ABSTRACT

Speed, as well as area and power consumption, is always the most important thing in VLSI (Very-large-scale integration) design. With the aim of achieving higher speed, a trade-off can be made by reducing the accuracy. This project is to design 32 by 32 bit approximate multipliers, in which faster and smaller approximate (4:2) compressors are applied in partial product reduction step. The successful design can be 20% faster than conventional Wall tree multipliers with an acceptable error rate in benchmark tests.

Main contributions and achievements:

- Seven different approximate (4:2) compressors have been designed (see pages 24-35).
- Tree topologies are classified into three types according to their allocation schemes. Bubbles, which have a close relationship among speed, accuracy and power dissipation, are introduced to analyze the structure (see pages 36-37).
- A new tree structure named bottom-up tree has been implemented. This tree is proved to have a higher speed and require fewer compressors (see pages 39-40).
- Compared to Wallace tree multipliers, the (4:2) based multiplier can be 32% faster while accuracy decreases 12% in benchmark tests.
- The merged compressor based tree provides a possible way of using high-speed low-accuracy elements in multiplier design. This tree can be 46% faster than Wallace tree while accuracy only decreases 12% in benchmark tests.

DECLARATION

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Jieming Ma, September 2010

TABLE OF CONTENTS

ABSTRACT……………………………………………………………………………..I
ACKNOWLEDGEMENTS……………………………………………………………II
DECLARATION…………………………………………………………………III
TABLE OF CONTENTS……………………………………………………………...IV
LIST OF TABLES……………………………………………………………………VI
LIST OF FIGURES…………………………………………………………………..VII

CHAPTER 1: INTRODUCTION……………………………………………………1
1.1 Aims and Objectives …………………………………………………………….. 1
1.2 Organization……………………………………………………………………2

CHAPTER 2: LITERATURE REVIEW………………………………………………3
2.1 Conventional Compressors………………………………………………………3
2.1.1 Conventional (3:2) counters……………………………………………………..3
2.1.2 Conventional (4:2) compressors…………………………………………………4
2.2 Approximation Circuits………………………………………………………… 7
2.3 Multiplier Topologies……………………………………………………………..8
2.3.1 Multiplication Algorithm………………………………………………………8
2.3.2 Wallace Tree……………………………………………………………………10
2.3.3 Dadda Tree………………………………………………………………………12

CHAPTER 3: PROJECT DESIGN……………………………………………………14
3.1 Design Methodology……………………………………………………………14
3.1.1 Logic styles ……………………………………………………………………..14
3.1.2 The method of estimating delay and area ………………………………......18
3.2 Module Design……………………………………………………………………..24
3.2.1 Approximate (4:2) Compressors……………………………………........25
3.2.1.1 High-accuracy approximate (4:2) Compressors…………….....28
3.2.1.2 Medium-accuracy approximate (4:2) Compressors………..…28
3.2.1.3 Low-accuracy (4:2) approximate Compressors……………......31
3.2.2 Accurate (3:2) Counters…………………………………………………….33
3.2.3 Accurate (5:3) Compressors………………………………………….......34
3.2.4 Performance Comparison………………………………………35
3.3 Topology Design…………………………………………………………………..36
3.3.1 Allocation Schemes……………………………………………………………36
3.3.2 Bottom-up Tree…………………………………………………………..........39
3.4 Verification Strategies…………………………………………………………..41

Chapter 4: EXPERIMENT RESULTS AND DISCUSSION……………….......…45
4.1 Conclusion I……………………………….....………………………45
4.2 Conclusion II………………………...…………………………………46
4.3 Conclusion III……………………………..………………………………47
4.4 Conclusion IV……………………………..……………………………48
4.5 Conclusion V………………………………...……………………………49

Chapter 5: CONCLUSION AND FUTRURE WORK……………………….......50

Bibliography 52

Appendix A: Cell Library 54

Appendix B: Design for High-accuracy approximate (4:2) compressor..............55

Appendix C: Design for Medium-accuracy approximate (4:2) compressor 56

Appendix D: Design for Low-accuracy approximate (4:2) compressor...............57

Appendix E: Design for accurate (3:2) counter...59

Appendix F: 32×32 bit Wallace tree..60

Appendix G: 32×32 bit Convectional (4:2) compressor based tree.......................61

Appendix H: 32×32 bit Bottom-up (4:2) compressor based tree..........................62

LIST OF TABLES

TABLE NO.

Table 2.1 Truth table for full adders……………………………………………………4
Table 2.2 Truth table for conventional (4:2) compressors……………………………...5
Table 2.3 The number of reduction stages for a Dadda multiplier (the reduction stages including the final addition step) ……………………………………………………13
Table 3.1 Qualitative logic style comparisons ………………......…………………18
Table 3.2 Multiplexer comparisons (all logic styles) ……………………………..........18
Table 3.3 Karnaugh map for a high-accuracy approximate (4:2) compressor……..... 25
Table 3.4 Logical Effort calculation for the high-accuracy approximate (4:2) compressor in Figure 3.7 ……………………………………………………...….26
Table 3.5 The Verilog HDL program for the high-accuracy approximate (4:2) compressor in Figure 3.7 ……………………….........................……………27
Table 3.6 Karnaugh map for another high-accuracy approximate (4:2) compressor....28
Table 3.7 Karnaugh map for a medium-accuracy approximate (4:2) compressor...….29
Table 3.8 Logical Effort calculation for the medium-accuracy approximate (4:2) compressor in Figure 3.9…………………………………………......................…30
Table 3.9 Karnaugh map for another medium-accuracy approximate (4:2) compressor………………………………………………………………………...30
Table 3.10 Karnaugh map for a low-accuracy approximate (4:2) compressor……....31
Table 3.11 Logical Effort calculation for the low-accuracy approximate (4:2) compressor in Figure 3.11 ……………………………………………….....….32
Table 3.12 Karnaugh map for other low-accuracy approximate (4:2) compressors.....33
Table 3.13 The performance of compressor family…………………………………....35
Table 4.1 Estimation results of (4:2) compressor based tree multipliers…………......45
Table 4.2 Tests for evaluating the errors in different reduction stages……………....49
Table 5.1 Performance comparison of tree multipliers ……………………………......50

LIST OF FIGURES

FIGURE NO.

Figure 2.1 Simplified schematic of (3:2) counters…………………………………..…3
Figure 2.2 Logic level CMOS (4:2) compressors……………………………….…..…5
Figure 2.3 A logic level optimized CMOS (4:2) compressor……………………..………6
Figure 2.4 A sample 32-bit approximation adder……………………...………………8
Figure 2.5 Dot representation of 4 bit by 4 bit multiplication………………………… 9
Figure 2.6 Partial product generation logic circuit for a 4bit by 4 bit multiplier …….10
Figure 2.7 Wallace tree multiplier. An example of a multiply tree that computes a product in 0(log n) steps……………………………………...…...………… 10
Figure 2.8 8-bit by 8-bit unsigned Wallace tree dot diagram……………...………… 11
Figure 2.9 Bit slice of a binary tree that reduces 16 partial products ……..........…..…12
Figure 2.10 8-bit by 8-bit unsigned Dadda tree dot diagram ……………………........…12
Figure 3.1 (a) General logic gate using pull-up and pull-down networks
(b) A simple example of static logic gates: an inverter ………….………14
Figure 3.2 (a) schematic of a dynamic inverter
(b) wave of percharge and evaluation…………………………………… 15
Figure 3.3 Pass transistor threshold drops ……………………………………...……16
Figure 3.4 Two-input multiplexer in (a) CMOS, (b) CMOS with pass-gates, (c) DPL, (d)LEAP, (e)CPL, (f)EEPL, (g)SRPL, and (h)PPL logic style …………......………17
Figure 3.5 2-input NOR gate sized for unit resistance …………………........………20
Figure 3.6 Layout placement of a conventional (5:3) compressor …………………...23
Figure 3.7 Schematic view of a high-accuracy approximate (4:2) compressor ……...26
Figure 3.8 Layout placement of a high-accuracy approximate (4:2) compressor.……26
Figure 3.9 Schematic view of a medium-accuracy approximate (4:2) compressor…..29
Figure 3.10 Layout placement of a high-accuracy approximate (4:2) compressor...…30
Figure 3.11 Schematic view of a low-accuracy approximate (4:2) compressor……...32
Figure 3.12 Layout placement of a low-accuracy approximate (4:2) compressor…....32
Figure 3.13 Schematic view of an accurate (3:2) counters…………………...……33
Figure 3.14 Layout placement of the accurate (3:2) counter in Figure 3.13 ………....34
Figure 3.15 The schematic of an accurate (5:3) compressor …………………………35
Figure 3.16 Layout placement of the accurate (5:3) compressor in Figure 3.15…......35

Figure 3.17 Allocation schemes: (a) Tight (b) Medium (c) Loose ……….......……36
Figure 3.18 (a) A medium scheme used in an 8 bit by 8 bit tree multiplier
(b) A tight scheme used in an 8 bit by 8 bit tree multiplier…….……..…38
Figure 3.19 The relationship among the bubble quantity, compressor usage rate and the number of partial product reduction stages ………………………………………38
Figure 3.20 The algorithm of calculating the stage height of a bottom-up tree. ……..39
Figure 3.21 Dot diagram of a (3:2) and (5:3) compressor based tree multiplier …..…40
Figure 3.22 Testbench for a (4:2) compressor…………………………….....….......41
Figure 3.23 Cumulative distribution of multiplication operands in benchmark...……43
Figure 3.24 The main part of multiplier testbench …………………………………...43
Figure 4.1 Comparison of the error rate between conventional (4:2) multipliers and bottom-up (4:2) multipliers ……………………………..………………………46
Figure 4.2 Error rate of each product bit ……………………………………………..47
Figure 4.3 The critical path of a tree multiplier 49…………………………………...48

CHAPTER 1: INTRODUCTION

1.1 Aims and Objectives

Since a huge number of multiplications are required in DSP (digital signal processing) applications, multipliers are deemed as a critical part in digital signal processor design (Dandapat, Ghosal, Sarkar, Mukhopadhyay,2010 [13]).

As is well known, accurate multiplication operation is always complicated and therefore requires a huge number of delays. According to Kwon, Nowka, and Swartzlander (2002 [14]), fast multiplication process normally can be divided into three separated steps:

1. Partial product generation
2. Partial product reduction
3. Final carry-propagating addition

Most of the delays are from the Partial production reduction step. In this step, partial products are reduced from the preceding step into the results that have fewer bits by using compressors.

