

# 國立交通大學

電子工程學系 電子研究所碩士班

## 碩士論文

自我補償之固定長度乘法器及其應用



Design of Self-Compensation Fixed-Width  
Multiplier and Its Applications

研究生：黃弘安

指導教授：張錫嘉

中華民國九十四年十月

自我補償之固定長度乘法器及其應用  
Design of Self-Compensation Fixed-Width  
Multiplier and Its Applications

學生：黃弘安

Student : Hong-An Huang

指導教授：張錫嘉

Advisor : Hsie-Chia Chang

國立交通大學

電子工程學系 電子研究所碩士班



Submitted to Department of Electronics  
College of Electrical Engineering and Computer Science  
National Chiao Tung University  
In Partial Fulfillment of the Requirements  
For the Degree of Master  
In  
Electronics Engineering

October 2005

Hsinchu, Taiwan, R.O.C.

# 自我補償之固定長度乘法器及其應用

學生：黃弘安

指導教授：張錫嘉

國立交通大學

電子工程學系 電子研究所碩士班



在本論文中，我們提出一個利用自我補償方法的固定長度乘法器架構。在這個架構中，藉由進位估算方程式，只需要少量的全加器就能夠計算所需要的進位補償值。為了減少因為刪除運算元件所造成的誤差，我們的架構會根據不同的乘法器長度而有其相對應的進位估算方程式，以達到最佳的效果。經由模擬結果發現，使用所提出的乘法器架構，刪除誤差可以降低到只有 Direct-truncated 乘法器的 15%，在面積部份，則是縮小到只有傳統 Booth 乘法器的 60%。此外，我們也將這個乘法器架構應用在 128 點 FFT 中，和使用 Direct-truncated 乘法器的 128 點 FFT 架構相比，我們的 SQNR (Signal to Quantization Noise Ratio) 高出了 10dB，而面積只增加了 2% 左右；相較於傳統的 Booth 乘法器架構，我們可以降低 10% 的面積，而且只減少約 1dB 的 SQNR。


# Design of Self-Compensation Fixed-Width Multiplier and Its Applications

Student: Hong-An Huang

Advisor: Hsie-Chia Chang

Institute of Electronics  
National Chiao Tung University

## ABSTRACT

The logo of National Chiao Tung University is a circular emblem. It features a gear-like outer border. Inside the circle, there is a stylized building or structure. At the bottom of the emblem, the year '1896' is inscribed.

This thesis introduces a self-compensation method for fixed-width multiplier which receives two  $n$ -bit inputs and produces an  $n$ -bit product. The truncated part that produces the carry-out bit is replaced with carry-estimation equations. In order to reduce the truncation errors, different input-width multipliers will correspond to different carry-estimation equations. Simulation results show that our self-compensation method can lead to 85% reduction of truncation errors while compared with direct-truncated multipliers, as well as 40% reduction in area of a multiplier while compared with traditional Booth multipliers. In contrast with the 128-FFT using direct-truncated multipliers, our 128-FFT approach has 10dB SQNR improvement and only 2% circuit penalty.

## 誌 謝

一轉眼，二年的研究所生活就過去，在這兩年中學到許多作學問的方法以及一些處世的道理。其中要感謝的人非常多，首先最要感謝的當然是我的指導教授張錫嘉博士。這兩年老師不但在研究上給予我許多的指導和方向，讓我能有一些研究成果，在生活上，老師也是一位很好的諮商顧問，當我們在生活上糟遇到問題，都會耐心的聽我們訴說，並且給我們適當的建議。再來要感謝的就是林建青學長，每當我遇到問題時，學長總是不厭其煩的幫我解答，讓我在研究的路上，能更加的順暢。此外，也要感謝 oasis 的同學和學弟們，這兩年來，因為有你們的陪伴，讓我的生活更加充實、快樂，真的很高興能認識你們這群好伙伴。最後，再一次的感謝在這兩年中，曾經幫助過我的每一位。



# CONTENTS

中文摘要.....	ii
英文摘要.....	iii
誌謝.....	iv
目錄.....	v
圖目錄.....	vii
表目錄.....	ix
Chapter 1 Introduction.....	- 1 -
1.1 Motivation .....	- 1 -
1.2 Thesis Organization.....	- 2 -
Chapter2 Existed Fixed-Width Multipliers .....	- 3 -
2.1 S-J Jou Approach.....	- 4 -
2.2 K-J Cho Approach.....	- 7 -
2.3 L-D Van Approach.....	- 15 -
Chapter3 Proposed Self-Compensation Multiplier.....	- 21 -
3.1 Calculation of Error-compensation bias .....	- 21 -
3.2 Proposed Structure.....	- 28 -
3.3 Performance Analysis.....	- 31 -
Chapter4 Application of Fixed-Width Multipliers in FFT .....	- 34 -
4.1 Introduction to 128-Point FFT.....	- 34 -
4.2 128-Point FFT Architecture.....	- 38 -
4.2.1 Module 1 .....	- 39 -
4.2.2 Module 2.....	- 40 -
4.2.3 Module 3.....	- 41 -

4.3 Simulation Results.....	- 42 -
Chapter5 Implementation Results for 128-point FFT .....	- 44 -
5.1 Approach 1.....	- 44 -
5.2 Approach 2.....	- 46 -
5.3 Comparison.....	- 51 -
Chapter6 Conclusion .....	- 52 -
BIBLIOGRAPHY .....	- 53 -



# *List of Figures*

FIG 1-1: 8-BIT BOOTH MULTIPLIER.....	- 1 -
FIG 2-1: PARTIAL PRODUCT OF 8-BIT BOOTH MULTIPLIER .....	- 3 -
FIG 2-2: EXAMPLE OF 6 X 8 BOOTH MULTIPLIERS .....	- 4 -
FIG 2-3: 6 X 8 FIXED-WIDTH MULTIPLIER WITH S-J JOU APPROACH .....	- 6 -
FIG 2-4: STRUCTURE OF K-J CHO SCHEME.....	- 8 -
FIG 2-5: PARTIAL PRODUCTS FOR $Y_3''Y_2''Y_1''Y_0'' = 0001$ .....	- 10 -
FIG 2-6: KARNAUGH MAP REPRESENTATION FOR (A)LP_CARRY_0 AND (B)LP_CARRY_1 FOR N = 10 .....	- 12 -
FIG 2-7: CIRCUIT OF APPROXIMATE CARRY FOR N = 10.....	- 12 -
FIG 2-8: APPROXIMATE CARRY GENERATION CIRCUITS (A)N = 10 (B)N = 14 .....	- 14 -
FIG 2-9: FIXED-WIDTH MULTIPLIER WITH K-J CHO APPROACH FOR N = 8.....	- 15 -
FIG 2-10 PARTIAL PRODUCT OF 8-BIT BAUGH-WOOLEY ARRAY MULTIPLIER .....	- 15 -
FIG 2-11: FIXED-WIDTH MULTIPLIER WITH L-D VAN APPROACH FOR N = 8 .....	- 20 -
FIG 3-1: PARTIAL PRODUCT OF 8-BIT BOOTH MULTIPLIER .....	- 22 -
FIG 3-2: THE ERROR-COMPENSATION TABLE FOR N = 8 .....	- 23 -
FIG 3-3: CIRCUIT OF CARRY-ESTIMATION EQUATION FOR N = 8 .....	- 28 -
FIG 3-4: CIRCUIT OF 8-BIT BOOTH MULTIPLIER WITH PROPOSED APPROACH .....	- 29 -
FIG 3-5: CIRCUITS OF CARRY-ESTIMATION EQUATIONS FOR (A) N = 10 (B) N = 12 (C) N = 14 (D) N = 16 .....	- 30 -
FIG 4-1: THE SIGNAL FLOW GRAPH OF RADIX-8 FFT ALGORITHM .....	- 37 -
FIG 4-2: THE SIGNAL FLOW GRAPH OF 128-POINT MIXED-RADIX FFT ALGORITHM .....	- 38 -
FIG 4-3: BLOCK DIAGRAM OF 128-POINT FFT .....	- 39 -
FIG 4-4: BLOCK DIAGRAM OF THE MODULE 1 .....	- 40 -



FIG 4-5: BLOCK DIAGRAM OF THE MODULE 2..... - 41 -

FIG 4-6: BLOCK DIAGRAM OF THE MODULE 3..... - 41 -

FIG 5-1 LAYOUT VIEW OF APPROACH 1 ..... - 45 -

FIG 5-2: STRUCTURE OF DDR REGISTER..... - 47 -

FIG 5-3: OPERATIONS OF PROPOSED DDR REGISTER AT (A) CLK = 0 (B) CLK = 1..... - 48 -

FIG 5-4: EXAMPLE OF PROPOSED DDR REGISTER ..... - 48 -

FIG 5-5 LAYOUT VIEW OF APPROACH 2 ..... - 50 -



# *List of Tables*

TABLE 2-1: PROBABLE VALUES OF $CARRY_{T-1}$ WITH DIFFERENT VALUES OF B AND T .....	- 6 -
TABLE 2-2: PARTIAL PRODUCT FOR EACH ENCODED $Y_1'$ WITH $N = 8$ .....	- 7 -
TABLE 2-3: 8-BIT NUMBERS WITH $Y_3''Y_2''Y_1''Y_0'' = 1000$ .....	- 9 -
TABLE 2-4: ROUNDED VALUE OF $E[\Lambda]$ FOR $N = 10$ .....	- 11 -
TABLE 2-5: REPRESENTATION OF APPROXIMATE CARRY VALUES .....	- 11 -
TABLE 3-1: THE VALUES OF $\Theta$ FOR DIFFERENT VALUES OF $P_{3\_0}P_{2\_2}P_{1\_4}P_{0\_6}$ .....	- 22 -
TABLE 3-2: THE AVERAGE CARRY FOR EACH VALUE OF $\Theta$ .....	- 23 -
TABLE 3-3: THE VALUES OF AVERAGE CARRY FOR EACH B FOR $N = 8$ .....	- 24 -
TABLE 3-4: THE RESULTS OF CARRY-ESTIMATION EQUATION FOR $N = 8$ .....	- 25 -
TABLE 3-5: THE RESULTS OF CARRY-ESTIMATION EQUATION FOR (A) $N = 10$ (B) $N = 12$ (C) $N = 14$ (D) $N = 16$ .....	- 26 -
TABLE 3-6: THE NUMBERS OF CASES FOR EACH B WITH $N = 10$ .....	- 27 -
TABLE 3-7: THE NUMBERS OF CASES FOR EACH B WITH $N = 14$ .....	- 27 -
TABLE 3-8: THE NUMBERS OF CASES FOR EACH B WITH $N = 16$ .....	- 27 -
TABLE 3-9: COMPARISON RESULTS OF AVERAGE ERROR .....	- 32 -
TABLE 3-10: COMPARISON RESULTS OF VARIANCE OF ERROR .....	- 33 -
TABLE 3-11: COMPARISON RESULTS OF GATE COUNTS .....	- 33 -
TABLE 4-1: SQNR OF 128-POINT FFT WITH DIFFERENT MULTIPLIER APPROACH .....	- 42 -
TABLE 4-2: GATE COUNT OF 128-POINT FFT FOR $N = 10$ .....	- 43 -
TABLE 4-3: SQNR OF 8192-POINT FFT WITH DIFFERENT MULTIPLIER APPROACH .....	- 43 -
TABLE 5-1: THE CHIP SUMMARY OF APPROACH 1 ARCHITECTURE .....	- 46 -
TABLE 5-2: FUNCTION OF LATCH .....	- 47 -
TABLE 5-3 FUNCTION OF PROPOSED DDR REGISTER .....	- 49 -

TABLE 5-4: THE CHIP SUMMARY OF APPROACH 2 ARCHITECTURE..... - 50 -  
TABLE 5-5: COMPARISON OF VARIOUS 128-POINT FFT ARCHITECTURES..... - 51 -





truncation errors.

In this thesis, the low-error area-efficient fixed-width multiplier based on Booth multiplier is proposed. The error-compensation bias of proposed approach is produced by the carry-estimation equations. The equations are adapted for different input width “n”. And these equations can be analyzed by few full adders. Thus, the area penalty caused by error-compensation is very small. By simulation results, the proposed fixed-width multiplier can not only reduce the truncation errors of direct-truncated multiplier but also decrease the area of standard Booth multiplier.

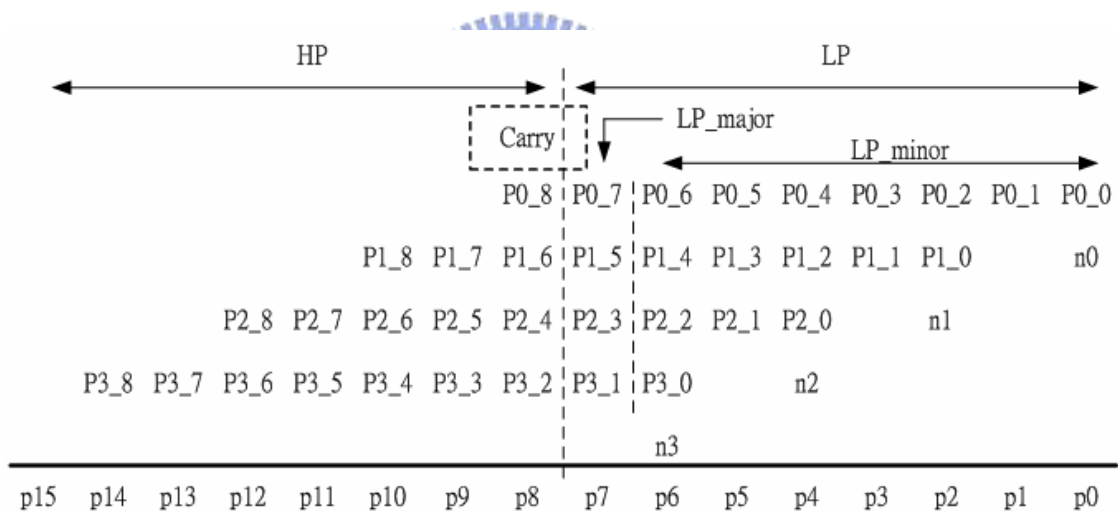
So as to compare the performance in real applications, our fixed-width multiplier is employed in 128-point FFT architecture. Compare to the direct-truncated multiplier, the proposed multiplier has higher SQNR with only 2% increase in circuit overhead.

## 1.2 Thesis Organization

The organization of this thesis is described as follows. In chapter 2, three existed fixed-width multipliers are introduced. Chapter 3 shows the proposed fixed-width multiplier . The applications of proposed fixed-width multiplier are described in chapter 4. In which, the proposed multiplier is employed in 128-point FFT architecture. The design and chip implementation are shown in Chapter 5. The structure of DDR register which can reduce the operation frequency is also described in Chapter 5. Finally, Chapter 6 is the conclusion.

# Chapter2 Existed Fixed-Width Multipliers

For lower computations area, the multiplications of DSP applications are usually have the fixed-width property. In other words, the bits right of decimal point are commonly omitted. Fig 2-1 shows the partial products of 8-bit Booth multiplier, it is divided into two parts, the low part (LP) and the high part (HP). The signal “Carry” means the carry from LP to HP. The adder cells required for the computation of LP in Fig 2-1 are usually truncated in DSP application. Because the carry from low part to high part was also skipped (Carry = 0), the significant truncation errors will be produced since no any error-compensation bias is employed.



**Fig 2-1: Partial product of 8-bit Booth multiplier**

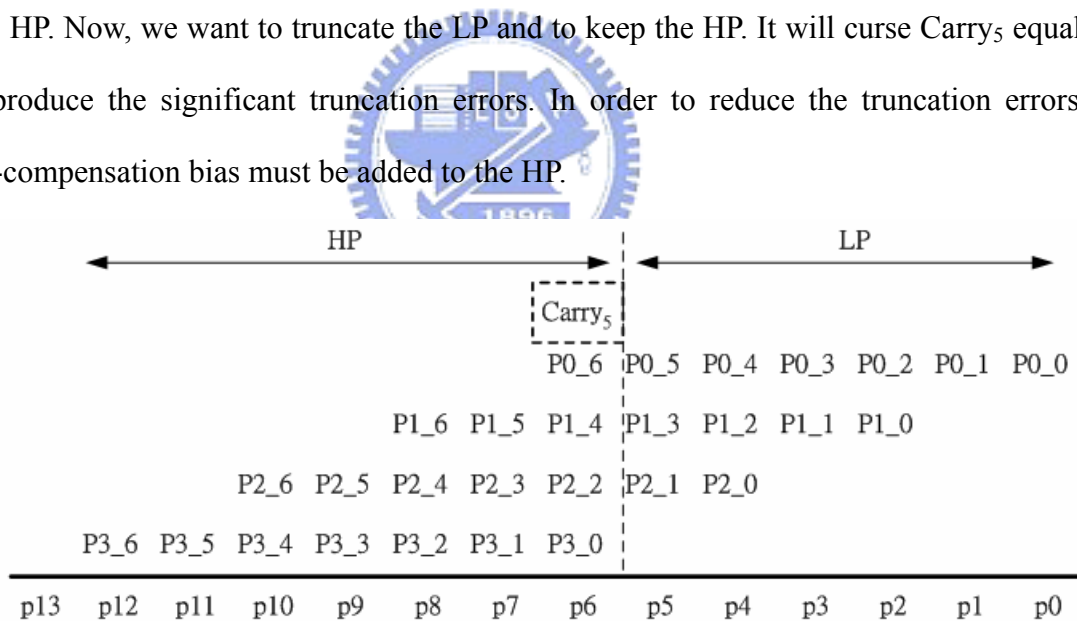
Many schemes are presented to calculate the error-compensation bias. In [1]-[3], a constant error-compensation bias is used to the retained cells. Because the bias do not adapt to the input signals, the truncation errors of these methods are large. In [4] and [5], an adaptive error-compensation bias approach which is obtained from the column of partial products adjacent to the truncated LSB is used to reduce the truncation error. In this chapter, three existed approach of fixed-width multiplier, S-J Jou approach, K-J Cho approach, and L-D Van

approach, will be introduced. These schemes can not only reduce the truncation error but also decrease the area of multiplications efficiently.

