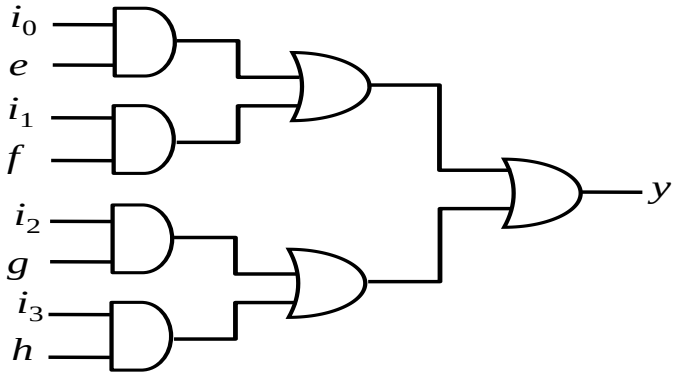

Figure 4.7: Modified logic structure of 4:1 MUX

Table 4.8: Comparison of logic complexity of MUX

Design	AND/OR gate	Critical Path delay
Conventional	9	$2(T_{AND}+T_{OR}+T_{NOT})$
Modified*	7	$T_{AND}+2T_{OR}+T_{deco.}$

*Decoder complexity is not included.
Legend: $T_{deco.:}$ Decoder delay.

The PPS unit of SAPC-based radix-8 Booth multiplier involves 4:1 MUXes (word-level). These word-level MUXes are implemented using the modified logic design of 4:1 MUX. Recently, [Galal *et al*. (2013)] have suggested a design for radix-8 Booth multiplied where the modified logic design of 4:1 MUX is used to implement the PPS unit and the decoder generating the selection control signals is embedded with the Booth encoder. However, the multiplier design of [Galal *et al*. (2013)] does not include the reset circuit in the PPS unit and not use a regular partial product array. These short comings are addressed in the proposed design.

4.4.2. REGULAR PARTIAL PRODUCT ARRAY

Partial product of Booth encoded multiplication is represented in 2's complement arithmetic. The PPS unit produces partial product values of size (n+2) bits corresponding to a particular Booth encoded digit $\{d_i\}$. These (n+2) partial product bits forms a row in the partial product row. For a ($n\times n$) multiplication, the PPS unit produces w partial product values corresponding to w Booth encoded digit and these w partial product values forms w rows in the partial product array. Each partial product row is left shifted by 3-bit positions with respect to its previous row. Due to 2's complement representation, 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 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. Figure 4.8 shows the partial product array of radix-8 Booth multiplication for n=12 where guard bits are used for sign extension [Kuang *et.al* (2009)]. The existing radix-8 Booth multipliers structures [Besli and Deshmukh (2013), Galal *et al*. (2013)] are based on the PPA shown in Figure 4.8. The control bit ($n_i = b_{3i+4}$) is added to each row of the PPA to convert 1's complement number into 2's complement number. Due to addition of n_i to the partial product row the size of

the PPA is increased by one extra row. The PPA accumulator requires an extra adder to accumulate the extra partial product row.