With the aim of achieving extra higher speed and smaller area, a trade-off can be made by reducing the accuracy. Recently, researchers have explored many adders by using approximation circuits. A case in point is Lu's adder (Lu, 2004 [1]), which is designed for speculation in a high performance processor.

This project is to explore some possible approaches to implement multipliers by using (4:2) approximate compressors cells. The objectives to meet this aim can be broken down as follows:

- Implement a set of approximate (4:2) compressors that have different gross error rate. Accelerate the new estimating (4:2) compressors to the greatest extent.
- Explore and build several popular parallel multiplier topologies.
- Design multipliers and take the estimating (4:2) compressors as the basic components. The estimating multipliers are aimed to be twice faster than their conventional partners and have the minimum gross error rate.
- Setup experiments to test the performance of the estimating multipliers. An successful design should have an acceptable gross error rate. High speed, low power and small size are also required.

1.2 Organization of this dissertation

This dissertation consists of three parts.

In literature review, the conventional (3:2) and (4:2) compressors, as well as some popular tree topologies, are introduced since they provide a basic idea of designing a multiplier.

Chapter 3 introduces the whole project. Firstly, Design methodology, such as the logic style and estimation method is explained at the very beginning. Secondly, a set of approximate (4:2) compressor designs are illustrated in the part of module design. Some accurate compressors, which are widely used in conventional multiplier designs, are also introduced as reference. Thirdly, multiplier topologies are analyzed according to the allocation schemes. At the end of Chapter 3, verification strategies are explained. Verification is important in this project since a large number of tests are required to estimate the accuracy of multipliers.

Different compressors have been tried in different tree topologies. The experiment results and discussion can be seen in chapter 4. A general conclusion and the future work are given in the last chapter.

Chapter 2: LITERATURE REVIEW

Multipliers are important functional elements of arithmetic units, digital signal processors, and other circuits that execute arithmetic operations (Ciletti, 2008 [18]).

High speed multipliers can be classified into two types: parallel and sequential multipliers. Sequential multipliers compute the product sequentially and usually utilizing storage elements so that hardware of the multiplier is reused during iteration (Stine, 2004 [9]). Though sequential multipliers are smaller, they are usually slower than parallel multipliers. Since one aim of this project is to accelerate the speed, the multipliers illustrated in this dissertation are all parallel multipliers.

The previous chapter presented tree separate steps in multiplication operation. Partial Product Reduction is usually considered to be time-consuming. Thus, this project is aimed to implement the multiplication in the second operation step. Compressors, which are the basic elements used for product reduction, will be introduced in the first part of this chapter. Efficient topologies also make great contribution to speed, power and size. At the second part, some of the popular tree topology will be described.

2.1 Convectional Compressors

2.1.1 Convectional (3:2) counters

(3:2) counters, also known as full adder, are frequently used in the multiplier design. Normally, the critical path of a parallel multiplier sums up the partial products of each column. (3:2) counters are usually the fundamental elements for this column addition.

A (3:2) counter has three inputs (A, B, C) and two outputs (S and Cout) as shown in Figure 2.1.

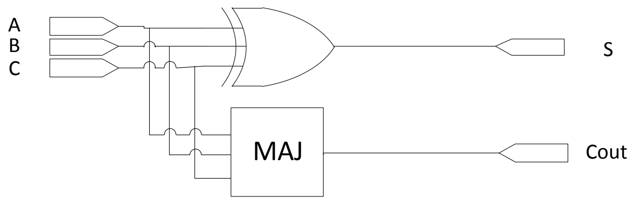


Figure 2.1 Simplified schematic of (3:2) counters

The logic of full adder can be expressed as following:

$$S = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C}$$

$$= A \oplus B \oplus C \qquad [2\text{-}1]$$

$$\begin{aligned} Cout &= AB+AC+BC \\ &= A \# B \# C \end{aligned} \qquad [2\text{-}2]$$

where $\oplus$ denotes exclusive-OR and # denotes majority gate.

A	B	C	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 2.1 Truth table for full adders

Majority gates produce 1 on condition that at least two inputs of a (3:2) counter are set to 1. It is widely used for implementing carry-out function in a full adder design. In a parallel multiplier, carry-out is usually connected to one input of a full adder located in the next column, while carry-in is usually from a Cout of the previous column. Table 2.1 shows the truth table of full adders. The sum output goes high in the event that the number of 1 is odd in input.

2.1.2 Convectional (4:2) compressors

(4:2) compressors are named after their function. In a (4:2) compressor based multiplier, an internal carry-in is always connected to one input of the compressor in the preceding column, while one carry-out is always passed to the next column. When these internal carry-in and carry-out are ignored, the conventional compressor could be considered to be a unit that is able to reduce four inputs to two outputs.

A conventional (4:2) compressor normally contains two full adders in a complex interconnection structure (Oklobdzija, 2000[8]). It has five inputs (A, B, C, D and Cin) and three outputs (COUT, CARRY, SUM) as shown in Figure 2.2. Therefore, the conventional (4:2) compressors are more aptly called (5:3) compressors. CIN is usually connected a carry-out generated from the preceding adder or compressor, while COUT is usually considered to be an internal carry-out.

.
The function of conventional (4:2) compressors can be described as follows:

SUM = A $\oplus$ B $\oplus$ C $\oplus$ D $\oplus$ CIN [2-3]
COUT = A # B # C [2-4]
CARRY = (A $\oplus$ B $\oplus$ C) # D # CIN [2-5]

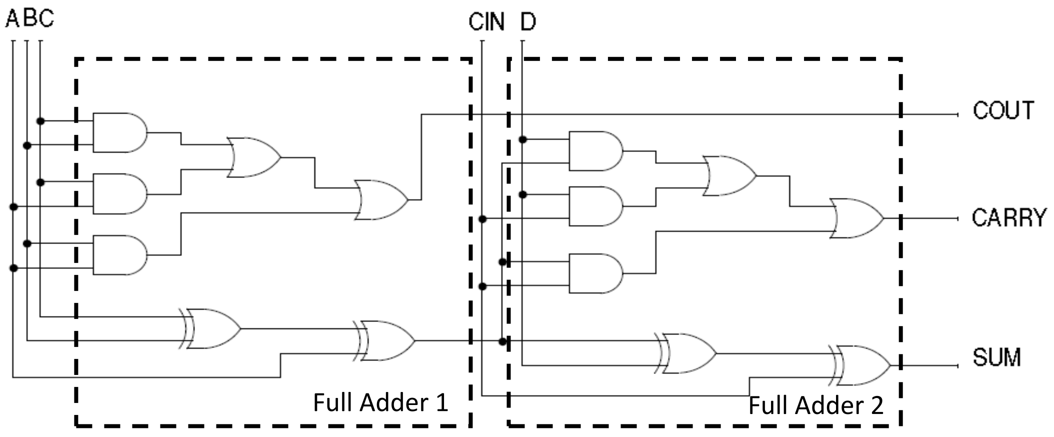


Figure 2.2 Logic level CMOS (4:2) compressors

The truth table for conventional (4:2) compressors is shown in Table 2.2.

INPUTS				CIN = 0		CIN = 1		COUT
A	B	C	D	Carry	Sum	Carry	Sum	
0	0	0	0	0	0	0	1	0
0	0	0	1	0	1	1	0	0
0	0	1	0	0	1	1	0	0
0	1	0	0	0	1	1	0	0
1	0	0	0	0	1	1	0	0
0	0	1	1	1	0	1	1	0
0	1	1	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
0	1	0	1	1	0	1	1	0
1	0	1	0	0	0	0	1	1
1	0	0	1	1	0	1	1	0
0	1	1	1	0	1	1	0	1
1	1	1	0	0	1	1	0	1
1	1	0	1	0	1	1	0	1
1	1	1	0	0	1	1	0	1
1	1	1	1	1	0	1	1	1

Table 2.2 Truth table for conventional (4:2) compressors

If ΔXOR denotes the delay of one XOR gate, the critical path delay of the conventional (4:2) compressors is as long as 4ΔXOR.

The accurate (4:2) compressors, which were introduced in [2] (Prasad and Parhi (2001) provided a scheme to reduce and flatten the delay. These designs are based on the modified complete logic function equations (equation [2-4], [2-5] and [2-6]) for the three outputs. The logic decomposition of these designs is shown in Figure 2.2.

$$SUM = A \oplus B \oplus C \oplus D \oplus CIN \qquad [2\text{-}6]$$

$$COUT = B \cdot C + A \cdot C + A \cdot B$$
$$= \overline{A} \cdot B \cdot C + A \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot C + A \cdot B$$
$$= (A \cdot \oplus \cdot B) \cdot C + A \cdot B$$
$$= (A \cdot \oplus \cdot B) \cdot C + (\overline{A} \cdot \overline{B} + A \cdot B) \cdot A$$
$$= (A \cdot \oplus \cdot B) \cdot C + \overline{(A \oplus B)} \cdot A \qquad [2\text{-}7]$$

$$CARRY = (A \oplus B \oplus C \oplus D) \cdot CIN + A \oplus B \oplus C \cdot D$$
$$= (A \oplus B \oplus C \oplus D) \cdot CIN + \overline{(A \oplus B \oplus C \oplus D)} \cdot D \qquad [2\text{-}8]$$

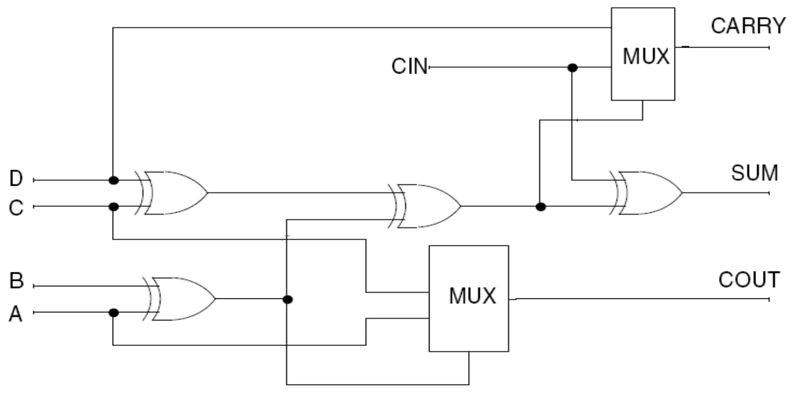


Figure 2.3 A logic level optimized CMOS (4:2) compressor
(Adapted from: Chang, C H.; Gu, J.M.; Zhang, 2004 [4])

According to Figure 2.3, it is clear that the critical path delay of the optimized (4:2) compressor is decreased to 3ΔXOR. Besides SUM, CARRY and COUT are also generated from exclusive-OR gates.