## 2.1 S-J Jou Approach

In this section, the S-J Jou approach [6] [7] which is based on Booth multiplier will be introduced. The error-compensation bias of S-J Jou approach is generated using statistical analysis and linear regression analysis. The process of S-J Jou is represented as follows.

Fig 2-2 shows the partial product of 6 x 8 Booth multiplier. The partial product is divided into two parts, the six least-significant columns are the low part (LP) and the eight most-significant columns are the high part (HP). And the signal, Carry<sub>5</sub>, means the carry from LP to HP. Now, we want to truncate the LP and to keep the HP. It will cause Carry<sub>5</sub> equal to 0 and produce the significant truncation errors. In order to reduce the truncation errors, the error-compensation bias must be added to the HP.



**Fig 2-2: Example of 6 x 8 Booth multipliers**

The carry from LP to HP can be obtained by Equation (2.1), where  $\lfloor x \rfloor$  represents the largest integer less than or equal to the number  $x$ .

$$\begin{aligned}
\text{Carry}_5 = & \lfloor 2^{-1}(P_{0_5} + P_{1_3} + P_{2_1}) + 2^{-2}(P_{0_4} + P_{1_2} + P_{2_0}) \\
& + 2^{-3}(P_{0_3} + P_{1_1}) + 2^{-4}(P_{0_2} + P_{1_0}) \\
& + 2^{-5}P_{0_1} + 2^{-6}P_{0_0} \rfloor
\end{aligned} \tag{2.1}$$

In general, Equation (2.1) can be written as Equation (2.2). The value of  $\tau$  means the number of columns of LP which will be truncated.

$$\begin{aligned}
\text{Carry}_{\tau-1} = & \lfloor 2^{-1}(P_{0_{\tau-1}} + P_{1_{\tau-3}} + \dots + P_{\lceil \tau/2 \rceil - 1_1}) \\
& + 2^{-2}(P_{0_{\tau-2}} + P_{1_{\tau-4}} + \dots + P_{\lceil \tau/2 \rceil - 1_0}) \\
& + \dots + 2^{-(\tau-1)}P_{0_1} + 2^{-\tau}P_{0_0} \rfloor \\
= & \lfloor 2^{-1}\beta + \lambda \rfloor
\end{aligned} \tag{2.2}$$

As seen in Equation (2.2), the carry is composed of  $\beta$  and  $\lambda$ .

$$\begin{aligned}
\beta = & P_{0_{\tau-1}} + P_{1_{\tau-3}} + \dots + P_{\lceil \tau/2 \rceil - 1_1} \\
\lambda = & 2^{-2}(P_{0_{\tau-2}} + P_{1_{\tau-4}} + \dots + P_{\lceil \tau/2 \rceil - 1_0}) + \dots + 2^{-(\tau-1)}P_{0_1} + 2^{-\tau}P_{0_0}
\end{aligned} \tag{2.3}$$

Where  $\lceil x \rceil$  represents the smallest integer that is larger than or equal to the number  $x$ . The value of  $\beta$  means the total number of “1” in the  $(\tau - 1)$ <sup>th</sup> column. If the value of  $\lambda$  can be expressed in terms of  $\beta$  and  $\tau$ , the error-compensation bias can be obtained in terms of only  $\beta$  and  $\tau$ . Before we introduce the process of S-J Jou approach, we assume that the probability of each input data bit equaling “1” is 0.5 and the probability of each partial product bit  $P_{i_j}$  equaling “1” is  $P(P_{i_j})$ . According to the  $P(P_{i_j})$  concept, the equation of  $\lambda$  can be rewritten as

$$\lambda = \sum_{k=1}^{\tau-1} \frac{1}{2^{k+1}} \times P(P_{i_j}) \times \left\lceil \frac{\tau - k}{2} \right\rceil \tag{2.4}$$

The values of  $P(P_{i_j})$  are different for different  $\beta$  and  $\tau$ . By using statistical analysis and linear regression line analysis,  $P(P_{i_j})$  can be approximated as a first-order polynomial.

$$P(P_{i_j}) = \frac{0.41}{\tau} \times \beta + 0.58(0.01 \times \tau + 0.37) \tag{2.5}$$

Taking Equation (2.4) and Equation (2.5) into Equation (2.3), the error-compensation bias can



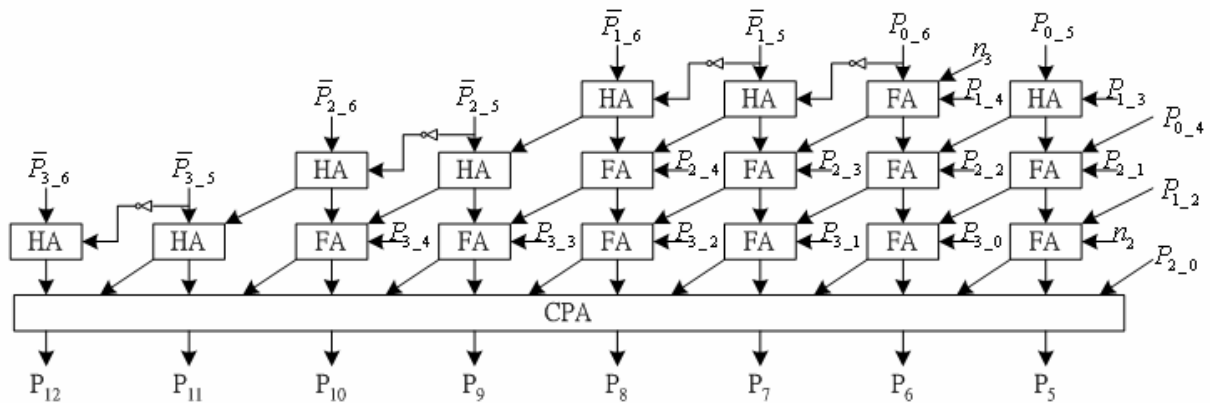
be obtained by Equation (2.6).

$$Carry_{\tau-1} = \left[ 2^{-1} \beta + \left\{ \sum_{k=1}^{\tau-1} \frac{1}{2^{k+1}} \left[ \frac{0.41}{\tau} \beta + 0.58(0.01\tau + 0.37) \left[ \frac{\tau-k}{2} \right] \right] \right\} + 0.5 \right] \quad (2.6)$$

The probable values of  $Carry_{\tau-1}$  for different  $\tau$  are listed in Table 2-1. It is obviously that the best error-compensation bias is  $\beta$  for any  $\tau$ .

**Table 2-1: Probable values of  $Carry_{\tau-1}$  with different values of  $\beta$  and  $\tau$**

$\tau$	$\beta+2$	$\beta+1$	$\beta$	$\beta-1$	$\beta-2$	$\beta-3$	Expected Value
4	0	2.34%	85.94%	11.72%	0	0	$\beta - 0.09$
6	1.27%	36.35%	56.88%	5.49%	0	0	$\beta + 0.33$
8	2.11%	37.06%	53.05%	7.75%	0.04%	0	$\beta + 0.33$
10	3.23%	36.78%	50.30%	9.54%	0.14%	0	$\beta + 0.33$
12	4.38%	36.24%	47.97%	11.09%	0.31%	3.58E-7	$\beta + 0.33$
14	5.52%	35.66%	45.88%	12.38%	0.55%	1.20E-5	$\beta + 0.33$



**Fig 2-3: 6 x 8 fixed-width multiplier with S-J Jou approach**

The circuit of 6 x 8 fixed-width Booth multiplier with S-J Jou approach is shown in Fig 2-3. As seen in Fig 2-4, the adder cells of LP are omitted and the carry from LP to HP is replaced by  $\beta (P_{0\_5} + P_{1\_3} + P_{2\_1})$ .

## 2.2 K-J Cho Approach

The S-J Jou approach is introduced in last section. In which, the error-compensation bias is generated using statistical analysis and linear regression analysis. And the probability of each partial product bit  $P_{i\_j}$  equaling “1” is different for different  $\beta$  and  $\tau$ . In this section, the second approach, K-J Cho approach [8] [9], will be introduced. In this approach, the error-compensation bias is obtained by using Booth encoder outputs. And the probability of each partial product bit  $P_{i\_j}$  equaling “1” is 1/2 for any  $\beta$  and  $\tau$ .

Table 2-2 shows the values of partial product of 8-bit Booth multipliers. Where

$$b_i' = -2 \cdot b_{2i+1} + b_{2i} + b_{2i-1} \quad (2.4)$$

If the value of  $b_i'$  is zero, each bit of partial product “ $P_i$ ” will be zero. Otherwise, the value of partial product “ $P_i$ ” will be based on input data “A”.

**Table 2-2: Partial product for each encoded  $y_i'$  with  $n = 8$**

$b_i'$	$P_{i\_8}$	$P_{i\_7}$	$P_{i\_6}$	$P_{i\_5}$	$P_{i\_4}$	$P_{i\_3}$	$P_{i\_2}$	$P_{i\_1}$	$P_{i\_0}$	$n_i$
0	0	0	0	0	0	0	0	0	0	0
1	$a_7$	$a_7$	$a_6$	$a_5$	$a_4$	$a_3$	$a_2$	$a_1$	$a_0$	0
-1	$\bar{a}_7$	$\bar{a}_7$	$\bar{a}_6$	$\bar{a}_5$	$\bar{a}_4$	$\bar{a}_3$	$\bar{a}_2$	$\bar{a}_1$	$\bar{a}_0$	1
2	$a_7$	$a_6$	$a_5$	$a_4$	$a_3$	$a_2$	$a_1$	$a_0$	0	0
-2	$\bar{a}_7$	$\bar{a}_6$	$\bar{a}_5$	$\bar{a}_4$	$\bar{a}_3$	$\bar{a}_2$	$\bar{a}_1$	$\bar{a}_0$	1	1

From Fig 2-1, the carry from low part from high part can be expressed as

$$Carry_7 = \left\lfloor \frac{1}{2} \beta + \lambda \right\rfloor \quad (2.5)$$

$$\beta = P_{0_6} + P_{1_4} + P_{2_2} + P_{3_0} \quad (2.6)$$

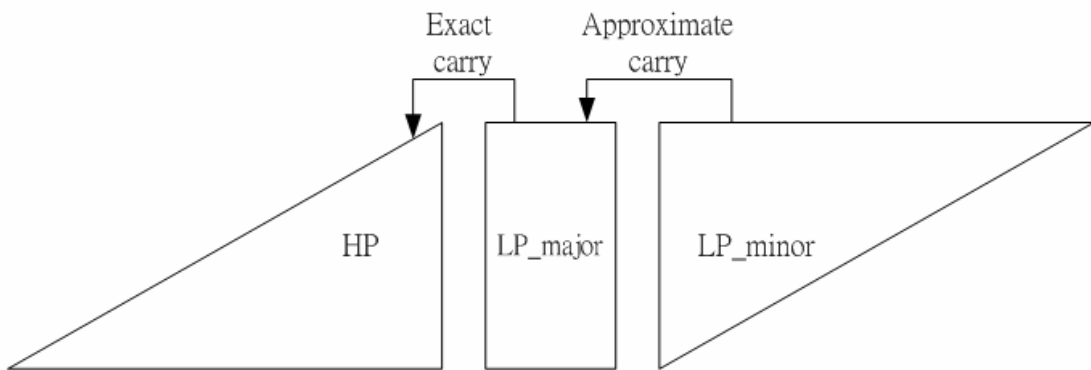
$$\begin{aligned} \lambda = & 2^{-2}(P_{0_5} + P_{1_3} + P_{2_1}) + 2^{-3}(P_{0_4} + P_{1_2} + P_{2_0} + n_2) \\ & + 2^{-4}(P_{0_3} + P_{1_1}) + 2^{-5}(P_{0_2} + P_{1_0} + n_1) \\ & + 2^{-6}(P_{0_1}) + 2^{-7}(P_{0_0} + n_0) \end{aligned} \quad (2.7)$$

The value of  $\beta$  is sum of the elements in LP\_major and the value of  $\lambda$  is the sum of the elements in LP\_minor.

Fig 2-4 shows the structure of K-J Cho approach. The adder cells of LP\_minor are omitted and the error-compensation bias of low part is defined as follow.

$$Carry_{t-1} = C_E \left\lfloor \frac{1}{2} \beta + C_A[\lambda] \right\rfloor \quad (2.8)$$

Where  $C_E[t]$  represents the exact carry value of  $t$  and  $C_A[t]$  means the approximate carry value of  $t$ . So,  $C_A[\lambda]$  means the approximate carry from LP\_minor to LP\_major.



**Fig 2-4: Structure of K-J Cho scheme**

In order to find the error-compensation bias, to define  $y_i''$  as

$$y_i'' = \begin{cases} 1, & \text{if } y_i' \neq 0 \\ 0, & \text{otherwise} \end{cases} \quad (2.9)$$

For example, if the value of  $y_3''y_2''y_1''y_0''$  is 1000, the coded number  $y_3'y_2'y_1'y_0'$  should have

four possible values: 1000, 2000, -1000, and -2000. There are only three 8-bit numbers which can have  $y_3''y_2''y_1''y_0'' = 1000$ . Table 2-3 shows the three 8-bit numbers.

**Table 2-3: 8-bit numbers with  $y_3''y_2''y_1''y_0'' = 1000$**

8-bit number								$y_3'$	$y_2'$	$y_1'$	$y_0'$
0	1	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	-2	0	0	0
1	1	0	0	0	0	0	0	-1	0	0	0

For the case  $y_3''y_2''y_1''y_0'' = 0001$ , there are also only three possible values of 8-bit numbers, which are shown as follow

$$\begin{aligned}
 00000001(0) &\rightarrow y_3'y_2'y_1'y_0' = 0001 \\
 11111110(0) &\rightarrow y_3'y_2'y_1'y_0' = 000\bar{2} \\
 11111111(0) &\rightarrow y_3'y_2'y_1'y_0' = 000\bar{1}
 \end{aligned} \tag{2.10}$$

The partial products for the three multiplier coefficients corresponding to  $y_3''y_2''y_1''y_0'' = 0001$  is shown in Fig 2-5. As we have assumed in last section, the probability of each input bit equaling “1” is 0.5. That is

$$E[a_i] = \frac{1}{2} \tag{2.11}$$

Thus, the rounded value of  $E[\lambda]$  for each of the three cases in Fig 2-5 can be computed as follows:

$$\{E[\lambda]\}_r = \begin{cases} 0, & \text{for } y_3'y_2'y_1'y_0' = 0001 \\ 1, & \text{for } y_3'y_2'y_1'y_0' = 000\bar{2}, 000\bar{1} \end{cases} \tag{2.12}$$

Where  $\{t\}_r$  means rounding operation for t.

In equation (2.12), there are two cases that  $\{E[\lambda]\}_r = 1$  and one case that  $\{E[\lambda]\}_r = 0$ . In other words, the probability of  $\{E[\lambda]\}_r$  equaling “1” is 2/3 which is bigger than 1/2. So, the

value of  $\{E[\lambda]\}_r$  can be set to 1 for  $y_3''y_2''y_1''y_0'' = 0001$ .

Notice that  $E[\lambda]$  is always zero for the three 8-bit numbers with  $y_3''y_2''y_1''y_0'' = 1000$ . Because no element of the partial product corresponding to  $y_3'$  is included in LP\_minor as can be seen in Fig 2-1. In general, the element of the partial product corresponding to  $y_{n/2-1}'$  is not included in LP\_minor for any input width "n".

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14		
$y_3'y_2'y_1'y_0' = 0001$															0	
								$a_7$	$a_7$	$a_6$	$a_5$	$a_4$	$a_3$	$a_2$	$a_1$	$a_0$
$y_3'y_2'y_1'y_0' = 000\bar{1}$															1	
								$\bar{a}_7$	$\bar{a}_7$	$\bar{a}_6$	$\bar{a}_5$	$\bar{a}_4$	$\bar{a}_3$	$\bar{a}_2$	$\bar{a}_1$	$\bar{a}_0$
$y_3'y_2'y_1'y_0' = 000\bar{2}$															1	
								$\bar{a}_7$	$\bar{a}_6$	$\bar{a}_5$	$\bar{a}_4$	$\bar{a}_3$	$\bar{a}_2$	$\bar{a}_1$	$\bar{a}_0$	1
HP									LP							

**Fig 2-5: Partial products for  $y_3''y_2''y_1''y_0'' = 0001$**

From previous discussions, it is obvious that the value of  $E[\lambda]$  is calculated by the LP\_minor of partial product. By using K-J Cho approach, to determine the error-compensation bias is more easily. Because the carry from LP\_minor to LP\_major is replaced by  $\{E[\lambda]_r\}$ , we only need to calculate the values of  $\{E[\lambda]_r\}$  for each case of  $y_{n/2-2}'' y_{n/2-3}'' \dots y_0''$ . Then, the circuit of carry generation can be designed based on the values of  $\{E[\lambda]_r\}$ .

