國立交通大學

電子工程學系 電子研究所碩士班

碩 士 論 文

截斷乘法器之進位估計與其快速傅立葉轉換應用
Carry Estimation of Truncated-Width Multiplier for FFT Application

研究生:趙祐徵

指導教授:張錫嘉

中華民國九十六年一月

截斷乘法器之進位估計與其快速傅立葉轉換應用

Carry Estimation of Truncated-Width Multiplier for FFT Application

學生：趙祐徵　　　　Student : You-Zheng Chao

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

碩士論文

A Thesis

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

January 2007

Hsinchu, Taiwan, R.O.C.

截斷乘法器之進位估計與其快速傅立葉轉換應用

學生：趙祐徵

指導教授：張錫嘉

電子工程學系 電子研究所碩士班

摘　要

在本論文中，我們提供了一個用統計方式來分析截斷乘法器補償。在截斷乘法器中，被截去部分的誤差可以用簡單的進位公式來補償。本論文對 Baugh-Wooley 以及 Booth 乘法器各提出了三種不同的補償方式。藉由以我們所提出的這些補償方式，與使用 direct-truncate 方法的 Baugh-Wooley 以及 Booth 乘法器相比較之下，各減少了約 85%以及 80%的誤差。此外，我們將此補償方法應用在 64 點 FFT 的乘法器裡面，可以得到與 post-truncate 相近的 SQNR(Signal to Quantization Noise Ratio)。當應用於 2048 點 FFT 時，我們所提出的方法可以大約降低約 4.7%的面積而其 SQNR 值亦不會與 post-truncate 有太大的差異。

Carry Estimation of Truncated-Width Multiplier for FFT Application

Student: You-Zheng Chao

Advisor: Hsie-Chia Chang

Institute of Electronics
National Chiao Tung University

ABSTRACT

This thesis provides a statistical analysis for truncated-width multiplier which receives two n-bit inputs and truncates n-bit output. The truncated parts which produce carry-in can be replaced by carry estimation methods. In order to reduce the truncation error, different compensation methods are provided for different bit-width. This thesis discusses Baugh-Wooley and Booth multiplier and provides three types of compensation method for these two multipliers. According to the simulation result, about 85% and 80% error of the direct-truncation Baugh-Wooley and Booth multipliers can be reduced. For the 64-point FFT case, the software simulation shows similar performance while comparing to post-truncate method. For the hardware of 2048-point FFT, our method can reduce about 4.7% gate count while comparing to post-truncate method without performance loss.

誌　　　　謝

轉眼間，兩年的碩士研究生涯即將告一段落，在這兩年中學習到很多做學問的方法以及為人處世的道理。要感謝的人非常多，首先我要誠摯的感謝指導教授張錫嘉老師。老師細心的教導並不厭其煩給予我正確的方向，讓我的研究過程能夠更加順利。老師對學問的嚴謹態度更是令我相當值得學習的典範。再來要感謝的就是彥欽學姊、建青以及陳元學長，有你們熱心的幫忙以及不斷的討論，總能讓我在迷惘時找到正確的方向，也讓我在研究的過程中，可以很快的解決各種問題。此外也要感謝Oasis實驗室以及Sun實驗室的同學、學弟們，有你們的陪伴，讓我的研究生涯更加充實以及快樂。最後，再一次的感謝在這兩年中，曾經幫助過我的每一位。

Carry Estimation of Truncated-Width Multiplier for FFT Application

Student: You-Zheng Chao

Advisor: Dr. Hsie-Chia Chang

Department of Electronics Engineering
National Chiao Tung University

Contents

List of Figures

ES A 1896

List of Tables

Chapter 1

Introduction

1.1 Research Motivation

In many digital signal procession (DSP) applications, multipliers have the fixed-width property in order to reduce the growing width of serially concatenated addition and multiplication. The fixed-width property means that : If both input bits of a multiplication equal to n-bit, its corresponding output bit number will be $2n$. In order to reduce the hardware complexity and keep fixed-width property, the $2n$-bit output will truncate k bits (usually $k = n$) by direct truncating or rounding the k least significant bits. Consider the case of $k = n$, the input and output will be kept the same bit-width. For example, assume a 8-bit fixed-width multiplier in Figure 1.1, one input is X(x_7∼x_0), a 8-bit integer; the other input is Y(y_7∼y_0), a 8-bit decimal.

By multiplying X by Y, we can get the 16 bits product which is composed of 8-bit decimal and 8-bit integer. In order to reduce the area and improve critical path of the multiplier, the least significant 8 bits ($P7 \sim P0$) will be truncated. But some error may be introduced by direct truncating the least significant 8 bits. Thus, to ease this phenomenon, carry compensation is required. A proper carry compensation can effectively reduce the error by adding a proper carry to the most significant part ($P15 \sim P8$).

In the thesis, an area-efficient low-error multiplier based on statistical analysis is proposed. The carry estimation method for different truncated-width multiplier is variable through different input width n. And it can be implemented with few full adder cells. From simulation result, the error by using direct-truncation can be reduced by proper

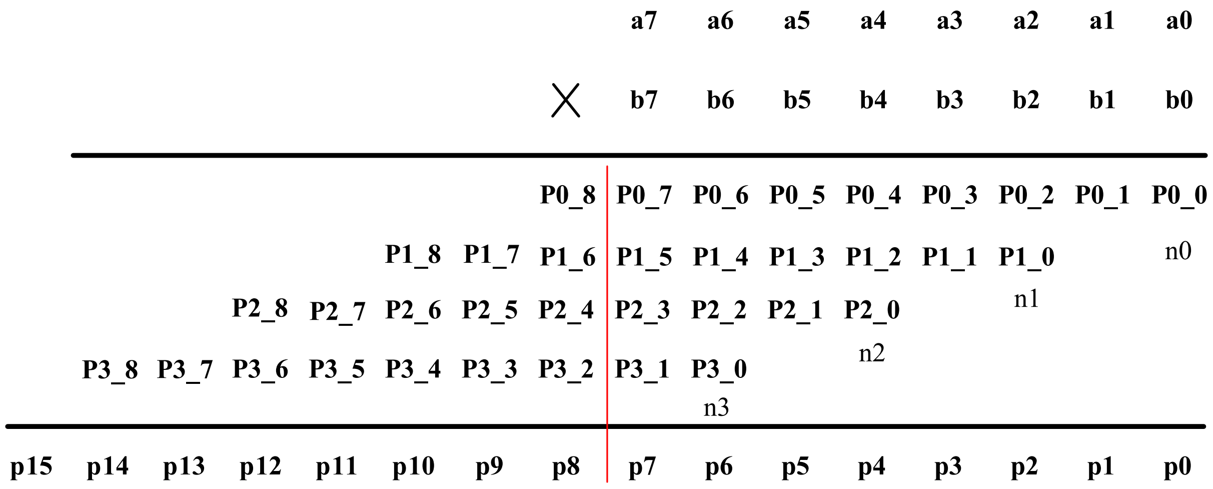

a7 a6 a5 a4 a3 a2 a1 a0
b7 b6 b5 b4 b3 b2 b1 b0
P0_8 P0_7 P0_6 P0_5 P0_4 P0_3 P0_2 P0_1 P0_0
n0
P1_8 P1_7 P1_6 P1_5 P1_4 P1_3 P1_2 P1_1 P1_0
n1
P2_8 P2_7 P2_6 P2_5 P2_4 P2_3 P2_2 P2_1 P2_0
n2
P3_8 P3_7 P3_6 P3_5 P3_4 P3_3 P3_2 P3_1 P3_0
n3
p15 p14 p13 p12 p11 p10 p9 p8 p7 p6 p5 p4 p3 p2 p1 p0

Figure 1.1: 8-bit fixed width multiplier

carry compensation.

1.2 Thesis Organization

The organization of this thesis is as follow. In chapter 2, basic concept of two multipliers and three existed compensation approach are introduced. Chapter 3 shows the statistical analysis of Baugh-Wooley and Booth encoding multiplier. Chapter 4 illustrates a software example for a 64-point FFT. The hardware example of a 2048-point FFT is shown in Chapter 5. Finally, Chapter 6 is the conclusion of this thesis.

Chapter 2

Existed Fixed Width Multipliers and Compensation Methods

2.1 Basic Multiplier Architecture

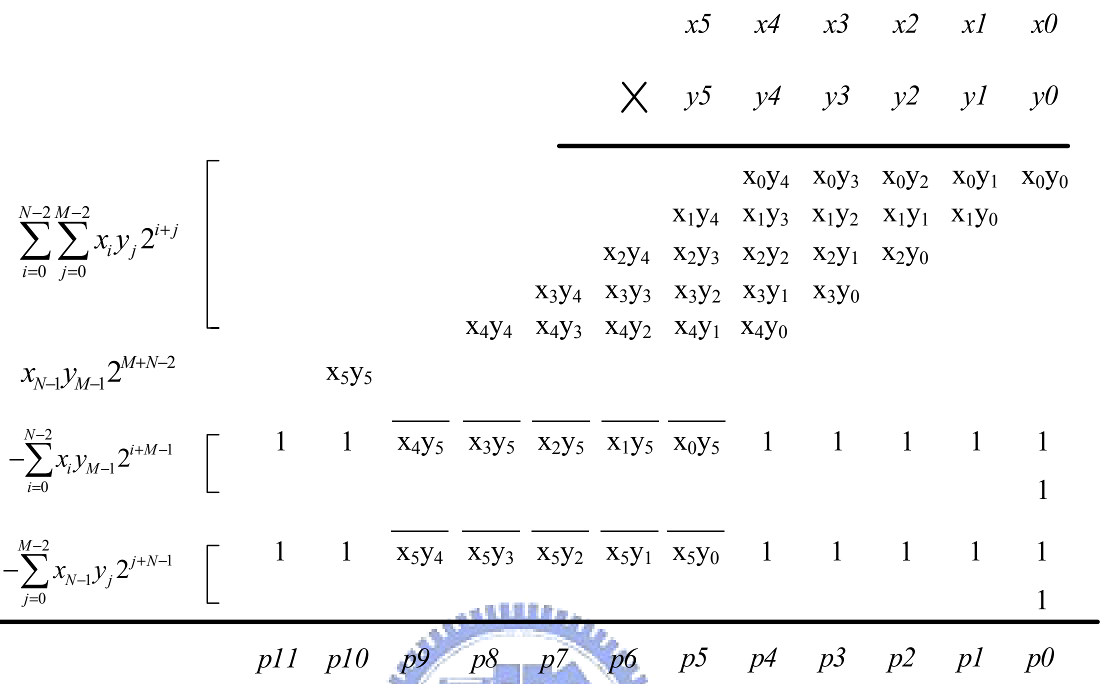
Multiplication is widely used and essential for digital signal processing. Multiplication algorithms can be designed in many different architectures for various application. The most basic way to perform multiplication is generating the rows of partial products and then summing them. For many signal processing application, multiplication operation is a signed operation, which means one or both operand of multiplication may be signed. The following paragraph will illustrate two different architecture for signed multiplication.

2.1.1 Baugh-Wooley multiplier

Multiplication of signed number may need abstraction operation while summing the partial products. But we can recall that the signed number is described in 2's complement form, which the most significant bit has the negative weight. Hence, the product is :

$$\begin{aligned}P &= \left(-y_{M-1}2^{M-1} + \sum_{j=0}^{M-2} y_j 2^j\right)\left(-x_{N-1}2^{N-1} + \sum_{i=0}^{N-2} x_i 2^i\right)\\ &= \sum_{i=0}^{N-2}\sum_{j=0}^{M-2} x_i y_j 2^{i+j} + x_{N-1}y_{M-1}2^{M+N-2} - \left(\sum_{i=0}^{N-2} x_i y_{M-1} 2^{i+M-1} + \sum_{j=0}^{M-2} x_{N-1} y_j 2^{j+N-1}\right)\end{aligned} \tag{2.1}$$

In equation (2.1), the last two terms with negative weight must be subtracted. The Baugh-Wooley multiplier algorithm [1] handles subtraction function as adding 2's complement of negative terms.(i.e., inverting each bit of negative terms, and adding one at least significant bit). Figure 2.1 shows the procedure of summing partial products.



$x5$ $x4$ $x3$ $x2$ $x1$ $x0$
$\times$ $y5$ $y4$ $y3$ $y2$ $y1$ $y0$
$\sum_{i=0}^{N-2}\sum_{j=0}^{M-2} x_i y_j 2^{i+j}$
x_0y_4 x_0y_3 x_0y_2 x_0y_1 x_0y_0
x_1y_4 x_1y_3 x_1y_2 x_1y_1 x_1y_0
x_2y_4 x_2y_3 x_2y_2 x_2y_1 x_2y_0
x_3y_4 x_3y_3 x_3y_2 x_3y_1 x_3y_0
x_4y_4 x_4y_3 x_4y_2 x_4y_1 x_4y_0
$x_{N-1}y_{M-1}2^{M+N-2}$
x_5y_5
$-\sum_{i=0}^{N-2} x_i y_{M-1} 2^{i+M-1}$
1 1 $\overline{x_4y_5}$ $\overline{x_3y_5}$ $\overline{x_2y_5}$ $\overline{x_1y_5}$ $\overline{x_0y_5}$ 1 1 1 1 1
1
$-\sum_{j=0}^{M-2} x_{N-1} y_j 2^{j+N-1}$
1 1 $\overline{x_5y_4}$ $\overline{x_5y_3}$ $\overline{x_5y_2}$ $\overline{x_5y_1}$ $\overline{x_5y_0}$ 1 1 1 1 1
1
$p11$ $p10$ $p9$ $p8$ $p7$ $p6$ $p5$ $p4$ $p3$ $p2$ $p1$ $p0$

Figure 2.1: Partial product for 2's complement Baugh-Wooley multiplier

The upper part of partial product represent unsigned multiplication, and the second row represents the most significant bit of the product. The final two pairs of rows means subtraction, which is transformed into 2's complement form. Notice that each term has leading and tailing 1's, which are inversion of implicant leading and tailing 0's. And the extra 1 in least significant part must be added while taking 2's complement.

The simplification of Figure 2.1, called modified Baugh-Wooley multiplier [2] which reduced partial products by summing the 1's, is shown in Figure 2.2. We can see that the most part of sign-extension bits are eliminated, which reduces area complexity and hardware cost.

$$
\begin{array}{cccccccccccc}
 & & & & & & x5 & x4 & x3 & x2 & x1 & x0 \\
 & & & & & \times & y5 & y4 & y3 & y2 & y1 & y0 \\
\hline
 & & & & & 1 & \overline{x_5y_0} & x_0y_4 & x_0y_3 & x_0y_2 & x_0y_1 & x_0y_0 \\
 & & & & & \overline{x_5y_1} & x_1y_4 & x_1y_3 & x_1y_2 & x_1y_1 & x_1y_0 & \\
 & & & & \overline{x_5y_2} & x_2y_4 & x_2y_3 & x_2y_2 & x_2y_1 & x_2y_0 & & \\
 & & & \overline{x_5y_3} & x_3y_4 & x_3y_3 & x_3y_2 & x_3y_1 & x_3y_0 & & & \\
 & & \overline{x_5y_4} & x_4y_4 & x_4y_3 & x_4y_2 & x_4y_1 & x_4y_0 & & & & \\
1 & x_5y_5 & \overline{x_4y_5} & \overline{x_3y_5} & \overline{x_2y_5} & \overline{x_1y_5} & \overline{x_0y_5} & & & & & \\
\hline
p11 & p10 & p9 & p8 & p7 & p6 & p5 & p4 & p3 & p2 & p1 & p0
\end{array}
$$

Figure 2.2: Simplified partial product for 2's complement Baugh-Wooley multiplier

2.1.2 Booth Encoding Multiplier

Multipliers in the previous section compute partial products are in radix-2 manner, that is, each bit has one corresponding partial product. In order to reduce the number of partial products, we can use modified Booth encoding [3] [4] technique. Consideration of two 2's complement number X and Y, with m and n bits separately.

$$
\begin{aligned}
X &= -x_{m-1} + \sum_{i=1}^{m-1} x_{m-1-i} 2^{-i} \\
Y &= -y_{n-1} + \sum_{j=1}^{n-1} y_{n-1-j} 2^{-j}
\end{aligned}
\tag{2.2}
$$

We must concatenate a "0" at the right end of Y for modified Booth encoding, as in Figure 2.3. Table 2.1 shows the encoding table of partial product.

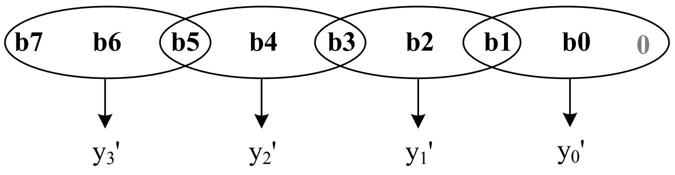


Figure 2.3: Grouping of multiplier bits for bit width n = 8

Table 2.1: Modified Booth encoding table

y_{2i+1}	y_{2i}	y_{2i-1}	y'_i	X_{sel}	$2X_{sel}$	NEG
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	1	0	1	1	0	0
0	1	1	2	0	1	0
1	0	0	-2	0	1	1
1	0	1	-1	1	0	1
1	1	0	-1	1	0	1
1	1	1	0	0	0	1

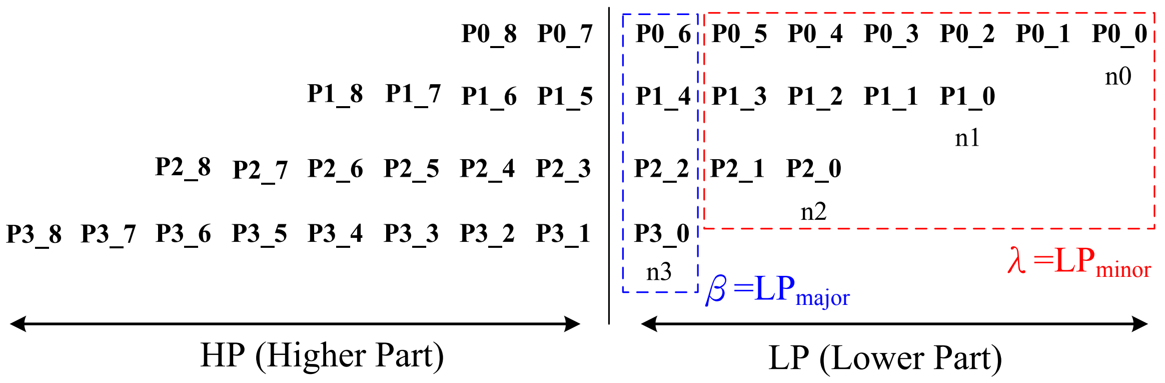
After modified Booth encoding, Y can express as following.

$$Y = \sum_{j=0}^{n/2-1} y'_{n/2-1-j} 2^{-(2j+1)} \tag{2.3}$$

where

$$y'_j = -2y_{2j+1} + y_{2j} + y_{2j-1} \tag{2.4}$$

By using Table 2.1 and Figure 2.4, we can get the simplified partial products. For example, a 8x8 multiplier's partial product is shown as follow.



P0_8 P0_7 P0_6 P0_5 P0_4 P0_3 P0_2 P0_1 P0_0
n0
P1_8 P1_7 P1_6 P1_5 P1_4 P1_3 P1_2 P1_1 P1_0
n1
P2_8 P2_7 P2_6 P2_5 P2_4 P2_3 P2_2 P2_1 P2_0
n2
P3_8 P3_7 P3_6 P3_5 P3_4 P3_3 P3_2 P3_1 P3_0
n3
β =LPmajor
λ =LPminor
HP (Higher Part)
LP (Lower Part)

Figure 2.4: HP and LP part for modified Booth Multiplier when n = 8

And its relationship between partial products and encoded y' is shown in Table 2.2.