2.2 Approximate circuits

With the rapid developments in science and technology, faster, smaller and lower power consumption multipliers are required in industry. While the accurate computing is usually complicated, hardware arithmetic units are expensive and hard to optimize. In some special application designs, with the aim of accelerating the speed and decreasing the power consumption, a trade-off may be made by reducing the accuracy (SanGregory, Brothers, Gallagher, and Siferd,1999 [5]). A case in point is data value speculation, which usually heighten the need for high-speed at the expense of accuracy.

Lu ([1]) applied the approximation concept to a few stages in a superscalar processor in 2004. As is well known, the speed of column additions are very different since they have different depth of circuits in gates. However, many of logic paths are not necessary to calculate the speed since they are fast enough. The path, whose duration is longest in all logic paths, is called critical path. It determines the speed of the entire function unit, and therefore seriously affects the performance. If the critical path could be shortened, the speed of function unit will be accelerated.

Approximation concept brings a possible approach to shorten the critical path. It is able to roughly calculate the results and increase the function unit's clock frequency. Lu's adder introduces k-bits look-ahead method instead of waiting for all proceeding bits. In other words, the carry propagation distance is much less than the full result width. The current carry result could be expressed as following:

$$C_i = f(a_{i-1}, b_{i-1}, a_{i-2}, b_{i-2}, \dots, a_{i-k}, b_{i-k}), \quad [2\text{-}9]$$

Where 0<k<i+1, and a_i, b_i=0 if i<0.

Though Lu's adder is much faster and smaller than accurate adders, errors will generate when a carry is cut off near MSB (Most Significant Bit). Figure 2.3 shows a sample 32-bit approximate adder. k is set to 4, which means the current bit will look ahead 4 bits. If the carry propagation is greater than 4, the approximate adder will provide a wrong result.

According to the simulation result reported in Lu (2004 [1]), the accuracy of the 32-bit approximate adder is about 65 percent with a 4-bit carry chain on condition that the test inputs are from random data. Lu also pointed that the accuracy will be much better (close to 90 percent) when this adder applied in real applications.

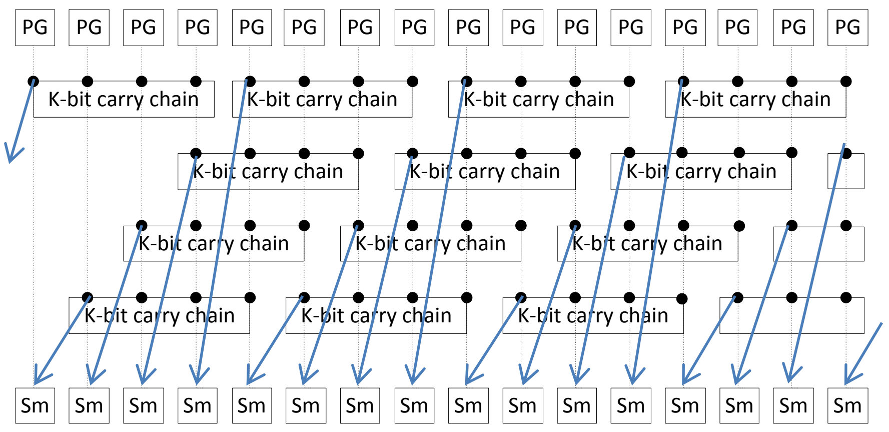


Figure 2.4 a sample 32-bit approximation adder. The adder has the usual carry, propagate, generate, and sum circuits but also implements a carry chain with 29 4-bit carry blocks and three boundary cells. (Adapted from Lu, 2004 [1])

2.3 Multiplier Topologies

Multiplication is one of the basic operations in elementary arithmetic. As we learned before, multiplication applies addition operation to produce the product from a multiplicand and a multiplier. The speed of multiplication plays a importance role in digital signal processing today, especially since the media signal processing became popular (Oklobdzija, 2000 [8]).

Array and tree are two of the most popular structures in parallel multiplier design. The delay of array multiplier is affected by the number of counters in the row. Though the iterative cells used in the array require less hardware but the speed is quite limited. Tree topology is usually considered to be faster, however, tree is complex and hard to lay out. Thus, both of the two structures have pros and cons. The choice is based upon factors such as latency, thourthput, energy, area, and design complexity (Weste and Harris, 2010 [16]).

2.3.1 Multiplication Algorithm

The simplest method of multiplication is to add together copies of the partial products, and then produce the final product as shown in Figure 2.5. Every black dot stands for a single bit, which can be either “1” or “0”. This type of the figure is usually called dot diagrams. It is an easy way to analyze the operation of multiplication.

In Figure 2.6, both of the multiplicand and the multiplier are 4 bit unsigned binary integer. This multiplication results in 4 partial products and each of them is 4 bits. When X stands for the multiplicand and Y stands for the multiplier, the product P can be estimated by Equation [2-10]. One of the partial products can be represented by$(X_i \cdot Y_j)$.

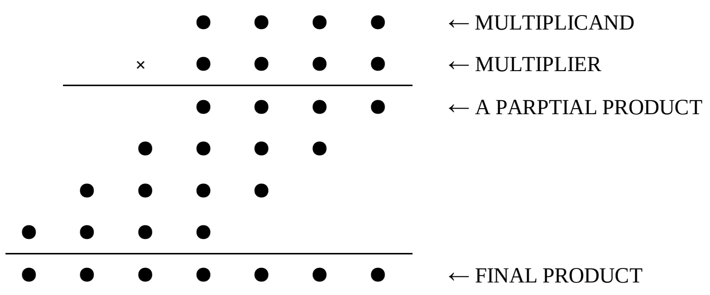


Figure 2.5 Dot representation of 4 bit by 4 bit multiplication

$$\begin{aligned} P &= X \cdot Y \\ &= \sum_{i-0}^{n-1} \sum_{j=0}^{m-1} X_i \cdot Y_j \cdot 2^{i+j} \\ &= \left(\sum_{i=0}^{n-1} X_i \cdot 2^i\right) \cdot \left(\sum_{j=0}^{m-1} Y_j \cdot 2^j\right) \end{aligned} \quad [2\text{-}10]$$

According to the basic multiplication algorithm, the hardware multiplier can be divided into three steps (Kwon, Nowka, and Swartzlander, 2002 [14]).

- Partial Product Generation: the main function of this circuit is to generate partial products$(X_i \cdot Y_j)$. The simple way is to use a set of AND gats. For example, in Figure 3.2, the partial product P_{00} is generated by (X_0 AND Y_0), the partial product P_{01} is generated by (X_0 AND Y_1), and so on.
- Partial Product Reduction: in this step, adders and compressors can be applied to reduce the partial products. The optimization for partial product reduction will be detailedly demonstrated in the next two sections.
- Final Carry-Propagate Addition: the sum and carry will be added to the product.

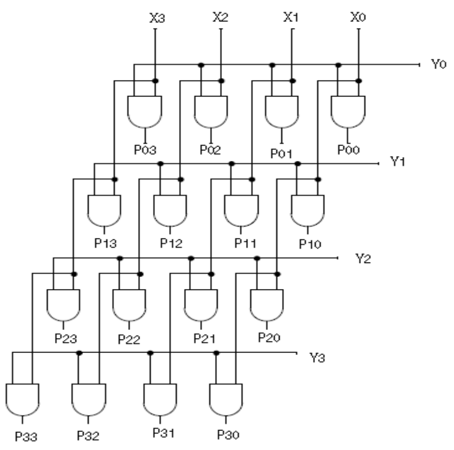


Figure 2.6 Partial product generation logic circuit for a 4bit by 4 bit multiplier

2.3.2 Wallace tree

The idea of tree topologies is to reduce the partial products down until they are reduced enough for use with high speed carry-propagation adders (Stine,2004 [9]). Figure 2.7 shows an example of a Wallace tree which computes a product in 0(log n) steps. Every adder or compressor in the tree is not an isolated part, because each of them is used to perform partial products reduction and there is a close relationship between each other. In other words, every component in the tree is commutative, and every component makes contribution to partial products reduction.

Though Wallace tree topologies may bring high speed to multipliers, it also increases the difficulty of VLSI lay out due to its irregular interconnection. Sometimes, design engineers turn to choose regular topologies such as array structure rather than irregular topologies.

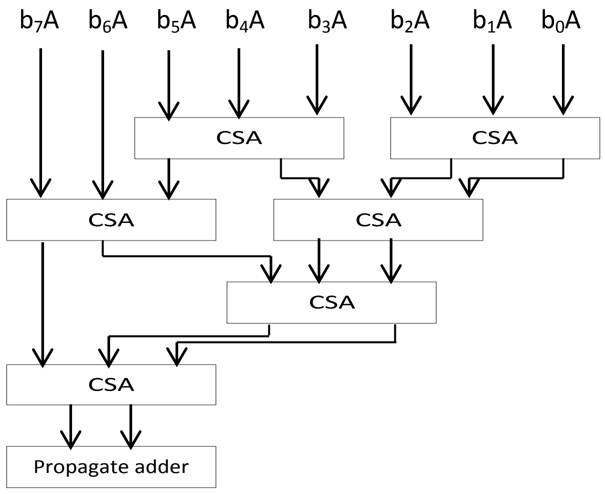


Figure 2.7 Wallace tree multiplier. An example of a multiply tree that computes a product in 0(log n) steps. (Adapted from: Hennessy and Patterson, 2002[17])

Wallace tree is popular since it is considered to be much faster than array structure. The basic idea of the Wallace tree is to reduce the partial products by a set of (3:2) counters as shown in Figure 2.7. In a Wallace tree multiplier, (3:2) counters reduce columns with three bits. If j stands for the reduction stage, the matrix height w can be calculated by the following recursive equation (Bickerstaff, Schulte, and Swartzlander, 2001 [12]):

$$W_j = n \qquad [2\text{-}11]$$

$$W_{j+1} = 2 \cdot \left\lfloor \frac{W_j}{3} \right\rfloor + (W_j \bmod 3)$$

For example, in a 4 by 4 bit Wallace tree as Figure 3.5, the matrix height is:

Reduction stage 0: $W_0 = 4$
Reduction stage 1: $W_1 = 3$
Reduction stage 2: $W_2 = 2$

A good way to analyze Wallace tree is to draw the dot diagram as shown in Figure 2.8. Crossed diagonal lines means the outputs of half adders, while the diagonal lines means the outputs of full adders. As we calculated by the recursive equation, the 4 bit by 4 bit Wallace tree takes 3 reduction stages with matrix height of 4, 3 and 2. In the second stage, the last row is left over and not reduced until the next stage.