The procedure of K-J Cho approach is explained in the following example.

Example 1: In this example, it will show the process of K-J Cho approach by using a 10 x 10 Booth multiplier. First, we should calculate the values of  $\{E[\lambda]\}_r$  for all the possible values of  $y_3''y_2''y_1''y_0''$  and the values of  $\{E[\lambda]\}_r$  are shown in Table 2-4. Notice that  $y_4''$  is not shown in Table 2-4 since there is no any element of the partial product corresponding to  $y_4'$  is included in LP\_minor.

**Table 2-4: Rounded value of  $E[\lambda]$  for  $n = 10$**

$y_3''y_2''y_1''y_0''$ (# of cases)	$\{E[\lambda]\}_r$ (# of cases)
0 0 0 0 (4)	0 (4)
0 0 0 1 (12)	0 (4) 1 (8)
0 0 1 0 (12)	0 (4) 1 (8)
0 0 1 1 (36)	1 (36)
0 1 0 0 (12)	0 (4) 1 (8)
0 1 0 1 (36)	1 (36)
0 1 1 0 (36)	1 (36)
0 1 1 1 (108)	1 (52) 2 (56)
1 0 0 0 (12)	0 (4) 1 (8)
1 0 0 1 (36)	1 (36)
1 0 1 0 (36)	1 (36)
1 0 1 1 (108)	1 (52) 2 (56)
1 1 0 0 (36)	1 (36)
1 1 0 1 (108)	1 (52) 2 (56)
1 1 1 0 (108)	1 (52) 2 (56)
1 1 1 1 (324)	2 (324)

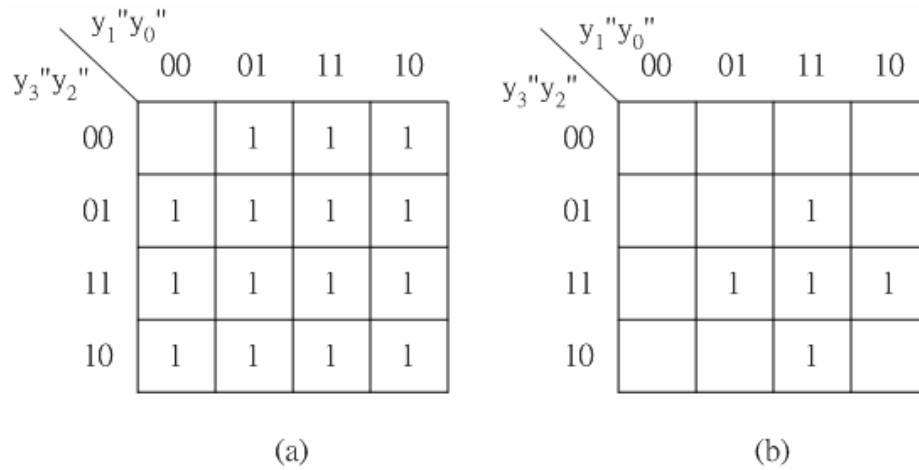
**Table 2-5: Representation of approximate carry values**

Rounded value	LP_carry_0	LP_carry_1
0	0	0
1	1	0
2	1	1

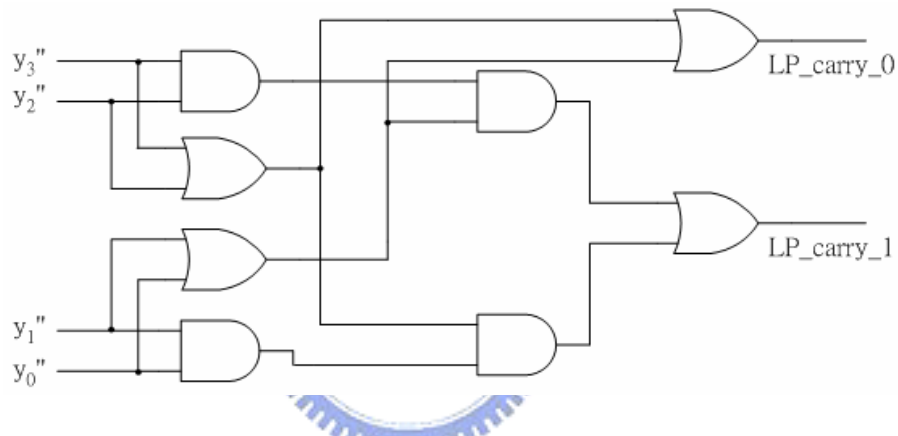
In Table 2-4, the biggest value of carry is two. Thus, two approximate carry signals (LP\_carry\_0 and LP\_carry\_1) are needed to represent the values of  $\{E[\lambda]\}_r$ . The values of the two carry signals are shown in Table 2-5. We can obtain the circuit of the approximate carry signals by using Karnaugh map as shown in Fig 2-6. In Fig 2-6, the values of approximate carry signals can be determined using probability analysis. For example, for  $y_3''y_2''y_1''y_0'' = 0001$ ,  $P[\{E[\lambda]\}_r=0] = 4/12$  and  $P[\{E[\lambda]\}_r=1] = 8/12$ . Thus, the value of approximate carry signals is determined to be 1. Then, LP\_carry\_0 and LP\_carry\_1 signals can be simplified from each map as

$$\begin{aligned}
 LP\_carry\_0 &= y_3'' + y_2'' + y_1'' + y_0'' \\
 LP\_carry\_1 &= y_3''y_2''(y_1'' + y_0'') + y_1''y_0''(y_3'' + y_2'')
 \end{aligned}
 \tag{2.13}$$

Fig 2-7 shows the circuit of equation (2.13) which is the approximate carry signals from LP\_minor to LP\_major. The approximate carry signals are added to LP\_major. Then, the resulted carry signals from LP\_major are added to HP as error-compensation bias.



**Fig 2-6: Karnaugh map representation for (a)LP\_carry\_0 and (b)LP\_carry\_1 for n = 10**



**Fig 2-7: Circuit of approximate carry for n = 10**

The procedure of Example 1 is illustrated as below:

I. For given input width “n”, the number of approximate carry signals is determined as

$$N_{AC} = \lfloor n/4 \rfloor$$

II. The approximate carry signals are denoted as LP\_carry\_0, LP\_carry\_1, ... , LP\_carry\_(N<sub>AC</sub> - 1)

III. To calculate the rounded values of  $\{E[\lambda]_r\}$  for each case of  $y_{n/2-2}'' y_{n/2-3}'' \dots y_0''$ .

IV. By applying Karnaugh map to the result in step III, approximate carry generation circuit can be designed.

To perform the exhaustive simulation for large width of input data will take a lot of time. A statistical analysis for obtaining the approximate carry values is introduced as below.

Given  $y_i''$  is 1, it can be shown that  $E[P_{i,j}] = 1/2$ . If  $y_2''y_1''y_0'' = 100$  in Fig 2-1,  $E[\lambda]$  can be computed by using equation (2.5)

$$\begin{aligned}
 E[\lambda] &= E[2^{-1}(P_{2,-1}) + 2^{-2}(P_{2,-0} + n_2)] \\
 &= 2^{-1}E[P_{2,-1}] + 2^{-2}(E[P_{2,-0}] + E[n_2]) \\
 &= 2^{-1}(2^{-1}) + 2 - 2(2^{-1} + 2^{-1}) \\
 &= 2^{-1}
 \end{aligned} \tag{2.14}$$

By the same way, it can be shown that  $E[\lambda]$  is also equal to  $1/2$  for  $y_2''y_1''y_0'' = 010$  and  $001$ .

So, for  $n = 8$ ,  $E[\lambda]$  can be expressed as

$$E[\lambda] = 2^{-1}(y_2'' + y_1'' + y_0'') \tag{2.15}$$

In general,  $E[\lambda]$  can be computed by equation (2.14)

$$\begin{aligned}
 E[\lambda] &= 2^{-1}(y_{n/2-2}'' + y_{n/2-3}'' + \dots + y_0'') \\
 &= 2^{-1} \cdot \sum_{i=0}^{n/2-2} y_i''
 \end{aligned} \tag{2.16}$$

In the following example, the procedure of this scheme for  $n = 10$  is explained.

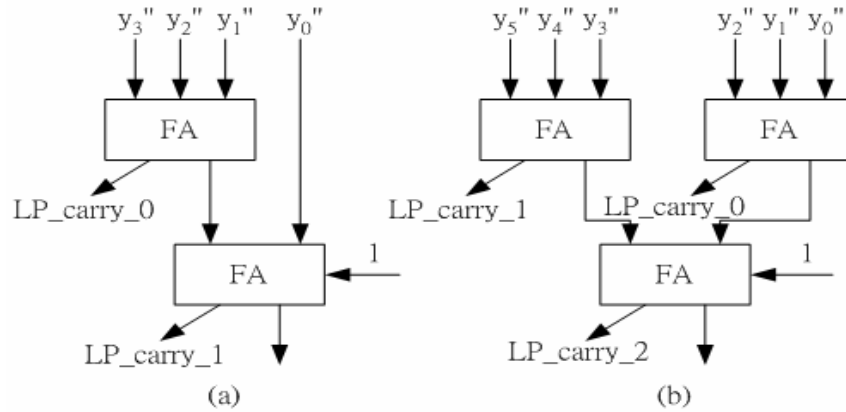
Example 2: For  $n = 10$

$$E[\lambda] = 2^{-1}(y_3'' + y_2'' + y_1'' + y_0'') \tag{2.17}$$

The maximum rounded value of  $E[\lambda]$  is 2. Hence, two signals are needed to represent the rounded value.

If the number of  $y_i''$  equaling “1” are one or two, the rounded value is equal to 1. Else if the number of  $y_i''$  equaling “1” are more than three, the rounded value is equal to 2. Then, the approximate carry generation circuit for  $n = 10$  can be obtain as shown in Fig 2-8(a). Using the same scheme, the approximate carry circuit for  $n = 14$  is shown in Fig 2-8(b).



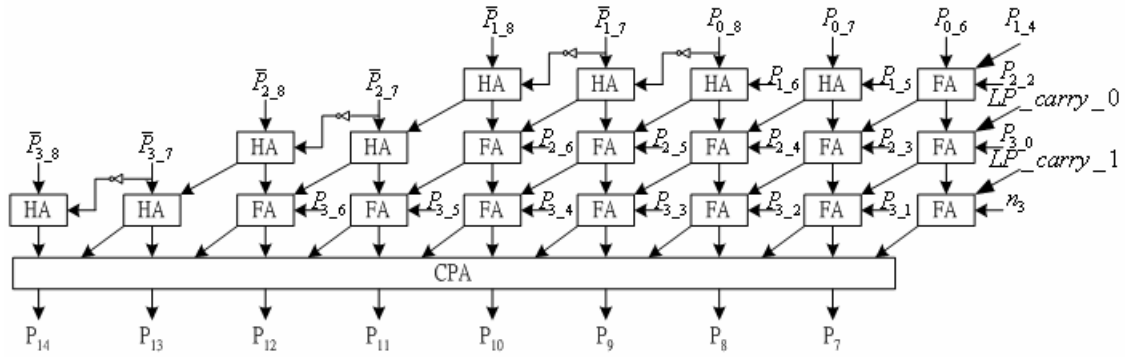


**Fig 2-8: Approximate carry generation circuits (a)  $n = 10$  (b)  $n = 14$**

The procedure of Example 2 described as below:

- I. The signals in the  $\{y_{n/2-2}'', y_{n/2-3}'', \dots, y_0''\}$  are divided into groups of three signals. If the number of signals in the set is  $3N + k$  ( $k = 1, 2$ ), the last group contains only  $k$  signals.
- II. The  $3N$  signals are added using  $N$  FAs. For  $k = 2$ , the two signals in the last group are added using a HA. For  $k = 1$ , the signal in the last group is passed to the next stage. The  $N$  (or  $N+1$  for  $k = 2$ ) carry signals from each adder are approximate carry signals.
- III. The sum signals generated in step II are added using the same principle as in step II. Then, the carry signals from each adder are approximate carry signals. The new sum signals are passed to the next stage.
- IV. Repeat step II until only one sum signal is left.
- V. Add "1" to the last adder.

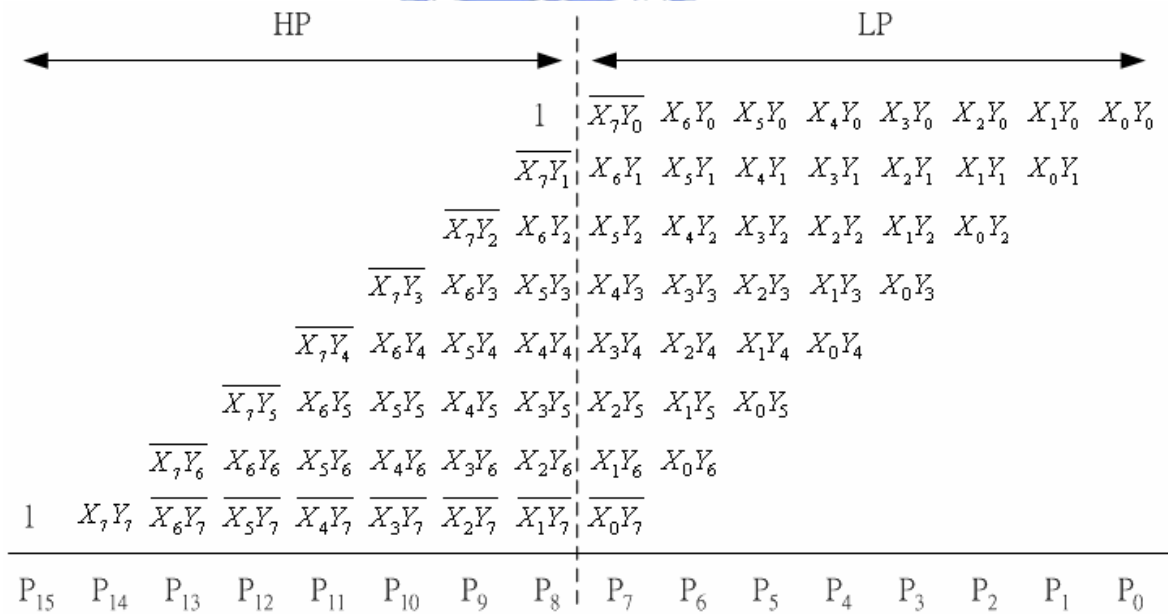
The circuit of  $8 \times 8$  fixed-width multiplier with K-J Cho approach is shown in Fig 2-9. From Fig 2-9, we can find that the adder cells of low part are skipped. The carry from low part to high part is replaced by the approximate carry signals ( $LP\_carry\_0$  and  $LP\_carry\_1$ ) which are generated by Fig 2-8(a).



**Fig 2-9: Fixed-width multiplier with K-J Cho approach for n = 8**

### 2.3 L-D Van Approach

In this section, the fixed-width multiplier proposed by L-D Van will be introduced [10] [11]. The L-D Van approach is based on Baugh-Wooley Array multiplier [12]. Fig 2-10 shows the partial product of 8-bit Baugh-Wooley Array multiplier. It can be divided into two parts, HP and LP, as the same as Booth multiplier.



**Fig 2-10 Partial product of 8-bit Baugh-Wooley Array multiplier**

In general, the carry from low part to high part of Baugh-Wooley Array multiplier can be defined as Equation (2.18). The two elements,  $\beta$  and  $\lambda$ , are represented in equation (2.19) and (2.20), respectively.

$$Carry_{\tau-1} = \left[ \frac{1}{2} \beta + \lambda \right]_r \quad (2.18)$$

$$\beta = \overline{x_{n-1}y_0} + x_{n-2}y_1 + x_{n-3}y_2 + \dots + x_1y_{n-2} + \overline{x_0y_{n-1}} \quad (2.19)$$

$$\lambda = 2^{-2}(x_{n-2}y_0 + x_{n-3}y_1 + \dots + x_0y_{n-2}) + \dots + 2^{-n}x_0y_0 \quad (2.20)$$

Before we introduce L-D Van approach, the terminology,  $\theta_{index,\tau}$ , should be indicated. It signifies the binary value of LP\_major for different values of  $\tau$ , where  $\tau$  means to keep  $(n + \tau)$  most-significant columns of partial product and to truncate the  $(\tau - 1)$ . least-significant columns. The value of  $\theta_{index,\tau}$  is indicated in Equation (2.21) and the binary parameters  $q_{n-1-\tau} q_{n-2-\tau} \dots q_0$  are belong to  $\{0, 1\}$ .

$$\theta_{index,\tau}(q_{n-1-\tau}, q_{n-2-\tau}, \dots, q_0) = \langle x_{n-1-\tau}y_0 \rangle^{q_{n-1-\tau}} + \langle x_{n-2-\tau}y_1 \rangle^{q_{n-2-\tau}} + \dots + \langle x_0y_{n-1-\tau} \rangle^{q_0} \quad (2.21)$$

Equation (2.22) illustrates the operation of  $\langle X \rangle^q$

$$\langle X \rangle^q = \begin{cases} X, & \text{if } q = 0 \\ \bar{X}, & \text{otherwise} \end{cases} \quad (2.22)$$

In which  $\bar{X}$  means the complement of the binary number X. For  $n = 8$ , the 129<sup>th</sup> index under keeping eight columns,  $\theta_{index=129,\tau=0}$ , can be written as

$$\theta_{index=129,\tau=0} = \overline{x_7y_0} + x_6y_1 + x_5y_2 + x_4y_3 + x_3y_4 + x_2y_5 + x_1y_6 + \overline{x_0y_7} \quad (2.23)$$

In the following discussion, two calculated methods of error-compensation bias for  $\tau = 0$  will be explained.