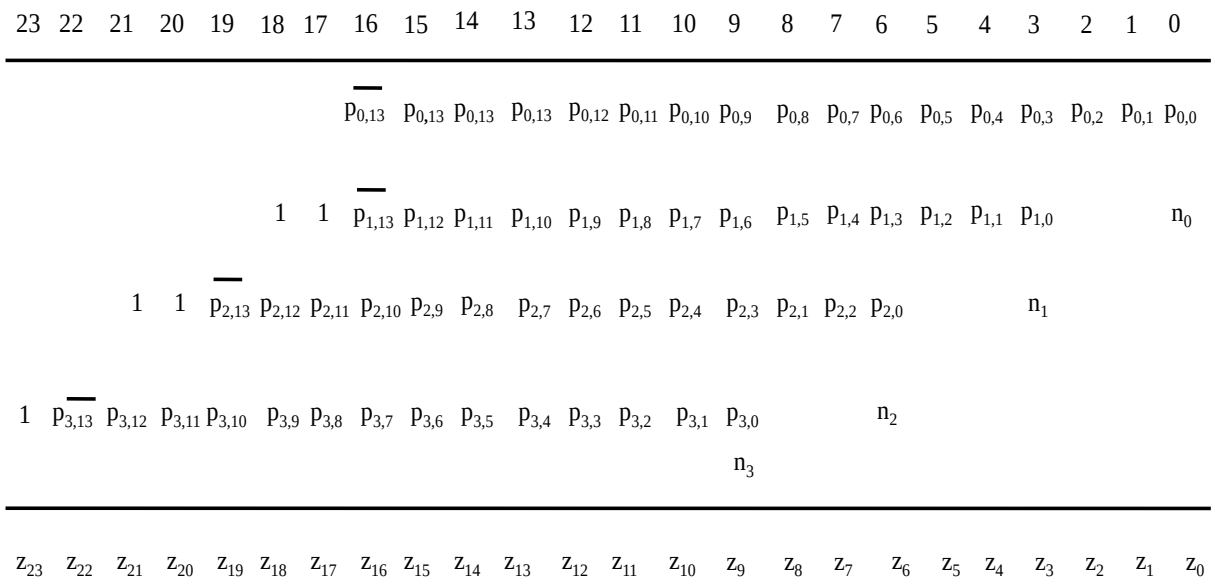

Figure 4.8: Partial product array of radix-8 Booth multiplier for 12 bit.

[Kuang *et.al* (2009)] have suggested a scheme to obtain a regular partial product array for Radix-4 Booth multiplication without increasing partial product rows. The scheme given in [Kuang *et.al* (2009)] is extended to find a regular partial product array for radix-8 Booth multiplication. The regular PPA of 12-bit radix-8 Booth multiplier is shown in Figure. 4.9. 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 is appended to the right of the (i+2)-th row LSB of PPA. In case of w-th row of PPA, four LSBs ($p_{w\text{-}1,3}$, $p_{w\text{-}1,2,}$ $p_{w\text{-}1,1}$, $p_{w\text{-}1,0}$) are modified into ($m_{w\text{-}1,3}$, $m_{w\text{-}1,2,}$ $m_{w\text{-}1,1}$, $m_{w\text{-}1,0}$) to absorb the control bit ($n_{w\text{-}1}$) and produces the carry bit ($r_{w\text{-}1}$ - not shown in Figure 4.9). The sign extension bits of first row of PPA are modified into (q_3, $q_{2,}$ q_1, q_0) after absorbing the carry bit ($r_{w\text{-}1}$). The modified bits ($m_{i,1}$, $m_{i,0}$), carry bit (r_i) are generated using the following logic expressions:

$$m_{i0} = p_{i0} \oplus n_i \qquad 4.20$$

$$m_{i1} = p_{i,1}\overline{p_{i,0}} + p_{i,1}\overline{n_i} + \overline{p_{i,1}}p_{i,0}n_i \qquad 4.21$$

$$r_i = p_{i,1}p_{i,0}n_i \qquad 0 \le i \le (n/3)\text{-}2 \qquad 4.22$$

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

$$m_{32} = p_{i,2}\overline{p}_{i,1} + p_{i,2}\overline{p}_{i,0} + p_{i,2}\overline{n}_3 + \overline{p}_{i,2}p_{i,1}p_{i,0}n_3 \qquad 4.23$$

$$m_{33} = p_{i,3}\overline{p}_{i,2} + p_{i,2}\overline{p}_{i,0} + p_{i,3}\overline{n}_3 + \overline{p}_{i,3}p_{i,2}p_{i,1}p_{i,0}n_3 \qquad 4.24$$