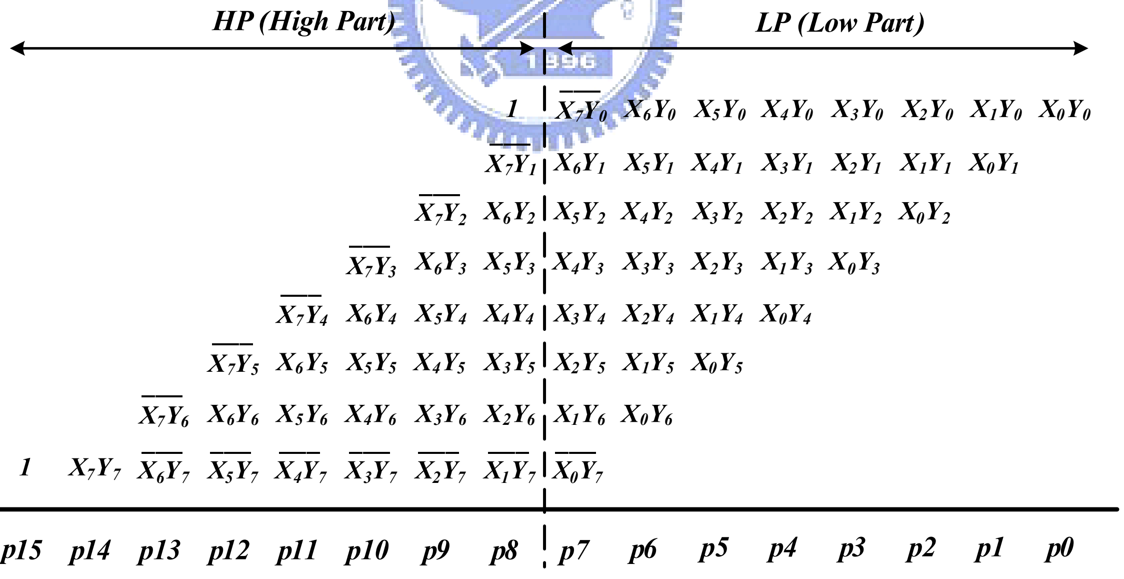
Table 2.2: Modified Booth encoded partial product

y'_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	$\overline{a_7}$	$\overline{a_7}$	$\overline{a_6}$	$\overline{a_5}$	$\overline{a_4}$	$\overline{a_3}$	$\overline{a_2}$	$\overline{a_1}$	$\overline{a_0}$	1
2	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	0	0
-2	$\overline{a_7}$	$\overline{a_6}$	$\overline{a_5}$	$\overline{a_4}$	$\overline{a_3}$	$\overline{a_2}$	$\overline{a_1}$	$\overline{a_0}$	1	1

2.2 Compensated Multiplier Architecture

2.2.1 L. D. Van's Fixed-Width Multiplier

The fixed-width multiplier proposed by L. D. Van will be introduced in this section, [5] [6]. The L. D. Van's approach is based on Baugh-Wooley multiplier [1]. Figure 2.5 shows the partial product terms of an 8x8 multiplier using Baugh-Wooley architecture. It can be divided into two parts, HP (high part) and LP (low part), which represents the most significant part and the least significant part of partial product.



HP (High Part)
LP (Low Part)
1 $\overline{X_7Y_0}$ X_6Y_0 X_5Y_0 X_4Y_0 X_3Y_0 X_2Y_0 X_1Y_0 X_0Y_0
$\overline{X_7Y_1}$ X_6Y_1 X_5Y_1 X_4Y_1 X_3Y_1 X_2Y_1 X_1Y_1 X_0Y_1
$\overline{X_7Y_2}$ X_6Y_2 X_5Y_2 X_4Y_2 X_3Y_2 X_2Y_2 X_1Y_2 X_0Y_2
$\overline{X_7Y_3}$ X_6Y_3 X_5Y_3 X_4Y_3 X_3Y_3 X_2Y_3 X_1Y_3 X_0Y_3
$\overline{X_7Y_4}$ X_6Y_4 X_5Y_4 X_4Y_4 X_3Y_4 X_2Y_4 X_1Y_4 X_0Y_4
$\overline{X_7Y_5}$ X_6Y_5 X_5Y_5 X_4Y_5 X_3Y_5 X_2Y_5 X_1Y_5 X_0Y_5
$\overline{X_7Y_6}$ X_6Y_6 X_5Y_6 X_4Y_6 X_3Y_6 X_2Y_6 X_1Y_6 X_0Y_6
1 X_7Y_7 $\overline{X_6Y_7}$ $\overline{X_5Y_7}$ $\overline{X_4Y_7}$ $\overline{X_3Y_7}$ $\overline{X_2Y_7}$ $\overline{X_1Y_7}$ $\overline{X_0Y_7}$
p15 p14 p13 p12 p11 p10 p9 p8 p7 p6 p5 p4 p3 p2 p1 p0

Figure 2.5: Partial product of 8-bit Baugh-Wooley Array multiplier

Generally speaking, the carry from the least significant part to the most significant part of Baugh-Wooley multiplier can be defined as Equation (2.5).

$$Carry = [\frac{1}{2}\beta + \lambda] \tag{2.5}$$

The two elements in the equation (2.5), β and λ, represent the lower part of the partial product, and can be seen as a part of Figure 2.5. They are respectively defined in Equation (2.6) and (2.7), which are shown in Figure 2.6.

$$\beta = \overline{x_{n-1}y_0} + x_{n-2}y_1 + x_{n-3}y_2 + \cdots + x_1y_{n-2} + \overline{x_0y_{n-1}} \tag{2.6}$$

$$\lambda = 2^{-2}(x_{n-2}y_0 + x_{n-3}y_1 + \cdots + x_0y_{n-2}) + \cdots + 2^{-2}x_0y_0 \tag{2.7}$$

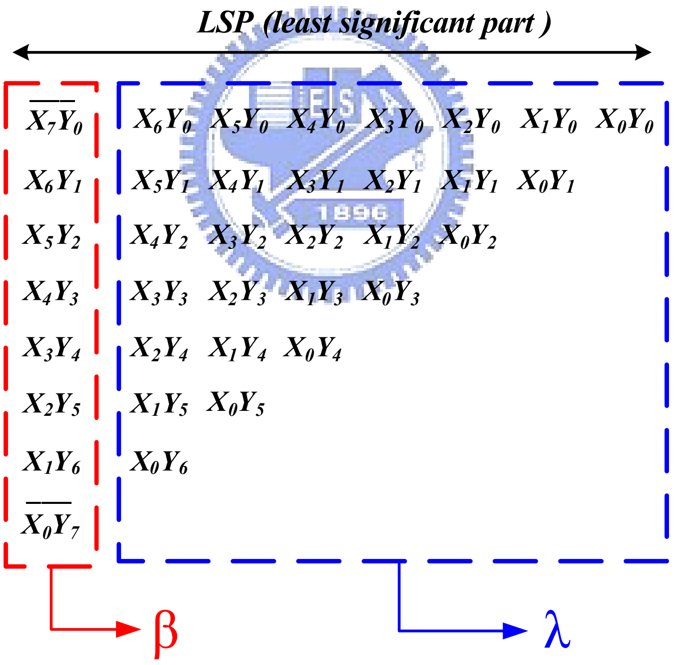

LSP (least significant part)
$\overline{X_7Y_0}$ X_6Y_0 X_5Y_0 X_4Y_0 X_3Y_0 X_2Y_0 X_1Y_0 X_0Y_0
X_6Y_1 X_5Y_1 X_4Y_1 X_3Y_1 X_2Y_1 X_1Y_1 X_0Y_1
X_5Y_2 X_4Y_2 X_3Y_2 X_2Y_2 X_1Y_2 X_0Y_2
X_4Y_3 X_3Y_3 X_2Y_3 X_1Y_3 X_0Y_3
X_3Y_4 X_2Y_4 X_1Y_4 X_0Y_4
X_2Y_5 X_1Y_5 X_0Y_5
X_1Y_6 X_0Y_6
$\overline{X_0Y_7}$
β
λ

Figure 2.6: β and λ in Baugh-Wooley multiplier

Before introducing L. D. Van's approach, a terminology, $\theta_{index,\tau}$, is shown in Equation (2.8), and is defined as follow:

$$\theta_{index,\tau}(q_{n-1-\tau}, q_{n-2-\tau}, \ldots, q_0) = \\ < x_{n-1-\tau}y_0 >^{q_{n-1-\tau}} + < x_{n-2-\tau}y_1 >^{q_{n-2-\tau}} + \cdots + < x_0 y_{n-1-\tau} >^{q_0} \tag{2.8}$$

The parameter,τ, means to truncate the (τ-1) least significant columns of partial part, and keep (n + τ) most-significant columns to be un-truncated, and the binary parameters $q_{n-1-\tau}q_{n-2-\tau}\ldots q_0$ are belong to (0,1).

Equation (2.9) illustrates the operation of $< X >^q$

$$< X >^q = \begin{cases} X, & \text{if } q = 0 \\ \overline{X}, & \text{otherwise} \end{cases} \tag{2.9}$$

The $\overline{X}$ above means the complement of the binary number X. For example, if $n = 8$, and keeping eight columns, the 129_{th} index, $\theta_{index=129,\tau=0}$, can be written as Equation (2.10).

$$\theta_{index=129,\tau=0} = \overline{x_7y_0} + x_6y_1 + x_5y_2 + x_4y_3 + x_3y_4 + x_2y_5 + x_1y_6 + \overline{x_0y_7} \tag{2.10}$$

Two calculation methods of error-compensation bias for $\tau = 0$ will be explained in the following discussion.

Referencing to the derivation in [6], Equation (2.5) can be rewritten as Equation (2.11)

$$Carry_{\tau-1} = \theta_{index,\tau=0} + [\frac{1}{2}\beta - \theta_{index,\tau=0} + \lambda]_\gamma \tag{2.11}$$

The following shows that Equation (2.5) can be replaced by Equation (2.12)

$$Carry_{\tau-1} = (< x_{n-2}y_1 >^{q_{n-2}} + < x_{n-3}y_2 >^{q_{n-3}} + \cdots + < x_1y_{n-2} >^{q_1}) + [K]_\gamma \tag{2.12}$$

Which K can be shown as Equation (2.13)

$$K = < x_{n-1}y_0 >^{q_{n-1}} + < x_0y_{n-1} >^{q_0} + \frac{1}{2}\beta - \theta_{index,\tau=0} + \lambda \tag{2.13}$$

After the index is chosen, the first term in the parenthesis of equation (2.12) can be easily determined. The second term, $[K]_\gamma$, can be calculated by the expected value of the partial product which can be obtained by full search. Based on the above equations, two types of carry-estimation formulas are proposed to get more accurate error-compensation value. These formulas are separately shown in Equation (2.14) and Equation (2.15).

$$Carry_{type1} = \begin{cases} (<x_{n-2}y_1>^{q_{n-2}} + <x_{n-3}y_2>^{q_{n-3}} + \cdots + <x_1y_{n-2}>^{q_1}) + [K_1]_\gamma, & \text{if } \theta_{index} = 0 \\ (<x_{n-2}y_1>^{q_{n-2}} + <x_{n-3}y_2>^{q_{n-3}} + \cdots + <x_1y_{n-2}>^{q_1}) + [K_2]_\gamma, & \text{if } \theta_{index} > 0 \end{cases} \tag{2.14}$$

$$Carry_{type1} = \begin{cases} (<x_{n-2}y_1>^{q_{n-2}} + <x_{n-3}y_2>^{q_{n-3}} + \cdots + <x_1y_{n-2}>^{q_1}) + [K_3]_\gamma, & \text{if } \theta_{index} < n \\ (<x_{n-2}y_1>^{q_{n-2}} + <x_{n-3}y_2>^{q_{n-3}} + \cdots + <x_1y_{n-2}>^{q_1}) + [K_4]_\gamma, & \text{if } \theta_{index} = n \end{cases} \tag{2.15}$$

Where K_1, K_2, K_3, and K_4 are the mean value of K for different range of θ_{index}. For all different indices in these above equations, the values of K_1 and K_2 can be determined by all input condition simulation. We choose the indices which satisfy $[K_1]_\gamma \in (0,1)$ and $[K_2]_\gamma \in (0,1)$ in order to reduce hardware complexity. For example, a 6x6 multiplier, there are only three indices to satisfy the conditions, $[K_1]_\gamma \in (0,1)$ and $[K_2]_\gamma \in (0,1)$. But when the bit width "n" is changed, the indices will no longer satisfy these conditions. So the second approach 'type 2' is used to find the fixed value of 'K' for different bit width 'n'. By exhaustive search for bit width n from 4 to 12, we can find that the specific index $\theta_{index=2^{n-1}+1}$ satisfy $[K_3]_\gamma = 1$ and $[K_4]_\gamma = 0$. Because the error-compensation bias is shown as Equation (2.12) and $\theta_{index=2^{n-1}+1} = \overline{x_{n-1}y_0} + x_{n-2}y_1 + \ldots + x_1y_{n-2} + \overline{x_0y_{n-1}}$ For $n \leq 12$, it can be described as Equation (2.16).

$$Carry_{type2,index=2^{n-1}+1} = \begin{cases} (<x_{n-2}y_1>^{q_{n-2}} + <x_{n-3}y_2>^{q_{n-3}} + \cdots + <x_1y_{n-2}>^{q_1}) + 1, & \text{if } \theta_{index=2^{n-1}+1} < n \\ (<x_{n-2}y_1>^{q_{n-2}} + <x_{n-3}y_2>^{q_{n-3}} + \cdots + <x_1y_{n-2}>^{q_1}), & \text{if } \theta_{index=2^{n-1}+1} = n \end{cases} \tag{2.16}$$

Exhaustive simulation for large bit width "n" will be a long time. So "Type 2" approach for large "n" will be introduced in the following. Two cases of "Type 2" approach, $\theta_{index=2^{n-1}}+1 < n$ and $\theta_{index=2^{n-1}}+1 = n$ will be separately explained.In order to reduce time for exhaustive simulation, "Type 2" approach for large "n" will be introduced. We will separately explain two conditions for "Type 2" approach.

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

Assumed the probability of each bit of input data equals to "1" is 1/2. Hence, the expect value $E[x_iy_j]$ and $E[\overline{x_iy_j}]$ are equal to 1/4 and 3/4 separately. According to these above expect values, the expected value of $\frac{1}{2}\beta$ can be represented as Equation(2.17).

$$\begin{aligned} E[\frac{1}{2}\beta] &= \frac{1}{2} \times (\frac{3}{4} + \frac{3}{4} + \frac{1}{4} \times (n-2)) \\ &= \frac{n}{8} + \frac{1}{2} \end{aligned} \tag{2.17}$$

Similarly, the expected value of λ, $E[\lambda]$ can be shown in the following Equation (2.18).

$$\begin{aligned} E[\lambda] &= \frac{1}{2^2} \times \frac{1}{4} \times (n-1) + \frac{1}{2^3} \times \frac{1}{4} \times (n-2) + \ldots + \frac{1}{2^n} \times \frac{1}{4} \times 1 \\ &= \frac{1}{4}(\frac{1}{2^2} \times (n-1) + \frac{1}{2^3} \times (n-2) + \ldots + \frac{1}{2^n} + 1) \\ &\cong \frac{n}{8} - \frac{1}{4}, \text{ if } n \geq 4 \end{aligned} \tag{2.18}$$

1896

From equation (2.13), the value of $[K_3]_\gamma$ for index $= 2^{n-1}+1$ is shown as Equation (2.19).

$$\begin{aligned} [K_3]_\gamma &= [E[K]]_\gamma \\ &= \left[E[\overline{x_{n-1}y_0} + \overline{x_0y_{n-1}} - \frac{1}{2}\beta + \lambda] \right]_\gamma \\ &= \left[\frac{3}{4} + \frac{3}{4} - \frac{n}{8} - \frac{1}{2} + \frac{n}{8} - \frac{1}{4} \right]_\gamma = 1 \end{aligned} \tag{2.19}$$

From above equations, we can obtain the error-compensation bias for bit width "n" without exhaustive simulation. Equation (2.20) shows the error-compensation bias for $\theta_{index=2^{n-1}+1} < n$, which is the same as Equation (2.16).

$$Carry_{type2,index=2^{n-1}+1} = (< x_{n-2}y_1 >^{q_{n-2}} + < x_{n-3}y_2 >^{q_{n-3}} + \cdots + < x_1y_{n-2} >^{q_1}) + 1, \quad \text{if } \theta_{index=2^{n-1}+1} < n \tag{2.20}$$

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

The case $\theta_{index=2^{n-1}+1} = n$ is met only when $\overline{x_0y_{n-1}} = \overline{x_{n-1}y_0} = 1$ and $x_1y_{n-2} = x_2y_{n-3} = \ldots = x_{n-2}y_1 = 1$. So, the expected value of $\frac{1}{2}\beta$ can be represented as Equation(2.21).

$$E[\frac{1}{2}\beta] = \frac{1}{2} \times 1 \times n = \frac{n}{2} \tag{2.21}$$

And the expected value of λ, $E[\lambda]$ can be shown as following.

$$\begin{aligned} E[\lambda] =& \frac{1}{2^2}\left(\frac{1}{3} \times 1 \times 2 + 1 \times (n-3)\right) + \frac{1}{2^2}\left(\frac{1}{3} \times 1 \times 2 + 1 \times (n-4)\right) \\ &+ \ldots + \frac{1}{2^{n-1}}\left(\frac{1}{3} \times 1 \times 2\right) + \frac{1}{2^n}\left(\frac{1}{9} \times 1 \times 1\right) \\ =& \frac{1}{2}n - \frac{3}{5} \text{if } n \geq 4 \end{aligned} \tag{2.22}$$

From Equation (2.21) and (2.22), the value of $[K_4]_\gamma$ for index $= 2^{n-1} + 1$ can be illustrated as following.

$$\begin{aligned} [K_4]_\gamma &= [E[K]]_\gamma \\ &= \left[E[\overline{x_{n-1}y_0} + \overline{x_0y_{n-1}} - \frac{1}{2}\beta + \lambda]\right]_\gamma = 0 \end{aligned} \tag{2.23}$$

The error-compensation which is the same as Equation (2.16) for case 2 is shown in Equation (2.24).

$$Carry_{type2,index=2^{n-1}+1} = (< x_{n-2}y_1 >^{q_{n-2}} + < x_{n-3}y_2 >^{q_{n-3}} + \cdots + <x_1y_{n-2} >^{q_1}) + 1, \quad \text{if } \theta_{index=2^{n-1}+1} = n \tag{2.24}$$

The following Figure 2.7 shows the hardware architecture of an 8-bit fixed-width multiplier with the 129th index. The A-A cell in the figure is used to determine the value of $\theta_{index=2^{n-1}+1}$ is equal to n or not.

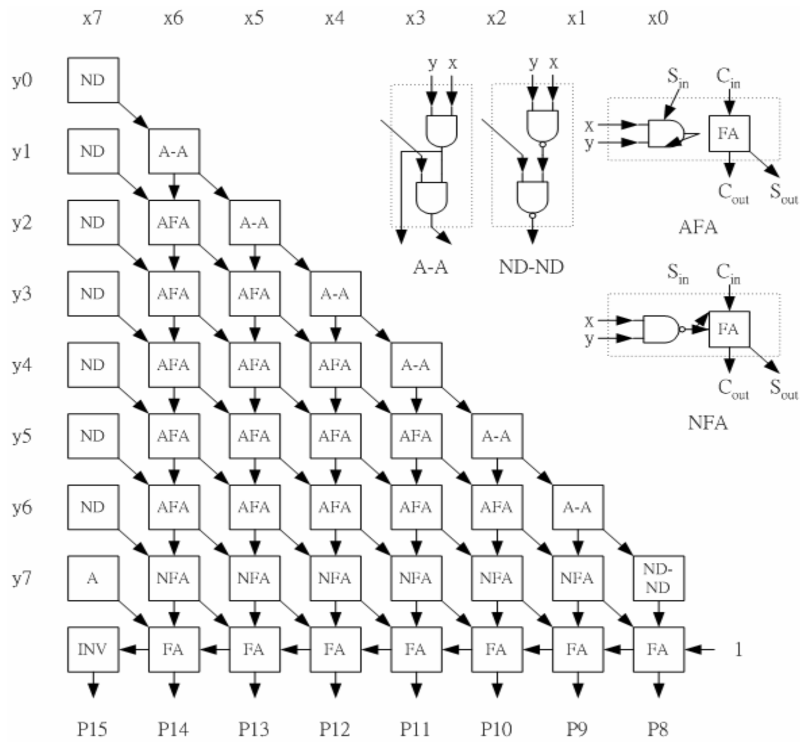


Figure 2.7: Fixed-width multiplier with L. D. Van approach for n = 8

2.2.2 S. J. Jou's Fixed-Width Multiplier

In this section, the S. J. Jou's approach [7] [8] based on Booth encoding technique will be discussed. The S. J. Jou's approach is form by statistical analysis and linear regression analysis. The following Figure 2.8 shows an example of 6x8 Booth encoding multiplier. The partial products are divided into two parts, the eight most significant bits as HP and the six least significant bits as LP. The $Carry_5$ in the figure means the carry from LP to HP. For fixed width purpose, we will truncate the LP and leave only HP. So the signal, $Carry_5$, will be forced to zero and some truncation error may be generated by this procedure. In order to reduce the truncation error, proper error compensation may be add to the HP.