According to the derivation result in [10], equation (2.18) can be rewritten as

$$Carry_{\tau-1} = \theta_{index,\tau=0} + \left[ \frac{1}{2} \beta - \theta_{index,\tau=0} + \lambda \right]_r \quad (2.24)$$

It can be replaced by

$$Carry_{\tau-1} = (\langle x_{n-2}y_1 \rangle^{q_{n-2}} + \langle x_{n-3}y_2 \rangle^{q_{n-3}} + \dots + \langle x_1y_{n-2} \rangle^{q_1}) + [K]_r \quad (2.25)$$

$$K = \langle x_{n-1}y_0 \rangle^{q_{n-1}} + \langle x_0y_{n-1} \rangle^{q_0} + \frac{1}{2}\beta - \theta_{index, \tau=0} + \lambda \quad (2.26)$$

In Equation (2.25), the first term can be easily determined while the index is decided. And the second term,  $[K]_r$ , can be approached by the expected value which can obtain by full search. In order to get more accurate error-compensation bias, two types of carry-estimation formula are proposed. The formulas are shown in Equation (2.27) and (2.28), separately.

$$Carry_{type1} = \begin{cases} (\langle x_{n-2}y_1 \rangle^{q_{n-2}} + \langle x_{n-3}y_2 \rangle^{q_{n-3}} + \dots + \langle x_1y_{n-2} \rangle^{q_1}) + [K_1]_r, & \text{if } \theta_{index} = 0 \\ (\langle x_{n-2}y_1 \rangle^{q_{n-2}} + \langle x_{n-3}y_2 \rangle^{q_{n-3}} + \dots + \langle x_1y_{n-2} \rangle^{q_1}) + [K_2]_r, & \text{if } \theta_{index} > 0 \end{cases} \quad (2.27)$$

$$Carry_{type2} = \begin{cases} (\langle x_{n-2}y_1 \rangle^{q_{n-2}} + \langle x_{n-3}y_2 \rangle^{q_{n-3}} + \dots + \langle x_1y_{n-2} \rangle^{q_1}) + [K_3]_r, & \text{if } \theta_{index} < n \\ (\langle x_{n-2}y_1 \rangle^{q_{n-2}} + \langle x_{n-3}y_2 \rangle^{q_{n-3}} + \dots + \langle x_1y_{n-2} \rangle^{q_1}) + [K_4]_r, & \text{if } \theta_{index} = n \end{cases} \quad (2.28)$$

Where  $K_1$ ,  $K_2$ ,  $K_3$ , and  $K_4$  are the average value of  $K$  for different range of  $\theta_{index}$

By full search simulation, we can get the values of  $K_1$  and  $K_2$  for each index. In order to reduce the complexity of circuit design, to choose the indices which satisfy  $[K_1]_r \in \{0, 1\}$  and  $[K_2]_r \in \{0, 1\}$  is a good idea. For the 6 x 6 multiplier, there are three indices to satisfy the conditions,  $[K_1]_r \in \{0, 1\}$  and  $[K_2]_r \in \{0, 1\}$ . However, these indices do not always satisfy the conditions while the width “n” is changed. In order to find the fixed value of  $K$  for different width “n”, the second approach “Type 2” is proposed. By using exhaustive search simulation generated from  $n = 4$  to  $n = 12$ , we can find that the specific index  $\theta_{index=2^{n-1}+1}$  is satisfy  $[K_3]_r = 1$  and  $[K_4]_r = 0$ . Because the error-compensation bias is shown as Equation (2.25) and

$\theta_{index=2^{n-1}+1} = \overline{x_{n-1}y_0} + x_{n-2}y_1 + \dots + x_1y_{n-2} + \overline{x_0y_{n-1}}$ , it can be described as Equation (2.29) for  $n \leq 12$ .

$$Carry_{Type2, index=2^{n-1}+1} = \begin{cases} (< x_{n-2}y_1 >^{q_{n-2}} + < x_{n-3}y_2 >^{q_{n-3}} + \dots + < x_1y_{n-2} >^{q_1}) + 1, \\ \quad \text{if } \theta_{index=2^{n-1}+1} < n \\ (< x_{n-2}y_1 >^{q_{n-2}} + < x_{n-3}y_2 >^{q_{n-3}} + \dots + < x_1y_{n-2} >^{q_1}), \\ \quad \text{if } \theta_{index=2^{n-1}+1} = n \end{cases} \quad (2.29)$$

To perform the exhaustive simulation for large width “n” will take a lot of time. In the following discussion, “Type 2” approach for large width “n” will be introduced. Two cases of “Type 2” approach,  $\theta_{index=2^{n-1}+1} < n$  and  $\theta_{index=2^{n-1}+1} = n$  will be explained, separately.

**Case 1:**  $\theta_{index=2^{n-1}+1} < n$

We have assumed that the probability of each bit of input data equaling “1” is 1/2. Hence, the value of  $E[x_i y_j]$  and  $E[\overline{x_i y_j}]$  are equal to 1/4 and 3/4. According to the values of  $E[x_i y_j]$  and  $E[\overline{x_i y_j}]$ , the expected value of  $\frac{1}{2}\beta$  can be represented as

$$E\left[\frac{1}{2}\beta\right] = \frac{1}{2} \times \left( \frac{3}{4} + \frac{3}{4} + \frac{1}{4} \times (n-2) \right) \\ = \frac{n}{8} + \frac{1}{2} \quad (2.30)$$

Similarly, the expected value of  $\lambda$  can be shown as

$$E[\lambda] = \frac{1}{2^2} \times \frac{1}{4} \times (n-1) + \frac{1}{2^3} \times \frac{1}{4} \times (n-2) + \dots + \frac{1}{2^n} \times \frac{1}{4} \times 1 \\ = \frac{1}{4} \left( \frac{1}{2^2} \times (n-1) + \frac{1}{2^3} \times (n-2) + \dots + \frac{1}{2^n} + 1 \right) \\ \cong \frac{n}{8} - \frac{1}{4}, \quad \text{if } n \geq 4 \quad (2.31)$$

From equation (2.26), the value of  $[K_3]_r$  for  $index = 2^{n-1} + 1$  is indicated as

$$\begin{aligned}
[K_3]_r &= [E[K]]_r \\
&= \left[ E[x_{n-1}y_0 + x_0y_{n-1} - \frac{1}{2}\beta + \lambda] \right]_r \\
&= \left[ \frac{3}{4} + \frac{3}{4} - \frac{n}{8} - \frac{1}{2} + \frac{n}{8} - \frac{1}{4} \right]_r = 1
\end{aligned} \tag{2.32}$$

Hence, we can obtain the error-compensation bias for large width “n” without using exhaustive search scheme. Equation (2.33) shows the error-compensation bias for  $\theta_{index=2^{n-1}+1} < n$  which is the same as equation (2.29)

$$\begin{aligned}
Carry_{Type2, index=2^{n-1}+1} &= (\langle x_{n-2}y_1 \rangle^{q_{n-2}} + \langle x_{n-3}y_2 \rangle^{q_{n-3}} + \dots + \langle x_1y_{n-2} \rangle^{q_1}) + 1, \\
&\text{if } \theta_{index=2^{n-1}+1} < n
\end{aligned} \tag{2.33}$$

**Case 2:**  $\theta_{index=2^{n-1}+1} = n$

The case  $\theta_{index=2^{n-1}+1} = n$  is met only when  $x_0y_{n-1} = x_{n-1}y_0 = 1$  and  $x_1y_{n-2} = x_2y_{n-3} = \dots = x_{n-2}y_1 = 1$ . So, the expected value of  $\frac{1}{2}\beta$  can be represented as

$$E\left[\frac{1}{2}\beta\right] = \frac{1}{2} \times 1 \times n = \frac{1}{2}n \tag{2.34}$$

And the expected value of  $\lambda$  can be shown as

$$\begin{aligned}
E[\lambda] &= \frac{1}{2^2} \left( \frac{1}{3} \times 1 \times 2 + 1 \times (n-3) \right) + \frac{1}{2^2} \left( \frac{1}{3} \times 1 \times 2 + 1 \times (n-4) \right) \\
&\quad + \dots + \frac{1}{2^{n-1}} \left( \frac{1}{3} \times 1 \times 2 \right) + \frac{1}{2^n} \left( \frac{1}{9} \times 1 \times 1 \right) \\
&= \frac{1}{2}n - \frac{5}{3}, \text{ if } n \geq 4
\end{aligned} \tag{2.35}$$

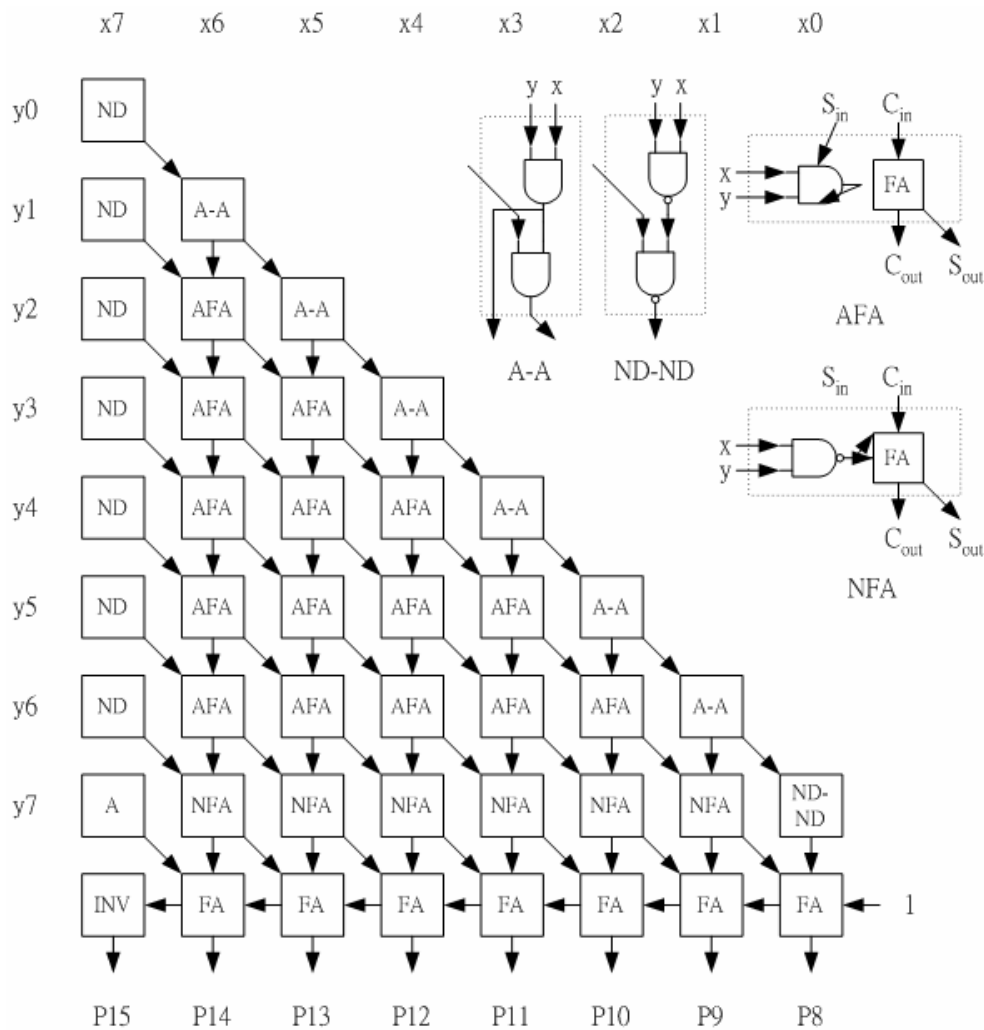
According to Equation (2.34) and (2.35), the value of  $[K_4]_r$  for index =  $2^{n-1} + 1$  is illustrated in Equation (2.36).

$$\begin{aligned}
 [K_4]_r &= [E[K]]_r \\
 &= \left[ E\left[ \overline{x_{n-1}y_0 + x_0y_{n-1}} - \frac{1}{2}\beta + \lambda \right] \right]_r = 0
 \end{aligned}
 \tag{2.36}$$

The error-compensation bias for case 2 is shown in Equation (2.37) which is the same as Equation (2.29)

$$\begin{aligned}
 Carry_{Type2, index=2^{n-1}+1} &= (\langle x_{n-2}y_1 \rangle^{q_{n-2}} + \langle x_{n-3}y_2 \rangle^{q_{n-3}} + \dots + \langle x_1y_{n-2} \rangle^{q_1}), \\
 &\text{if } \theta_{index=2^{n-1}+1} = n
 \end{aligned}
 \tag{2.37}$$

Fig 2-11 shows the circuit of 8-bit fixed-width multiplier with the 129<sup>th</sup> index. The function of A-A cell is to judge whether the value of  $\theta_{index=2^{n-1}+1}$  is equal to n or not.



**Fig 2-11: Fixed-width multiplier with L-D VAN approach for n = 8**

# Chapter3 Proposed Self-Compensation Multiplier

In chapter 2, three existed Fixed-width multiplier are introduced, S-J Jou approach, K-J Cho approach, and L-D Van approach. S-J Jou approach and K-J Cho approach are based on Booth multiplier and the L-D Van approach is based on Baugh-Wooley Array multiplier. In this chapter, a new approach of fixed-width multiplier which is based on Booth multiplier will be introduced. The error-compensation bias of proposed approach is produced by the carry-estimation equations. In order to reduce the truncation error, the carry-estimation equations are adapted for  $n = 8$  to  $n = 16$ . These equations for different  $n$  can be analyzed by few logic gates. Hence, the circuit complexity of proposed approach is closed to direct-truncation approach.

The error-compensation bias is introduced in section 3.1 and the circuit of proposed structure is illustrated in section 3.2. Finally, the comparison of performance for each approach is shown in section 3.3.

## 3.1 Calculation of Error-compensation bias

Fig 3-1 shows the partial product of 8-bit Booth multiplier and the partial product is divided into two parts, low part (LP) and high part (HP). The LP can be further divided into two parts, the first column of LP is LP\_major and the remaining columns of LP are LP\_minor. The carry-out bit from LP to HP is written as equation (3.1), where  $\beta$  the sum of LP\_major and  $\lambda$  is the sum of LP\_minor.

$$Carry_7 = \left\lfloor \frac{1}{2} \beta + \lambda \right\rfloor \quad (3.1)$$



$$\beta = P_{0\_7} + P_{1\_5} + P_{2\_3} + P_{3\_1} \quad (3.2)$$

$$\begin{aligned} \lambda = & 2^{-2}(P_{0\_6} + P_{1\_4} + P_{2\_2} + P_{3\_0} + n_3) + 2^{-3}(P_{0\_5} + P_{1\_3} + P_{2\_1}) \\ & + 2^{-4}(P_{0\_4} + P_{1\_2} + P_{2\_0} + n_2) + 2^{-5}(P_{0\_3} + P_{1\_1}) \\ & + 2^{-6}(P_{0\_2} + P_{1\_0} + n_1) + 2^{-7}(P_{0\_1}) \\ & + 2^{-8}(P_{0\_0} + n_0) \end{aligned} \quad (3.3)$$

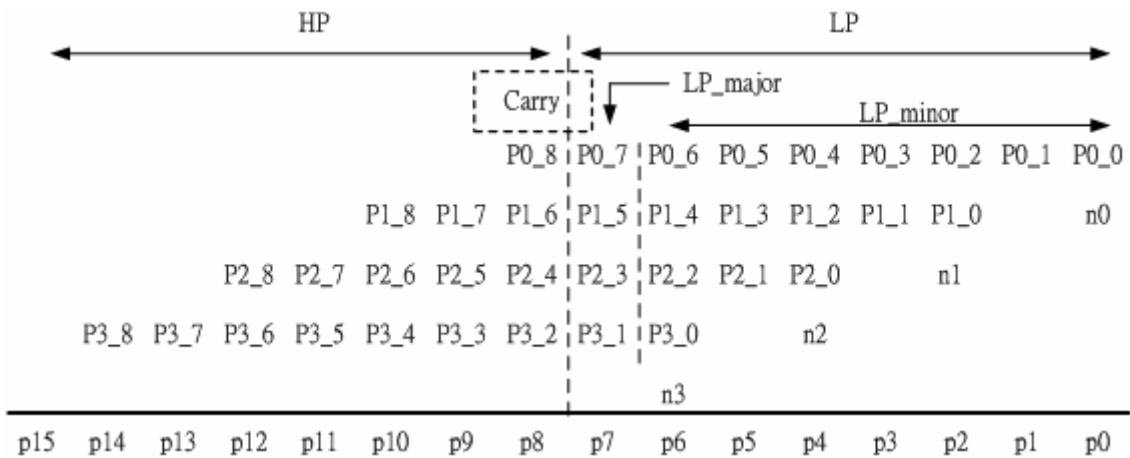
In order to find the error-compensation bias, LP\_major index,  $\theta$ , is defined as equation follow.

$$\theta = P_{0\_7} + 2^1 \cdot P_{1\_5} + 2^2 \cdot P_{2\_3} + 2^3 \cdot P_{3\_1} \quad (3.4)$$

the values of  $\theta$  for different values of LP\_major are shown in Table 3-1.