$$r_3 = p_{3,1}p_{3,2}p_{3,1}p_{3,0}n_3 \qquad 4.25$$

$$q_0 = s_0 \odot r_3 \qquad 4.26$$

$$q_3 = \overline{s_0}\,\overline{r_3} \qquad 4.27$$

$$q_2, q_1 = \overline{q}_3 \qquad 4.28$$

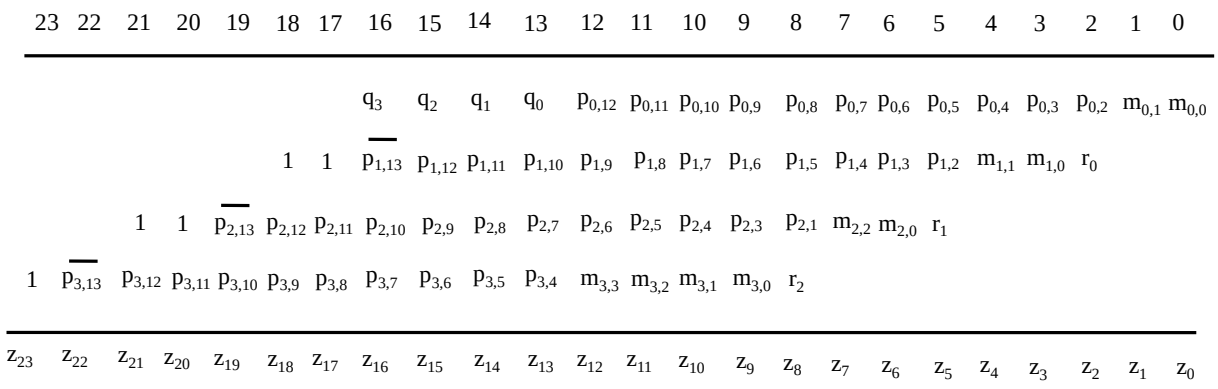

Figure 4.9: Regular partial product array of radix-8 Booth multiplier for 12 bit.

4.4.3. PROPOSED DESIGN

The block diagram of proposed radix-8 Booth multiplier design is shown in Figure 4.10 for 12-bit multiplication. The block diagram of proposed design is identical to the block diagram of straight forward radix-8 Booth multiplication (as shown in Figure 4.5) except that the proposed design uses a partial-product resize unit (PPRU) and the adder unit accumulate *w* partial product rows.

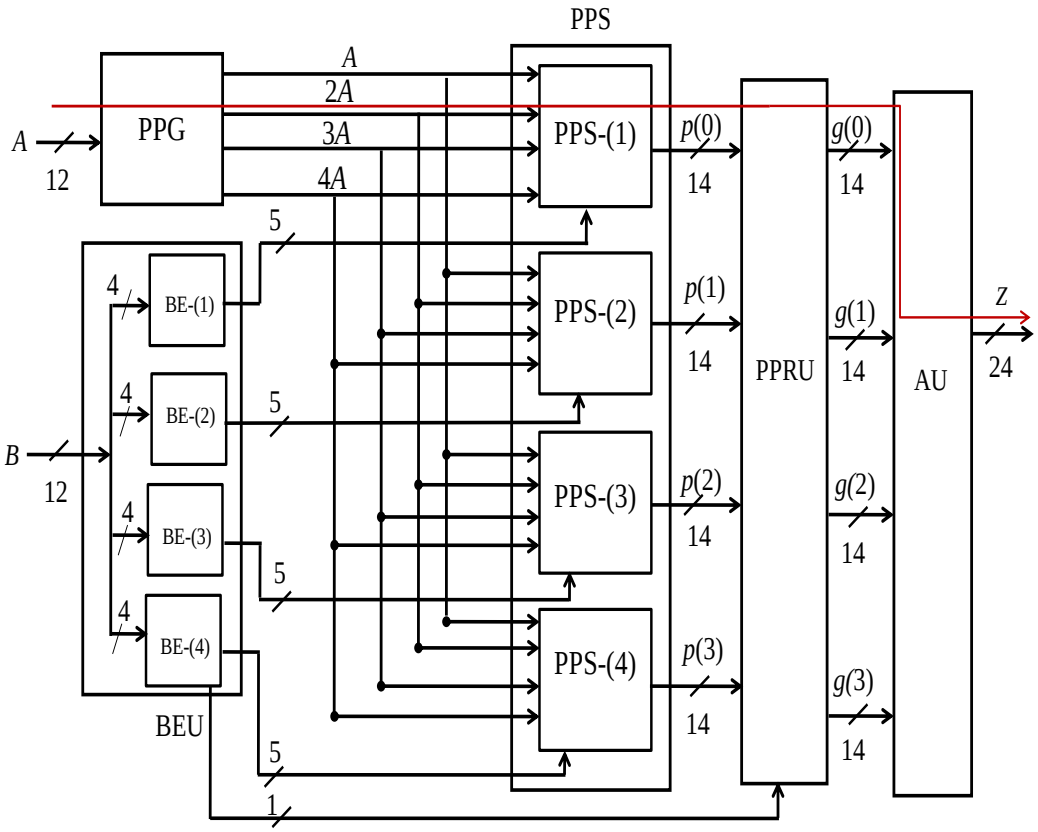

Figure 4.10: Block diagram of proposed radix-8 Booth multiplier structure for 12-bit. The red colored arrow shows the critical path.