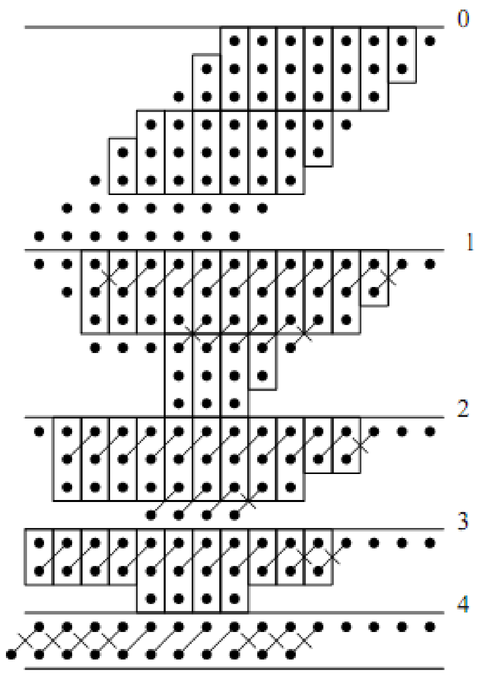


Figure 2.8 8-bit by 8-bit unsigned Wallace tree dot diagram
(Source: Eriksson, 2003 [19])

While the optimized (4:2) compressors can achieve higher speed than (3:2) counters, they can be applied in the Wallace tree instead of full adders. In other words, the multiplier group rows into sets of four. Within the four row set, (4:2) compressors reduce columns with four bits. The proposed structure is drawn in Figure 2.9. This structure is called binary tree in (Flynn and Oberman, 2001 [10]).

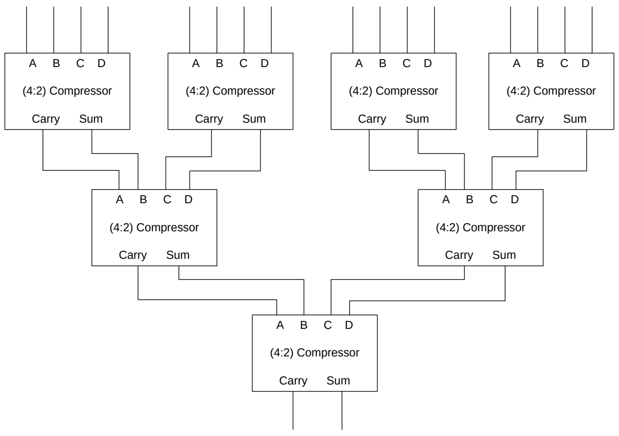


Figure 2.9 Bit slice of a binary tree that reduces 16 partial products (Adapted from: Flynn and Oberman, 2001[10])

2.3.2 Dadda tree

In a similar way, Dadda tree also reduces partial products by using (3:2) and (2:2) counters. The main difference between Dadda and Wallace tree is the number of counters used in the whole multiplier.

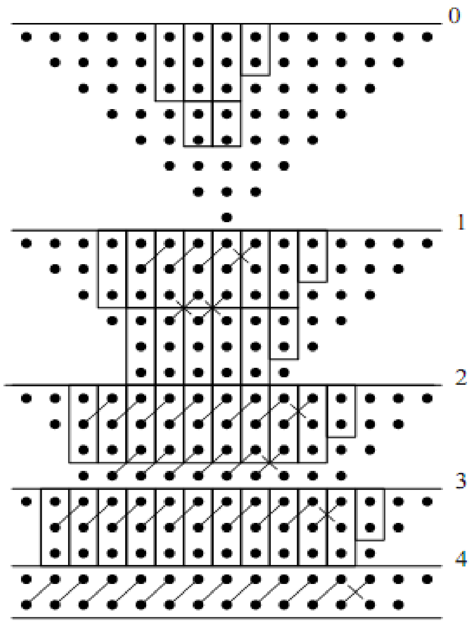


Figure 2.10 8-bit by 8-bit unsigned Dadda tree dot diagram
(Source: Eriksson, 2003 [19])

An 8-bit by 8-bit unsigned Dadda tree can be seen in Figure 2.10. The last step, which is able to complete 2-bit addition, is the Final Carry-Propagate Addition stage. The reduction stages of the partial products follow the following series,

2; 3; 6; 9; 13; 19; 28; 42; 63; …

Alternatively, when j stands for the reduction stage, the serious could be described as following:

$$W_{j-1} = 2\,; \qquad [2\text{-}12]$$

$$W_{j-2} = \frac{3}{2} \cdot W_{j-1} \qquad (j \geq 1)$$

The following illustrates the steps to build a Dadda tree:

a) Let W_0 = 2 since it is the final addition stage. The matrix height of preceding stage is equal to ($\frac{3}{2} \cdot W_j$). j could be defined in Table 2.3.

Reduction stages	The number of bits in the multiplier(N)
1	$0 \leq N \leq 2$
2	$N = 3$
3	$N = 4$
4	$5 \leq N \leq 6$
5	$7 \leq N \leq 9$
6	$10 \leq N \leq 13$
7	$14 \leq N \leq 19$
8	$20 \leq N \leq 28$
9	$29 \leq N \leq 42$
10	$43 \leq N \leq 63$
11	$64 \leq N \leq 94$

Table 2.3: The number of reduction stages for a Dadda multiplier (the reduction stages including the final addition step)

b) Use (3:2) and (2:2) counters in stage 0. The reduced matrix should be no more than W_j as described in formula [2-10].

c) Let j = (j +1). Repeat step (b) until the matrix height is reduced to two bits.

Chapter 3: Project Design

3.1 Design Methodology

3.1.1 Logic styles

The logic style used in logic gates basically influences the speed, size, power dissipation, and the wiring complexity of a circuit as well as ease-of-use and generality of gates in cell-based design techniques (Zimmermann and Fichtner, 1997[22]).

Complementary CMOS is one of most popular logic styles in VLSI (Very-large-scale integration) design. Generally, a static CMOS gate has an nMOS pull-down network and a dual pMOS pull-up network as shown in Figure 3.1.

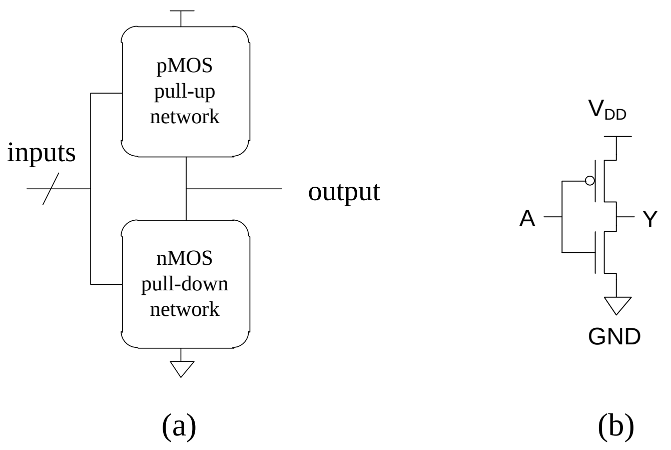


Figure 3.1 (a) General logic gate using pull-up and pull-down networks
(b) A simple example of static logic gates: an inverter
(Source: Weste and Harris, 2010 [16])

An inverter is a simple example of static logic gates. Its pull-down network is a single nMOS transistor and its pull-up network is a single pMOS transistor. The source of the nMOS is connected to the ground while the drain of pMOS is connected to V_{DD}.

Even in complex function circuit design, static CMOS gates are easy to use. As is well known, complementary CMOS gates are inverting, which means AND and OR functions must be transferred to NAND and NOR. DeMorgan's law is helpful for bubble pushing.

Dynamic logic is distinguished from static logic since logic circuits are clocked and work in two phases, which are percharge and evaluation. Figure 3.2(a) shows a dynamic inverter. During percharge phase, the clock is zero, so the pMOS transistor is

ON and the output is one. During evaluation phase, the clock is one, so the pMOS is OFF but the output is according to the status of nMOS. The wave of percharge and evaluation can be seen in Figure 3.2(b). Dynamic logic gates are usually considered to be faster since they have small input capacitance and no contention during switch. However, when dynamic logic gates are applied in a system which has large clock loads and high signal transition activities, they require a large amount of power dissipation. Therefore researchers suppose dynamic circuits are not suitable for low-power circuit gate (Chandrakasan and Brodersen, 1995[23]; Zimmermann and Fichtner, 1997[22]; Weste and Harris, 2010[16]).

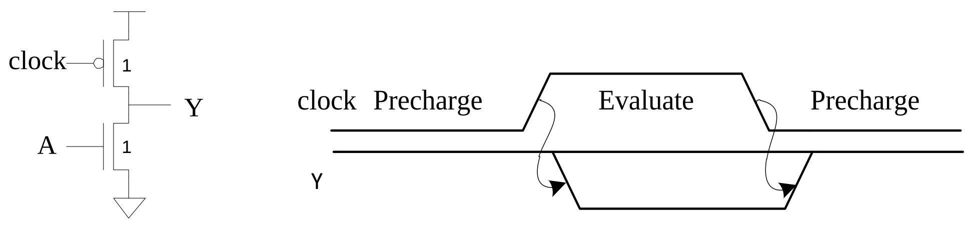
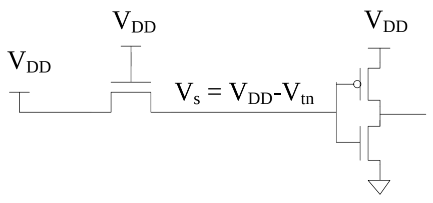


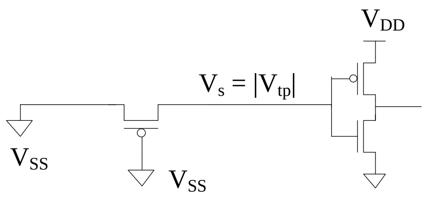
Figure 3.2 (a) schematic of a dynamic inverter
(b) wave of percharge and evaluation
(Source: Weste and Harris, 2010 [16])

Pass-transistor is another popular logic style. Compared to the complementary CMOS gates, the source of the logic transistor network is connected an input signals instead of V_{DD}. Pass-transistor logic style normally has a small number of transistors since one single pMOS or nMOS transistor could perform the switch operation already. Thus, pass-transistor logic may be an approach to achieve high speed, low power and small area.