The signal $Carry_5$, which form LP to HP, can be obtained from Equation (2.25). And $\lfloor x \rfloor$ is the floor function, which means the largest integer smaller than or equal to the value of x.

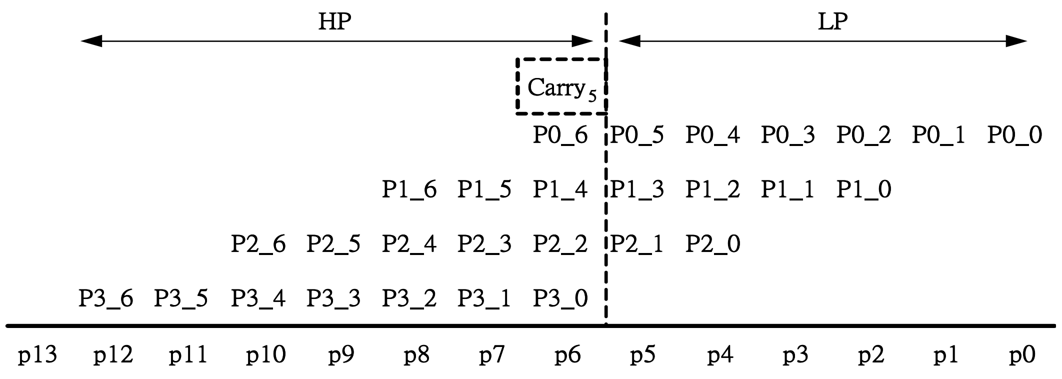

HP
LP
Carry5
P0_6 P0_5 P0_4 P0_3 P0_2 P0_1 P0_0
P1_6 P1_5 P1_4 P1_3 P1_2 P1_1 P1_0
P2_6 P2_5 P2_4 P2_3 P2_2 P2_1 P2_0
P3_6 P3_5 P3_4 P3_3 P3_2 P3_1 P3_0
p13 p12 p11 p10 p9 p8 p7 p6 p5 p4 p3 p2 p1 p0

Figure 2.8: Example of 6 x 8 Booth multipliers

$$\begin{aligned} Carry_5 =& \lfloor 2^{-1}(P_{0,5}+P_{1,3}+P_{2,1})+2^{-2}(P_{0,4}+P_{1,2}+P_{2,0}) \\ &+2^{-3}(P_{0,3}+P_{1,1})+2^{-4}(P_{0,2}+P_{1,0}) \\ &+2^{-5}P_{0,1}+2^{-6}P_{0,0}\rfloor \end{aligned} \tag{2.25}$$

Equation (2.25) can be generalized as Equation (2.26), and τ means the number of bits which will be truncated.

ES 1896

$$\begin{aligned} Carry_{\tau-1} =& \lfloor 2^{-1}(P_{0,\tau-1}+P_{1,\tau-3}+\cdots+P_{\lceil\tau/2\rceil-1,1}) \\ &+2^{-2}(P_{0,\tau-2}+P_{1,\tau-4}+\cdots+P_{\lceil\tau/2\rceil-1,0}) \\ &+\cdots+2^{-(\tau-1)}P_{0,1}+2^{-\tau}P_{0,0}\rfloor \\ &= \lfloor 2^{-1}\beta+\lambda\rfloor \end{aligned} \tag{2.26}$$

The β and λ in Equation (2.26) can represented as follows.

$$\begin{aligned} &\beta = P_{0,\tau-1}+P_{0,\tau-3}+\cdots+P_{\lceil\tau/2\rceil,1} \\ &\lambda = 2^{-2}(P_{0,\tau-2}+P_{1,\tau-4}+\cdots+P_{\lceil\tau/2\rceil,0})+\cdots+2^{-(\tau-1)}P_{0,1}+2^{-\tau}P_{0,0} \end{aligned} \tag{2.27}$$

$\lceil x\rceil$ is the ceiling function, which means the smallest integer larger than or equal to the value of x. The value of β means the number of "1" in the $(\tau-1)_{th}$ column, and the value of λ means the sum of remaining columns. Before we introduce the S. J. Jou's approach, we must assume the probability of each input bit equaling to "1" is 1/2 and the probability of each partial product bit $P_{i,j}$ equaling "1" is $P(P_{i,j})$. From equation (2.27) we can find that the λ is a quite long term. In order to simplify it, we can simplify λ in terms of β and τ. According to the $P(P_{i,j})$ concept, we can rewrite λ as

$$\lambda = \sum_{k=1}^{\tau-1} \frac{1}{2^{k+1}} \times P(P_{i,j}) \times \left\lceil \frac{\tau - k}{2} \right\rceil \tag{2.28}$$

The value of $P(P_{i,j})$ varies with β and τ. By using statistical and linear regression line analysis, $P(P_{i,j})$ can be approximated as follow.

$$P(P_{i,j}) = \frac{0.41}{\tau} \times \beta + 0.58(0.01 \times \tau + 0.37) \tag{2.29}$$

Taking the above two equations into Equation (2.27), the error compensation equation can be shown as Equation (2.30).

$$Carry_{\tau-1} = \left\lfloor 2^{-1}\beta + \left\{ \sum_{k=1}^{\tau-1} \frac{1}{2^{k+1}} \left[\frac{0.41}{\tau}\beta + 0.58(0.01\tau + 0.37) \left\lceil \frac{\tau - k}{2} \right\rceil \right] \right\} + 0.5 \right\rfloor \tag{2.30}$$

And the value of $Carry_{\tau-1}$ for probable τ are listed in Table . It is obviously that the β is the best error-compensation value for any value of τ.

Table 2.3: Probable value of $Carry_{\tau-1}$ for different β and τ

τ	$\beta+2$	$\beta+1$	β	$\beta-1$	$\beta-2$	$\beta-3$	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	$\beta+0.33$

The circuit of 6x8 fixed width multiplier with S. J. Jou's approach is shown in Figure 2.9. The adder cells in the LP are omitted and are replaced by $\beta(P_{0,5} + P_{1,3} + P_{2,1})$.

2.2.3 K. J. Cho's Fixed-Width Multiplier

In this section, the K. J. Cho's approach [9] [10] with Booth encoding technique will be discussed. The partial products of modified Booth encoding multiplier can be divided

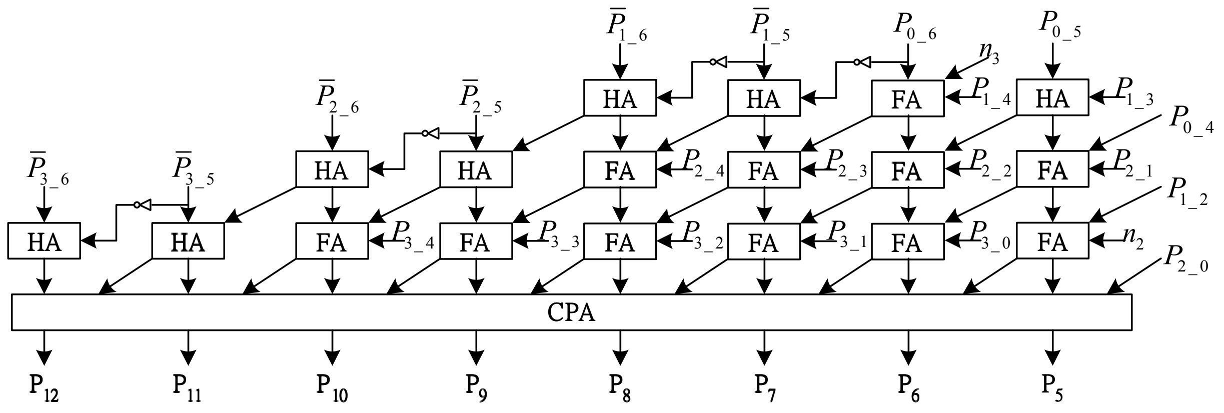


Figure 2.9: 6x8 fixed width Booth multiplier with S. J. Jou's approach

into two parts, HP and LP, as shown in Figure 2.4. And the carry from LP to HP can be represented as

$$Carry_7 = \left\lfloor \frac{1}{2}\beta + \lambda \right\rfloor \tag{2.31}$$

where β and λ are

$$\beta = P_{0,6} + P_{1,4} + P_{2,2} + P_{3,0} \tag{2.32}$$

$$\begin{aligned} \lambda \;=\; & 2^{-2}(P_{0,5} + P_{1,3} + P_{2,1}) + 2^{-3}(P_{0,4} + P_{1,2} + P_{2,0} + n_2) \\ & +2^{-4}(P_{0,3} + P_{1,1}) + 2^{-5}(P_{0,2} + P_{1,0} + n_1) \\ & +2^{-6}(P_{0,1}) + 2^{-7}(P_{0,0} + n_0) \end{aligned} \tag{2.33}$$

The value of β is the sum of LP_{major} and the value λ is the sum of LP_{minor}, as shown in Figure 2.10. The adder cells in LP_{minor} are discarded and replaced by approximate carry, shown in Equation (2.34).

$$Carry_{\tau-1} = C_E[\frac{1}{2}\beta + C_A[\lambda]] \tag{2.34}$$

Where the $C_E[x]$ means the exact carry, and the $C_A[x]$ means the approximate carry of x.

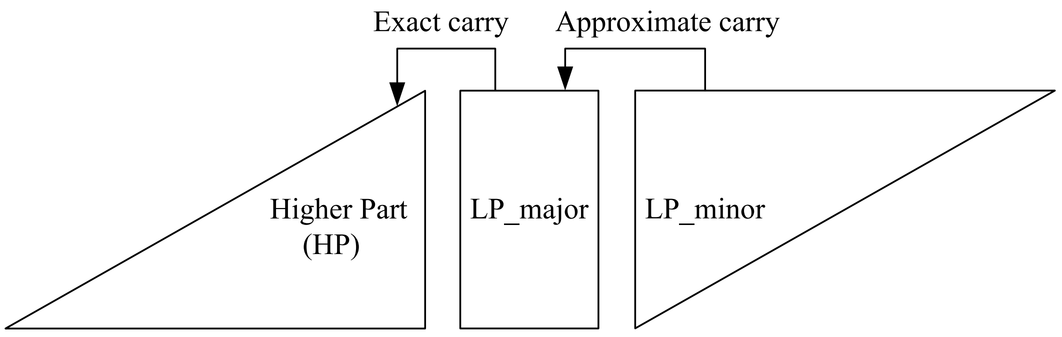


Figure 2.10: The structure of K. J. Cho's scheme

The procedure proposed by K. J. Cho can be summarized as follows.

1. Divide LP into LP_{major} and LP_{minor}.

2. Compute the approximate carry value of LP_{minor}.

3. Add the approximate carry to LP_{major}.

4. The carry from the add operation in step 3 is the error compensation bias.

In order to find the carry estimation bias in the LP, we first define y''_i as

$$y''_i = \begin{cases} 1, \text{if } y'_i \neq 0 \\ 0, otherwise \end{cases} \tag{2.35}$$

From Table 2.1, we can see that y''_i can be computed from

$$y''_i = X_{i,sel} \vee 2X_{i,sel} \tag{2.36}$$

For example, there are only four case for $y''_3y''_2y''_1y''_0 = 1000$, which are 1000, 2000, $\bar{1}000$ and $\bar{2}000$. For another example, consider the case of $y''_3y''_2y''_1y''_0 = 0001$, and there are three possible cases are shown as follows.

$$\begin{aligned} 00000001(0) &\rightarrow y''_3y''_2y''_1y''_0 = 0001 \\ 11111110(0) &\rightarrow y''_3y''_2y''_1y''_0 = 000\bar{2} \\ 11111111(0) &\rightarrow y''_3y''_2y''_1y''_0 = 000\bar{1} \end{aligned} \tag{2.37}$$

And the corresponding cases of partial products are

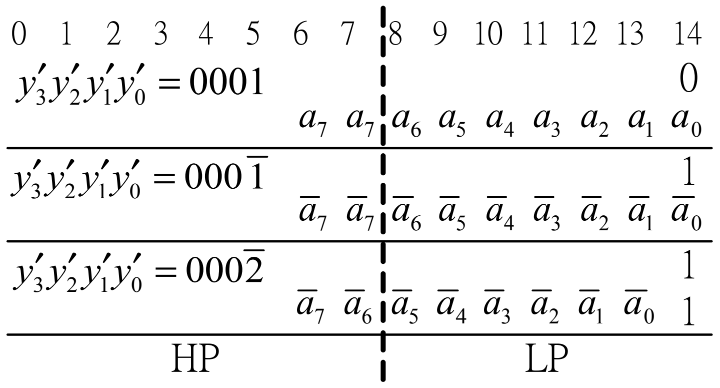

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
$y'_3y'_2y'_1y'_0 = 0001$
0
$a_7\ a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0$
$y'_3y'_2y'_1y'_0 = 000\bar{1}$
1
$\bar{a}_7\ \bar{a}_7\ \bar{a}_6\ \bar{a}_5\ \bar{a}_4\ \bar{a}_3\ \bar{a}_2\ \bar{a}_1\ \bar{a}_0$
$y'_3y'_2y'_1y'_0 = 000\bar{2}$
1
$\bar{a}_7\ \bar{a}_6\ \bar{a}_5\ \bar{a}_4\ \bar{a}_3\ \bar{a}_2\ \bar{a}_1\ \bar{a}_0\ 1$
HP
LP

Figure 2.11: The partial product of $y''_3y''_2y''_1y''_0 = 0001$

Assumed that the probability of each input bit equaling to "1" is 1/2, so

$$E[a_i] = E[b_i] = \frac{1}{2} \tag{2.38}$$

Thus, the rounded value of $E[\lambda]$ for each case in Figure 2.11 can be computed as follows

$$\{E[\lambda]\}_r = \begin{cases} 0, & \text{for } y'_3y'_2y'_1y'_0 = 0001 \\ 1, & \text{for } y'_3y'_2y'_1y'_0 = 000\bar{1}, 000\bar{2} \end{cases} \tag{2.39}$$

Where the $\{x\}$ is the rounding operation of x.

Notice that $E[\lambda]$ is always zero when $y''_3y''_2y''_1y''_0 = 1000$. That is, no matter how the y'_3 changes, $E[\lambda]$ will not vary with it. Because no partial product elements corresponding to y'_3 is included in λ, as shown in Figure 2.4. In general, the element of the partial product corresponding to $y'_{\frac{n}{2}-1}$ will not be included in LP_minor(λ) for any input width "n".

From the previous discussion, we can see that the value of $E[\lambda]$ is determined by the partial product of LP_minor. The K. J. Cho's method determines the error compensation more accurately, because the carry from LP_minor to LP_major is replaced by $\{E[\lambda]\}$. We only need to calculate the value of $\{E[\lambda]\}$ from $y''_{\frac{n}{2}-2}y''_{\frac{n}{2}-3}\cdots y''_0$. The procedure of K. J. Cho's approach is explained in the following example.

Example 1 : For a 10x10 fixed width Booth multiplier, Table 2.4 shows the value of $\{E[\lambda]\}$ corresponding to all conditions of $y''_3y''_2y''_1y''_0$. Notice that y''_4 is not included in Table 2.4. Because no partial product corresponding to y''_4 has effect on LP_minor.

Table 2.4: Rounded value of $E[\lambda]$ for bit width 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)

Consider the Table 2.4, we can see that the necessary number of carry signal is two. Thus, two signals(LP_carry_0 & LP_carry_1) shown in Table 2.5 are needed to represent the rounded value of $E[\lambda]$. From Table 2.4 and Table 2.5, Karnaugh map representation for these two signal can be obtained in Figure 2.12. The value of approximate carry in Figure 2.12 can be determined by using probability analysis. For example, for $y_3''y_2''y_1''y_0'' = 0001$, P[$\{E[\lambda] = 0\}$]=4/12 and P[$\{E[\lambda] = 1\}$]=8/12. Thus, the approximate value of carry is determined to be 1. So the LP_carry_0 and LP_carry_1 signals can be simplified from each Karnaugh map as

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

$$\begin{aligned} \text{LP\_carry\_0} &= y_3'' + y_2'' + y_1'' + y_0'' \\ \text{LP\_carry\_1} &= y_3''y_2''(y_1'' + y_0'') + y_1''y_0''(y_3'' + y_2'') \end{aligned} \tag{2.40}$$

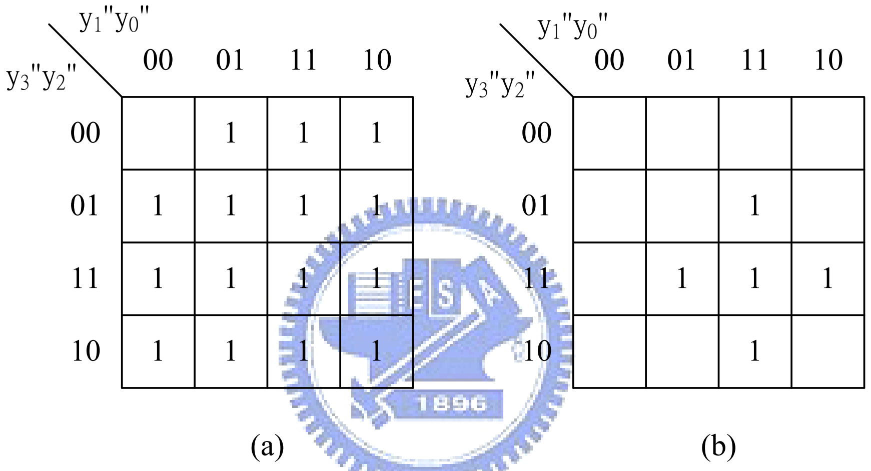


Figure 2.12: Karnaugh map for (a) LP_carry_0 and (b) LP_carry_1 for bit width n = 10

Figure 2.13 shows the circuit of Equation (2.40), which is the approximate carry from LP_minor to LP_major. Thus, the carry signal result from LP_major are added to HP as the error compensation bias.

The procedure of example 1, called approximate carry generation procedure (ACGP) I, is listed as follows.

1. For input bit width n, the number of approximate carry signal is determined by $N_{AC} = \lfloor n/4 \rfloor$.

2. Denote the approximate carry signals in order. For example, LP_carry_0, LP_carry_1, . . ., LP_carry_$(N_{AC} - 1)$.

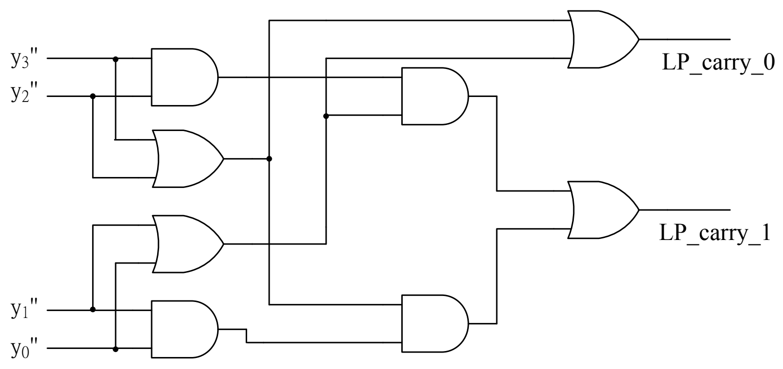


Figure 2.13: Approximate carry circuit for n = 10

3. Calculate the rounded values of $E[\lambda]$ for all cases of $y''_{\frac{n}{2}-2} + y''_{\frac{n}{2}-3} \cdots y''_0$
4. Generate carry generation circuit by applying Karnaugh map to the result of step 3.