Table 4.9: Radix-8 Booth Encoding using modified control functions

$b_{3i+2}b_{3i+1}b_{3i}b_{3i-1}$	e_i	f_i	g_i	h_i	n_i	PP
0000	0	0	0	0	0	0
0001	1	0	0	0	0	+A
0010	1	0	0	0	0	+A
0011	0	1	0	0	0	+2A
0100	0	1	0	0	0	+2A
0101	0	0	1	0	0	+3A
0110	0	0	1	0	0	+3A
0111	0	0	0	1	0	+4A
1000	0	0	0	1	1	-4A
1001	0	0	1	0	1	-3A
1010	0	0	1	0	1	-3A
1011	0	1	0	0	1	-2A
1100	0	1	0	0	1	-2A
1101	1	0	0	0	1	-A
1110	1	0	0	0	1	-A
1111	0	0	0	0	0	0

4.4.3.1 BOOTH ENCODER (BE) DESIGN

The decoder function is embedded with the SAPC encoding in the proposed radix-8 Booth encoding scheme as shown in Table 4.9. As shown in Figure 4.3 the Booth encoder unit (BEU) is comprised of w=4 number of Booth encoders (BEs) corresponding to four Booth encoded digits. BE receives four bits of Booth encoder digit and generates five control signals. The (i+1)-th BE (for $0 \le i \le 3$) produces five control signals $\{e_i, f_i, g_i, h_i, n_i\}$ from the digit $d_i=\{b_{3i+2}\, b_{3i+1}\, b_{3i}\, b_{3i-1}\}$ using the logic expressions of (4.29)-(4.32). The structure of (i+1)-th BE is shown in Figure 4.11.

$$e_i = \overline{(b_{3i-1} \odot b_{3i}) + (\overline{b_{3i+1} \odot b_{3i+2}})} \quad 4.29$$

$$f_i = \overline{(\overline{b_{3i-1} \odot b_{3i}}) + (b_{3i} \odot b_{3i+1})} \quad 4.30$$

$$g_i = \overline{(b_{3i-1} \odot b_{3i}) + (b_{3i+1} \odot b_{3i+2})} \quad 4.31$$

$$d_i = \overline{(\overline{b_{3i-1} \odot b_{3i}}) + (\overline{b_{3i+1} \odot b_{3i+2}}) + (b_{3i+1} \odot b_{3i+2})} \quad 4.32$$

Where + and Θ denote the OR and XNOR operations, respectively.

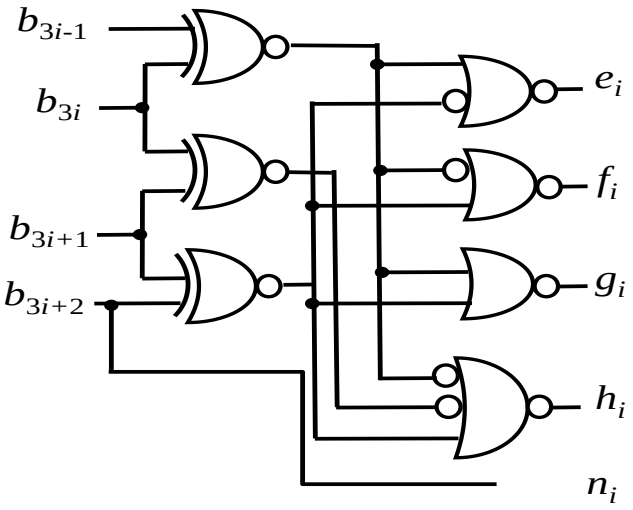

Figure 4.11: Internal structure of (i+1)-th of Booth encoder (BE)