**Table 3-1: The values of  $\theta$  for different values of  $P_{3\_0}P_{2\_2}P_{1\_4}P_{0\_6}$**

$P_{3\_1}$	$P_{2\_3}$	$P_{1\_5}$	$P_{0\_7}$	$\theta$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15



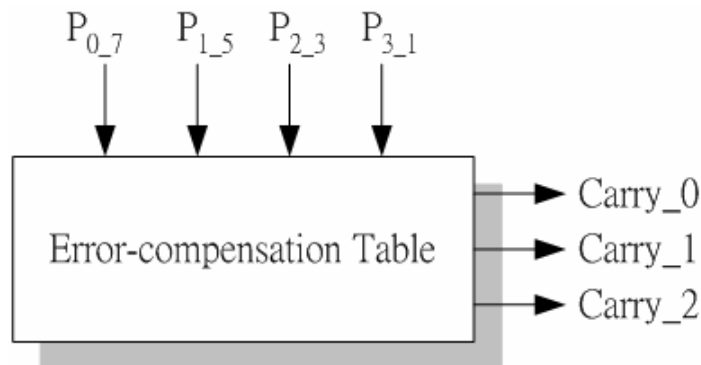
**Fig 3-1: Partial product of 8-bit Booth multiplier**

It is obviously that if one value of  $\theta$  is selected, the value of  $\beta$  is fixed. For example, if the value of  $\theta$  is equal to 7, the value of  $\beta$  must be equal to 3. We can obtain the average carry for each value of  $\theta$  by using full search simulation. Because the value of  $\beta$  is fixed, we only need to calculate the average carry from LP\_minor to HP ( $\lambda$ ). The results of full search simulation for  $n = 8$  are represented in Table 3-2.

**Table 3-2: The average carry for each value of  $\theta$**

$\theta$	Average Carry
0	1
1	1
2	1
3	2
4	1
5	2
6	2
7	2
8	1
9	2
10	2
11	2
12	2
13	2
14	2
15	3

Now, we can get the error-compensation bias by using look-up table method. The input of table is the four elements of LP\_major and three approximate carry signals are needed to represent the values of error-compensation bias. The block diagram of error-compensation bias is illustrated in Fig 3-2.




**Fig 3-2: The error-compensation table for  $n = 8$**

As mentioned in former discussion, the values of approximate carry signals are obtained by looking up table. But the area of error-compensation table will increase when n is large. Hence, it is unwise to obtain the error-compensation bias by this method. The new approach should be proposed to improve the hardware complexity of look-up table method.

In order to find the new approach, we try to find the rule between average carry and  $\theta$ . Fortunately, the average carry is related to the number of “1” in LP\_major which is the value of  $\beta$ . If the values of  $\theta$  have the same number of “1” in LP\_major, the values of average carry for these  $\theta$  will be all equally. For example, the values of average carry are all equal to 1 for  $\theta = 1, 2, 4, \text{ and } 8$  in which the number of “1” in LP\_major is all equal to 1. The relation between average carry and the number of “1” in LP\_major is shown in Table 3-3 for  $n=8$ .

**Table 3-3: The values of Average carry for each  $\beta$  for  $n = 8$**



$\beta$	$\theta$	Average carry
0	0	1
1	1, 2, 4, 8	1
2	3, 5, 6, 9, 10, 12	2
3	7, 11, 13, 14	2
4	15	3

From Table 3-3, we can derive a carry-estimation equation which can computed the average carry for  $n = 8$ . The carry-estimation equation is written as

$$Carry_7 = \left\lfloor \frac{\beta}{2} \right\rfloor + 1 \quad (3.5)$$

The results of this equation are the error-compensation bias for  $n = 8$ . Table 3-4 shows the results of carry-estimation equation for  $n = 8$ .

**Table 3-4: The results of carry-estimation equation for n = 8**

$\beta$	0	1	2	3	4
Average Carry	1	1	2	2	3
$C_{carry_7} = \left\lfloor \frac{\beta}{2} \right\rfloor + 1$	1	1	2	2	3

As seen in Table 3-4, the results of carry-estimation equation are all equal to the values of average carry which are obtain by full search simulation. Hence, we can calculate the error-compensation bias by equation (3.5). The carry-estimation equations for different width “n” can be obtained by similarly process. Equation (3.6) and equation (3.7) show the carry-estimation equations for n = 10 to n = 16. And the results of these carry-calculate equations are represented in Table 3-5.

$$Carry_{n/2-1} = \left\lfloor \frac{\beta+1}{2} \right\rfloor + 1, \text{ for } n = 10, 12, \text{ and } 14 \quad (3.6)$$

$$Carry_{15} = \left\lfloor \frac{\beta}{2} \right\rfloor + 2, \text{ for } n = 16 \quad (3.7)$$

However, the values of carry-estimation equations do not always match the values of average carry. For example, the result of carry-estimation equation is equal to 2 for n = 10 and  $\beta = 1$ . But the value of average carry is equal to 1. In order to derive the influence of the truncation error caused by this inequality situation, the probability of the mismatch situation is calculated. First, the number of cases for each value of  $\beta$  is computed. Equation (3.8) shows the equation which can calculate the number of cases for each  $\beta$ .

$$N_{\beta} = C_{\beta}^{n/2} \quad (3.8)$$

**Table 3-5: The results of carry-estimation equation for  
(a) n = 10 (b) n = 12 (c) n = 14 (d) n = 16**

$\beta$	0	1	2	3	4	5
Average Carry	1	1	2	3	3	4
$Carry_9 = \lfloor \frac{\beta+1}{2} \rfloor + 1$	1	2	2	3	3	4

(a)

$\beta$	0	1	2	3	4	5	6
Average Carry	1	2	2	3	3	4	4
$Carry_{11} = \lfloor \frac{\beta+1}{2} \rfloor + 1$	1	2	2	3	3	4	4

(b)

$\beta$	0	1	2	3	4	5	6	7
Average Carry	1	2	2	3	3	4	5	5
$Carry_{13} = \lfloor \frac{\beta+1}{2} \rfloor + 1$	1	2	2	3	3	4	4	5

(c)

$\beta$	0	1	2	3	4	5	6	7	8
Average Carry	1	2	2	3	4	4	5	5	6
$Carry_{15} = \lfloor \frac{\beta}{2} \rfloor + 2$	2	2	3	3	4	4	5	5	6

(d)

Table 3-6 shows the numbers of cases for each  $\beta$  with  $n = 10$ . It is obviously that, the probability of  $\beta$  equaling “1” is only  $\frac{5}{32}$ . In other words, the probability of the mismatch condition is only  $\frac{5}{32}$  (15%) for  $n = 10$ . Thus, the increase of the truncation error caused by the mismatch situation will be few.

**Table 3-6: The numbers of cases for each  $\beta$  with  $n = 10$**

$\beta$	0	1	2	3	4	5
# of cases	1	5	10	10	5	1

Similarly, the mismatch condition occurs for  $\beta = 6$  and  $n = 14$ . We can also find the number of cases for each  $\beta$  by using equation (3.8). Table 3-7 shows the numbers of cases for each  $\beta$  with  $n = 14$ .

In Table 3-7, we can see that the probability of  $\beta$  equaling “6” is only  $\frac{7}{128}$ . That is, the probability of the mismatch condition is only  $\frac{7}{128}$  (5.4%). Hence, the increase of the truncation error caused by inequality condition is also very little for  $n = 14$ .

**Table 3-7: The numbers of cases for each  $\beta$  with  $n = 14$**

$\beta$	0	1	2	3	4	5	6	7
# of cases	1	7	21	35	35	21	7	1

The inequality situation is also occurred for  $n = 16$ . There are two conditions,  $\beta = 0$  and  $\beta = 2$ , that the results of carry-estimation equation are not equal to average carry. By using equation (3.8), the numbers of cases for each  $\beta$  with  $n = 16$  can be obtained from Table 3-8.

**Table 3-8: The numbers of cases for each  $\beta$  with  $n = 16$**

$\beta$	0	1	2	3	4	5	6	7	8
# of cases	1	8	28	56	70	56	28	8	1

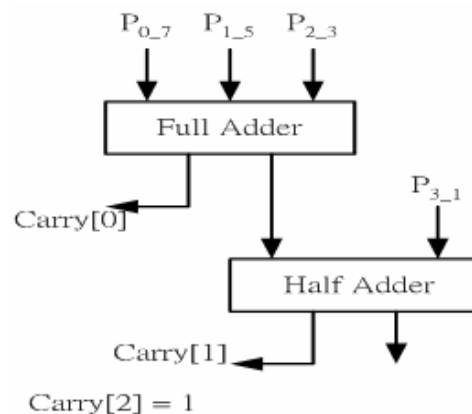
The probability of the inequality condition is only 11%. Hence, the increase of the truncation error for  $n = 16$  is still very little.

In former discussion, we can know that the error-compensation bias can be obtained by using the carry-estimation equations for  $n = 8$  to  $n = 16$ . And the results of the

carry-estimation equations are almost the same as average carry which is obtained by full search simulation.

### 3.2 Proposed Structure

In this section, the circuits of carry-estimation equations for  $n = 8$  to  $n = 16$  are introduced. The carry-estimation equation for  $n = 8$  is shown in equation (3.5) which is composed by  $\lfloor \beta/2 \rfloor$  and “plus 1”. The function of  $\lfloor \beta/2 \rfloor$  can be calculated by the carry of LP\_major. The process of calculation of  $\lfloor \beta/2 \rfloor$  is shown as follow. First, the elements of LP\_major are summed by some full adders and half adders. The carry signals from each adder are the carry from LP\_major to HP, and the sum signals are added. The new carry signals from the added sum signals are also the carry from LP\_major to HP. Repeat former steps until only one sum signal is left. As regards the function of plus “1”, we only need to assign the third carry signal “Carry[2]” equal to 1 for  $n = 8$ . Fig 3-3 illustrates the circuit of carry-estimation equation for  $n = 8$ .

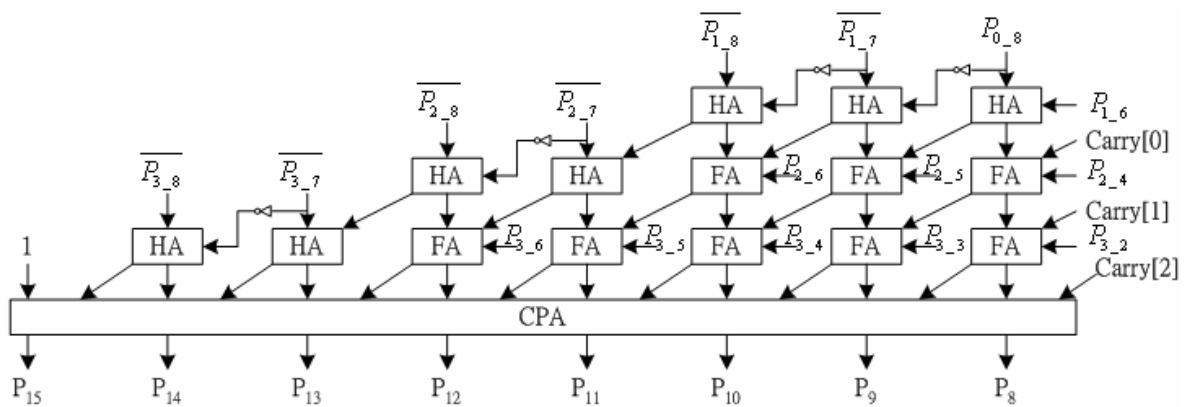


**Fig 3-3: Circuit of carry-estimation equation for  $n = 8$**

As seen in Fig 3-3, the circuit only needs one full adder and one half adder. Thus, the penalty of area for computed error-compensation bias is very small.

Fig 3-4 illustrates the circuit of 8-bit Booth multiplier with proposed approach. In Fig 3-4,

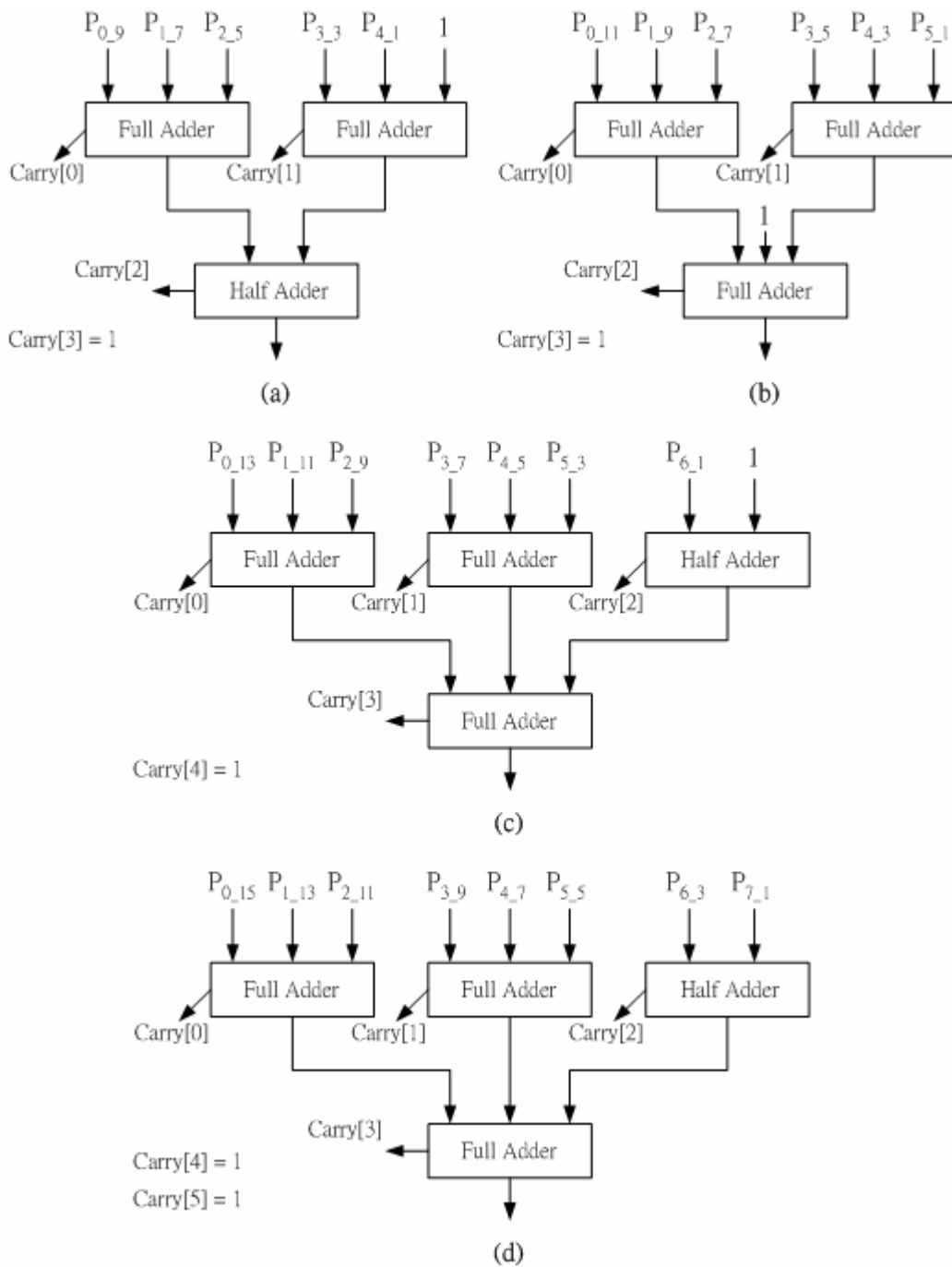
the adder cells required for LP are all omitted, and the carry-out from LP to HP is estimated by equation (3-5) which corresponds to Carry[0], Carry[1], and Carry[2] in Fig 3-4. There are eight full adders, seven half adders, and one 8-bit CPA in this multiplier. The ratio of area between the circuit of carry-estimation equation and the circuit of 8 x 8 booth multiplier is very small. Thus, the area of proposed multiplier is only a bit larger than the area of direct-truncated multiplier.



**Fig 3-4: Circuit of 8-bit Booth multiplier with proposed approach**

Fig 3-5 shows the circuits of carry-estimation equations for  $n = 10$  to  $n = 16$ . We can see that the required adders do not increase much as the width “ $n$ ” increases. The required adders are three full adders for  $n = 12$ . Even for  $n = 16$ , the required adders are only four full adders which are one more full adder than the condition of  $n = 12$ . According to the few required adders for large width “ $n$ ”, the influence of the increased area caused by the circuit of carry-estimation equation can be skipped for large width “ $n$ ”. Thus, the ratio of area between proposed multiplier and direct-truncated multiplier is almost equal to “1” for large width “ $n$ ”. That is, the area of proposed approach is close to the area of direct-truncated multiplier.





**Fig 3-5: Circuits of carry-estimation equations for (a)  $n = 10$  (b)  $n = 12$  (c)  $n = 14$  (d)  $n = 16$**

### 3.3 Performance Analysis

The performance of different approaches will be represented in this section. The performance is evaluated in terms of average error, the variance of errors, and gate count. Note that the comparison of gate counts only contains the approach based on Booth multiplier. However, the approach proposed by L-D Van [11] is based on Baugh-Wooley Array multiplier, it does not include in the comparison of gate counts.

The absolute error between the standard Booth multiplier and fixed-width multiplier is defined as

$$\varepsilon = |P_{\text{Standard}} - P_{\text{Fixed-width}}| \quad (3.9)$$

where  $P_{\text{standard}}$  represents the computational result of standard Booth multiplier and  $P_{\text{Fixed-width}}$  represents the result of fixed-width multiplier.