It is very time consuming to perform exhaust simulation for large bit width n. So a statistical analysis for obtaining approximate carry will be introduced as following.

Given that y''_i is 1, it can be shown that $E[P_{i,j}] = 1/2$. If $y''_2 y''_1 y''_0 = 100$ in Figure 2.4, $E[\lambda]$ can be calculate by Equation (2.31) as follows:

$$\begin{aligned} E[\lambda] &= E[2^{-1}(P_{2,1} + 2^{-2}(P_{2,0} + n_2))] \\ &= 2^{-1}E[P_{2,1}] + 2^{-2}(E[P_{2,0}] + E[n_2]) \\ &= 2^{-1} + 2 - 2(2^{-1} + 2^{-1}) \\ &= 2^{-1} \end{aligned} \tag{2.41}$$

By the same way, the values of $E[\lambda]$ for $y''_2 y''_1 y''_0 = 010$ and $y''_2 y''_1 y''_0 = 001$ are both equal to 2^{-1}. For n = 8, $E[\lambda]$ can be expressed as

$$E[\lambda] = 2^{-1}(y''_2 + y''_1 + y''_0) \tag{2.42}$$

In general case, $E[\lambda]$ can be represented as

$$E[\lambda] = 2^{-1} \sum_{i=0}^{n/2-2} y''_i \tag{2.43}$$

Where n in the above equation is the bit width of multiplier. The following shows an example for n = 10.

Example 2 : Consider the case of bit width n = 10, the rounded value $E[\lambda]$

$$E[\lambda] = 2^{-1} \sum_{i=0}^{10/2-2} y_i'' = 2^{-1}(y_3'' + y_2'' + y_1'' + y_0'') \tag{2.44}$$

The maximum rounded value of $E[\lambda]$ is 2. So, two signals are needed for the rounded value. Notice that if only one y_i'' in Equation (2.44) equals to 1, the rounded value is 1. And if three of more y_i'' equal to 1, the rounded value is 2. Means that the number of carry signal can be determined as follows.

$$E[\lambda] = \lfloor 2^{-1}(y_3'' + y_2'' + y_1'' + y_0'') \rfloor \tag{2.45}$$

By using this scheme, the approximate carry generation circuit for $n = 10$ and $n = 14$ can be obtained and shown in Figure 2.14.

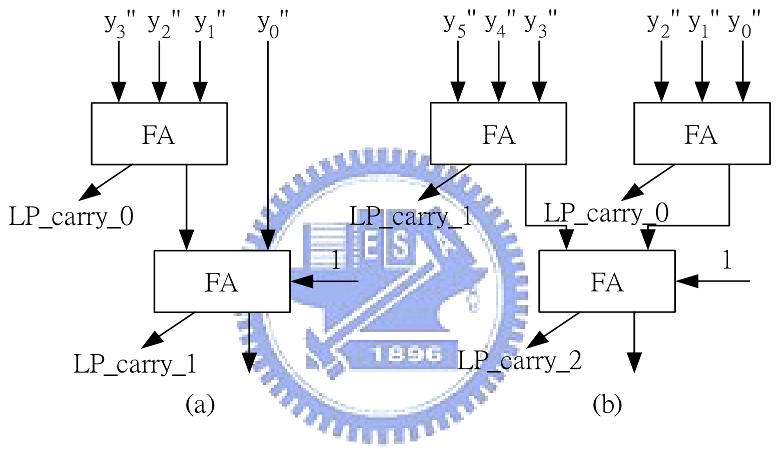


Figure 2.14: Approximate carry generation circuit for (a) n = 10 (b) n = 14

The procedure of example 2, called approximate carry generation procedure (ACGP) II, is listed as follows.

1. Divide $\{y''_{\frac{n}{2}-2} y''_{\frac{n}{2}-3} \cdots y_0''\}$ into groups of three signals. If the number of signal in the set is $3N + k (k = 1, 2)$, then the last group has only k signals.

2. N FAs are required for $3N$ signals. For $k = 2$, the last two signals needs a HA. For $k = 1$, it needs only a carry input at the next stage. The $N(N + 1$ for $K = 2)$ carry signals form the approximate carry signals.

3. The sum signals from step 2 are added as step 2. Then the carry signals from these adders are approximate carry signals. The new sum signals generated from present stage are passed to the next stage.

4. Repeat step 2 until only one sum signal remains.

5. Add 1 to the last adder.

The circuit for bit width $n = 8$ with K. J. Cho's approach is shown in Figure 2.15. We can see that the lower part of adder cells are reduced and replaced by carry approximation signals (LP_carry_0 and LP_carry_1), which are generated from Figure 2.14.

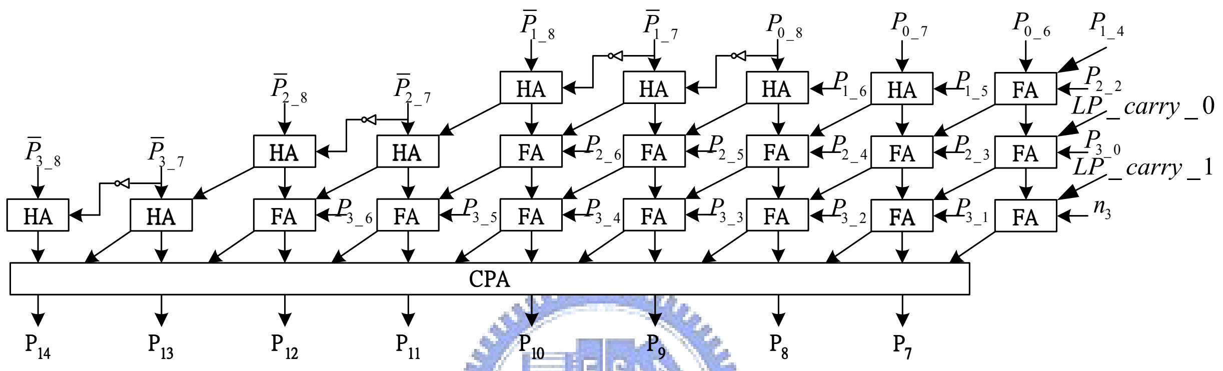


Figure 2.15: Fixed-width multiplier with K. J. Cho's approach for bit width n = 8

Chapter 3

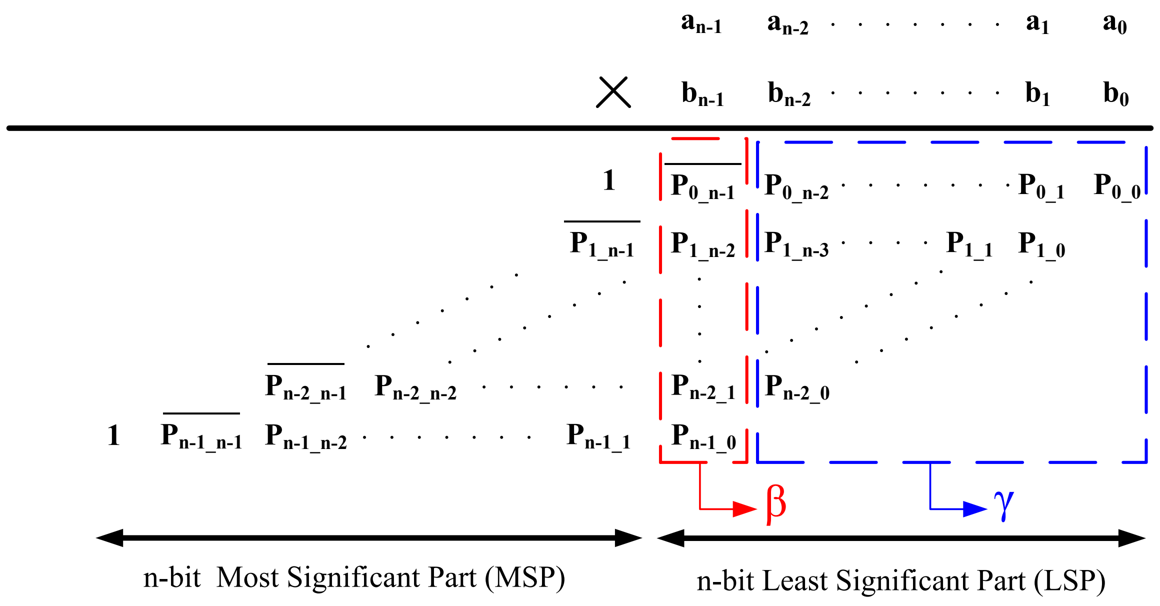
Statistical Analysis of Truncated Width Multiplier

3.1 Analysis of Carry Estimation for Baugh-Wooley Multiplier

Consider the n-bit Baugh-Wooly 2's complement multiplier. Suppose the two input defined as $A = -a_{n-1}2^{n-1} + \sum_{j=0}^{n-2} a_j 2^j$ and $B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$, where $a_j, b_j \in \{0, 1\}$, then the product is

$$\begin{aligned} A \times B = {} & a_{n-1}b_{n-1}2^{n-2} + \sum_{j=0}^{n-2}\sum_{i=0}^{n-2} a_j b_i 2^{j+i} \\ & + 2^{n-1}\left(-2^{n-1} + \sum_{j=0}^{n-2} \overline{a_j b_{n-1} 2^j} + 1\right) + 2^{n-1}\left(-2^{n-1} + \sum_{i=0}^{n-2} \overline{a_{n-1} b_j 2^i} + 1\right) \end{aligned} \tag{3.1}$$

Define $P_{ij} = a_j b_i$ and $\overline{P_{ij}}$ denote the bit inverse of P_{ij}. The partial product can be shown as Figure 3.1. To keep the fixed width property, the input width and output width must be the same. So we should keep the n-bit most significant part (MSP). Different truncation method will result in different truncation error and computation complexity. The post-truncate method remove the last n-bit least significant part (LSP) after the overall summation is completed, produce the smaller truncation error. However, its requires the most computation complexity. On the other hand, the direct-truncation method discard the computation of LSP part, which leads large truncation error and has the least computation complexity.



a_{n-1} a_{n-2} a_1 a_0
b_{n-1} b_{n-2} b_1 b_0
1
P_{0_n-1} P_{0_n-2} P_{0_1} P_{0_0}
P_{1_n-1} P_{1_n-2} P_{1_n-3} P_{1_1} P_{1_0}
P_{n-2_n-1} P_{n-2_n-2} P_{n-2_1} P_{n-2_0}
1 P_{n-1_n-1} P_{n-1_n-2} P_{n-1_1} P_{n-1_0}
β
γ
n-bit Most Significant Part (MSP)
n-bit Least Significant Part (LSP)

Figure 3.1: Partial product for $A \times B$ n-bit fixed width Baugh-Wooley multiplier

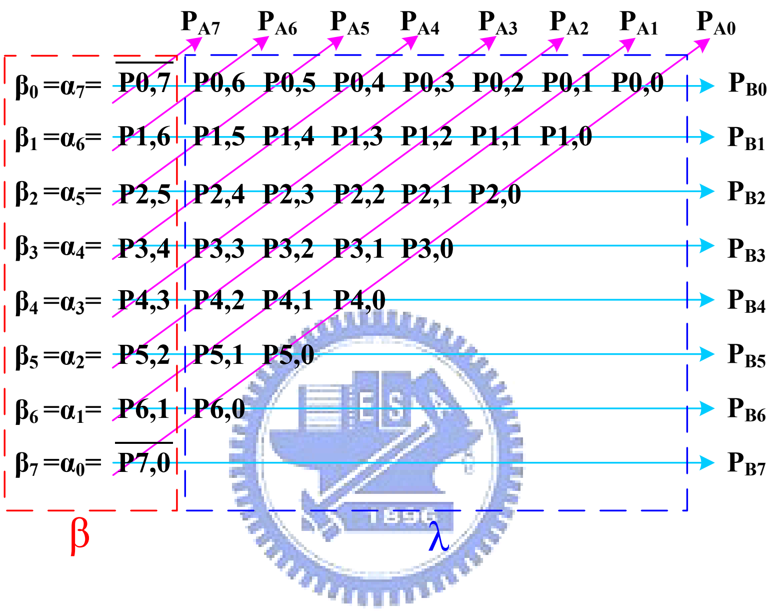
1896

Let *Sum* denote the MSP part of $A \times B$, and from Figure 3.1 and Equation 3.1, the product of post-truncated method can be shown as

$$\begin{aligned} A \times B_{post\_truncation} &= Sum + 2^n \left[\frac{1}{2} \sum_{i=0}^{n-1} P_{i,n-i-1} + \sum_{i=0}^{n-1} \sum_{j=0}^{n-i-2} P_{i,j} 2^{j-n} \right]_r \\ &= Sum + 2^n \left[\frac{\beta}{2} + \lambda \right]_r \\ &= Sum + 2^n \sigma \end{aligned} \tag{3.2}$$

where $\sigma = \left[\frac{\beta}{2} + \lambda\right]_r$ is the carry-in to the *Sum*. The $[x]_r$ operation in Equation 3.2 is the round operation, which rounds x to its nearest number. From Equation 3.2 we can see β domains the value of carry-in σ. So we can utilize the information of β to compensate the truncation error.

Define $\alpha_j = P_{n-j-1,j}$ and $\beta_i = P_{i,n-i-1}$ [11] first, then the element of partial product $P_{i,j} = a_j b_i$ depends both on a_j and b_i, and $P_{i,j}$ is related to $\{P_{0,j}, P_{1,j}, \ldots, P_{n-1,j}\}$ and $\{P_{i,0}, P_{i,1}, \ldots, P_{i,n-1}\}$. We can define $P_{A_j} = \{P_{0,j}, P_{1,j}, \ldots, P_{n-1,j}\}$ for $j = 0, 1, \ldots, n-1$ and $P_{B_i} = \{P_{i,0}, P_{i,1}, \ldots, P_{i,n-1}\}$ for $i = 0, 1, \ldots, n-1$, so the partial product of $A \times B$ will be composed of $P_{A_0}, P_{A_1}, \ldots, P_{A_{n-1}}$ or $P_{B_0}, P_{B_1}, \ldots, P_{B_{n-1}}$. Figure 3.2 shows an example for $n = 8$ and the dependency among the partial products with $\alpha_j \in P_{A_j}$ and $\beta_i \in P_{B_i}$. Thus, the element α_j (or β_i) that constitutes $\beta = \sum_{j=0}^{n-1} \alpha_j = \sum_{i=0}^{n-1} \beta_i$ contains the information of all $P_{i'j} \in P_{A_j}$ (or $P_{ij'} \in P_{B_i}$). From above discussion, we can find that λ strongly depends on β and can be estimated from the information implied in β. Three carry estimation method will be presented in the following section. Without loss of generality, the input of multiplier should be independent and uniform distributed, means that the probability $P(a_j b_i) = P(a_j)P(b_i)$, $P(a_j = 0) = P(a_j = 1) = \frac{1}{2}$ and $P(b_i = 0) = P(b_i = 1) = \frac{1}{2}$.



PA7 PA6 PA5 PA4 PA3 PA2 PA1 PA0
β0=α7= P0,7 P0,6 P0,5 P0,4 P0,3 P0,2 P0,1 P0,0 PB0
β1=α6= P1,6 P1,5 P1,4 P1,3 P1,2 P1,1 P1,0 PB1
β2=α5= P2,5 P2,4 P2,3 P2,2 P2,1 P2,0 PB2
β3=α4= P3,4 P3,3 P3,2 P3,1 P3,0 PB3
β4=α3= P4,3 P4,2 P4,1 P4,0 PB4
β5=α2= P5,2 P5,1 P5,0 PB5
β6=α1= P6,1 P6,0 PB6
β7=α0= P7,0 PB7
β
λ

Figure 3.2: The LSP of partial product for an n = 8 Baugh-Wooley multiplier. $P_{A_j} = \{P_{0,j}, P_{1,j}, \ldots, P_{n-1,j}\}$ and $P_{B_i} = \{P_{i,0}, P_{i,1}, \ldots, P_{i,n-1}\}$.

3.1.1 Type I : Carry Estimation Conditioned on a_j or b_i

Consider any element P_{ij} of partial product, it is generated from $a_j \times b_i$, so the conditional expectation $E[P_{ij}|a_j = 0] = 0$ and $E[P_{ij}|a_j = 1] = \frac{1}{2}$. Therefore, any elements $P_{ij} \in P_{A_j}$ can be estimated as $a_j \cdot E[P_{ij}|a_j = 1] + (1 - a_j) \cdot E[P_{ij}|a_j = 0] = \frac{a_j}{2}$ Then λ can be estimated as

$$\begin{aligned}\lambda_{BW_I} &= \sum_{j=0}^{n-2}\sum_{k=2}^{n-j}\frac{a_j}{2}2^{-k} \\ &= \frac{1}{2}\sum_{j=0}^{n-2}a_j\sum_{k=2}^{n-j}2^{-k} \\ &= \frac{1}{4}\sum_{j=0}^{n-2}a_j - \frac{1}{4}\sum_{j=0}^{n-2}a_j2^{-(n-j-1)}\end{aligned} \tag{3.3}$$

The second term of Equation (3.3) can be approximated (rounding) by its expectation

$$\begin{aligned}\left[E\left[\frac{1}{4}\sum_{j=0}^{n-2}a_j2^{-(n-j-1)}\right]\right]_r &= \left[\frac{1}{8}\sum_{j=0}^{n-2}2^{-(n-j-1)}\right]_r \\ &= \left[\frac{1}{8}\left(2^{-(n-1)}+2^{-(n-2)}+\ldots+2^{-1}\right)\right]_r \\ &= \left[\frac{1}{8}\left(1-2^{-(n-1)}\right)\right]_r \\ &= 0\end{aligned} \tag{3.4}$$

So the carry in to MSP can be estimated as follows.

$$\sigma^{(a)}_{BW_I} = \left[\frac{1}{2}\left(\beta+\frac{1}{2}\sum_{j=0}^{n-2}a_j\right)\right]_r \tag{3.5}$$

By using similar procedure, we can get the carry-in from b_i

$$\sigma^{(b)}_{BW_I0} = \left[\frac{1}{2}\left(\beta+\frac{1}{2}\sum_{i=0}^{n-2}b_i\right)\right]_r \tag{3.6}$$

From the initial assumption, A and B are uniform distributed, so Equation (3.5) and Equation (3.6) will result in the same carry-in. But the computation complexity will apparently grow as n increases.