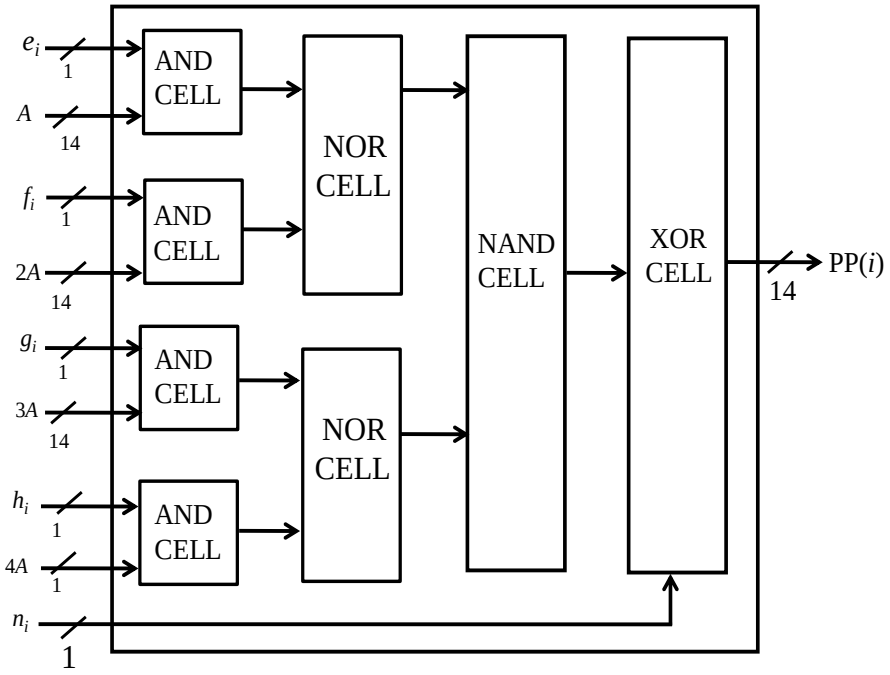

Figure 4.12: Modified structure of PPS unit

4.4.3.2 PARTIAL SELECTION UNIT (PPS) DESIGN

The partial product selection unit is comprised of (w=4) partial product selectors. The internal structure of (i+1)-th partial product selector is shown Figure 4.12 which is derived using the

Boolean expression of (4.19). The (i+1)-th partial-product selector receives five control {e_i, f_i, g_i, h_i, n_i} from (i+1)-th BE and selects one partial product from the set {4A, 3A, 2A, A}.

4.4.3.3 ADDER UNIT DESIGN

According to (4.20)-(4.28), partial product bits p_{ij} and sign control bits are required to obtain the modified bits of regular partial product array. The Boolean expressions of (4.20)-(4.28) are implemented outside the adder unit using a separate logic unit named partial product resize unit (PPRU). The $g(0)$, $g(1)$, $g(2)$ and $g(3)$ are represent modified partial product rows as shown in Figure 4.13. The regular partial product rows are shift-added in the adder unit (AU) to produce the 2n-product bits. There are many ways to implement the AU. However, Wallace tree adder is considered here to implement the AU**.** The bit flow diagram AU is shown in Figure 4.13.