The average error is defined as equation (3.10) where  $E[x]$  is the expected value of  $x$ .

$$\bar{\varepsilon} = E[\varepsilon] \quad (3.10)$$

Besides the comparison of average error, the variance of error for each approach is compared, too. The computation of the variance of error is described as equation (3.11).

$$\nu = E[(\varepsilon - \bar{\varepsilon})^2] \quad (3.11)$$

It is obvious that the fixed-width multiplier with smaller truncation error has more accurate results. Similarly, the approach with smaller variance of error has stable results.

The comparison of average error and the variance of error are represented in Table 3-9 and Table 3-10, respectively. And the comparison of gate counts is shown in Table 3-11. From Table 3-9, the error of proposed approach is only 11% of direct-truncated multiplier for  $n = 16$ . Besides average error, the variances of our multiplier are only 4.9% of direct-truncated multiplier. It means that the proposed approach is more stable than direct-truncated multiplier.

The comparisons of gate counts are based on Booth multiplier. The gate count of proposed approach is a bit larger than S-J Jou approach but is smaller than K-J Cho approach. For large width “n”, the gate count of our approach is very close to S-J Jou approach, because the number of adder for carry-estimation equation is very small.

In conclusion, the proposed approach has three features:

- I. Low average error
- II. More stable than other approach
- III. Area efficient

**Table 3-9: Comparison results of average error**

	n = 8	n = 10	n = 12	n = 16
Direct-Truncated	100% (576.25)	100% (2816.35)	100% (13312.34)	100% (278525)
S-J Jou Approach	18.59%	16.94%	15.65%	13.76%
K-J Cho Approach	14.79%	12.50%	11.00%	11.31%
L-D Van Approach	18.22%	16.20%	14.33%	12.41%
Proposed Approach	15.14%	16.26%	12.41%	11.22%

**Table 3-10: Comparison results of variance of error**

	n = 8	n = 10	n = 12	n = 16
Direct-Truncated	100% (54284)	100% (1172842)	100% (23650942)	100% (8561689532)
S-J Jou Approach	11.67%	10.70%	10.15%	8.37%
K-J Cho Approach	7.02%	5.53%	4.81%	4.72%
L-D Van Approach	11.11%	9.56%	8.66%	7.65%
Proposed Approach	7.35%	9.05%	6.10%	4.89%

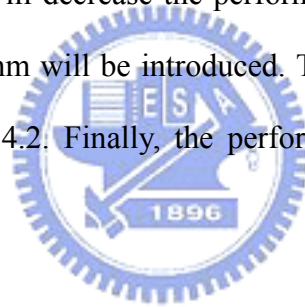


**Table 3-11: Comparison results of gate counts**

	n = 8	n = 10	n = 12	n = 16
Booth Multiplier	100% (671)	100% (925)	100% (1681)	100% (2948)
S-J Jou Approach	52.75%	61.33%	50.70%	59.74%
K-J Cho Approach	62.20%	72.58%	55.05%	70.42%
Proposed Approach	60.24%	62.62%	54.94%	60.72%

# Chapter4 Application of Fixed-Width Multipliers in FFT

The low-error area efficient fixed-width multiplier is proposed in chapter 3. The average error of proposed structure is only about 15% of direct-truncated multiplier. As well as the area of proposed one is only about 60% of standard Booth multiplier. In order to reduce the hardware complexity, the multiplication operations in FFT usually have the fixed-width property. Thus, the proposed fixed-width multiplier is employed in the 128-point FFT architecture [13]. The truncation errors will be introduced because of the usage of the fixed-width multiplier and it will decrease the performance of the 128-point FFT. In section 4.1, the 128-point FFT algorithm will be introduced. Then the architecture of 128-point FFT will be introduced in section 4.2. Finally, the performance of 128-point FFT is shown in section 4.3.



## 4.1 Introduction to 128-Point FFT

In this section, the 128-point FFT algorithm proposed by Y-W Lin [13] will be introduced. Given a sequence  $x(n)$ , the  $N$ -point DFT is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} \quad (k = 0, 1, \dots, N-1) \quad (4.1)$$

Where  $x(n)$  and  $X(k)$  are complex numbers. And the values of  $W_N^{kn}$  is

$$W_N^{nk} = \cos(2\pi nk/N) - j \sin(2\pi nk/N) \quad (4.2)$$

In equation (4.1) the computational complexity is  $O(N^2)$  through directly performing the

required computation. The computational complexity can be reduced to  $O(N \log_r^N)$  by using the radix-r FFT algorithm. In general, higher-radix FFT algorithm has less number of complex multiplications while compared with radix-2 FFT algorithm. Hence, the radix-8 FFT algorithm is employed in the 128-point FFT. But the 128-point FFT is not the power of 8, the mixed-radix FFT algorithm which include radix-2 FFT and radix-8 FFT algorithm should be chosen. The mixed-radix 128-point FFT algorithm is derived as below.

First, let the constant in equation (4.1) as

$$\begin{aligned}
 N &= 128 \\
 n &= 64n_1 + n_2 \quad \begin{cases} n_1 = 0, 1 \\ n_2 = 0, 1, \dots, 63 \end{cases} \\
 k &= k_1 + 2k_2 \quad \begin{cases} k_1 = 0, 1 \\ k_2 = 0, 1, \dots, 63 \end{cases}
 \end{aligned} \tag{4.3}$$

Then, equation (4.1) can be rewritten as

$$\begin{aligned}
 X(2k_2 + k_1) &= \sum_{n_2=0}^{63} \sum_{n_1=0}^1 x(64n_1 + n_2) W_{128}^{(64n_1 + n_2)(2k_2 + k_1)} \\
 &= \sum_{n_2=0}^{63} \underbrace{\left\{ \sum_{n_1=0}^1 x(64n_1 + n_2) W_2^{n_1 k_1} \right\}}_{2 \text{ point DFT}} \underbrace{\left\{ W_{128}^{n_2 k_1} \right\}}_{\text{twiddle factor}} \underbrace{\left\{ W_{64}^{n_2 k_2} \right\}}_{64 \text{ point DFT}} \\
 &= \sum_{n_2=0}^{63} BU_2(k_1, n_2) W_{64}^{n_2 k_2}
 \end{aligned} \tag{4.4}$$

In equation (4.4), the 128-point DFT can be considered as a two-dimensional DFT, 2-point DFT and 64-point DFT. The inputs of 128-point DFT are computed by radix-2 FFT algorithm at first. Then, the results of radix-2 FFT are multiplied by twiddle factor. Finally, the results of multiplication should be calculated by 64-point DFT algorithm which can be decomposed into 8-point DFT recursively 2 times. In order to derive the 64-point FFT algorithm by using radix-2<sup>3</sup> FFT algorithm, the constant  $n_2$  and  $k_2$  in equation (4.3) can be defined as

$$\begin{aligned} n_2 &= 32\alpha_1 + 16\alpha_2 + 8\alpha_3 + \alpha_4, & \alpha_1, \alpha_2, \alpha_3 &= 0, 1; \alpha_4 = 0, 1, \dots, 7 \\ k_2 &= \beta_1 + 2\beta_2 + 4\beta_3 + 8\beta_4, & \beta_1, \beta_2, \beta_3 &= 0, 1; \beta_4 = 0, 1, \dots, 7 \end{aligned} \quad (4.5)$$

Using equation (4.5), equation (4.4) can be rewritten as

$$\begin{aligned} X(2(\beta_1 + 2\beta_2 + 4\beta_3 + 8\beta_4) + k_1) &= \\ \sum_{\alpha_4=0}^7 \sum_{\alpha_3=0}^1 \sum_{\alpha_2=0}^1 \sum_{\alpha_1=0}^1 BU_2(k_1, 32\alpha_1 + 16\alpha_2 + 8\alpha_3 + \alpha_4) & \\ \times W_{64}^{(32\alpha_1 + 16\alpha_2 + 8\alpha_3 + \alpha_4)(\beta_1 + 2\beta_2 + 4\beta_3 + 8\beta_4)} & \end{aligned} \quad (4.6)$$

Where the twiddle factor can be decomposed as

$$\begin{aligned} W_{64}^{(32\alpha_1 + 16\alpha_2 + 8\alpha_3 + \alpha_4)(\beta_1 + 2\beta_2 + 4\beta_3 + 8\beta_4)} &= \\ W_2^{\alpha_1\beta_1} W_4^{\alpha_2\beta_1} W_2^{\alpha_2\beta_2} W_8^{\alpha_3(\beta_1 + 2\beta_2)} W_2^{\alpha_3\beta_3} W_{64}^{\alpha_4(\beta_1 + 2\beta_2 + 4\beta_3)} W_8^{\alpha_4\beta_4} & \end{aligned} \quad (4.7)$$

Thus, equation (4.6) becomes

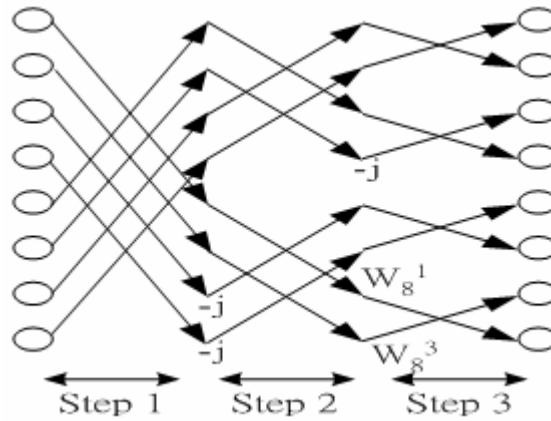
$$X(2(\beta_1 + 2\beta_2 + 4\beta_3 + 8\beta_4) + k_1) = \sum_{\alpha_4=0}^7 BU_8(k_1, \beta_1, \beta_2, \beta_3, \alpha_4) W_8^{\alpha_4\beta_4} \quad (4.8)$$

Where

$$\begin{aligned} BU_8(k_1, \beta_1, \beta_2, \beta_3, \alpha_4) &= \\ \sum_{\alpha_3=0}^1 \sum_{\alpha_2=0}^1 \sum_{\alpha_1=0}^1 \left\{ \underbrace{BU_2(k_1, \alpha_1, \alpha_2, \alpha_3, \alpha_4)}_{1st \ step} \underbrace{W_2^{\alpha_1\beta_1} W_4^{\alpha_2\beta_1} W_2^{\alpha_2\beta_2} W_8^{\alpha_3(\beta_1 + 2\beta_2)}}_{2nd \ step} \underbrace{W_2^{\alpha_3\beta_3} W_{64}^{\alpha_4(\beta_1 + 2\beta_2 + 4\beta_3)}}_{3rd \ step} \right\} & \end{aligned} \quad (4.9)$$

In equation (4.9), the 8-point DFT are divided into three steps by using radix-2 index map. Fig 4-1 shows the signal flow graph of the radix-8 FFT algorithm. In which, the radix-8 algorithm is decomposed into three steps. Each step has four butterfly operations. After the butterfly operations, the multiplications of twiddle factors in each step should be performed. There are only three twiddle factors,  $-j$ ,  $W_8^1$ , and  $W_8^3$  in radix-8 algorithm. The multiplication of “-j” only needs to exchange the real part with imaginary part. Thus, it does not need any multiplier. The multiplications of the twiddle factors,  $W_8^1$  and  $W_8^3$ , can be

replaced by some additions. Because the twiddle factors can be written as  $\sqrt{2}/2(1-j)$  and  $-(\sqrt{2}/2(1-j))$ , respectively. The value of  $\sqrt{2}/2$  is equal to 0.70710678 which can be written as  $2^{-1}+2^{-3}+2^{-4}+2^{-6}+2^{-8}$  can be complemented only by five shifters and four adders .



**Fig 4-1: The signal flow graph of radix-8 FFT algorithm**

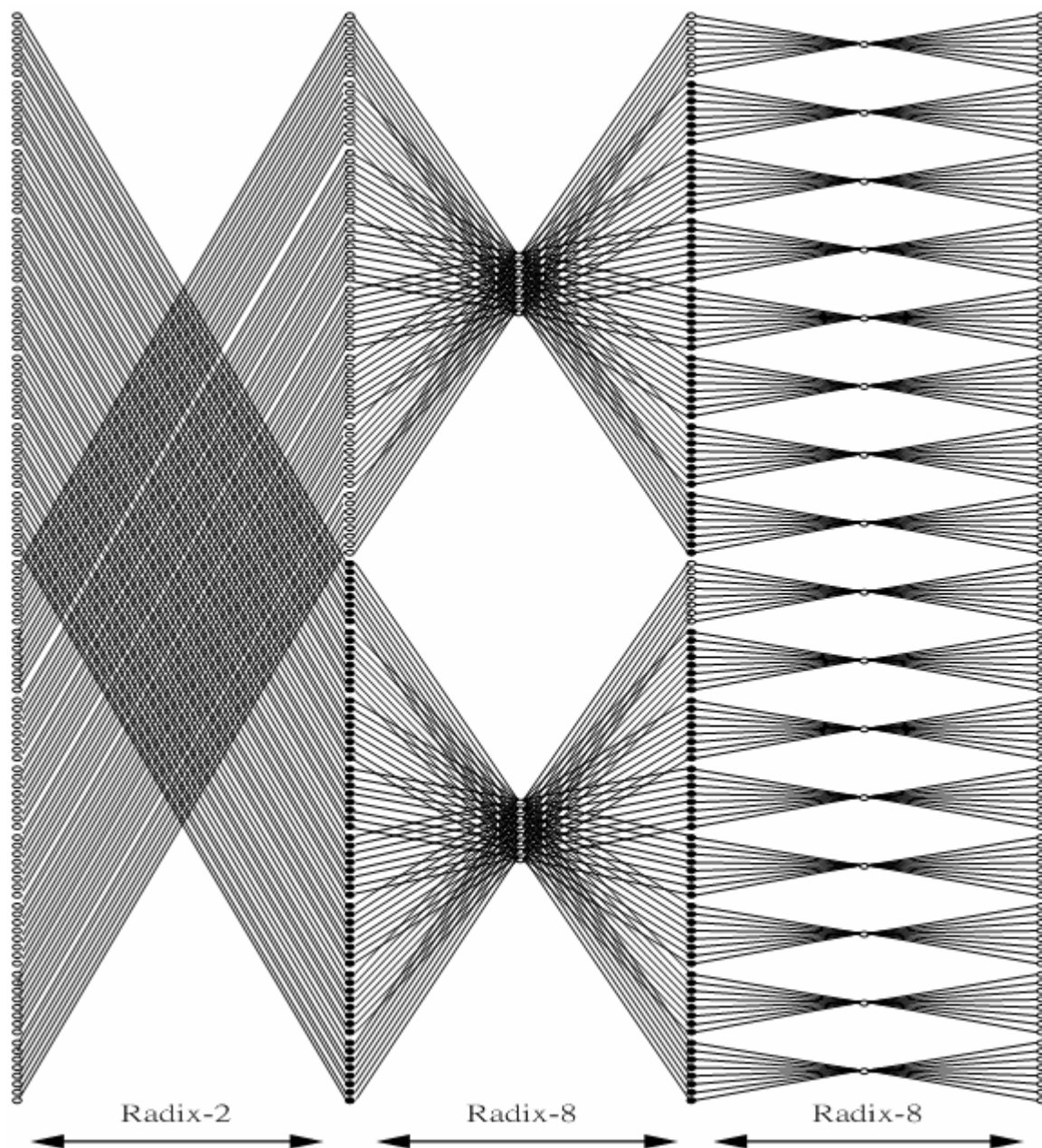
The signal flow graph of 128-point mixed-radix FFT algorithm is shown as Fig 4-2. In which, the 128-point FFT is composed by three stage. The first stage is performed by radix-2 FFT algorithm and the radix-8 FFT algorithm shown in Fig 4-1 is employed in the second and third stages. The black point in each stage means that one twiddle factor will be multiplied at that point. In the first stage, there are sixty-four butterfly units and the two inputs of  $i_{th}$  butterfly unit are  $i_{th}$  and  $(64+i)_{th}$  input data where  $i = 0 \sim 63$ . Then the results of first stage should be calculated by the second and third stages. There are sixteen radix-8 FFT units in the second and third stages, respectively. The orders of radix-8 FFT inputs are different in each stage. In the second stage, the inputs of each radix-8 FFT unit are shown as below

$$\left\{ \begin{array}{l} (64i + j)_{th}, (64i + j + 8)_{th}, (64i + j + 16)_{th}, (64i + j + 24)_{th}, \\ (64i + j + 32)_{th}, (64i + j + 40)_{th}, (64i + j + 48)_{th}, (64i + j + 56)_{th} \end{array} \right\} \quad (4.10)$$

$i = 0, 1; j = 0, 1, \dots, 7; \quad \text{for } (8i + j)_{th} \text{ radix-8 FFT unit}$

But in the third stage, the eight inputs are  $(8i)_{th} \sim (8i + 7)_{th}$  input data for  $i = 0 \sim 15$ .