3.1.2 Type II : Carry Estimation Conditioned on α_j or β_i

Now consider $\alpha_j = P_{n-1-j,j}$ and $\beta = P_{i,n-i-1}$ in Figure 3.1, notice that $\alpha_j = \beta_{n-j-1}$. The expectation values of any partial product conditioned on α_j are $E[P_{ij}|\alpha_j = 0] = \frac{1}{6}$ and $E[P_{ij}|\alpha_j = 1] = \frac{1}{2}$. So P_{ij} can be determined by the conditional expectation $\alpha_j \cdot E[P_{ij}|\alpha_j = 1] + (1-\alpha_j)\cdot E[P_{ij}|\alpha_j = 0] = \frac{\alpha_j}{3}+\frac{1}{6}$.

$$\begin{aligned}
\lambda_{BW_{II}}^{(\alpha)} &= \sum_{j=0}^{n-2}\left(\frac{\alpha_j}{3}+\frac{1}{6}\right)\sum_{k=2}^{n-j}2^{-k} \\
&= \frac{1}{2}\sum_{j=0}^{n-2}\left(\frac{\alpha_j}{3}+\frac{1}{6}\right)\left(1-2^{-(n-1-j)}\right) \\
&\approx \frac{1}{2}\sum_{j=0}^{n-2}\left(\frac{\alpha_j}{3}+\frac{1}{6}\right) \\
&= \frac{1}{6}\sum_{j=0}^{n-2}\alpha_j+\frac{(n-1)}{12} \\
&= \frac{(\beta-\alpha_{n-1})}{6}+\frac{(n-1)}{12} \quad \left(\beta=\sum_{j=0}^{n-1}\alpha_j\right)
\end{aligned} \tag{3.7}$$

The Type II carry estimation denoted by $\sigma_{BW_{II}}^{(\alpha)}$ is

$$\begin{aligned}
\sigma_{BW_{II}}^{(\alpha)} &= \left[\frac{\sum_{j=0}^{n-1}\alpha_j}{2}+\lambda_{BW_{II}}^{(\alpha)}\right]_r \\
&= \left[\sum_{j=1}^{n-2}\alpha_j+\frac{\alpha_0}{2}+\frac{\alpha_{n-1}}{2}+\lambda_{BW_{II}}^{(\alpha)}-\frac{1}{2}\sum_{j=1}^{n-2}\alpha_j\right]_r \\
&= \sum_{j=1}^{n-2}\alpha_j+[\delta]_r
\end{aligned} \tag{3.8}$$

where $\delta=\frac{\alpha_0}{2}+\frac{\alpha_{n-1}}{2}+\lambda_{BW_{II}}^{(\alpha)}-\frac{1}{2}\sum_{j=1}^{n-2}\alpha_j$.

From the same procedure, the carry-in estimated by β_i can also be conditioned by the expectation value of $E\left[P_{ij}|\beta_i\right]$, which is shown as follows.

$$\lambda_{BW_{II}}^{(\beta)}=\frac{(\beta-\beta_{n-1})}{6}+\frac{(n-1)}{12} \tag{3.9}$$

3.1.3 Type III : Carry Estimation Conditioned on α_j and β_i

Since P_{ij} depends on α_j and β_i, we can estimate value of P_{ij}, denoted by $\hat{P}_{ij}$, by the conditional expectation $E[P_{ij}|\alpha_j\beta_i]$. Notice that $\alpha_j=\beta_{n-j-1}$.

$$\begin{aligned}
\widehat{P}_{ij} &= \alpha_j\beta_i\cdot E\left[P_{ij}|\alpha_j\beta_i=1\right]+(1-\alpha_j\beta_i)\cdot E\left[P_{ij}|\alpha_j\beta_i=0\right] \\
&= \beta_{n-j-1}\beta_i\cdot E\left[P_{ij}|\beta_{n-j-1}\beta_i=1\right]+(1-\beta_{n-j-1}\beta_i)\cdot E\left[P_{ij}|\beta_{n-j-1}\beta_i=0\right]
\end{aligned} \tag{3.10}$$

Table 3.1 lists the probabilities of $P\left(P_{ij}=1|\alpha_j\beta_i=0\right)$ and $P\left(P_{ij}=1|\alpha_j\beta_i=1\right)$. From Equation 3.10 we can see the carry estimation of Type III, called $\lambda^{(\alpha\beta)}_{BW_{III}}$, is related to the vector of $\{\beta_0,\beta_1,\ldots,\beta_{n-1}\}$. So, comparing Type II and Type III can be thought as one-dimension and two-dimension estimation. Since two-dimension estimation contains more information of the partial product, it will result in less truncation error. From the Type II estimation shown in Equation (3.9), $\lambda^{(\beta)}_{BW_{III}}$ should be related to $(\beta-\beta_{n-1})$ since they have both conditioned on β. In order to express $\lambda^{(\alpha\beta)}_{BW_{III}}$ simply, we define θ as follows.

$$\theta=\sum_{i=0}^{n-1}\beta_i 2^i=\sum_{i=0}^{n-1}\alpha_{n-i-1}2^i \tag{3.11}$$

Table 3.1: Conditional probability of the partial products for Type III

Probability	$i\in\{1,2,\ldots,n-2\}$ and $j\in\{1,2,\ldots,n-2\}$	$i\in\{0,n-1\}$ and $j\in\{1,2,\ldots,n-2\}$ or $j\in\{0,n-1\}$ and $i\in\{1,2,\ldots,n-2\}$	$i\in\{0,n-1\}$ and $j\in\{0,n-1\}$
$Pr\left(P_{ij}=1\|\alpha_j\beta_i=0\right)$	$\frac{1}{5}$	$\frac{3}{13}$	$\frac{3}{7}$
$Pr\left(P_{ij}=1\|\alpha_j\beta_i=1\right)$	1	$\frac{1}{3}$	$\frac{1}{9}$

Thus, there are two important factor for $\lambda^{(\alpha\beta)}_{BW_{III}}$, the vector of $\{\beta_0,\beta_1,\ldots,\beta_{n-1}\}$ and $(\beta-\beta_{n-1})$. Note that several combinations of $\{\beta_0,\beta_1,\ldots,\beta_{n-1}\}$ will result in the same $(\beta-\beta_{n-1})$ value. From Figure 3.3 we can observe that $\lambda^{(\alpha\beta)}_{BW_{III}}$ and $(\beta-\beta_{n-1})$ are highly correlated. If any two different θ lead to the same $(\beta-\beta_{n-1})$, the corresponding $\lambda^{(\alpha\beta)}_{BW_{III}}$ will also be similar. In other words, once $(\beta-\beta_{n-1})$ is given, we can find the maximum and the minimum number of λ, called λ_{max} and λ_{min}, to meet the corresponding $(\beta-\beta_{n-1})$. Moreover, the λ_{max} and λ_{min} can be also viewed as the boundaries of the carry estimation.

We can rewrite the σ in Equation 3.2 by Equation 3.12, note that $\left[\frac{\beta}{2}\right]_r=\left\lfloor\frac{\beta+1}{2}\right\rfloor_r$.

$$\begin{aligned}\sigma&=\left[\frac{\beta}{2}+\lambda\right]_r\\&=\left[\frac{\beta+1}{2}+\frac{2\lambda-1}{2}\right]_r\\&\approx\left\lfloor\frac{\beta+1}{2}\right\rfloor+\left[\frac{2\lambda-1}{2}\right]_r\end{aligned}\tag{3.12}$$

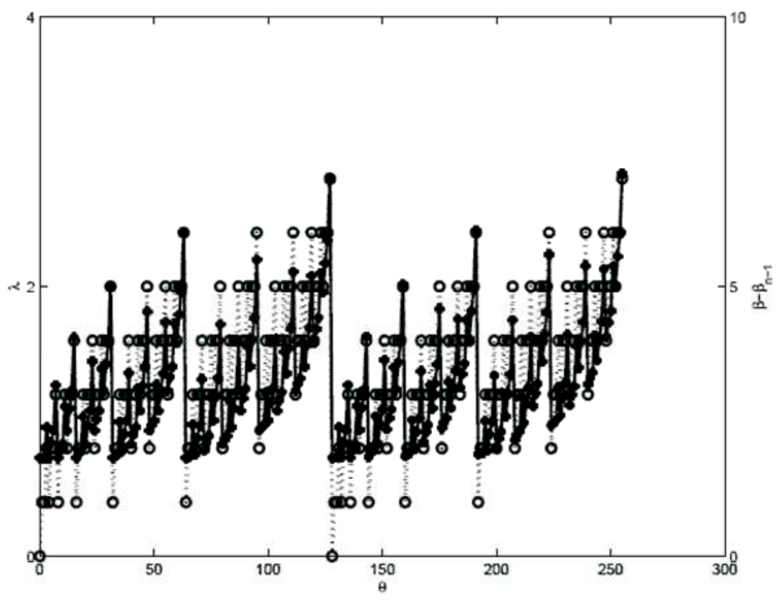


Figure 3.3: Relationship between $\lambda_{BW_{III}}$, θ, and $\beta - \beta_{n-1}$ for $n = 8$. The solid line and the dash line represent the value of $\lambda_{BW_{III}}$ and $\beta - \beta_{n-1}$

Subsequently, Type III carry estimation of value $(2\lambda - 1)$ will be estimated as Equation (3.13).

$$2\lambda^{(\alpha\beta)}_{BW_{III}} - 1 \approx \left[\frac{1}{2}\left((2\lambda_{max} - 1) + (2\lambda_{min} - 1)\right)\right]_r \tag{3.13}$$

Figure 3.4 shows an 8-bit example of $(2\lambda_{max} - 1)$ and $(2\lambda_{min} - 1)$ from Figure 3.3. For $\beta - \beta_{n-1} = 0, 1, 2, \ldots, 7$ in this example, $2\lambda^{(\alpha\beta)}_{BW_{III}} - 1$ will be 0,1,1,2,2,3,4,4, respectively. However, it is difficult to determine $2\lambda^{(\alpha\beta)}_{BW_{III}} - 1$ by Equation (3.12) and (3.13). So a look-up table method is required for Type III estimation. For example, consider the case of $n = 8$, $\left[2\lambda^{(\alpha\beta)}_{BW_{III}} - 1\right]_r = 0, 1, 1, 2, 2, 3, 4, 4 \approx \left\lfloor \frac{1}{2}(\beta - \beta_{n-1} + 1) \right\rfloor$. From the similar procedure, $\left[2\lambda^{(\alpha\beta)}_{BW_{III}} - 1\right]_r$ for different n is shown in Table 3.2.

3.2 Analysis of Carry Estimation for modified Booth Multiplier

Radix-4 Booth multiplier converts the multiplication of $A \times B$ into $A \times y$ by Table 3.3, where $B = \{b_{n-1}, b_{n-2}, \ldots, b_0\}$ and $y = \left\{y_{\lceil n/2\rceil-1}, y_{\lceil n/2\rceil-2}, \ldots, y_0\right\}$. Note that the b_{-1} is always 0 and the number of rows in partial product are reduced into $\left\lceil \frac{n}{2} \right\rceil$. Same as

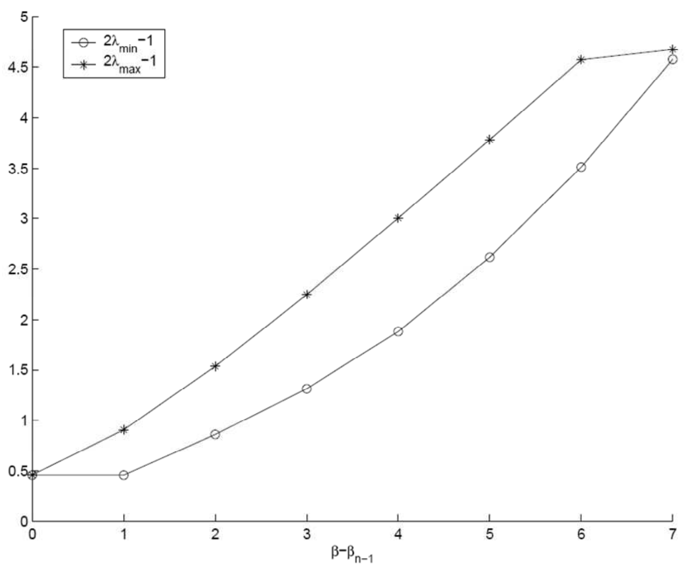


Figure 3.4: All cases of $\beta - \beta_{n-1}$ for λ_{max} and λ_{min} at $n = 8$

Baugh-Wooley multiplier, the partial product of modified Booth multiplier, $P_{ij} = a_j y_i$, can be also divided into two parts, n-bit MSP and n-bit LSP. The most significant column of LSP is represented by β and the summation of rest $(n-1)$ columns is denoted as λ. The β dominates the carry-in to the MSP, so the compensation for Booth multiplier can be expressed as a function of β.

Similar to Baugh-Wooley multiplier, we define $P_{Aj} = \{P_{0,j}, P_{1,j}, \ldots, P_{n,j}\}$, $P_{Bi} = \{P_{i,0}, P_{i,1}, \ldots, P_{i,n-1}\}$, $\alpha_j = P_{n-1-j,j}$ and $\beta_i = P_{i,n-1-i}$. Since $\{\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\}$ and $\left\{\beta_0, \beta_1, \ldots, \beta_{\lceil n/2 \rceil - 1}\right\}$ contains the information of $a_0, a_1, \ldots, a_{n-1}$ and $\beta_0, \beta_1, \ldots, \beta_{\lceil n/2 \rceil - 1}$, respectively, we can estimate P_{ij} by the conditional expectations of $E\left[P_{ij} | y_i\right]$, $E\left[P_{ij} | \alpha_j\right]$ and $E\left[P_{ij} | \beta_i\right]$. The three estimation method will be presented as follows.

3.2.1 Carry estimation by Conditioning on y_i

For the encoded Booth multiplication $A \times y_i$, if $y_i = 0$, all elements in the row will be zero. So the carry should be estimated only when $y_i \neq 0$. All elements P_{ij} can be estimated by conditional expectation by $E\left[P_{ij} | y_i \neq 0\right]$. From Table 3.4 and Table 3.5 we can verify $E\left[P_{ij} | y_i \neq 0\right] = \frac{1}{2}$. Moreover, let $y_{i''}$ denote the event $y_i \neq 0$, then P_{ij} can be estimated by $\frac{y_{i''}}{2}$. Besides, the n_i in Booth encoding equals to 1 only when $y_i < 0$. It also has conditional expectations $E\left[n_i | y_{i''} = 1\right] = \frac{1}{2}$ and $E\left[n_i | y_{i''} = 0\right] = 0$. Hence, n_i can also

Table 3.2: The estimated $2\lambda_{BW_{III}}^{(\alpha\beta)}-1$ for different bit width n

Bit-width (n)	$2\lambda_{BW_{III}}^{(\alpha\beta)} - 1$
8	$\left\lfloor \frac{1}{2}(\beta - \beta_{n-1} + 1) \right\rfloor$
10	0, if $\beta - \beta_{n-1} = 0$ $\left\lfloor \frac{1}{2}(\beta - \beta_{n-1} + 2) \right\rfloor$, otherwise
12	$\left\lfloor \frac{1}{2}(\beta - \beta_{n-1} + 2) \right\rfloor$
14	$\left\lfloor \frac{1}{2}(\beta - \beta_{n-1} + 3) \right\rfloor$
16	$\left\lfloor \frac{1}{2}(\beta - \beta_{n-1} + 3) \right\rfloor$

estimated by $\frac{y_{i''}}{2}$. Then the estimated value λ will be

$$\begin{aligned}\lambda_{Booth_I} &= \sum_{i=0}^{\lceil n/2\rceil-1}\left(\frac{y_{i''}}{2}\sum_{k=2}^{n-2i}2^{-k} + \frac{y_{i''}}{2}2^{-(n-2i)}\right) \\ &= \frac{1}{4}\sum_{i=0}^{\lceil n/2\rceil-1} y_{i''}\end{aligned} \tag{3.14}$$

1896

The carry in to MSP is $\sigma_{Booth_I} = \left[\frac{\beta}{2} + \lambda\right]_r = \left[\frac{1}{2}\left(\beta + \frac{1}{2}\sum_{i=0}^{\lceil n/2\rceil-1} y_{i''}\right)\right]_r$. The Type I carry in signal can be directly generated from the Booth encoding output, but it requires much computation complexity for $\frac{1}{2}\sum_{i=0}^{\lceil n/2\rceil-1} y_{i''}$ as bit width n increases.

3.2.2 Carry estimation by Conditioning on β_i

Since $\beta_i = a_k y_i$ which contains the information of y_i, we can estimate the conditional expectation by $E[P_{ij}|\beta_i]$. In order to get $E[P_{ij}|\beta_i]$, we must calculate these conditional probabilities of $P(P_{ij}|\beta_i = 1)$ and $P(P_{ij}|\beta_i = 0)$ as Equation (3.15) and (3.16).

Table 3.3: Radix-4 modified Booth encoding table

b_{2i+1}	b_{2i}	b_{2i-1}	y_i	n_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	2	0
1	0	0	-2	1
1	0	1	-1	1
1	1	0	-1	1
1	1	1	0	0

Table 3.4: Probabilities of y_i for radix-4 Booth encoding

	$P(y_i = -2)$	$P(y_i = -1)$	$P(y_i = 0)$	$P(y_i = 1)$	$P(y_i = 2)$
$i = 0$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	0
$0 < i < \lceil n/2 \rceil$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{8}$

$$\begin{aligned}
&P\left(P_{ij} = a_j y_i = 1, \beta_i = a_k y_i = 0\right) \\
&= P\left(a_{j-1} = 1, a_{k-1} = 1, y_i = -2\right) + P\left(a_j = 0, a_k = 1, y_i = -1\right) \\
&+ P\left(a_j = 1, a_k = 0, y_i = 1\right) + P\left(a_{j-1} = 1, a_{k-1} = 0, y_i = 2\right) \\
&= \frac{3}{16}
\end{aligned} \tag{3.15}$$

$$\begin{aligned}
&P\left(P_{ij} = a_j y_i = 1, \beta_i = a_k y_i = 1\right) \\
&= P\left(a_{j-1} = 0, a_{k-1} = 0, y_i = -2\right) + P\left(a_j = 0, a_k = 0, y_i = -1\right) \\
&+ P\left(a_j = 1, a_k = 1, y_i = 1\right) + P\left(a_{j-1} = 1, a_{k-1} = 1, y_i = 2\right) \\
&= \frac{3}{16}
\end{aligned} \tag{3.16}$$

And the probability of $P(\beta_i)$ is

Table 3.5: The value of P_{ij} form $a_j a_{j-1}$ and y_i

P_{ij}	$a_j a_{j-1}$			
y_i	00	01	10	11
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
-2	1	0	1	0
-1	1	1	0	0

$$\begin{aligned} P\left(\beta_i = a_k y_i = 1\right) =& P\left(a_{k-1} = 0, y_i = -2\right) + P\left(a_k = 0, y_i = -1\right) \\ & + P\left(a_k = 1, y_i = 1\right) + P\left(a_{k-1} = 1, y_i = 2\right) \\ =& \frac{3}{8} \end{aligned} \tag{3.17}$$

So $P\left(\beta_i = 0\right) = \frac{5}{8}$, we can get $E\left[P_{ij}|\beta_i = 0\right] = \frac{P(P_{ij}=1,\beta_i=0)}{P(\beta_i=0)} = \frac{3}{10}$ and $E\left[P_{ij}|\beta_i = 1\right] = \frac{P(P_{ij}=1,\beta_i=1)}{P(\beta_i=1)} = \frac{1}{2}$. On the other hand, consider the conditional probabilities of n_i

$$P\left(n_i = 1|\beta_i = 0\right) = \frac{P\left(y_i = -1, a_k = 1\right) + P\left(y_i = -2, a_{k-1} = 1\right)}{5/8} = \frac{3}{10} \tag{3.18}$$

and

$$P\left(n_i = 1|\beta_i = 1\right) = \frac{P\left(y_i = -1, a_k = 0\right) + P\left(y_i = -2, a_{k-0} = 1\right)}{3/8} = \frac{1}{2} \tag{3.19}$$

Thus, $E\left[n_i|\beta_i = 0\right] = \frac{P(n_i=1,\beta_i=0)}{P(\beta_i=0)} = \frac{3}{10}$ and $E\left[n_i|\beta_i = 1\right] = \frac{P(n_i=1,\beta_i=1)}{P(\beta_i=1)} = \frac{1}{2}$, and Type II carry estimation $\lambda_{Booth_{II}}$ can be shown as follows.

$$\begin{aligned} \lambda_{Booth_{II}} &= \sum_{i=0}^{\lceil n/2 \rceil - 1} \left[\left(\frac{\beta_i}{2} + \frac{3}{10}\left(1 - \beta_i\right) \right) \sum_{k=2}^{n-2i} 2^{-k} + \left(\frac{\beta_i}{2} + \frac{3}{10}\left(1 - \beta_i\right) \right) 2^{-(n-2i)} \right] \\ &= \frac{\beta}{10} + \frac{3}{20}\left\lceil \frac{n}{2} \right\rceil \end{aligned} \tag{3.20}$$