However, a single pass transistor can pull the voltage in a special preferred direction. An nMOS transistor can only act as a good switch when passing a logic 0. But it will have a threshold drop when passing a logic 1. Normally, nMOS transistors can pull up no more than $(V_{DD}\text{-}V_{tn})$. pMOS transistors can pass a logic 1 well while it can only pull down to $|V_{tp}|$. This characteristic of pass transistors will seriously limit the use of them. Figure 3.5 shows the pass transistor with threshold drop. As illustrated in Figure 3.3(a), the pass transistor is passing a logic 1 and its output can only rise to $(V_{DD}\text{-}V_{tn})$. Moreover, the threshold voltage will be pulled up by the body effort since V_{sb} is greater than zero. The degraded output voltage is not able to fully cut off the pMOS transistor of the sequent inverter. This will cause the inverter to malfunction. Something similar happened with pass pMOS transistors. According to Figure 3.3(b), the pass pMOS transistor poorly pulls down the voltage of output. The floating output voltage will also make inverter work abnormally. Transmission gate, which combines of an nMOS and a pMOS transistor, is a possible solution to avoid threshold drop. But each of the two transistors requires a complementary control input and thus an extra inverter should be applied.



(a)



(b)

Figure 3.3 Pass transistor threshold drops

In order to further analyze the performance of different pass logic gates, Zimmermann (1997) designed a set of evaluation tests. Figure 3.4 shows the multiplexed implemented in different approaches. (a) is a static CMOS multiplex while (b) applies transmission gates with two inverters. Double pass-transistor logic (DPL), which is shown in (c), uses both pMOS and nMOS transistors in the circuit to provide full swing. Unlike DPL, Single-rail pass-transistor logic (LEAP) only has an inter-cell wiring (see (d)). Complementary pass-transistor logic (CPL) is shown in (e), it comprises two pairs of pMOS and nMOS transistors for complementary input signals and two outputs inverters for driving the outputs. The push-pull pass-transistor logic (PPL) is similar to CPL. The slight difference between PPL and CPL is the output circuit. PPL does not have inverters in its output nodes. Swing restored pass-transistor logic (SRPL) and energy economized pass-transistor logic (EEPL) are also derived from CPL, their schematics are shown in Figure 3.4 (g) and 3.4(f) respectively. All of the above pass-transistor logic styles are included in Zimmermann's evaluation tests.

Table 3.1 and Table 3.2 compare the performance of pass-transistor family with that of complementary CMOS logic. Among the pass-transistor logic styles, CPL requires less power and area. The multiplexer implemented by CPL also achieve perfect speed. Complementary CMOS, however, is better than all of the pass-transistor logic gates in speed, area and power consumption, as well as robustness. Moreover, the complementary CMOS has good noise margins, and is insensitive to device variations, easy to design, widely supported by CAD tools, and readily available in standard cell libraries (Weste and Harris, 2010[16]).

Based on above analysis, this project applies complementary CMOS as preferred logic style.

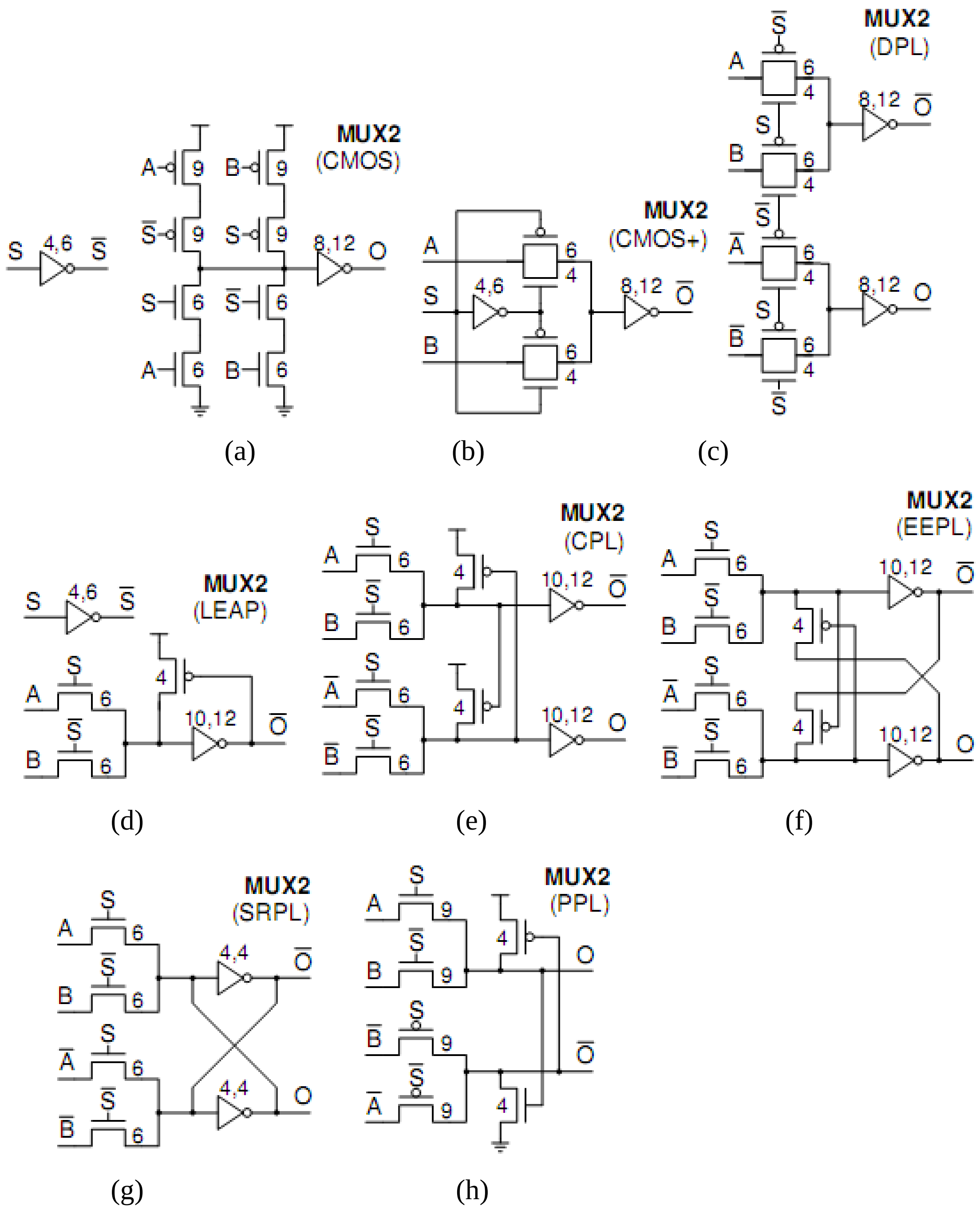


Figure 3.4 Two-input multiplexer in (a) CMOS, (b) CMOS with pass-gates, (c) DPL, (d) LEAP, (e) CPL, (f) EEPL, (g) SRPL, and (h) PPL logic style
(Source: Zimmermann and Fichtner, 1997[22])

Logic style	# MOS networks	Output driving	I/O decoupl.	Swing restor.	# rails	robustness
CMOS	n+p	med.-good	yes	no	single	high
CPL	2n	good	yes	yes	dual	medium
SRPL	2n	poor	no	yes	dual	low
DPL	2n+2p	good	yes	no	dual	high
LEAP	N	good	yes	yes	single	medium
EEPL	2n	good	yes	yes	dual	medium
PPL	n+p	poor	no	yes	dual	low

Table 3.1 Qualitative logic style comparisons
(Source: Zimmermann and Fichtner, 1997[22])

gate type	logic style	delay(ns) 3.3 V	delay(ns) 1.5V	Power(µW) 3.3V	Power(µW) 1.5V	PT(norm.) 3.3V	PT(norm.) 1.5V	# trans.	size (λ^2)
MUX2	CMOS	1.15	4.44	10.4	2.0	1.00	1.00	12	4111
	CMOS[1]	1.19	4.94	10.4	1.9	1.03	1.07	10	3969
	CMOS+	1.59	6.50	10.3	1.9	1.37	1.43	8	4455
	CPL	1.28	6.21	19.0	3.4	2.03	2.42	10	5528
	EEPL	2.02	10.27	23.0	4.9	3.88	5.72	10	6528
	SRPL	5.86	29.75	26.2	3.7	12.81	12.52	8	6009
	PPL	7.77	-[2]	32.7	-[2]	21.16	-	6	4301
	LEAP	2.07	-[2]	12.6	-[2]	2.18	-	7	4333
	DPL	1.34	5.33	17.3	3.3	1.93	1.98	12	6133

[1] without output inverter

[2] does not work for $V_{dd}<V_{tn}+|V_{tp}|$

Table 3.2 Multiplexer comparisons (all logic styles)
(Source: Zimmermann and Fichtner, 1997[22])

3.1.2 The method of estimating delay and area

Resistive-capacitive (RC) delay models can be used to estimate delay well since it is able to approximate the nonlinear transistor I-V and C-V.

R, which is the ratio of V_{ds} to I_{ds}, is usually called effective resistance in MOS transistor. If a unit nMOS transistor has resistance R, the nMOS transistor which has a k-times width will has resistance R/k. Therefore, resistance is usually considered to be inversely proportional to width. It is important to note that the pMOS transistor which has a k-times width has resistance R/2k due to the lower mobility of pMOS.

C is total capacitance on output, including gate and diffusion capacitance. Unlike effective resistance, gate and diffusion capacitance is proportional to width. For example, the transistor which has a k-times unit width has capacitance kC when C is the capacitance of gate, source or drain in a unit transistor.

A first-order system can be simply presented as following:

$$H(s) = \frac{1}{1+sRC} \qquad [3\text{-}1]$$

Its step response is

$$V_{out}(t) = V_{DD}e^{-\frac{t}{\tau}} \qquad [3\text{-}2]$$

where τ is equal to (R·C). As the propagation delay (t_{pd}) is the maximum time from the input $V_{dd}/2$ to the output $V_{dd}/2$, $V_{dd}/2$ could be introduced to equation [3-2]. Then t_{pd} could be rewritten as following:

$$t_{pd} = RC\ln 2 \qquad [3\text{-}3]$$

For simplicity, ln2 is usually ignored.

$$t_{pd} = RC \qquad [3\text{-}4]$$

It described the delay of a self-driving nMOS transistor.
For RC trees, Elmore delay is a tool that is commonly used to roughly calculate delays. Usually, the tree has only one source. Capacitors are located at different leaf nodes and every node has a matching resistance to the source. The calculation method of Elmore delay is to sum up the product of capacitance and resistance on every leaf node. The mathematical expression is

$$t_{pd} = \sum_i R_{is}C_i \qquad [3\text{-}5]$$

By using RC delay models, different delay has various τ values. For further simplicity, a new term called logical effort is introduced since the delay could be normalized relative to the characteristic of unit inverters. An alternative expression way of τ, which illustrate the delay of a gate, is

$$d = f + p \qquad [3\text{-}6]$$

where f is the effort delay (f) and p is the parasitic delay of the gate. The effort delay is a value relative to two variables, which are logical effort (g) and electrical effort (h).