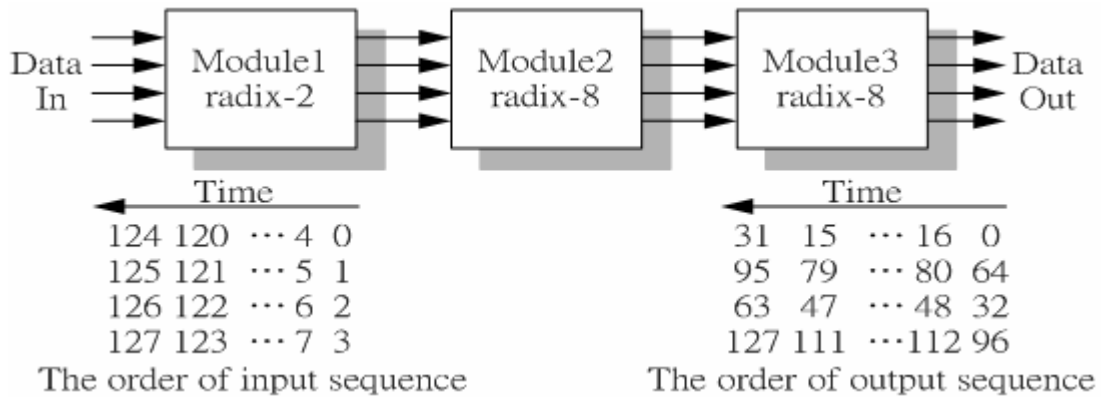
**Fig 4-2: The signal flow graph of 128-point mixed-radix FFT algorithm**

## 4.2 128-Point FFT Architecture

In order to reduce the area of 128-point FFT, the proposed multiplier is employed in the 128-point FFT architecture. The 128-point FFT architecture proposed by Y-W Lin's [13] is introduced in this section.

Fig 4-3 shows the 128-point FFT architecture which is divided into three modules. The first

module is complemented by radix-2 FFT algorithm, and the radix-8 FFT algorithm is used in the second and the third modules. In this architecture, the high throughput rate is achieved by using four parallel data paths; the order of the output sequence is the bit reversal of the order of the input sequence as seen in Fig 4-3.

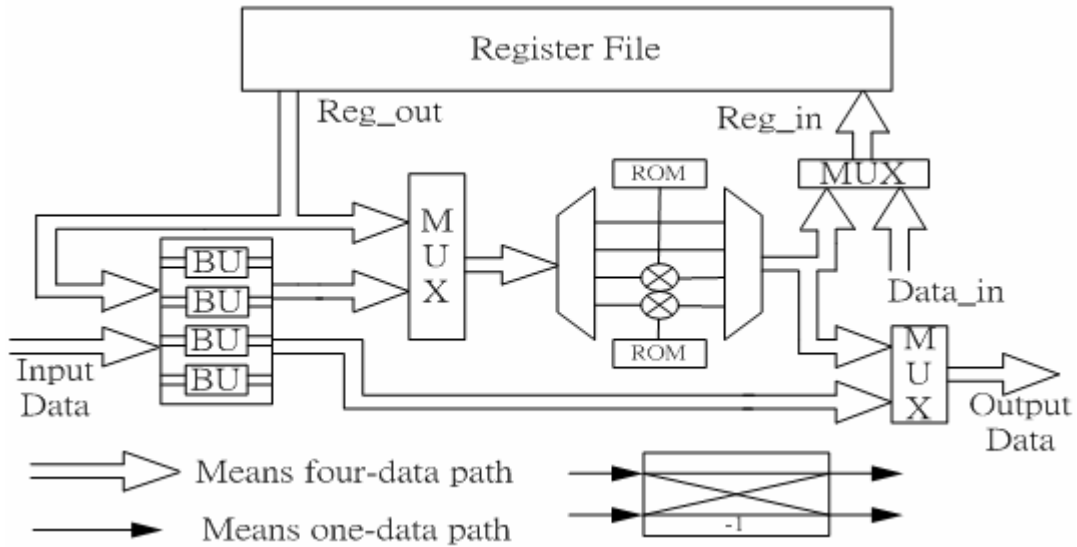


**Fig 4-3: Block diagram of 128-point FFT**

#### 4.2.1 Module 1

Fig 4-4 shows the architecture of Module 1 which consists of 128 registers which can store 64 complex data, four two-input butterfly units (BU), two complex multipliers, and two ROMs. The ROMs are used to store the twiddle factors. The 128 registers are used to store inputs data and the outputs of BU. The operations of BU are the complex addition and the complex subtraction from two input data. Because the two inputs of each BU are  $in(i)$  and  $in(64+i)$  where  $i$  is from 0 to 63. This corresponds to the first stage of Fig 4-2. The order of four parallel input sequences in Module 1 is  $in(4m)$ ,  $in(4m+1)$ ,  $in(4m+2)$ , and  $in(4m+3)$  where  $m$  is from 0 to 31. Thus, the 64 input data at first 16 cycles should be stored in the register file. At next 16 cycles, the eight inputs of the four BU are received from the register file and the inputs data, respectively. Then eight outputs data are generated by the four BU. The four outputs of the complex addition are sent to the Module 2 directly, and the other four outputs of complex subtraction are stored in the register file. Before the four outputs are stored, two of them are multiplied by twiddle factors. After 32 cycles, the other two outputs are multiplied

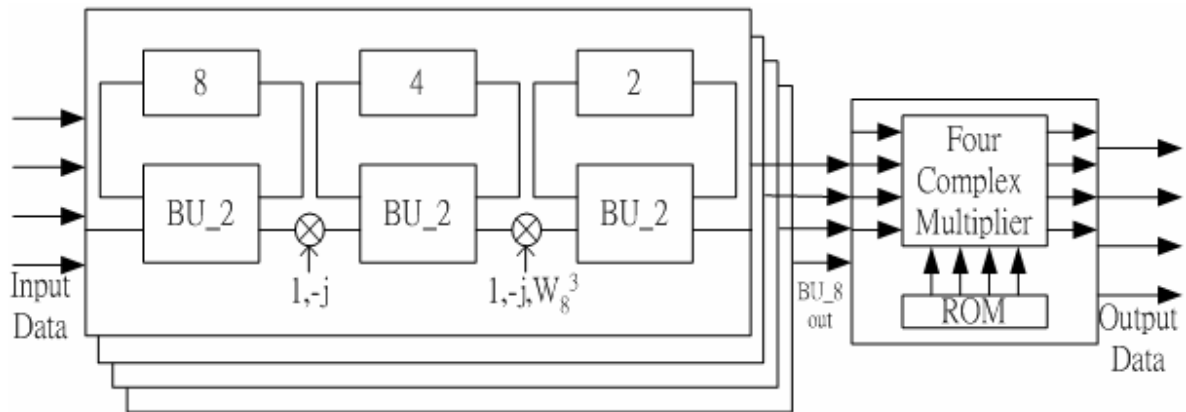
by twiddle factors. Then, the four outputs are fed into the Module 2. By this multiplication approach can not only reduce the four complex multipliers to two complex multipliers but also achieve 100% utilization of the complex multipliers.



**Fig 4-4: Block diagram of the Module 1**

#### 4.2.2 Module 2

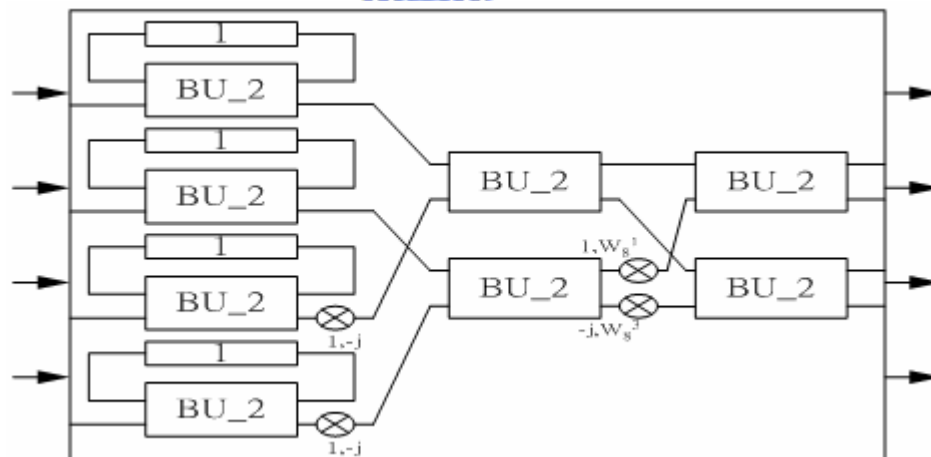
The block diagram of the Module 2 is illustrated in Fig 4-5. It consists of four BU\_8 structures and four complex multipliers. The architecture of BU\_8 is directly mapped from 3-step radix-8 FFT algorithm as seen in Fig 4-1. And the numbers of registers in each step are eight, four, and two, respectively. These registers are used to store the input of two-input BU until the other available input is received. The outputs of two-input BU in first and second steps should be multiplied by the twiddle factors,  $1$ ,  $-j$ ,  $W_8^1$ , and  $W_8^3$ . As mentioned in Section 4.1, the multiplications of these twiddle factors can be implemented without any multipliers. But the four outputs of BU\_8 need to be multiplied by the nontrivial twiddle factors with four complex multipliers.



**Fig 4-5: Block diagram of the Module 2**

### 4.2.3 Module 3

The Module 3 is also realized by radix-8 FFT algorithm. Fig 4-6 shows the block diagram of the Module 3. The structure of the Module 3 is different from that of Module 2, because the orders of input data of the Module 2 and the Module 3 are different. The structure should be adapted for the different orders of output as shown in Fig 4-6. The outputs data in first and second steps only need to be multiplied by the twiddle factors,  $1, -j, W_8^1$ , and  $W_8^3$ . Thus, no any multiplier is used in the Module 3.



**Fig 4-6: Block diagram of the Module 3**

### 4.3 Simulation Results

The algorithm and architecture of 128-point FFT are introduced in section 4.1 and section 4.2, respectively. In this section, the performance of 128-point FFT will be represented. Table 4-1 shows the SQNR of different 128-point FFT approaches. The first column indicates what kinds of multipliers are used in the 128-point FFT architecture. “Booth” means the traditional Booth multiplier; “Direct\_t” means the direct-truncated multiplier; and “Proposed” means the 128-point FFT with proposed multiplier. The first row shows the width of twiddle factors and input data. Because there are 2 bits right of decimal point of twiddle factors. The truncation-bit is only  $n - 2$  bits. For example, if the width of twiddle factors is 10, the truncation-bit is only 8 bits which are left of decimal point. In Table 4-1, the SQNR of proposed approach is only 1dB less than the traditional Booth multiplier. But it is about 10dB larger than the direct-truncated multiplier.

**Table 4-1: SQNR of 128-point FFT with different multiplier approach**

	18	16	14	12	10
Booth	80.43dB	68.38dB	56.33dB	44.33dB	33.33dB
Direct_t	68.29dB	57.27dB	46.82dB	35.55dB	24.24dB
Proposed	79.09dB	67.79dB	55.52dB	43.04dB	32.40dB

Table 4-2 represents the gate count of 128-point FFT with the three different multipliers for  $n = 10$ . Compare to the traditional Booth multipliers, our approach improves by 10% reduction in gate count. And it is only 2% bigger than the direct-truncated multiplier.

As mentioned in section 4.1, the 128-point mixed-radix FFT algorithm is divided into three stages. The multiplications of twiddle factors should be performed in the end of each stage. In Fig 4-2, the multiplications should be performed in the end of first stage and the end of second stage. Because the fixed-width multipliers which cause the truncation errors are

employed, the multiplications of twiddle factors will reduce the performance of FFT. The computations of twiddle factors will increase, if the points of FFT are raised. Thus, the more point of FFT is needed; the less performance will be expected. In order to observe the performance of high-point FFT by using Fixed-width multipliers, the proposed approach is employed in the 8192-point FFT algorithm [14]. Table 4-3 shows the simulation results of 8192-point FFT.

The SQNR of proposed approach is about 2dB less than the standard Booth multiplier. But it is still larger than the direct-truncated multiplier approach.

**Table 4-2: Gate count of 128-point FFT for n = 10**

	Combinational	Sequential	Total
Booth	57397(1)	35923	93320(1)
Direct_t	47768(0.83)	35021	82789(0.88)
Proposed	48850(0.85)	35612	84462(0.90)

**Table 4-3: SQNR of 8192-point FFT with different multiplier approach**

	18	16	14	12	10
Booth	87.63dB	75.55dB	63.23dB	51.57dB	39.53dB
Direct_t	80.87dB	69.06dB	57.40dB	46.35dB	35.20dB
Proposed	85.00dB	72.34dB	59.59dB	48.49dB	37.45dB

# Chapter5 Implementation Results for 128-point FFT

This chapter will describe the CHIP implementation and its design methodology. In section 5.1, the implementation of 128-point FFT with proposed multiplier is introduced. In section 5.2, the implementation of approach 2 is described. The architecture of approach 2 is the same as approach 1. But the registers of approach 2 are replaced by DDR (Double Data Rate) registers can catch data not only in positive edge clock but also in negative edge clock. The function of DDR registers will be introduced in section 5.2. Finally, the simulation result of two proposed architecture are represented in section 5.3.

## 5.1 Approach 1



The proposed approach 1 implements the 128-point FFT architecture with proposed fixed-width multiplier. The structure of approach 1 has been described in section 4.2. The width of input data and twiddle factors are 10-bit and the width of output data is 14-bit. In order to avoid the overflow of computations of BU, the outputs of BU is 1-bit larger than the inputs of BU. In the 128-point FFT algorithm, seven computations of BU are needed. Thus, the outputs of 128-point FFT are 7-bit larger than the inputs. In the cause of low hardware complexity, the least significant 3-bit of the four outputs of stage2 as seen in Fig 4.3 are truncated. The bits of outputs of first, second, and third stages are, 11-bit, 14-bit, and 14-bit, respectively.

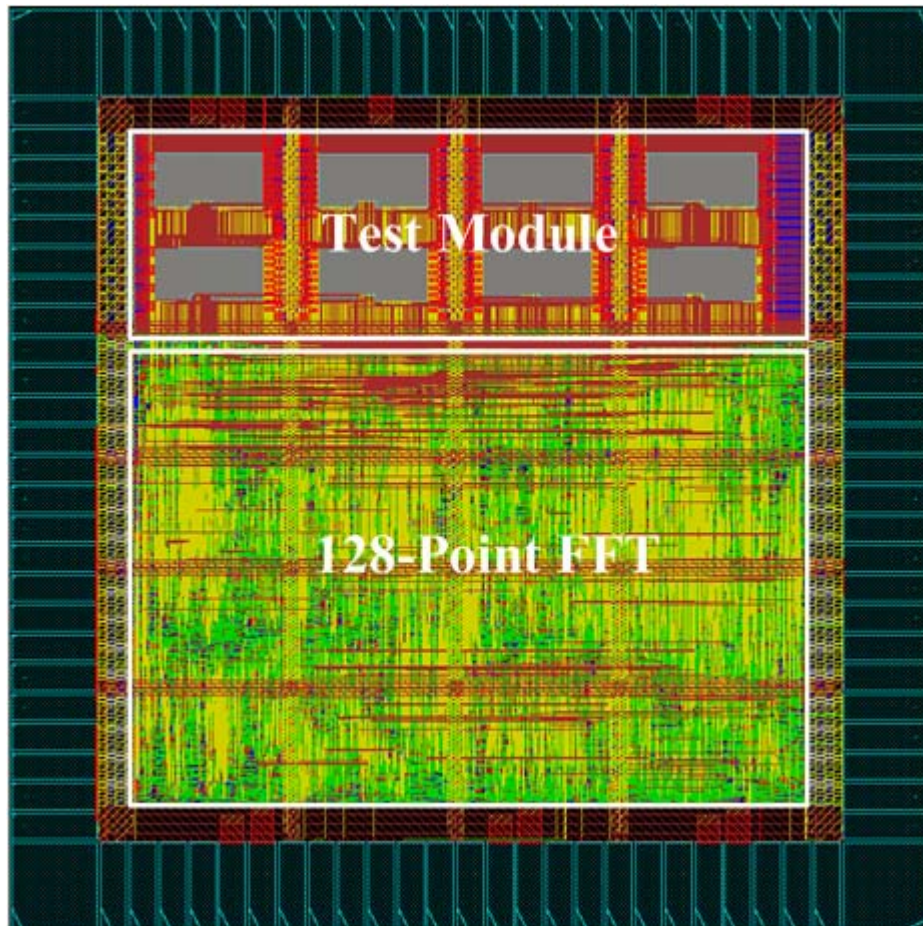
As mentioned in section 4.2, four data path is employed in our 128-point FFT architecture. Thus, the 14 x 256-bit SRAM is used to save the chip pins. The inputs data are stored serially in the SRAM from the 10-bit chip input pins before the operation of calculation. Then the four complex data in parallel are fed to the 128-point FFT structure. After the computations of



128-point FFT, the four complex outputs of 128-point FFT are stored in the SRAM. Finally, the outputs of 128-point FFT are read serially from the SRAM.

This 128-point FFT architecture is implemented by 0.18 $\mu$ m one-poly six-metal (1P6M) standard cell technology. Fig 5-1 shows the layout view of approach 1.

Table 5-1 shows the chip summary of approach 1. The total gate count is about 164K with test module 81K and the maximum clock rate is 195 MHz. And, the core size is 1.46 x 1.46 mm<sup>2</sup>. The maximum power consumption is 500mW at clock rate 195 MHz. The chip is packaged in a 128 CQFP package.