3.2.3 Carry estimation by Conditioning on α_j

Since $P_{ij} = a_j y_i$ and $\alpha_j = a_j y_{i'}$, where $i \neq i'$, so P_{ij} is related on α_j and we can estimate P_{ij} by the conditional expectation $E\left[P_{ij}|\alpha_j\right]$. Before we compute the value of $E\left[P_{ij}|\alpha_j = 1\right]$ and $E\left[P_{ij}|\alpha_j = 0\right]$, $P\left(P_{ij} = 1|\alpha_j = 1\right)$ and $P\left(P_{ij} = 1|\alpha_j = 0\right)$ must be computed first. From Table 3.6 and 3.7, the conditional probabilities are

$$P\left(P_{ij} = 1|\alpha_j = 1\right) = \frac{P\left(P_{ij} = 1, \alpha_j = 1\right)}{P\left(\alpha_j = 1\right)} = \frac{18/128}{3/8} = \frac{3}{8} \tag{3.21}$$

$$P\left(P_{ij} = 1|\alpha_j = 0\right) = \frac{P\left(P_{ij} = 1, \alpha_j = 0\right)}{P\left(\alpha_j = 0\right)} = \frac{15/64}{5/8} = \frac{3}{8} \tag{3.22}$$

Table 3.6: Conditions for both $P_{ij} = 1$ and $\alpha_j = P_{i'j} = 1$

	$y_{i'} = -2$	$y_{i'} = -1$	$y_{i'} = 1$	$y_{i'} = 2$
$y_i = -2$	$a_{j-1} = 0$ $\left(\frac{1}{128}\right)$	$a_{j-1} = 0, a_j = 0$ $\left(\frac{1}{128}\right)$	$a_{j-1} = 0, a_j = 1$ $\left(\frac{1}{128}\right)$	NA (0)
$y_i = -1$	$a_{j-1} = 0, a_j = 0$ $\left(\frac{1}{128}\right)$	$a_j = 0$ $\left(\frac{1}{32}\right)$	NA (0)	$a_{j-1} = 1, a_j = 0$ $\left(\frac{1}{128}\right)$
$y_i = 1$	$a_{j-1} = 0, a_j = 1$ $\left(\frac{1}{128}\right)$	NA (0)	$a_j = 1$ $\left(\frac{1}{32}\right)$	$a_{j-1} = 1, a_j = 1$ $\left(\frac{1}{128}\right)$
$y_i = 2$	NA (0)	$a_{j-1} = 1, a_j = 0$ $\left(\frac{1}{128}\right)$	$a_{j-1} = 1, a_j = 1$ $\left(\frac{1}{128}\right)$	$a_{j-1} = 1$ $\left(\frac{1}{128}\right)$

Table 3.7: Conditions for both $P_{ij} = 1$ and $\alpha_j = P_{i'j} = 0$

	$y_{i'} = -2$	$y_{i'} = -1$	$y_{i'} = 0$	$y_{i'} = 1$	$y_{i'} = 2$
$y_i = -2$	NA (0)	$a_{j-1} = 0, a_j = 1$ $\left(\frac{1}{128}\right)$	$a_{j-1} = 0$ $\left(\frac{1}{64}\right)$	$a_{j-1} = 0, a_j = 0$ $\left(\frac{1}{128}\right)$	$a_{j-1} = 0$ $\left(\frac{1}{128}\right)$
$y_i = -1$	$a_{j-1} = 0, a_j = 1$ $\left(\frac{1}{128}\right)$	NA $\left(\frac{1}{32}\right)$	$a_j = 0$ $\left(\frac{1}{32}\right)$	$a_j = 0$ $\left(\frac{1}{32}\right)$	$a_{j-1} = 0, a_j = 0$ $\left(\frac{1}{128}\right)$
$y_i = 1$	$a_{j-1} = 1, a_j = 1$ $\left(\frac{1}{128}\right)$	$a_j = 1$ $\left(\frac{1}{32}\right)$	$a_j = 1$ $\left(\frac{1}{32}\right)$	NA (0)	$a_{j-1} = 0, a_j = 1$ $\left(\frac{1}{128}\right)$
$y_i = 2$	$a_{j-1} = 1$ $\left(\frac{1}{128}\right)$	$a_{j-1} = 1, a_j = 1$ $\left(\frac{1}{128}\right)$	$a_{j-1} = 1$ $\left(\frac{1}{64}\right)$	$a_{j-1} = 1, a_j = 0$ $\left(\frac{1}{128}\right)$	NA (0)

Equations (3.21) and (3.22) means that the events of P_{ij} and α_j are independent if the inputs are uniformly and independently distributed. Then, the expected value of P_{ij} is

$$E\left[P_{ij}|\alpha_j = 1\right] = E\left[P_{ij}|\alpha_j = 0\right] = E\left[P_{ij}\right] = \frac{3}{8} \tag{3.23}$$

Equation 3.23 indicates that all the P_{ij} in λ can be estimated by a constant value, $\frac{3}{8}$. From similar procedure, the expect value of n_i, $E\left[n_i\right]$, can also be estimated by $\frac{3}{8}$.

Then the carry estimation of Type III for an n-bit radix-4 Booth multiplier is calculated as follows.

$$\begin{aligned}\lambda_{Booth_{III}} &= \frac{3}{8}\sum_{i=0}^{\lceil (n-1)/2\rceil-1}\sum_{k=2}^{n-2i}2^{-k} + \frac{3}{8}\sum_{i=0}^{\lceil (n-1)/2\rceil-1}2^{-(n-2i)} \\ &= \frac{3}{16}\left\lceil\frac{n}{2}\right\rceil\end{aligned} \tag{3.24}$$

3.3 Generalized Carry Estimation

The carry estimation methods mentioned in above two sections both truncate halt bits of multiplication result. In some cases, fewer bits are required to be truncated. For $1 \le z \le n$, an n-bit fixed-width multiplier that truncates $(n-z)$ bits are illustrated as in Figure 3.5. The carry from β to γ can be represented as follows.

$$\sigma_z = 2^{n-z+1}\left[\frac{\beta}{2}+\lambda\right]_r \approx 2^{n-z+1}\left(\left\lceil\frac{\beta+1}{2}\right\rceil + \left[\frac{2\lambda-1}{2}\right]_r\right) \tag{3.25}$$

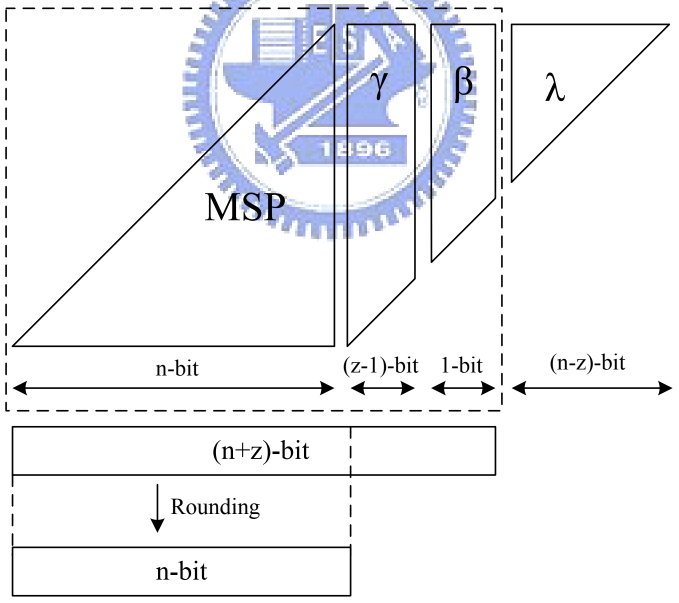


Figure 3.5: Add $n+z$ columns and round the result to n-bit

From the same concept, since β still provides the information of λ, the carry estimation equations can be derived and listed in Table 3.8, 3.9 and 3.10.

Table 3.8: Estimation for Baugh-Wooley multipliers while truncating the least $(n-z)$-bit

Type	Estimation of λ
Type $I^{(a)}$	$2^{-z}\sum_{j=0}^{n-z-1} a_j$
Type $I^{(b)}$	$2^{-z}\sum_{i=0}^{n-z-1} b_i$
Type $II^{(\alpha)}$	$2^{-z}\sum_{j=0}^{n-z-1}\left(\frac{\alpha_j}{3}+\frac{1}{6}\right)\left(1-2^{-(n-z-j)}\right)$
Type $II^{(\beta)}$	$2^{-z}\sum_{i=0}^{n-z-1}\left(\frac{\beta_i}{3}+\frac{1}{6}\right)\left(1-2^{-(n-z-i)}\right)$

3.4 Simulation Result

Exhaustive simulations are required for comparing the performance of each compensation method. In order to provide a quantitative performance measurement, we define the mean absolute error as follows. Assume the multiplier and multiplicand are both n-bit.

$$\overline{|\epsilon|} = \frac{1}{2^{2n}} \sum_{i=-2^{n-1}}^{2^{n-1}-1} \sum_{j=-2^{n-1}}^{2^{n-1}-1} |M_n - i \times j| \tag{3.26}$$

Where M_n is the product of the n-bit fixed-width multiplier to be compared, and the comparison of mean absolute error for $z = 1$ are listed in Table 3.11. Since Booth encoding reduces the number of partial products, the error of Booth multiplier is smaller. By observing Table 3.11, about 85% and 80% error of the Baugh-Wooley and Booth multipliers are compensated regardless of the bit-width n.

Table 3.9: The estimated $2\lambda^{(\alpha\beta)}_{BW_{III}} - 1$ for different bit width n and z

Bit-width (n)	$2\lambda^{(\alpha\beta)}_{BW_{III}} - 1, z = 2$	$2\lambda^{(\alpha\beta)}_{BW_{III}} - 1, z = 3$
8	0, if $\beta - \beta_{n-z} = 0$ $\left\lfloor \frac{1}{2}(\beta - \beta_{n-1}) + 1 \right\rfloor$, otherwise	$\left\lfloor \frac{1}{2}(\beta - \beta_{n-1}) + 1 \right\rfloor$
10	$\frac{1}{2}\lfloor \beta - \beta_{n-z} \rfloor$ +mod$((\beta - \beta_{n-1}), 2)$	1, if$(\beta - \beta_{n-1}) = 1$ $\left\lfloor \frac{1}{2}(\beta - \beta_{n-z}) \right\rfloor$, otherwise
12	$\left\lfloor \frac{1}{2}(\beta - \beta_{n-1} + 2) \right\rfloor$	$\left\lfloor \frac{1}{2}(\beta - \beta_{n-z}) \right\rfloor$, if $\beta - \beta_{n-z} > 3$ $\left\lfloor \frac{1}{2}(\beta - \beta_{n-z} + 1) \right\rfloor$, otherwise
14	$\left\lfloor \frac{1}{2}(\beta - \beta_{n-z} + 2) \right\rfloor$, if $0 \leq \beta - \beta_{n-z} < 4$ $\left\lfloor \frac{1}{2}(\beta - \beta_{n-z} + 1) \right\rfloor$, if $4 \leq \beta - \beta_{n-z} < 9$ $\left\lfloor \frac{1}{2}(\beta - \beta_{n-z}) \right\rfloor$, otherwise	$\left\lfloor \frac{1}{2}(\beta - \beta_{n-z} + 2) \right\rfloor$, if $0 \leq \beta - \beta_{n-z} < 2$ $\left\lfloor \frac{1}{2}(\beta - \beta_{n-z} + 1) \right\rfloor$, if $2 \leq \beta - \beta_{n-z} < 7$ $\left\lfloor \frac{1}{2}(\beta - \beta_{n-z}) \right\rfloor$, otherwise
16	$\left\lfloor \frac{1}{2}(\beta - \beta_{n-z} + 2) \right\rfloor$	$\left\lfloor \frac{1}{2}(\beta - \beta_{n-z} + 2) \right\rfloor$

Table 3.10: Estimation for Booth multipliers while truncating the least $(n - z)$-bit

Type	Estimation of λ
Type I	$2^{-(z+1)} \sum_{i=0}^{\lceil n/2 \rceil - \lfloor z/2 \rfloor} y_{i''}$
Type II	$2^{-z}\left(\frac{\beta}{10} + \frac{3}{20}\left\lceil \frac{n}{2} \right\rceil\right)$
Type III	$2^{-z}\left(\frac{3}{8}\left\lceil \frac{n}{2} \right\rceil\right)$

Table 3.11: Mean absolute error

Method	$n = 8$	$n = 10$	$n = 12$	$n = 14$	$n = 16$
Post-Trun.	63.75	255.75	1023.75	4095.75	16383.75
Direct Trun. B.W.	576.25	2816.25	13312.25	61440.25	278528.2
B.W. $\lambda = 0$	146.38	803.17	4161.13	20597.50	98511.41
B.W. Van's [6]	105.96	456.26	1943.43	8217.69	34554.25
Proposed B.W. Type-I	92.05	403.46	1743.25	7456.74	31651.57
Proposed B.W. Type-II	102.81	403.15	1750.22	7513.40	30792.76
Proposed B.W. Type-III	90.18	393.89	1673.38	7313.21	30120.3
Direct-Trun.Radix-4 Booth	384.25	1920.25	9216.25	43008.25	196608.2
Radix-4 Booth,$\lambda = 0$	139.68	725.74	3622.57	17474.12	82030.32
Radix-4 Booth,Jou's [8]	107.1	477.09	2083.53	8978.42	38315.33
Proposed Radix-4 Booth Type-I	84.59	350.78	1461.55	6040.97	24965.26
Proposed Radix-4 Booth Type-II	88.77	393.60	1667.44	6745.14	28706.97
Proposed Radix-4 Booth Type-III	88.77	406.16	1654.26	6771.68	31267

Chapter 4

Software Simulation for 64-point FFT

4.1 Introduction to Fast Fourier Transform (FFT) Algorithm

Discrete Fourier Transform(DFT) is widely used in digital signal procession application. For a N-point DFT, where N is a number with power of two, we can use the Fast Fourier Transform(FFT) algorithm to reduce the computation time and complexity.

Given a sequence $x[n]$, the N-point DFT is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{kn}, k = 0, 1, 2, \ldots, N-1 \tag{4.1}$$

where $X[k]$ and $x[n]$ are complex numbers and W_N^{kn} is

$$W_N^{kn} = e^{-j(2\pi/N)} = cos\left(\frac{2\pi nk}{N}\right) - j \cdot sin\left(\frac{2\pi nk}{N}\right) \tag{4.2}$$

When we directly use Equation (4.1) to compute the value of N-point DFT, its computational complexity is $O(N^2)$. But if we use radix-r FFT algorithm, the computational complexity will apparently reduce to $O(N\log_r^N)$. For example, consider the decimation in time FFT algorithm, which divided $x[n]$ into two sequence, one for odd points and the other for even points, then Equation (4.1) can be written as

$$X[k] = \sum_{r=0}^{(N/2)-1} x[2r]W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1]W_{N/2}^{rk} = G[k] + W_N^k H[k] \tag{4.3}$$

The corresponding figure is shown in Figure 4.1. So each N-point DFT can be replaced by two N/2-point DFT and several adder operation. From similar procedure, we can further reduce the N/2-point DFT into N/4-point DFT. Finally, for $N = 8$, the simplified FFT architecture is shown in Figure 4.2.

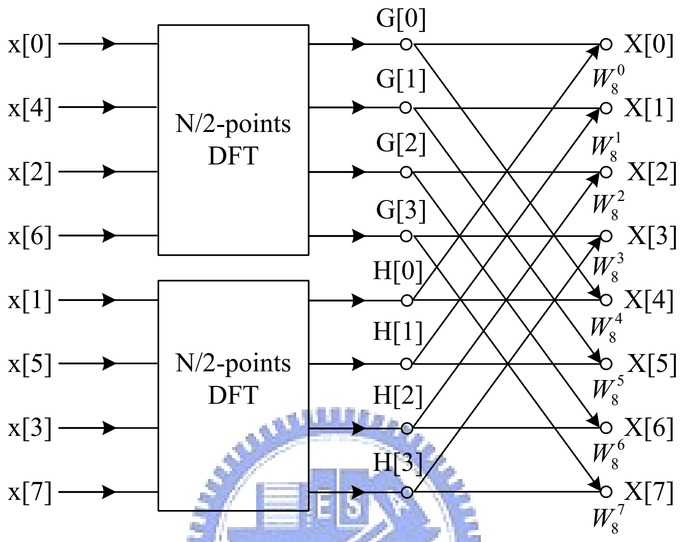


Figure 4.1: Decomposition of N-point DFT into N/2-point DFT by decimation in time

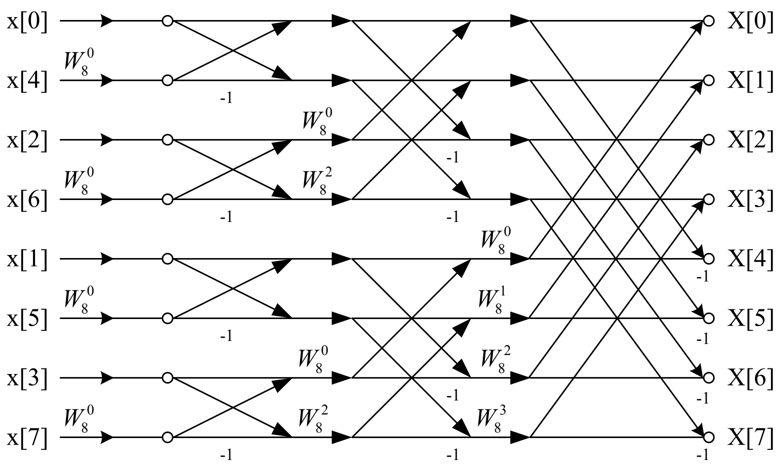


Figure 4.2: Decimation in time FFT butterfly architecture for $N = 8$

4.1.1 Radix 2^3 Architecture for 64-point FFT

In order to derive the 64-point FFT algorithm by using radix 2^3 FFT algorithm [12], we must define

$$\begin{aligned} n &= 32\alpha_1 + 16\alpha_2 + 8\alpha_3 + \alpha_4,\ \ \alpha_1, \alpha_2, \alpha_3 = 0, 1; \alpha_4 = 0, 1, \ldots, 7 \\ k &= \beta_1 + 2\beta_2 + 4\beta_3 + 8\beta_4,\ \ \beta_1, \beta_2, \beta_3 = 0, 1; \beta_4 = 0, 1, \ldots, 7 \end{aligned} \tag{4.4}$$

By using Equation (4.4), Equation (4.1) can be rewritten as

$$\begin{aligned} & X\left[\beta_1 + 2\beta_2 + 4\beta_3 + 8\beta_4\right] = \\ & \sum_{\alpha_4=0}^{7} \sum_{\alpha_3=0}^{1} \sum_{\alpha_2=0}^{1} \sum_{\alpha_1=0}^{1} x\left[32\alpha_1 + 16\alpha_2 + 8\alpha_3 + \alpha_4\right] \\ & \times W_{64}^{(\beta_1+2\beta_2+4\beta_3+8\beta_4)(32\alpha_1+16\alpha_2+8\alpha_3+\alpha_4)} \end{aligned} \tag{4.5}$$

The twiddle factor in Equation (4.5) can be decomposed as

$$\begin{aligned} & W_{64}^{(\beta_1+2\beta_2+4\beta_3+8\beta_4)(32\alpha_1+16\alpha_2+8\alpha_3+\alpha_4)} = \\ & W_2^{\alpha_1\beta_1} W_4^{\alpha_2\beta_1} W_2^{\alpha_2\beta_2} W_8^{\alpha_3(\beta_1+\beta_2)} W_2^{\alpha_3\beta_3} W_{64}^{\alpha_4(\beta_1+2\beta_2+4\beta_3)} W_8^{\alpha_4\beta_4} \end{aligned} \tag{4.6}$$

Thus, Equation (4.5) becomes

$$X\left[\beta_1 + 2\beta_2 + 4\beta_3 + 8\beta_4\right] = \sum_{\alpha_4=0}^{7} BU_8\left[\beta_1, \beta_2, \beta_3, \alpha_4,\right] W_8^{\alpha_4\beta_4} \tag{4.7}$$

BU_8 is the 8-point FFT butterfly architecture, and it can be divided into 3 steps by using radix-2 index map, called radix 2^3 butterfly architecture. The following equation shows the property.

$$\begin{aligned} & BU_8\left[\beta_1, \beta_2, \beta_3, \alpha_4,\right] = \\ & \sum_{\alpha_3=0}^{1} \sum_{\alpha_2=0}^{1} \sum_{\alpha_1=0}^{1} \left\{ \underbrace{\underbrace{\underbrace{BU_2\left[\alpha_1, \alpha_2, \alpha_3, \alpha_4\right] W_2^{\alpha_1\beta_1} W_4^{\alpha_2\beta_1}}_{1st\ step} W_2^{\alpha_2\beta_2} W_8^{\alpha_3(\beta_1+\beta_2)}}_{2nd\ step} W_2^{\alpha_3\beta_3} W_{64}^{\alpha_4(\beta_1+2\beta_2+4\beta_3)}}_{3rd\ step} \right\} \end{aligned} \tag{4.8}$$

Each step has four butterfly operations. After the butterfly operation, the twiddle factor is multiplied to corresponding butterfly output point, as shown in Figure 4.3. Except the twiddle factor multiplication, there are only three twiddle factors in radix 2^3 algorithm, W_8^4, W_8^1 and W_8^3. $W_8^4 = -j$, so we just need to exchange real part and imaginary part. The other two twiddle factor W_8^1 and W_8^3, which equal to $\frac{\sqrt{2}(1-j)}{2}$ and $-\frac{\sqrt{2}(1-j)}{2}$ separately, can be replace by some add operations. Because the $\frac{\sqrt{2}}{2}$ which equals to 0.70710678 can be approximated by $2^{-1} + 2^{-3} + 2^{-4} + 2^{-6} + 2^{-8}$, which can be implemented by five shifters and four adders. As a result, the 64-point FFT with 2-level radix 2^3 algorithm is shown in Figure 4.4.