Usually, the effort delay is the product of g and h.

$$f = g \cdot h \qquad [3\text{-}7]$$

According to Sutherland's definition (1998, [21]), Logical effort (g) of a gate is the ratio of the input capacitance of the gate to the input capacitance of an inverter that can diver the same output current. For example, a 2-input NOR gate (see Figure 3.5) has five-unit capacitance while the unit inverter has three-unit capacitance. The logical effort of this NOR gate is 5/3.

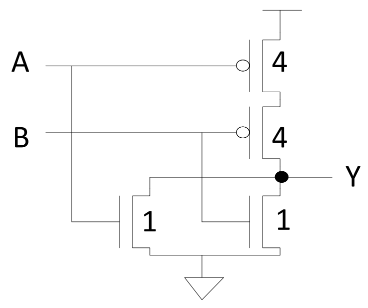


Figure 3.5 2-input NOR gate sized for unit resistance

When the gate drives zero load, the delay of the gate is called parasitic delay. Parasitic delay (p) is total diffusion capacitance on the output node relative to p_{inv}, which is the ratio of diffusion capacitance to gate capacitance in a unit inverter. In a 2-input NOR gate, total capacitance is 6 units and p_{inv} is 3 units. So the parasitic delay of this NOR gate is 2.

Electrical effort is the quotient of the output capacitance divided by the input capacitance and it can be expressed as:

$$h = \frac{C_{out}}{C_{in}} \qquad [3\text{-}8]$$

Before introducing the logical effort of multistage networks, another basic concept, which is termed branching effort (b), is worth explaining first.

When a logic gate drives some inputs of other logic gates, some of the drive current is available along the path, and some is off the path. The branching effort (b) at the output of a logic gate can be expressed as following:

$$b = \frac{C_{on-path} + C_{off-path}}{C_{on-path}} \qquad [3\text{-}9]$$

In multistage networks, the upper-case G is used to denote the path logical effort as:

$$G = \prod g_i \qquad [3\text{-}10]$$

The path logical effort (G) is the products of the logical efforts of each gate along the path. Similarly, the path electrical effort (H) is the products of the electrical efforts of each gate along the path and the path branching effort (B) is the products of the branching efforts of each gate along the path. Then the path electrical effort and path branching effort can be expressed as:

$$H = \prod h_i \qquad [3\text{-}11]$$
$$B = \prod b_i \qquad [3\text{-}12]$$

After explaining the path logical effort, path electrical effort and path branching effort, the path effort (F) could be defined as:

$$F = G \cdot B \cdot H \qquad [3\text{-}13]$$

In an adder or multiplier design, the path electrical effort (H) is usually set to 1 since the module is connected to a copy of itself.

The path delay (D) is the sum of the delays in each gate. Alternatively, it can be defined as the sum of the path effort delay (D_f) and path parasitic delay (P).

$$\begin{aligned} D &= \sum d_i = D_F + P \qquad [3\text{-}14] \\ D_F &= \sum f_i \\ D_F &= \sum f_i \end{aligned}$$

A simplified method of calculating the delays is to use the concept termed effort load. The branching effort at one gate on the critical path can be re-defined as following:

$$\begin{aligned} b_i &= \frac{C_{on-path} + C_{off-path}}{C_{on-path}} \\ &= \frac{load_i}{g_{i+1}} \qquad [3\text{-}15] \end{aligned}$$

$load_i$ is the effort represents the total capacitance on node i. In other words, it is equal to the sum of $C_{on-path}$ and $C_{off-path}$. g_{i+1} denotes the logical effort of the subsequent node.

Now let H is equal to 1, the formula [3-13] can be expressed as:

$$\begin{aligned} F &= G \cdot B \\ &= \prod g_i \, b_i \end{aligned}$$

$$= g_1 b_N \prod \text{load}_j \quad (1\leq i\leq N, \quad 1\leq j\leq N\text{-}1) \qquad [3\text{-}16]$$

Note that the module in the multiplier is connected to a copy of itself. Thus b_N is equal to load_0/g_1, where load_0 is the load effort relative to the inputs of subsequent module. Substitute b_N to equation [3-16].

$$F = \prod \text{load}_k \quad (0\leq k\leq N) \qquad [3\text{-}17]$$

Once the path effort has been determined, the ideal number of CMOS stages could be calculated by the following equation:

$$N = \text{round}(\log_\rho F) \qquad [3\text{-}18]$$

ρ is normally called best stage effort. It is a constant illustrates the number of the CMOS stages required in a least-delay path. The value of ρ is according to characteristic of unit inverters. In this project, the value of ρ is set to 4.

When the number of the ideal stage is more than the stages required in the logic design, inverters could be added to minimize the delay. However, the ideal stage cannot be always applied in the real design, since the ideal stage number may be less than the least number of stages required in the logic design.

After knowing how many stages required in the design, the stage effort could be determined. As is well known, if the product of a set of numbers is a constant, the sum of these numbers can be minimized by equalizing them. The path effort can be calculated by equation [3-13] and the path effort delay is the sum of the stage effort. Therefore, the stage effort can be known from equation [3-19] and the delay can be then calculated by equation [3-20] if the minimum delay is required.

$$\propto = F^{\frac{1}{N}} \qquad [3\text{-}19]$$

$$D = N \cdot \propto + P \qquad [3\text{-}20]$$

Ordinarily, the delay of the circuits is indicated by FO4 delays (the delay of a fanout-of-h inverter). Since FO4 has a delay of 5τ, we can get the FO4 delays of D in equation [3-20] by simply dividing D by 5.

.

Nowadays, the size of the chip has become smaller and the wires linking the transistors have become narrower. Crosstalk and interconnect resistance plays a much more important role in VLSI design since they are seriously affect the chip's energy and delay.

For simple, the computational methods of estimating delays of interconnect applied in this project is according to the real results of lab.

The delay of interconnect is roughly proportional to the length of wires. According to the lab experiments, the 20 x-grid length takes about 1-unit-inverter delay. If a wire across N x-grids, the delay approximates to N/20 unit inverter delay (D_{inv}).

$$D_{wire} = \frac{N}{20} \cdot D_{inv} \qquad [3\text{-}21]$$

Area calculation is also based on the lab experiments in this dissertation . All of cells have the same height of 10 y-grids, which is equal to 280 nm (2.8 um). The width of each cell is specified to a fixed value (see Appendix A) and thus it is easy to calculate the module width by summing over the basic cells' width along the layout placement.

Here is a simple example shows how to calculate the width of modules. Figure 3.6 illustrates the layout placement of a conventional (5:3) compressor.

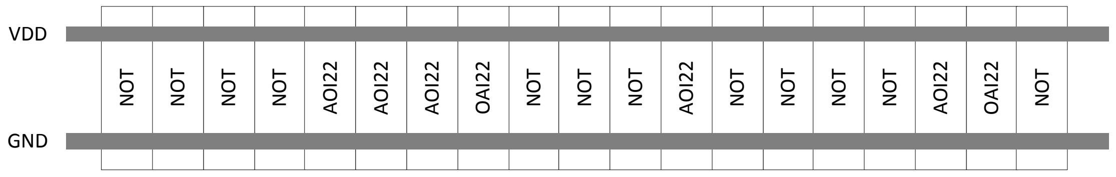


Figure 3.6 Layout placement of a conventional (5:3) compressor

There are 19 basic cells used in this module, including 12 NOT, 5 AOI22 and 2 OAI22. AOI22 is the 4-input gate implementing z = !(a.b + c.d) and OAI22 is the 4-input gate implementing z = !((a+b).(c+d)). The total length of this module is:

$$\begin{aligned} W_{total} &= 12 \cdot W_{NOT} + 5 \cdot W_{AOI22} + 2 \cdot W_{OAI22} \\ &= 12 \times 2 + 5 \times 5 + 2 \times 5 \ \text{grids} \\ &= 59 \ \text{x} - \text{grids} \end{aligned} \qquad [3\text{-}22]$$

Note that an x-grid in the lab is poly-poly repeat and each grid is around 420nm. Therefore the total length can be expressed as following:

$$W_{total} = 59 \ \times 420 \ \text{nm} = 24.78 \ \text{um}$$

Total area is

$$Area_{(5:3)compressor} = 2.8 \ \times 24.78 = 69.4 \ um^2 \qquad [3\text{-}23]$$

3.2 Module Design

This section introduces a set of modules used in the multiplier design. Generally, these modules can be classified into two types, which are accurate modules and approximate modules.

Accurate modules are the circuits which are able to perform the full arithmetic operation while approximate modules can mimic the function and perform rough calculations. According to the accuracy, the approximate compressors are divided into three grades as following:

High-accuracy compressors:	$90\% < \text{accuracy} \le 100\%$
Medium-accuracy compressors:	$70\% < \text{accuracy} \le 90\%$
Low-accuracy compressors:	$\text{accuracy} \le 70\%$

3.2.1 Approximate (4:2) Compressors

Compressors are usually named after the bit number of inputs and outputs. An (n:m) compressor has n input bits and m output bits.

In conventional compressor design, the number of input and output bits has following relationship:

$$\mathrm{m} \ge log_2(n+1) \quad (m, n \ge 0) \qquad [3\text{-}24]$$

The output bit can be equal or greater than $\log_2(n+1)$ if the output can provide an accurate sum. However, it is worth pointing out that more output bits will decrease the efficiency of multiplication, since more calculation stages are required in partial product reduction. Therefore the output bit of an ideal accurate multiplier is as following:

```
if $log_2(n+1)$ is integer
    $\mathrm{m} = log_2(n+1)$
else
    $\mathrm{m} = \mathrm{round}\,(log_2(n+1)) + 1$        [3-25]
```

This project explores several (4:2) approximate compressors (n=4, m=2).
Obviously, $\mathrm{m} < log_2(n+1)$. The common defect of (4:2) approximate compressor family is that it is not able to generate two carries when all four inputs are set to logical 1.