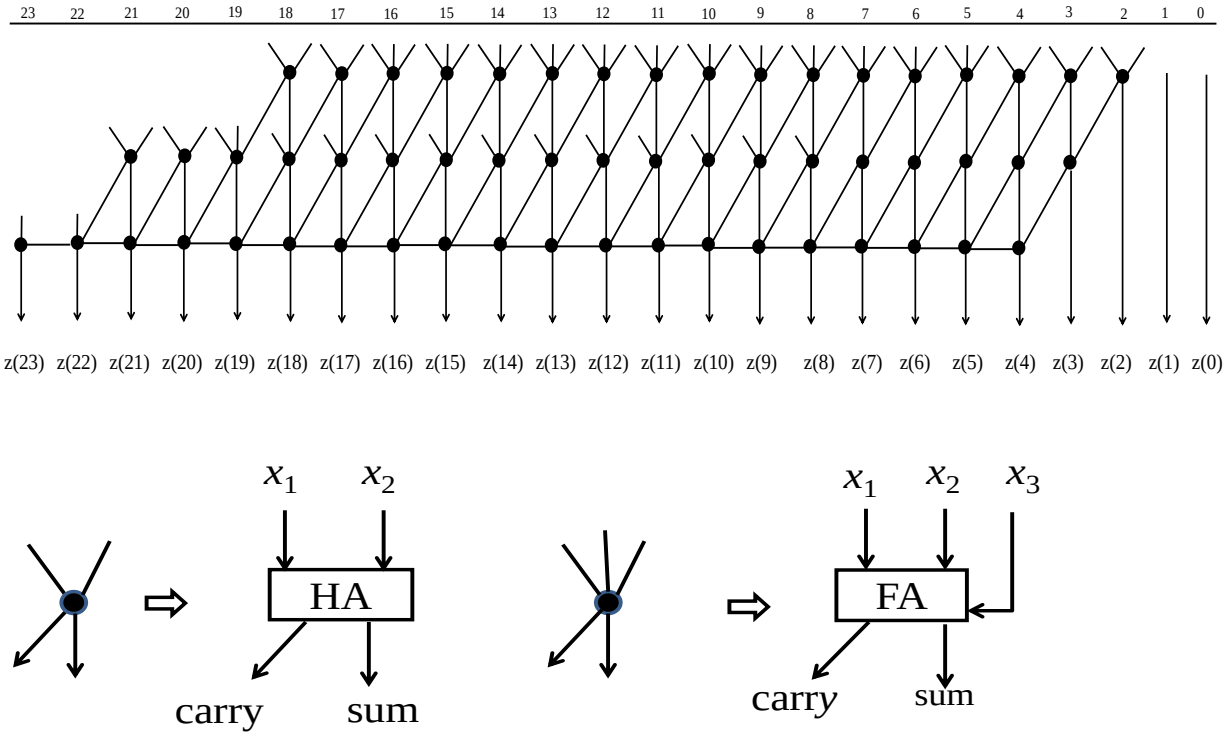

Figure 4.13: (a) Bit flow diagram of adder unit of radix-8 Booth multiplier for 12-bit. Critical path is shown in red colored arrow. (b) Node operation.

4.5 HARDWARE AND TIME COMPLEXITIES

4.5.1 HARDWARE COMPLEXITY

Hardware complexity of proposed radix-8, [Galal *et al*. (2013)] radix-8 and radix-4 multiplier for different bit-widths are estimated in terms of gate counts and the estimates are listed in Table 4.10. The proposed radix-8 multiplier structure has one PPG unit, one BEU, one PPS unit, one PPRU and one AU. The BEU consists of w BE where each BE further comprised of 5 NOR gates and 3 XNOR gates. Therefore, the BEU is comprised of ($5w$) NOR gates and ($3w$) XNOR gates. Similarly, the PPS unit consists of w PPS and each PPS is further comprised of 4 AND cells, 2 NOR cells, one NAND cell and one XOR cell, where one AND/NOR/NAND/XOR cell is consists of (n+2) 2-input gates, where $w=\lceil n/3 \rceil$, and n is the input operand bit-width. Therefore, the PPS unit is comprised of [$7w(n+2)$] AND/ NOR/ NAND cells and [$w(n+2)$] XOR cells. The PPG unit consists of one (n+2)-bit ripple carry adder (RCA), where the (n+2)-bit RCA is implemented using (n+1)-bit FAs and one HA. Since the adder unit can be implemented using different designs, here Wallace tree adder is considered to implement the adder unit. The adder unit complexity is estimated using the bit-flow diagram of Figure 4.10 in terms of FA and HA counts. For n=12 bit, the adder unit involves (42 FAs and 14 HAs) and for n=16 bit, the adder unit involves (66 FAs and 29 HAs). The PPRU involves 46 logic gates for n=12 bit and 54 logic gates for n=16 bit.

The radix-8 multiplier structure [Galal *et al*. (2013)] consists of one PPG unit, one BEU, one PPS unit and one AU. The logic complexity of PPG unit and BEU is same as those of the proposed structure. The PPS of multiplier structure of [Galal *et al*. (2013)] involves one less XOR gate than those of the proposed structure. However, the logic complexity of AU of [Galal *et al*. (2013)] is higher than the logic complexity of AU of proposed structure since the AU of [Galal *et al*. (2013)] involves one extra adder stage than those required by the AU of proposed structure. For n=12, the AU of [Galal *et al*. (2013)] involves (45 FAs and 2 HAs) and for n=16 bit, the AU involves 85 FAs and 46 HAs. The radix-4 multiplier design proposed by [Kuang *et al*. (2009)] is considered as the bench mark design for performance comparison. The hardware complexity of radix-4 multiplier design of [Kuang *et al*. (2009)] is estimated in the similar line of radix-8 multiplier design. In this case, the BEU consists of (n/2) BE where each BE further

comprised of 5 NOR gates and one XOR gates. Therefore, the BEU is comprised of (5n/2) NOR gates and (n/2) XOR gates. Similarly, the PPS unit consists of (n/2) PPS and each PPS further comprised of 4 AND/OR cells one XOR cell, where each AND/OR/XOR cell is comprised of (n+1) 2-input gates. Therefore, the PPS unit is comprised [2n(n+1)] AND/OR gates and [n(n+1)/2] XOR gates. The adder unit involves (70 FAs and 15 HAs) and (105 FAs and 18 HAs) for n=12 bit and 16 bit, respectively. The PPRU involves 32 and 43 logic gates for n=12 bit and 16 bit, respectively.