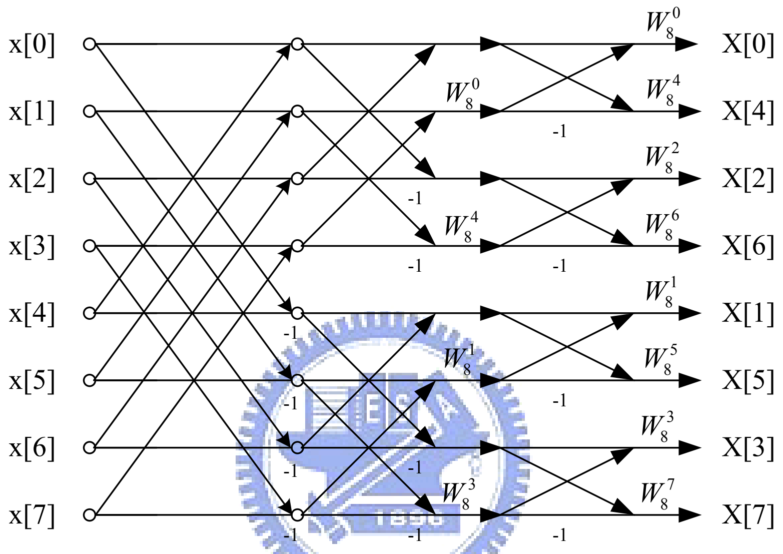


Figure 4.3: Butterfly of radix 2^3 Algorithm

4.2 Software Simulation for a 64-point FFT

The 64-point FFT with radix-2^3 algorithm has only one complex multiplier array stage. It locates between the two radix-2^3 stage, which shown in Figure 4.4. In this 64-point FFT simulation, we choose the input bit width n equal to 8 and the quantified twiddle factor bit width 9. Notice the input bit width through three stage of addition in Figure 4.3 will grow to 11 bits. So the input of complex multiplier array will be 11×9. The following subsections introduces some compensation method with various truncation bits.

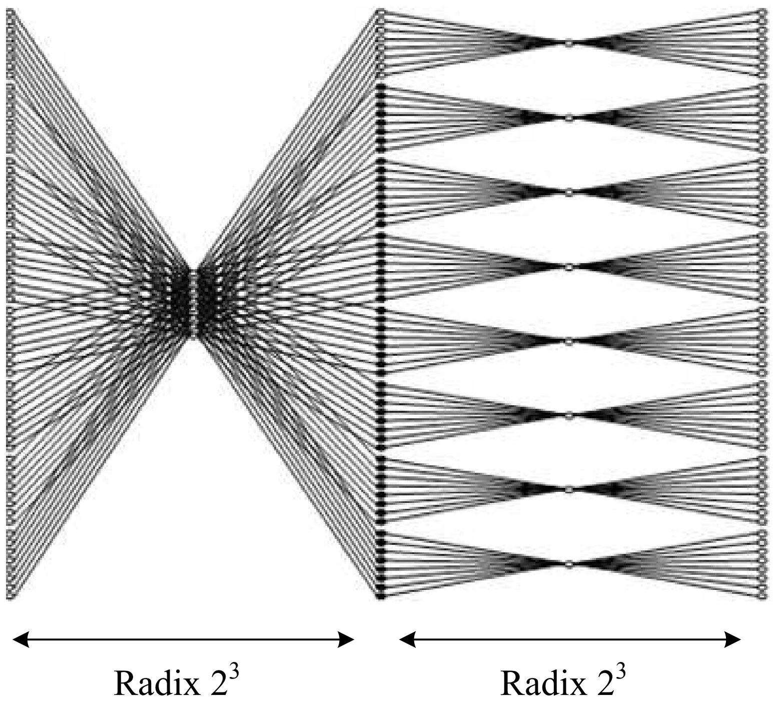


Figure 4.4: 64-point FFT Butterfly with radix 2^3 Algorithm

4.2.1 Method 1 : Direct Imaginary Part Compensation

The complex multiplication can be detail list in following equation

$$(a + j \cdot tw_1) \times (b + j \cdot tw_2) = (a \cdot b - tw_1 \cdot tw_2) + j \times (a \cdot tw_2 + b \cdot tw_1) \tag{4.9}$$

where a and b are input to the complex multiplier; tw_1 and tw_2 are the quantified twiddle factor. For fixed-width property, some least significant bits must be truncated, and the corresponding truncation error will arise. Assume the error in each truncation of multiplier equals to ϵ, then Equation 4.9 with the truncation error can be rewritten as follows.

$$\begin{aligned}(a + j \cdot tw_1) \times (b + j \cdot tw_2) =& \{[(a \cdot b)_{trun} + \epsilon_1] - [(tw_1 \cdot tw_2)_{trun} + \epsilon_2]\} \\ &+ j \times \{[(a \cdot tw_2)_{trun} + \epsilon_3] + [(b \cdot tw_1)_{trun} + \epsilon_4]\}\end{aligned} \tag{4.10}$$

Rewrite the real part and the imaginary part of Equation 4.10 we can get

$$Real\ Part : (a \cdot b)_{trun} + (tw_1 \cdot tw_2)_{trun} + \epsilon_1 - \epsilon_2 \tag{4.11}$$

$$Imaginary\ Part : (a \cdot tw_2)_{trun} + (b \cdot tw_1)_{trun} + \epsilon_3 + \epsilon_4 \tag{4.12}$$

The quantified twiddle factor in the 64-point FFT will be kept to some constant value. Consider the post-truncation method, it truncate the unnecessary bits after multiplication. By software simulation for post-truncation method, we can get the corresponding truncation error. It results in different value for different truncation bit, as listed in the following Table 4.1.

Table 4.1: Truncation error v.s. truncation bits of the post-truncation for a 64-point FFT

11×9 with k-bit truncation	$k = 8$	$k = 9$	$k = 10$	$k = 11$
$a \cdot b$ (ϵ_1)	0.4300	0.4619	0.4755	0.4839
$tw_1 \cdot tw_2$ (ϵ_2)	0.4247	0.4375	0.4376	0.4375
$a \cdot tw_2$ (ϵ_3)	0.4277	0.4323	0.4316	0.4357
$b \cdot tw_1$ (ϵ_4)	0.4283	0.4590	0.4735	0.4851

1896

We can notice that the truncation error in the real part of complex multiplication can be eliminated, and the error in the imaginary part will be accumulated. As shown in the following Table 4.2.

Table 4.2: Truncation error of real part and imaginary part

11×9 with k-bit truncation	$k = 8$	$k = 9$	$k = 10$	$k = 11$
Real Part $(\epsilon_1 - \epsilon_2)$	0.0053	0.0244	0.0379	0.0464
Imaginary Part $(\epsilon_3 + \epsilon_4)$	0.8560	0.8913	0.9051	0.9208

From above analysis, we can directly compensate 1 at the imaginary part of the complex multiplier. By the SQNR (Signal to Quantization Noise Ratio) function defined in Equation (4.13), we can see the improvement of the compensation for a 64-point FFT, as listed in Table 4.3.

$$SQNR = 10 \times log_{10} \frac{\sum \left| X_{ref}^2 \right|}{\sum \left| (X_{ref} - X)^2 \right|} \tag{4.13}$$

Table 4.3: SQNR comparison with imaginary part compensation

SQNR of 11×9 with k-bit truncate	$k = 8$	$k = 9$	$k = 10$	$k = 11$	$k = 12$
Post truncation	47.1259	42.4836	36.9213	30.9923	24.9635
Post truncation imaginary part + 1	49.6604	46.6120	41.7880	36.0879	30.1041

From the above table, we can see the compensation has strongly improvement for SQNR.

4.2.2 Method 2 : Proposed Single Multiplier Compensation

For Booth encoding multiplier, we choose Type-III estimation in Equation (3.24) to estimate it. In the case of 11×9, the n is equal to 10 and the Type-III estimation is equal to

$$\sigma_{Type_{III}} = \frac{\beta}{2} + \frac{3}{16}\left\lceil \frac{10}{2} \right\rceil = \frac{\beta}{2} + \frac{15}{16} \approx \frac{\beta}{2} + 1 \tag{4.14}$$

The corresponding SQNR are listed as follows.

Table 4.4: SQNR comparison with Booth Type III compensation

SQNR of 11×9 with k-bit truncate	$k = 8$	$k = 9$	$k = 10$	$k = 11$	$k = 12$
Booth direct truncation	37.7897	32.1659	26.2788	20.2672	14.2282
Post truncation	47.1259	42.4836	36.9213	30.9923	24.9635
Type III estimation $\sigma = \frac{\beta}{2} + 1$	45.4798	43.1634	38.1305	32.3758	26.2117

4.2.3 Comparison

Figure 4.5 shows the diagram of SQNR v.s. truncation bits. We can see the proposed Booth compensation method will have better SQNR while comparing to the post-truncation method. And the direct imaginary part compensation outperforms than all other methods. But it requires the most hardware since there is no any truncation of calculating the LSP and need an extra hardware to implement the +1 circuit. The following chapter will discuss about the hardware complexity among these truncation methods by using a 2048-point FFT example.

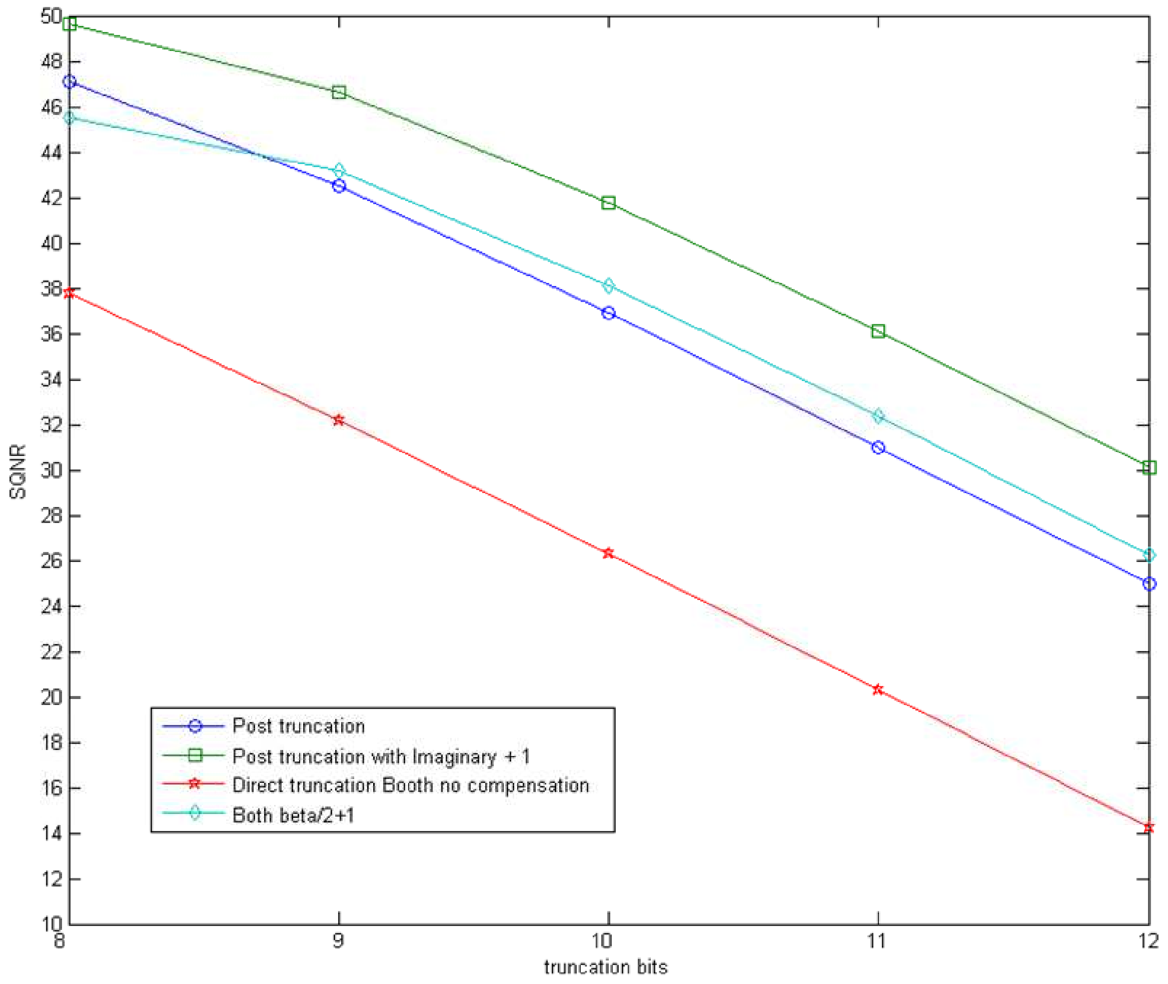


Figure 4.5: SQNR v.s. truncation bits

Chapter 5

Hardware Application for 2048-point FFT

5.1 Architecture of a 2048-point FFT

Since 2048 is not a power of 8, we can decomposed the 2048-point FFT into three stages of radix-2^3 and one stage of radix-2^2, as the following equation.

$$X\left(k_1+8k_2+64k_3+512k_4\right)=$$
$$\underbrace{\sum_{n_4=0}^{3}\underbrace{\Biggl\{\sum_{n_3=0}^{7}\underbrace{\Biggl\{\sum_{n_3=0}^{7}\underbrace{\Bigl\{\sum_{n_3=0}^{7}x(n')\Bigr\}}_{stage\ 1}W_8^{k_2n_2}W_{256}^{k_2(4n_3+n_4)}\Biggr\}}_{stage\ 2}W_8^{k_3n_3}W_{32}^{k_3n_4}\Biggr\}}_{stage\ 3}W_8^{k_4n_4}}_{stage\ 4} \tag{5.1}$$

where

$$\begin{cases} x(n')=x\left(256n_1+32n_2+4n_3+n_4\right)W_8^{k_1n_1}W_{2048}^{k_1(32n_2+4n_3+n_4)} \\ k_1,k_2,k_3=0,1,2\ ;\ k_4=0,1,2,3 \end{cases} \tag{5.2}$$

And the block diagram of the the FFT/IFFT processor [13] is shown in Figure 5.1. It consists of four FFT/IFFT control units, a main memory unit, a processing engine

(PE), and a cache. In this design, a novel block scaling method and a new ping-pong cache-memory architecture are proposed. Since FFT and IFFT have only difference in complexconjugated twiddle factors, the IFFT can be implemented by conjugating FFT input and output [14] as shown in Figure 5.1. And the modules of the design is discussed as follows.

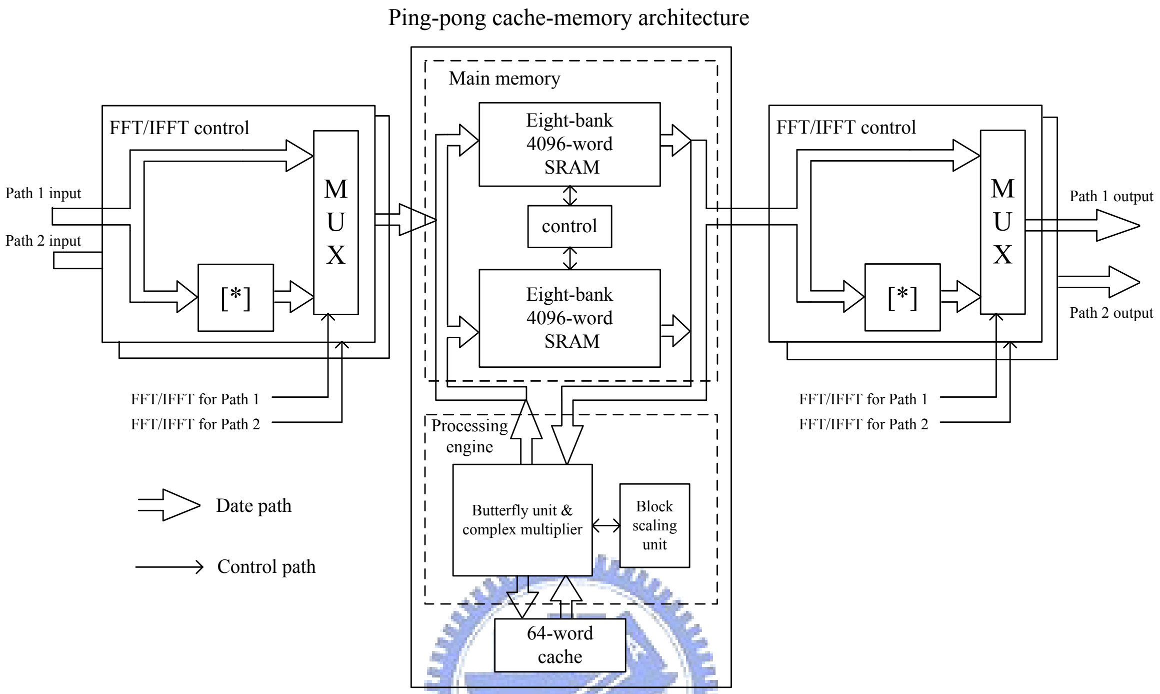


Figure 5.1: Block diagram of the two-stream FFT/IFFT processor

5.1.1 Main Memory

For memory-based FFT, continue flow (CF) memory [15] architecture is used to reduce the memory size. Although CF FFT can reduce memory size by doing I/O operation concurrently in a single memory, it requires additional control units. Because the original CF FFT uses radix-4 and radix-2 algorithms which have different bit-reverse orders. In this design, radix-2^3 and radix-2^2 algorithms are used and have the same bit-reverse order as radix-2 algorithm [16].

5.1.2 Ping-Pong Cache Memory Architecture

Cached-memory FFT [17] [18] is proposed for low power consumption by reducing the memory accesses. Data are first read from main memory and then sent to the cache, as shown in Figure 5.2. Although cached-memory FFT can reduce memory accesses effectively, a complex controlled concurrent read/write cache with unit is required to increase the throughput. Thus the ping-pong cache-memory architecture which uses a simple cache with single read/write operations is proposed, as shown in Figure 5.3. The data read from the main memory are used by PE first and then written to the cache. After the cache is full, data in the cache are read by PE and the computed results are stored back to the main memory. By using the architecture, half the memory accesses can be reduced.