In this section, seven different (4:2) approximate compressors will be introduced. Each of them will generate errors in different conditions. It is interesting to see how these approximate compressors affect the performance of multipliers.

3.2.1.1 High-accuracy approximate (4:2) Compressors

As we defined in the beginning of this section, high-accuracy is from 90% to 100%. In this dissertation, accuracy can be expressed by the following formula:

$$\text{accuracy} = \frac{the\ number\ of\ true\ results}{the\ total\ number\ of\ tests} \qquad [3\text{-}26]$$

Similar to accuracy, error rate and estimation rate are two parameters to measure the effectiveness of a multiplier. Error rate is the ratio of the number of erroneous results to the total number of tests. Estimation rate is the proportion between erroneous results and correct results.

$$\text{error rate} = \frac{the\ number\ of\ erroneous\ results}{the\ total\ number\ of\ tests} \qquad [3\text{-}27]$$

$$\text{estimation rate} = \frac{the\ number\ of\ erroneous\ results}{the\ number\ of\ true\ results} \qquad [3\text{-}28]$$

(4:2) compressors have four inputs and thus have 2^4 different possibilities. When the total number of tests is equal to the number of possibilities, there is only one scheme that can make the accuracy achieve high-accuracy level. That is the compressor which will generate the error only when four inputs are logic 1.

This section will introduce one possible high-accuracy approximate compressor. It gives the sum 1+1+1+1 as 2 (decimal). Table 3.3 shows the Karnaugh Map

AB \ CD	00	01	11	10
00	0	1	2	1
01	1	2	3	2
11	2	3	2	3
10	1	2	3	2

Table 3.3 Karnaugh map for a high-accuracy approximate (4:2) compressor

The corresponding schematic, layout placement and Verilog program are shown in Figure 3.7, Figure 3.8 and Table 3.5 respectively. The function of the high-accuracy approximate (4:2) compressor can be expressed as following:

$$\text{SUM} = \overline{\overline{(A \oplus B) \oplus (C \oplus D)}} \qquad [3\text{-}29]$$

$$\text{CARRY} = \overline{\overline{(A + B) \cdot (C + D)} \cdot \overline{A \cdot B + C \cdot D}} \qquad [3\text{-}30]$$

The approximate compressor in Figure 3.7 has two XOR-delay stages. Though the results of the sum are all accurate and there is no difference compared to the conventional compressor design, less carry out number is sufficient to accelerate the multiplication speed since fewer bits are connected to the subsequent column of a multiplier.

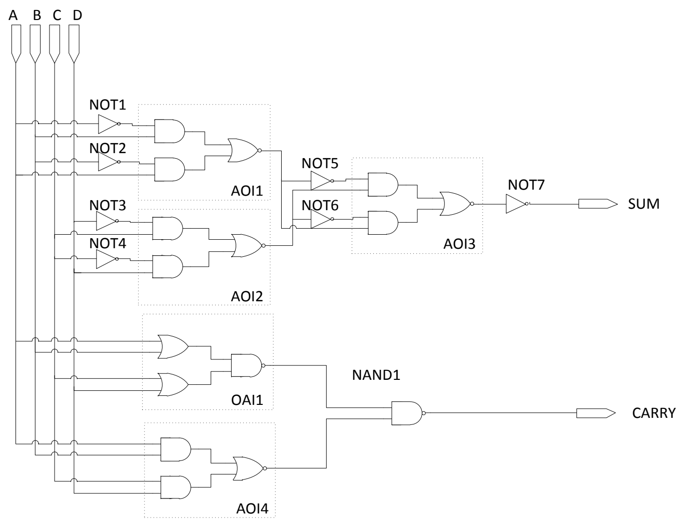


Figure 3.7 Schematic view of a high-accuracy approximate (4:2) compressor

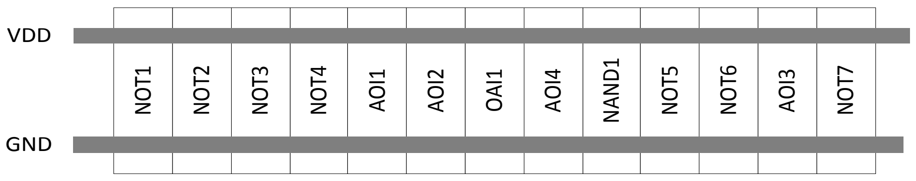


Figure 3.8 Layout placement of a high-accuracy approximate (4:2) compressor

cell	Fanout load	Effort load	Parasitic delay
NOT1	AOI1+track	6/3+6/20=2.3	3/3=1
NOT2	AOI1+track	6/3+4/20=2.2	3/3=1
NOT3	AOI2+track	6/3+6/20=2.3	3/3=1
NOT4	AOI2+track	6/3+4/20=2.2	3/3=1
AOI1	AOI3+NOT5+track	(6+3)/3+19/20=3.95	12/3=4
AOI2	AOI3+NOT6+track	(6+3)/3+15/20=3.75	12/3=4
OAI1	NAND1+track	4/3+4/20=1.53	12/3=4
AOI4	NAND1+track	4/3+0/20=1.33	12/3=4
NAND1	AOI1+NOT2+OAI1+AOI4+track	(6+3+6+6)/3+30/20=8.5	6/3=2
NOT5	AOI3+track	6/3+2/20=2.1	3/3=1
NOT6	AOI3+track	6/3+0/20=2	3/3=1
AOI3	NOT7+track	3/3+0/20=1	12/3=4
NOT7	AOI1+NOT1+OAI1+AOI4+track	(6+3+6+6)/3+20/20=8	3/3=1

[1] *assume cout is connected to input B* [2] *assume sum is connected to input A*

Table 3.4 Logical effort calculation for the high-accuracy approximate (4:2) compressor in Figure 3.7.

```
module com42_1_1 (sum,cout,a,b,c,d);
output sum,cout;
input a,b,c,d;
not (w1,a);
not (w2,b);
not (w3,d);
not (w4,c);
AOI22 A1(w5,w1,b,w2,a);
AOI22 A2(w6,w3,c,w4,d);
not (w7,w5);
not (w8,w6);
AOI22 A3(w9,w7,w6,w8,w5);
not (sum,w9);
OAI22 O1(w10,a,b,c,d);
AOI22 A4(w11,a,b,c,d);
nand (cout,w11,w10);
```

```
module AOI22(out,a,b,c,d);
output out;
input a,b,c,d;
and (w1,a,b);
and (w2,c,d);
nor (out,w1,w2);
endmodule

module OAI22(out,a,b,c,d);
output out;
input a,b,c,d;
or (w1,a,b);
or (w2,c,d);
nand (out,w1,w2);
endmodule
```

Table 3.5 The Verilog HDL program for the high-accuracy approximate (4:2) compressor in Figure 3.7.

Table 3.4 shows the logic effort calculation for high-accuracy approximate (4:2) compressor in Figure 3.5. It is necessary to point out that the method of calculation used in this project is the simplified one illustrated in 3.1.2. In other words, the approximate (4:2) compressor is always connected to the inputs of a copy of itself and therefore the path electrical effort (H) is set to 1. The width of the basic cells refers to the standard cell library which can be seen in Appendix A.

Assume the output sum is always connected to input a, the path effort of the critical path is:

$$F=\prod g\cdot b=2.3\times 3.95 \times 2.1 \times 1 \times 8=152.63$$

The ideal logic stage is 3. However, the minimum logic state required in the circuit is 5 after bubble pushing. So the logic stage is still 5.

$$N=\log_4 152.63=3.6\approx 3$$

Then the stage effort is:

$$\alpha=152.63^{\frac{1}{5}}=2.7$$

The delay can be calculated by the following equation.

D=N · α+∑p= 5×2.7+1+4+1+4+1=24.7 delay units ≈ 4.93 FO4 delays ≈ 246.7ps

Another high-accuracy approximate (4:2) compressor is also explored in this project. Similar to the compressor illustrated in above, this compressor fail to calculate the results of 1+1+1+1. The small difference is the erroneous result. Instead of 2 (decimal), 0 (decimal) is generated when each input is set to 1. Its Karnaugh map is shown in Table 3.6. The detailed design can be seen in Appendix B.

CD / AB	00	01	11	10
00	0	1	2	1
01	1	2	3	2
11	2	3	0	3
10	1	2	3	2

Table3.6 Karnaugh map for another high-accuracy approximate (4:2) compressor

3.2.1.2 Medium-accuracy approximate (4:2) Compressors

The accuracy of all of the medium-accuracy approximate (4:2) compressors in this project is 75%. In fact, there are a large number of medium-accuracy approximate (4:2) compressors can be designed according to the various accuracy. However, the aim of this project is to explore a (4:2) compressor that is able to achieve higher speed than that of high-accuracy approximate (4:2) compressors.

As described in the previous section, the high-accuracy approximate (4:2) compressor cut off one carry output from the conventional 4-input compressor design. This scheme can efficiently reduce the bit load of each column of a multiplier. However, the critical path of the multiplier is along a series of SUM connections. Thus minimizing the delay of SUM logic is a direct way of accelerating the speed.

The medium-accuracy approximate (4:2) compressors illustrated in this project provide a possible way of speeding up the SUM logic. It is able to simplify the circuits from 2-XOR-delay stages to 1-XOR-1-NAND (or NOR)-delay stages. Certainly, the 1-XOR-1-NAND (or NOR)-delay logic is not a complete function circuit. It mimics the SUM function as (A⊕B⊕C⊕D) and brings occasional errors.

In the (4:2) approximate compressor design, the errors can be roughly classified into two types. The first is SUM errors which are from SUM logic circuits. The second is the common errors of (4:2) compressors, which exists in every (4:2) compressor. The medium-accuracy approximate (4:2) compressors illustrated in this project integrate the two types into one condition and therefore achieve the error rate of 75%.

Since there are four probable errors in the SUM logic, a set of different 4-error compressors can be designed by setting various erroneous results. In this section, an example is shown. Table 3.7 describes the function of the approximate compressor. Errors will be generated when four inputs are set to "0000", "1100", "0011" and "1111". In fact, the medium-accuracy approximate (4:2) compressor will only generate the errors in the four situations if 1-XOR-1-NAND (or NOR)-delay logic is applied in the circuit design.

AB \ CD	00	01	11	10
00	1	1	3	1
01	1	2	3	2
11	3	3	3	3
10	1	2	3	2

Table 3.7 Karnaugh map for a medium-accuracy approximate (4:2) compressor

The schematic, layout placement and effort load are shown in Figure 3.9, Figure 3.10 and Table 3.8. The delay of this medium-accuracy approximate (4:2) compressor is 2.6 FO4 delays, which is 33% faster than the high-accuracy approximate (4:2) compressor in 3.2.1.