4.5.2 CRITICAL PATH DELAY (CPD)

The CPD of radix-8 Booth multiplier is defined as T={T_{PPG}+T_{BS}+T_{PPRU}+T_{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 (n+2)-bit RCA is the T_{PPG} and it is given as T_{PPG} =(2n+3)-T_X, T_{BS} is estimated as {T_{AND}+T_{NOR}+T_{NAND}+T_X} and the T_{PR} is estimated as {3T_{AND}+T_{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. The delay of AU is estimated as 5T_X+18T_{AND}+18T_{OR} (as shown in figure 4.10) for n=12 and T_{AU} =13T_X+24T_{AND}+24T_{OR}, for n=16 bit. Therefore, the proposed radix-8 multiplier has CPD T= (29T_X+22T_{AND}+19T_{OR}+T_{NOR}+T_{NAND}) for n=12-bit and T=(47T_X+28T_{AND}+24T_{OR}+T_{NOR}+T_{NAND}) for n=16 bit. The CPD of radix-8 multiplier [Galal *et al*. (2013)] is T= {T_{PPG}+T_{BS}+T_{AU}}. The delay of PPG unit, BSU is same as the proposed radix-8 multiplier structure, but the delay of AU unit is T_{AU} = (T_{AU} =7T_X+19T_{OR}+19T_{OR}) and T_{AU} = (17T_X+22T_{AND}+25T_{OR}) for n=12bit and 16 bit, respectively. Therefore, the CPD T= (31T_X+23T_{AND}+20T_{OR}+T_{NOR}+T_{NAND}) and T= (49T_X+30T_{AND} +26T_{OR} +T_{NOR}+T_{NAND}), for n=12 bit and 16-bit, respectively. The CPD of radix-4 Booth multiplier of [Kuang *et al*. (2009)] is defined as {T_{BS}+T_{BS}+T_{PPRU}+T_{AU}}. For n=12, T_{BE} is estimated as {T_{AND}+T_{OR}}, the T_{BS} is estimated as {T_X+T_{OR}+T_{NAND}} and T_{PPRU} is estimated as {T_{OR}+3T_{AND}+T_{NAND}}.The T_{AU} =9T_X+28T_{AND}+28T_{OR} and T_{AU}=(11T_X+37T_{AND}+37T_{OR}) for n=12 bit and n=16 bit, respectively. Therefore, the CPD of radix-4 multiplier is estimated as T= (10T_X+32T_{AND}+30T_{OR}+2T_{NAND}) and T=(12T_X+41T_{AND}+40T_{OR}+2T_{NAND}) for n=12 bit and 16 bit respectively.

4.5.3 PERFORMANCE COMPARISON

The theoretical estimates of hardware and time complexities of proposed radix-8 multiplier design, the existing radix-8 multiplier design of [Galal *et al*. (2013)] and the radix-4 design of [Kuang *et al*. (2009)] are listed in Table 4.10 for word length n=12 and 16. Assuming each FA is implemented using 2 XOR gates and 3 AND/OR gates, and HA is implemented using one XOR gate and one AND gate, the hardware complexity is estimated in terms of gate counts. As shown in Table 4.10, for n=12, the proposed multiplier design involves 22 less XOR/XNOR gates, 21 less AND/OR/NOR gates, and CPD less by ($2T_X+T_{AND}+T_{OR}$) compared to [Galal *et al*. (2013)]. For n=16, the proposed multiplier design involves 36 less XOR/XNOR gates, 33 less AND/OR/NOR gates, and CPD less by ($2T_X+T_{AND}+T_{OR}$) compared to [Galal *et al*. (2013)]. The saving in gates and CPD is achieved in the proposed design mainly due to the regular partial product array. Compared to radix-4 Booth multiplier [Kuang *et al*. (2009)], the proposed radix-8 multiplier design for n=12 involves 18 more XOR/XNOR gates, 77 less AND/OR/NOR gates, and the CPD less by ($T_{AND}+T_{OR}$). However, for n=16, the proposed structure involves 140 more XOR/XNOR gates, 138 more AND/OR/NOR gates, and the CPD higher by $6T_X$. In general, the proposed radix-8 multiplier design is expected to offer a better area-delay efficient structure than the existing radix-4 Booth multiplier design for n=12-bit and involves higher area-delay product compared to radix-4 multiplier design for higher bit-width. This is mainly due to the increase in PPS unit complexity of proposed radix-8 multiplier design for higher bit-width.