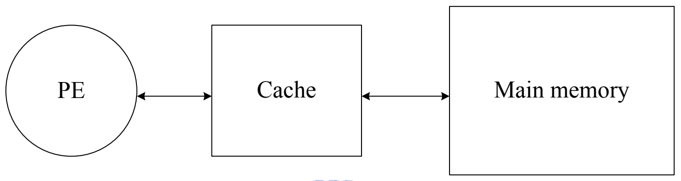


Figure 5.2: Cache-memory architecture

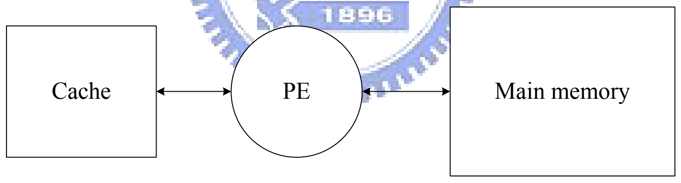


Figure 5.3: Ping-pong cache-memory architecture

5.1.3 Processing Engine (PE)

The processing engine with block scaling approach is shown in Figure 5.4. Consider the case of 2048-point FFT, the inputs have the same decimal point at the fist processing stage, so the data alignments are skipped. The input data are sent into radix-2^3 Butterfly Unit (BU) directly and then passed to the first overflow detection and scaling unit (ODSU1) in Figure 5.4. If an overflow is detected, all inputs will be scaled and the corresponding shift value in exponent will be saved in the block scaling unit. Afterward, the output of ODSU1 is sent to the complex multipliers for twiddle factor multiplications. The outputs of the complex multipliers are passed to the second overflow detection and scaling unit (ODSU2) in Figure 5.4. The second and third stages are similar to stage 1. For stage 4, the radix-2^2 operation is performed and only scaling is performed in ODSU1. Complex multiplications and ODSU2 are skipped because there is no twiddle factor multiplication at the final stage.

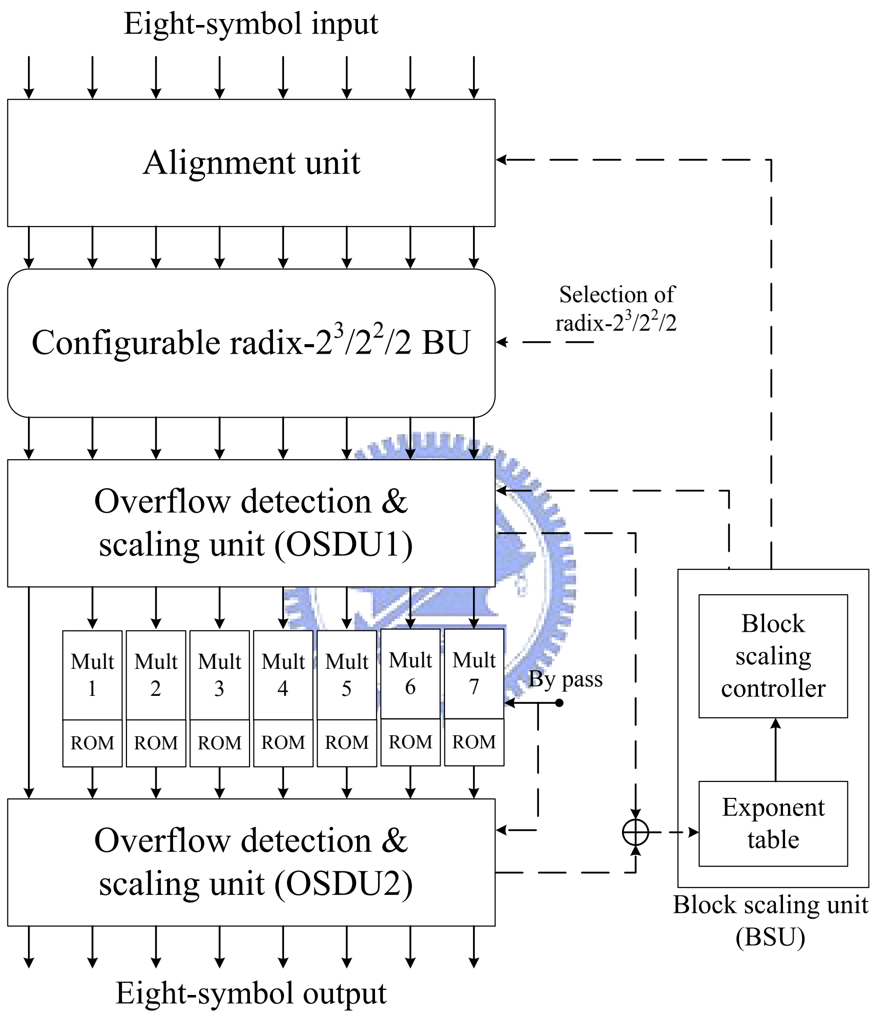


Figure 5.4: Ping-pong cache-memory architecture

5.2 Proposed Multiplier Architecture for 2048-point FFT

In this section, we will discuss multipliers in this 2048-point FFT. The multiplication in the FFT is 12 × 9 and truncate 6 bits, and the output of the multiplication is 15 bits. In the complex multiplication operation defined in Equation 4.9, the real part an imaginary part must be add or subtract. After summing or subtracting the truncated result, it truncates additional 2 bits finally, as shown in Figure 5.5. Several methods for this procedure is listed in the following sub-sections.

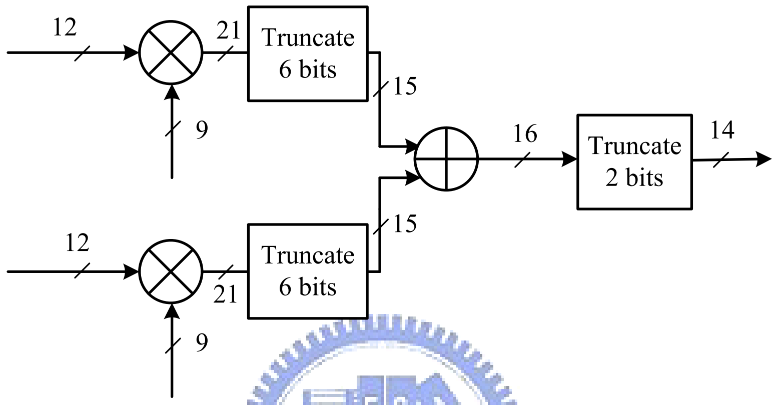


Figure 5.5: Multiplier Architecture for the 2048-point FFT

1896

5.2.1 Method 1 : Design-Ware Direct Truncate Compensation

For the 2048-point FFT application, 6 bits are truncated in each multiplication. If we directly truncate the unused part of the partial product and directly compensate 1 to the remaining parts, the area of multiplier will be saved without much performance loss. As shown in the following table.

Table 5.1: Comparison between truncated and un-truncated multiplier

	Area (Gate count)	SQNR (dB, for FFT)
Post-truncation	737.6	47.6818
Direct truncate, compensate 1	718.3	48.2845

5.2.2 Method 2 : Single Multiplier Compensation

Consider the Type-III compensation of Booth multiplier, it corresponding partial product expected value can be shown as follows.

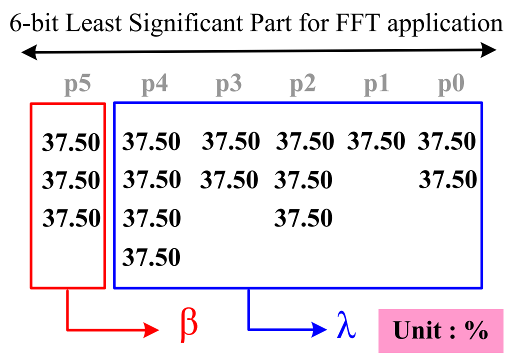


Figure 5.6: Corresponding expected value of partial product in 6-bit truncation

From Figure 5.6 we can calculate the compensation equals to $carry\_in = \frac{\beta}{2} + \frac{9}{16} = \frac{\beta}{2} + 0.5625$. Consider all cases of β we can get

$$\begin{cases} \beta = 0, carry\_in = 0.5625 \approx 1 \\ \beta = 1, carry\_in = 1.0625 \approx 1 \\ \beta = 2, carry\_in = 1.5625 \approx 2 \\ \beta = 3, carry\_in = 2.0625 \approx 2 \end{cases} \tag{5.3}$$

From equation (5.3) we can conclude the carry estimation equation as follows.

$$carry\_in = \left\lfloor \frac{\beta}{2} \right\rfloor + 1 \tag{5.4}$$

And its mean absolute error defined in Equation (3.26) compared with post-truncation is listed in the following Table 5.2.

Table 5.2: Mean absolute error comparison

$12 \times 9 \rightarrow 15$ (truncate 6 bits)	Mean absolute error
Direct-truncate	72.25(100%)
Post-truncate	15.75(21.80%)
Proposed with $carry\_in = \lfloor \frac{\beta}{2} \rfloor + 1$	22.77(31.51%)

5.3 Comparison

The following Table 5.3 shows the gate counts of different multipliers. If we directly truncate the least significant 6 bits of the partial product, about 2.6% gate counts will be saved. For Booth encoding multiplier, about 12.8% gate counts will be saved.

Table 5.3: Multiplier gate count comparison

12×9	Gate count
Design-ware multiplier	737.6
Design-ware direct truncate 6 bits compensation	718.3
Booth multiplier	948.6
Proposed Booth multiplier compensation truncate 6 bits	826.9

The SQNR of different compensation method are listed in the following Table 5.4. We can see the proposed Booth compensation has the similar SQNR value to post-truncate multiplier.

The hardware complexity of proposed multiplier and design-ware post-truncated multiplier using TSMC 0.18μm 1P6M technology is listed in Table 5.5.

We can see the both method 1 & 2 can reduce about 4.7% hardware of combinational circuits since it truncates a part of partial products.

Table 5.4: SQNR comparison

Multiplier type	Compensation	SQNR (dB)
Design-ware multiplier	Direct truncate	47.6818
	Direct truncate, directly compensate 1	48.2845
	Post-truncate	48.0919
	Post-truncate, imaginary + 1	48.4029
Booth multiplier	Direct truncate	47.5245
	Post truncate	48.4029
	Single multiplier compensation	48.4496

Table 5.5: Hardware complexity (gate count) comparison

	Combinational	Sequential	Total
Post-truncation	71657(18.88%)	307973(81.12%)	379630(100%)
Design-ware direct-trun., then compensate 1	68193(18.13%)	307948(82.87%)	376132(100%)
Proposed single multiplier compensation	68294(18.15%)	307959(82.85%)	376253(100%)

The chip layout view by using method 1 & 2 is shown in Figure 5.7 & 5.8. The chip summary of the design-ware post-truncated multiplier and proposed single multiplier compensation multiplier are listed in Table 5.6.

Table 5.6: The chip summary of design-ware post-truncated multiplier architecture v.s. proposed Booth multiplier architecture

Type	Design-ware post-truncation	Design-ware direct-truncation, then compensate 1	Proposed
Process	UMC 0.18μm 1P6M		
Memory size	512 × 24 bits × 16		
Operation frequency	25MHz		
Core area	2.3mm × 2.3mm		
Total gate count	374552	376252	370257
Average core power	109mW	136mW	127mW

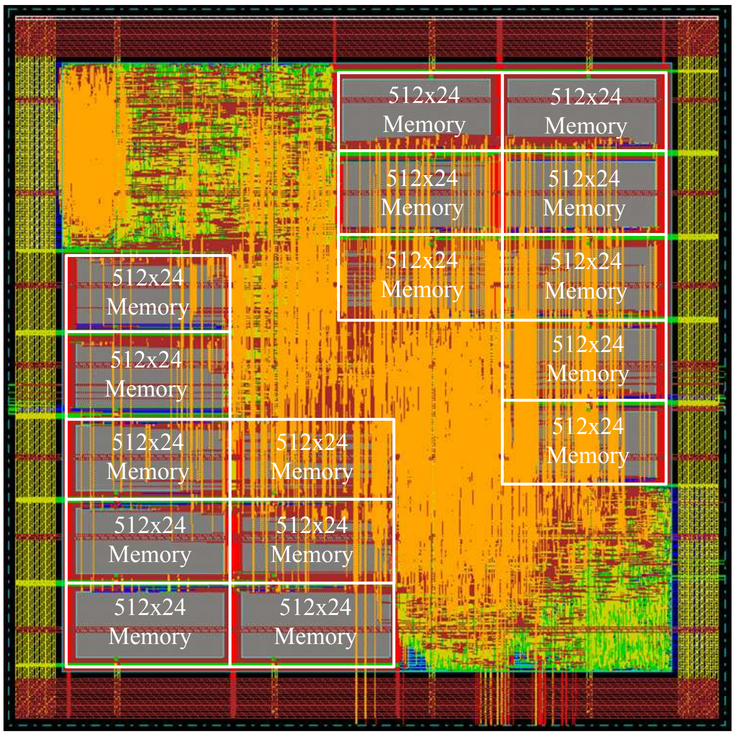


Figure 5.7: Layout view of proposed multiplier for 2048-point FFT

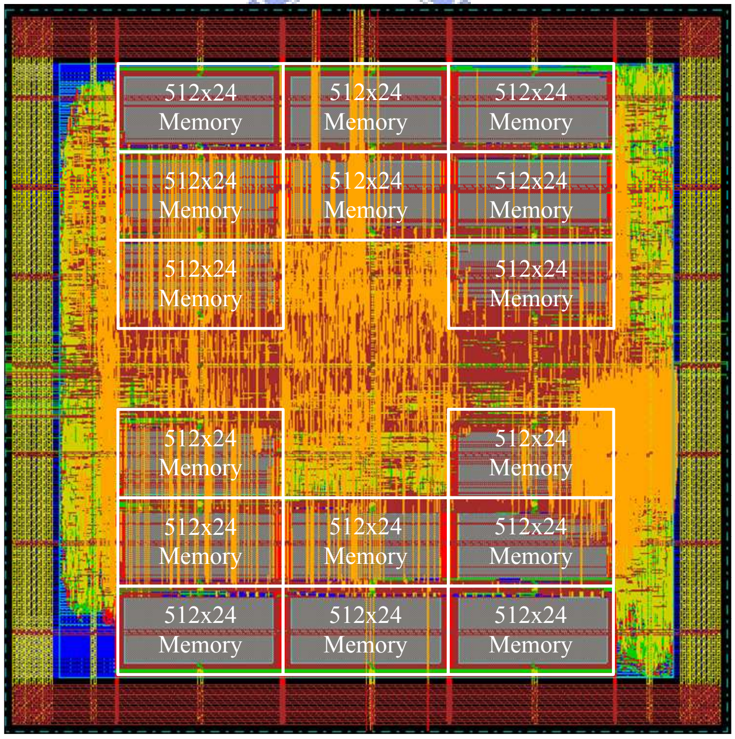


Figure 5.8: Layout view of Design-ware direct-truncate, then compensation 1 multiplier for 2048-point FFT

Chapter 6

Conclusion

In this thesis, the carry estimation methods base on statistical analysis is proposed. There is dependency among the truncated partial product of multipliers. Therefore the dependency of the Baugh-Wooley and the Booth multipliers are discussed and three types of carry estimation methods based on different conditions are proposed. The Type-III compensation of Booth multiplier is the simplest type for implementation, because it always deduced to a constant compensation. Furthermore, the estimation value can be changed as the truncating bits vary. 85% and 80% error of the Baugh-Wooley and Booth multipliers can be improved from the simulation result.

For real case application, the software simulation for 64-point FFT is discussed in Chapter 4. This chapter provides two compensation methods: the directly compensation for post-truncate multiplier can increase 2dB or more while comparing to post-truncate method; and the Type-III compensation for Booth multiplier can have similar SQNR performance while comparing to post-truncate method.

In Chapter 5, the hardware of a 2048-point FFT is implemented by 0.18μm 1P6M CMOS technology. By comparing to the design-ware multiplier, the direct truncate and compensate 1 method can reduce about 2.7% area for a single multiplier with about 0.6dB loss in SQNR. The single multiplier compensation method can reduce about 4.7% gate count without loss in performance. In conclusion, our approach provide a lower area comparing to the post-tuncated multiplier and high-performance closing to post-truncated method.

Bibliography

[1] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," in *IEEE Trans. Comp.*, no. 12, Dec. 1973, pp. 1045–1047.

[2] M. Hatamian and G.Cash, "A 70-MHz 8-bit x 8-bit parallel pipelined multiplier in 2.5-μm CMOS," *JSSC*, vol. 21, no. 4, pp. 505–513, Aug. 1986.

[3] A. Booth, "A single binary multiplication technique," *Quarterly J. Mechanics and Applied Mathematics*, vol. IV, no. 2, pp. 236–240, Jun. 1951.

[4] O. L. MacSorley, "High speed arithmetic in binary computers," *Proc. IRE*, vol. 49, pp. 67–91, Jan. 1961.

[5] L. D. Van, S. S. Wang, and W. S. Feng, "Design of the lower-error fixed-width multiplier and its application," in *IEEE Trans. Circuits Syst. II*, vol. 47, Oct. 2000, pp. 1112–1118.

[6] L. D. Van and C. C. Yang, "Generalized low-error area-efficient fixed-width multipliers," in *IEEE Trans. Circuits Syst. I*, vol. 52, Aug. 2005, pp. 1608–1619.

[7] S. J. Jou and H. H. Wang, "Fixed-width multiplier for DSP application," in *IEEE Int. Symp. Computer Design*, Sept. 2000, pp. 318–322.

[8] S. J. Jou, M. H. Tsai, and Y. L. Tsao, "Low-error reduced-width booth multipliers for DSP application," in *IEEE Trans. Circuits Syst. I*, vol. 50, Nov. 2003, pp. 1470–1474.

[9] 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, Oct. 2002, pp. 45–50.

[10] K. J. Cho, J. G. Chung, K. C. Lee, and K. K. Parhi, "Design of low-error fixed-width modified Booth multiplier," in *IEEE Trans. VLSI Syst.*, May. 2004, pp. 522–531.

[11] Y. C. Liao, H. C. Chang, and C. W. Liu, "Carry estimation for two's complement fixed-width multipliers," in *IEEE Signal Processing Systems (SiPS)*, Oct. 2006.

[12] 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.

[13] Y. Chen, Y. W. Lin, and C. Y. Lee, "A block scaling FFT/IFFT processor for WiMAX applications," in *IEEE Asian Solid-State Circuits Conference*, Nov. 2006.

[14] K. Maharatna, E. Grass, and U. Jagdhold, "A 64-point Fourier transform chip for high-speed wireless LAN application using OFDM," *IEEE J. Solid-State Circuits*, vol. 39, pp. 484–493, Mar. 2003.

[15] B. G. Jo and M. H. Sunwoo, "New continuous-flow mixed radix (CFMR) FFT using novel in-place strategy," in *IEEE Trans. Circuits Syst.*, vol. 52, May. 2005, pp. 911–919.

[16] H. Shousheng and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)modulation," in *Proc. Int. Symp. Signals, Systems, and Electronics*, Oct. 1998, pp. 257–262.

[17] B. M. Bass, "A low-power, high-performance, 1024-point FFT processor," *IEEE J. Solid-State Circuits*, vol. 34, pp. 380–387, Mar. 1999.

[18] Y. Lin, H. Liu, and C. Lee, "A dynamic scaling processor for DVB-T applications," *IEEE J. Solid-State Circuits*, vol. 39, pp. 2005–2013, Nox. 2004.

E S A

作者簡介

姓名：趙祐徵

出生：台中縣

學歷：大度國小 大道國中 台中一中

89.9 ~ 93.6 國立中興大學電機工程學系

93.9 ~ 95.7 國立交通大學 電子研究所（Oasis Lab）