**Fig 5-1 Layout view of approach 1**



**Table 5-1: The chip summary of approach 1 architecture**

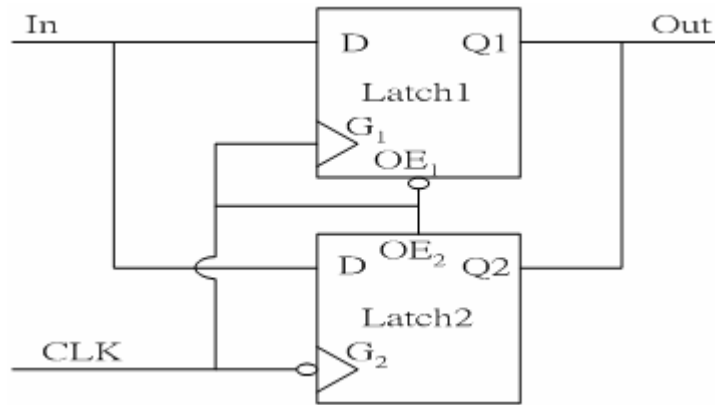
Memory size	13 x 256 bits
Core area (mm <sup>2</sup> )	1.46mm x 1.46mm
Total gate count	83K + 81K Test Module
Maximum Operating Frequency	195 MHz
Date rate (ample/s)	1.2G* 780M**
Average Power	363mW@300MHz* 246mW@195MHz**

\* Typical Case 1.8V

\*\* Worst Case 1.62V

## 5.2 Approach 2

The structure of approach 2 is almost the same as approach 1 besides the registers. The DDR registers are employed in approach 2. The proposed DDR registers can catch data either at positive edge clock or at negative edge clock. Thus, two operations can be computed during one clock cycle. We can achieve the same throughput rate as D Flip-Flop structure at half operation frequency by using DDR registers. Fig 5.2 shows the structure of DDR register which is composed by two parallel latches.



**Fig 5-2: Structure of DDR register**

The function of latch is shown in Table 5-2. If  $OE = 0$ , the state of the latch is “Z” and the output,  $Q[n+1]$ , is high impedance. Else if  $OE = 1$  and  $G = 1$ , the state of the latch is “Store” and  $Q[n+1]$  is equal to input, D. Otherwise, the state of the latch is “Latch” and  $Q[n+1]$  is equal to the last output,  $Q[n]$ .

**Table 5-2: Function of Latch**

OE	G	D	$Q[n+1]$
0	X	X	Z
1	1	0	0
1	1	1	1
1	0	X	$Q[n]$

The operation of proposed DDR register at  $CLK = 0$  is shown in Fig 5-3(a). At  $CLK = 0$ , the state of Latch 1 is “Latch” and the output,  $Q_1$ , is equal to  $Q_1[n]$ . The state of Latch 2 is “Z” and the output,  $Q_2$ , is high impedance. Thus, the output of proposed DDR register is equal to  $Q_1[n]$ . By the same way, the state of Latch 1 and Latch 2 are “Z”, and “Latch” respectively when  $CLK = 1$ . So, the output of DDR register is equal to  $Q_2[n]$ . Fig 5-4 is an example of proposed DDR register. And the function of proposed DDR register is described in Table 5-3.

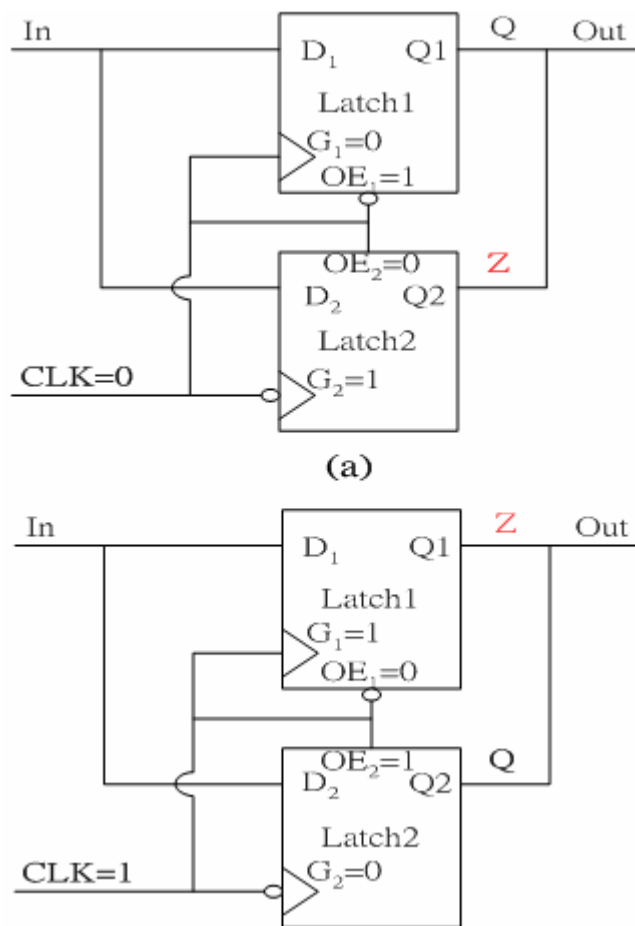


Fig 5-3: Operations of proposed DDR register at (a) CLK = 0 (b) CLK = 1

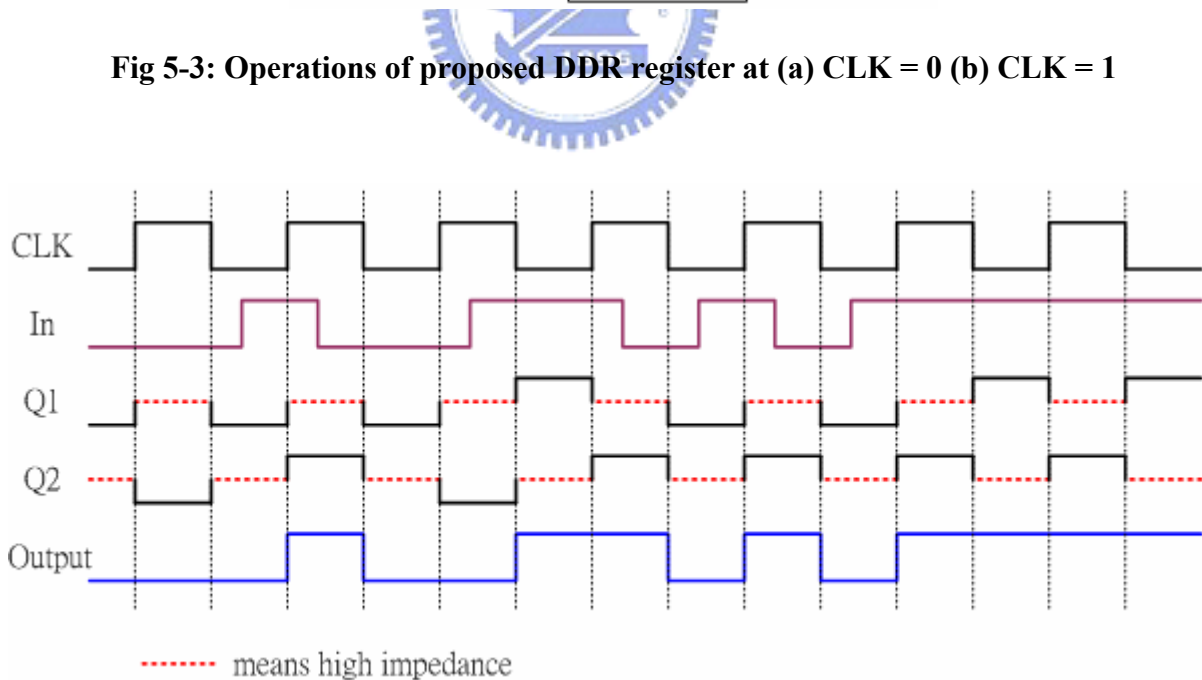


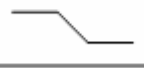



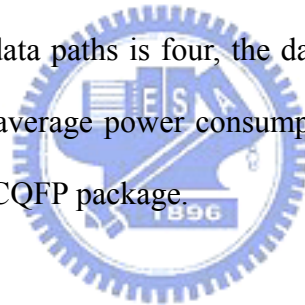
Fig 5-4: Example of proposed DDR register

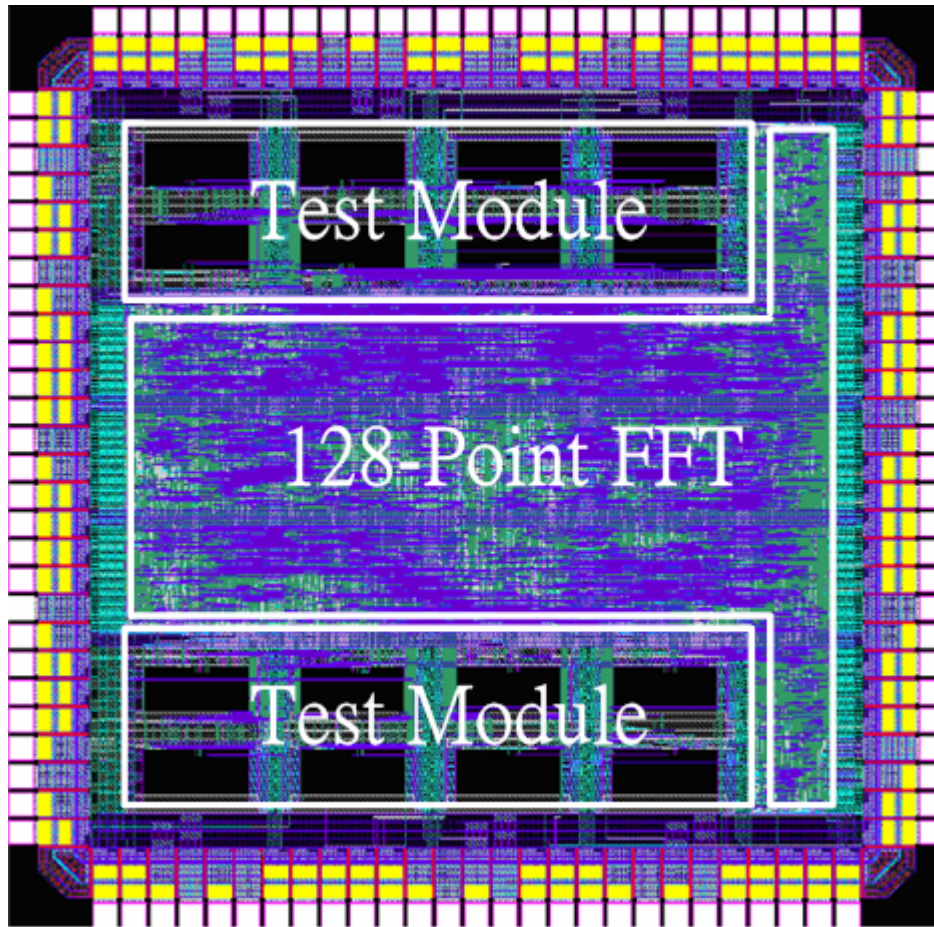
**Table 5-3 Function of proposed DDR register**

In	CLK	Out[n+1]
0		0
1		1
0		0
1		1

Approach 2 is also implemented by 0.18 $\mu$ m one-poly six-metal (1P6M) standard cell technology. Fig 5-5 shows layout view of approach 2.

Table 5-4 shows the chip summary of approach 2. The total gate count is about 222K with test module 108K, and the maximum clock rate is 90 MHz. Because the DDR registers are employed and the number of data paths is four, the data rate is 720M samples/sec. The core size is 2.24 x 2.24 mm<sup>2</sup>. The average power consumption is 533mW at clock rate 90 MHz. The chip is packaged in a 128 CQFP package.





**Fig 5-5 Layout view of approach 2**

**Table 5-4: The chip summary of approach 2 architecture**

Memory size	13 x 256 bits
Core area (mm <sup>2</sup> )	2.24mm x 2.24mm
Total gate count	113K + 108K Test Module
Maximum Operating Frequency	90 MHz
Date rate (M samples/s)	720
Average Power	533mW (Include RAM)

## 5.3 Comparison

Table 5-5 lists the comparisons of various 128-point FFT approaches. The proposed approach is simulated in 0.18 $\mu$ m (1P6M) worst case. From this table, it is obviously that Approach 1 has the minimum core size. And the power is smaller than Booth approach.

**Table 5-5: Comparison of various 128-point FFT architectures**

	Y-W Lin[13]	K-H Lin[15]	Booth	Approach 1
Process	0.18 $\mu$ m 1P6M	0.18 $\mu$ m 1P6M	0.18 $\mu$ m 1P6M	0.18 $\mu$ m 1P6M
Input width	10-bit	8-bit	10-bit	10-bit
SQNR	30dB	31dB	33dB	32dB
Data path	4	4	4	4
Maximum Data rate	1G sample/s	800M sample/s**	1.2G sample/s* 780M sample/s**	1.2G sample/s* 780M sample/s**
Average Power	175mW @250MHz	127mW @132MHz**	443mW @300MHz* 286mW @195MHz**	336mW @300MHz* 246mW @195MHz**
Chip size (mm <sup>2</sup> )	1.76 x 1.76 (include RAM)	1.56 x 1.56 (include RAM)	1.24 x 1.24	1.18 x 1.18 (1.46 x 1.46 include RAM))

\* Typical Case 1.8V

\*\* Worst Case 1.62V

# Chapter6 Conclusion

In this paper, the low-error area-efficient fixed-width multiplier is proposed. The proposed fixed-width multiplier can not only reduce the truncation error but also decrease the circuit complexity. The average error of proposed fixed-width multiplier is only 15% of direct-truncated multiplier. And the area of our approach is only 60% of the standard Booth multiplier.

In order to observe the performance in real applications, our multiplier is used in 128-point FFT architecture. The SQNR of our approach is only 1dB less than the traditional Booth multiplier. Compared to the direct-truncated multiplier approach, our approach has 10dB SQNR improvement with only 2% increased in circuit overhead. In conclusion, our approach can not only achieve the low-area approximated to the direct-truncated multiplier but also reach the high-performance close to the Booth multiplier approach.

In order to reduce the operation frequency, the DDR register structure is employed. It can reduce the operation frequency to only 50% of Flip-Flop structures.

Finally, the structure of approach 1 and approach 2 are implemented by 0.18 $\mu$ m 1P6M CMOS technology as shown in Section 5.1 and Section 5.2, separately.

# BIBLIOGRAPHY

- [1] Y. C. Lim, "Single precision multiplier with reduced circuit complexity for signal processing applications," *IEEE Trans. Comp.*, vol. 41, pp. 1333-1336, Oct. 1992.
- [2] M. J. Schulte and E. E. Swartzlander Jr., "Truncated multiplication with correction constant," *VLSI Signal Processing VI*, pp. 388-396, 1993.
- [3] S. S. Kidambi, F. El-Guibaly, and A. Antoniou, "Area-efficient multipliers for digital signal processing applications," *IEEE Trans. Circuits Syst. II*, vol. 43, pp. 90-94, Feb. 1996.
- [4] E. J. King and E. E. Swartzlander Jr., "Data-dependent truncation scheme for parallel multipliers," in *Proc. 31<sup>st</sup> Asilomar conf. Signals, Systems, Computers*, Pacific Grove, CA, pp. 1178-1182, 1997.
- [5] E. E. Swartzlander Jr., "Truncated multiplication with approximate rounding," in *Proc. 33<sup>rd</sup> Asilomar Conf. Signals, Systems, Computers*, Pacific Grove, CA, pp. 1480-1483, 1999.
- [6] S. J. Jou and H. H. Wang, "Fixed-width multiplier for DSP application," in *IEEE Int. Symp. Computer Design*, pp. 318-322, Sept. 2000.
- [7] S. J. Jou, M. H. Tsai, and Y. L. Tsao, "Low-error reduced-width booth multipliers for DSP applications," *IEEE Trans. Circuits Syst. I*, vol. 50, pp.1470-1474, Nov. 2003.
- [8] K. J. Cho, K. C. Lee, J. G. Chung, and K. K. Parhi, "Low error fixed-width modified Booth multiplier," in *Proc. IEEE Workshop on Signal Processing Systems*, San Diego, CA, pp. 45-50, Oct. 2002.
- [9] K. J. Cho, K. C. Lee, J. G. Chung, and K. K. Parhi, "Design of low-error fixed-width modified Booth multiplier," *IEEE Trans. VLSI Syst.*, vol. 12, pp. 522-531, May 2004.
- [10] L. D. Van, S. S. Wang, and W. S. Feng, "Design of the lower-error fixed-width multiplier and its application," *IEEE Trans. Circuits Syst. II*, vol. 47, pp. 1112-1118, Oct. 2000.
- [11] L. D. Van, and C. C. Yang, "Generalized low-error area-efficient fixed-width multipliers,"



*IEEE Trans. Circuits Syst. I*, vol. 52, pp. 1608-1619, Aug. 2005.

[12]C. R. Baugh and B. A. Wooley, “A two’s complement parallel array multiplication algorithm,” *IEEE Trans. Comp.*, vol. C-22, no. 12, pp.1045-1047, Dec. 1973.

[13]Y. W. Lin, H. Y. Liu, and C. Y. Lee, “A 1 GS/s FFT/IFFT processor for UWB applications,” *IEEE J. Solid-State Circuits*, vol. 40, pp. 1726-1735, Aug. 2005.

[14]Y. W. Lin, H. Y. Liu, and C. Y. Lee, “A dynamic scaling FFT processor for DVB-T applications,” *IEEE J. Solid-State Circuits*, vol. 39, pp. 2005-2013, Nov. 2004.

[15]K. H. Lin, “Design of FFT/IFFT module for Ultra Wideband System,” Master thesis, National Chiao Tung University, Hsinchu, Taiwan, June 2005.



## 作者簡介

姓名：黃弘安

出生：宜蘭縣

學歷：成功國小 國華國中 宜蘭高中

88.9 ~ 92.6 國立中央大學電機工程學系

92.9 ~ 94.7 國立交通大學 電子研究所 (oasis lab)

## 得獎

92 學年度大專院校 FPGA 設計競賽 Xilinx 組優等