Table 4.10: Comparison of hardware and time complexities of Booth multiplier

Design	Bit-Width	OR/NOR/AND /NAND gate	XOR/XNOR gate	Critical Path
Radix-4 [Kung *et. al* (2009)]	12	873	308	$10T_X+32T_{AND}+30T_{OR}+2T_{NAND}$
	16	1485	484	$12T_X+41T_{AND}+40T_{OR}+2T_{NAND}$
Radix-8 [*Galal et. al* (2013)]	12	796	344	$31T_X+23T_{AND}+20T_{OR}+T_{NOR}+T_{NAND}$
	16	1623	657	$49T_X+30T_{AND}+26T_{OR}+T_{NOR}+T_{NAND}$
Proposed radix-8	12	774	323	$29T_X+22T_{AND}+19T_{OR}+T_{NOR}+T_{NAND}$
	16	1587	624	$47T_X+28T_{AND}+24T_{OR}+T_{NOR}+T_{NAND}$

Legend: 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 4.11: Comparison of synthesis results of Booth multiplier for different bit-width

Multiplier design	Bit-Width	Area (μm^2)	DAT (ns)	ADP ($\mu m^2 \times ms$)	Power (mW)
Radix-4 [Kuang *et. al* 2009)]	12	7344.22	16.11	118315	39.9
	16	10342.43	18.32	189473	42.1
Radix-8 [Galal *et. al* (2013)]	12	7578.35	16.32	123678	40.6
	16	12478.52	21.16	264045	47.7
Proposed Radix-8	12	7032.30	15.83	111321	38.3
	16	11091.94	20.52	227606	45.2

4.6 SYNTHESIS RESULTS

The proposed radix-8 multiplier design, existing radix-8 multiplier design of [Galal *et al*. (2013)] and radix-4 multiplier design of [Kuang *et al*. (2009)] are coded in VHDL for word length n=12 and 16. All these designs are synthesized in Synopsys Design Compiler using SAED 90nm CMOS library. A virtual clock signal is used to estimate the critical path delay in terms of minimum clock period (MCP). The area, MCP and power estimates of these designs reported by Design Compiler are listed in Table 4.11 for comparison. Power is estimated using a normalized 40 MHz (25 ns) clock signal. As shown in Table 4.11, the proposed radix-8 multiplier structure involves nearly 8% less area and 3% less MCP than the existing radix-8 multiplier design of [Galal *et al*. (2013)] on average for different bit-width. Compared with the existing radix-4 Booth multiplier design of [Kuang *et al*. (2009)], the proposed radix-8 multiplier design involves marginally less area and less MCP for n=12, and marginally higher area and MCP for n=16. The proposed radix-8 multiplier design for n=12, involves nearly 9.8% less area-delay product (ADP) and consumes nearly 5.6% less power than those of existing multiplier design of [Galal *et al*. (2013)]. The proposed radix-8 multiplier design involves 6.1% less ADP and consumes 4.1% less power than the existing radix-4 Booth multiplier design of [Kuang *et al*. (2009)] for n=12.

4.7 CONCLUSION

In this chapter, radix-8 Booth multiplication algorithm using symmetric anti-symmetric product coding (SAPC) is presented. The number of partial product terms used by partial product selector of radix-8 Booth recorded multiplication is reduced by one-forth using SAPC than those used in direct Booth encoding. Therefore, the SAPC simplifies the partial product generator design and partial product selector design substantially. A detail complexity analysis of radix-4 and radix-8 Booth multiplication is presented. The complexity analysis reveals that the partial product selection unit of radix-8 multiplier consumes relatively more combinational logic than the saving from the adder unit due to higher radix size. Based on these findings, a regular partial product array for radix-8 Booth multiplication is presented with a small overhead complexity. An optimized logic design is presented for the partial product selection unit. Comparison result shows that the proposed radix-8 multiplier design involves 43 and 69 less gates compared to the best available radix-8 multiplier design [Galal *et al*. (2013)] for n=12 and 16, respectively. Synthesis result shows that the proposed radix-8 multiplier design for n=12, involves nearly 9.8% less area-delay product (ADP) and consumes nearly 5.6% less power than those of existing multiplier design of [Galal *et al*. (2013)]. For n=12, the proposed radix-8 multiplier design involves 6.1% less ADP and consumes 4.1% less power than the state-of-the art radix-4 Booth multiplier design of [Kuang *et al*. (2009)]. This is an interesting feature of the proposed radix-8 multiplier design. The proposed radix-8 multiplier design may be considered instead of the most popular radix-4 Booth multiplier for 12-bit multiplications.