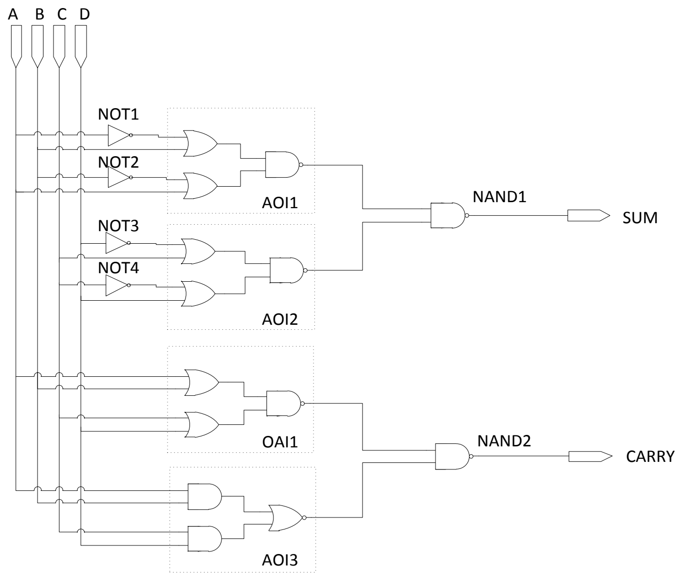


Figure 3.9 Schematic view of a medium-accuracy approximate (4:2) compressor

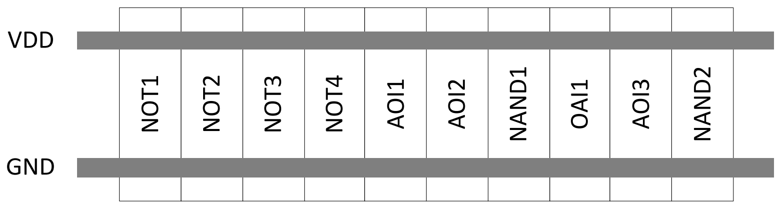


Figure 3.10 Layout placement of a high-accuracy approximate (4:2) compressor

cell	Fanout load	Effort load	Parasitic delay
NOT1	AOI1+track	6/3+6/20=2.3	3/3=1
NOT2	AOI1+track	6/3+4/20=2.2	3/3=1
NOT3	AOI2+track	6/3+6/20=2.3	3/3=1
NOT4	AOI2+track	6/3+4/20=2.2	3/3=1
AOI1	NAND1+track	4/3+4/20=1.53	12/3=4
AOI2	NAND1+track	4/3+0/20=1.33	12/3=4
NAND1	NOT1+AOI1+OAI1+AOI3+track	(3+6+6+6)/3+34/20=8.7	6/3=2
OAI1	NAND2+track	4/3+4/20=1.53	12/3=4
AOI3	NAND2+track	4/3+0/20=1.33	12/3=4
NAND2	NOT2+AOI1+OAI1+AOI3+track	(3+6+6+6)/3+23/20=8.15	6/3=2

[1] assume cout is connected to input b [2] assume sum is connected to input a

Table 3.8 Logical Effort calculation for the medium-accuracy approximate (4:2) compressor in Figure 3.9

Delay calculation in the critical path:

$F=\prod g\cdot b=2.3*1.53*8.7=30.61$

$N=\log_4 30.61=2.47\approx 2$

$\alpha=30.61^{\frac{1}{3}}=3.1$

$D=N\cdot\alpha+\sum p= 3\times 3.1+1+4+2=16.4$ delay units $\approx$ 3.28 FO4 delays $\approx$ 163.84ps

Another design for medium-accurate approximate (4:2) compressor can be found in Appendix C. It changes some CARRY logic in the situation where SUM errors will occur. Its Karnaugh map can be seen in Table 3.9.

CD AB	00	01	11	10
00	1	1	1	1
01	1	2	3	2
11	1	3	3	3
10	1	2	3	2

Table 3.9 Karnaugh map for another medium-accuracy approximate (4:2) compressor

3.2.1.3 Low-accuracy approximate (4:2) Compressors

The approximate (4:2) compressors which have even lower accuracy are also explored in this project. Generally, lower accuracy may lead to a simplified circuit. But the critical path of the multiplier is along a series of compressors and the accuracy of a multiplier's column can be expressed as following:

$$\text{accuracy}_{column} = accuracy_{SUM\_com}\cdot accuracy_{SUM\_com}\cdots accuracy_{SUM\_com}$$

where $accuracy_{SUM\_com}$ denotes for the accuracy of the SUM logic of a compressor.

Therefore, compressors, whose accuracy is low, will cause a large number of errors in partial product reduction stage. This project only explores the approximate compressors whose error rate is up to 31.25%. In other words, the low-accuracy approximate compressors in this project will generate errors in five situations. The designs of low-accuracy compressor are very similar to the designs of medium-accuracy compressors. They also applies 1-XOR-1-NAND (or NOR) –delay logic and thus their SUM logic generate errors in four situations. Certainly, the normal defect of (4:2) compressors cannot be avoided. The difference between the low-accuracy and medium-accuracy modules in this project is the placement of errors. The medium-accuracy modules, which illustrated in the previous section, hide the common defect in the SUM errors while low-accuracy modules set common error and SUM errors happened in different situation. Table 3.10 shows the Karnaugh map of a low-accuracy approximate (4:2) compressor.

CD / AB	00	01	11	10
00	0	0	2	0
01	1	2	3	2
11	2	2	2	2
10	1	2	3	2

Table 3.10 Karnaugh map for a low-accuracy approximate (4:2) compressor

The schematic, layout placement and logical effort can be seen in Figure 3.11, Figure 3.12 and Table 3.11 respectively.

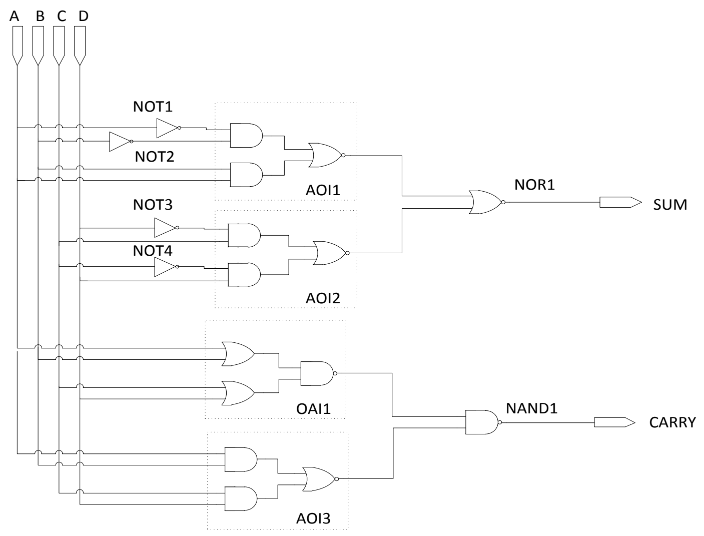


Figure 3.11 Schematic view of a low-accuracy approximate (4:2) compressor

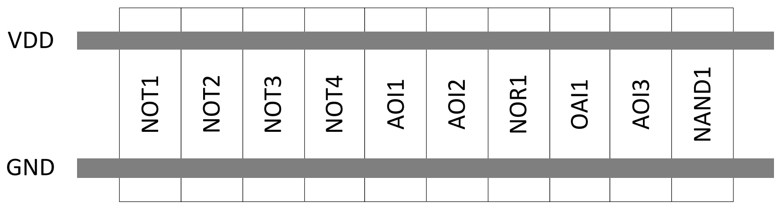


Figure 3.12 Layout placement of a low-accuracy approximate (4:2) compressor

cell	Fanout load	Effort load	Parasitic delay
NOT1	AOI1+track	6/3+6/20=2.3	3/3=1
NOT2	AOI1+track	6/3+4/20=2.2	3/3=1
NOT3	AOI2+track	6/3+6/20=2.3	3/3=1
NOT4	AOI2+track	6/3+4/20=2.2	3/3=1
AOI1	NOR1+track	5/3+4/20=1.87	12/3=4
AOI2	NOR1+track	5/3+0/20=1.67	12/3=4
NOR1	NOT1+AOI1+OAI1+AOI3+track	(3+6+6+6)/3+34/20=8.7	9/3=3
OAI1	NAND1+track	4/3+4/20=1.53	12/3=4
AOI3	NAND1+track	4/3+0/20=1.33	12/3=4
NAND1	NOT2+AOI1+OAI1+AOI3+track	(3+6+6+6)/3+23/20=8.15	6/3=2

1 assume cout is connected to input b *2 assume sum is connected to input a*

Table 3.11 Logical Effort calculation for the low-accuracy approximate (4:2) compressor in Figure 3.11

The delay of the low-accuracy compressors is similar to that of the medium-accuracy compressors. They are 27% faster than the high-accuracy compressors.

$F=\prod g\cdot b=2.3*1.87*8.7=37.42$

$N=\log_4 37.42=2.6\approx 2$

$\alpha=37.42^{\frac{1}{3}}=3.3$

$D=N\cdot\alpha+\sum p=3\times 3.3+1+4+3=17.9$ delay units $\approx$ 3.58 FO4 delays

More designs for low-accuracy approximate compressors can be found in Appendix D. Their functions are described in Table 3.12.

CD \ AB	00	01	11	10
00	0	1	2	1
01	1	3	3	3
11	2	3	2	3
10	1	3	3	3

CD \ AB	00	01	11	10
00	0	1	2	1
01	0	2	2	2
11	2	3	2	3
10	0	2	2	2

Table 3.12 Karnaugh map for other low-accuracy approximate (4:2) compressors

3.2.2 Accurate (3:2) Counters

In order to distinguish approximate modules, full adders are termed accurate (3:2) counters in this dissertation. As explained in 3.1, the logic style applied in this project is complementary CMOS. Thus all of the gates are inverting. Recall that the full adder logic is

$$S = a \oplus b \oplus c \qquad [3\text{-}31]$$
$$Cout = a\#b\#c \qquad [3\text{-}32]$$

After bubble pushing, the function can be expressed as following equations:

$$S = \overline{\overline{\bar{a}b + \bar{b}a} \cdot \bar{c} + (\bar{a}b + \bar{b}a) \cdot c} \qquad [3\text{-}33]$$

$$Cout = \overline{\overline{ab + (a+b) \cdot c}} \qquad [3\text{-}34]$$

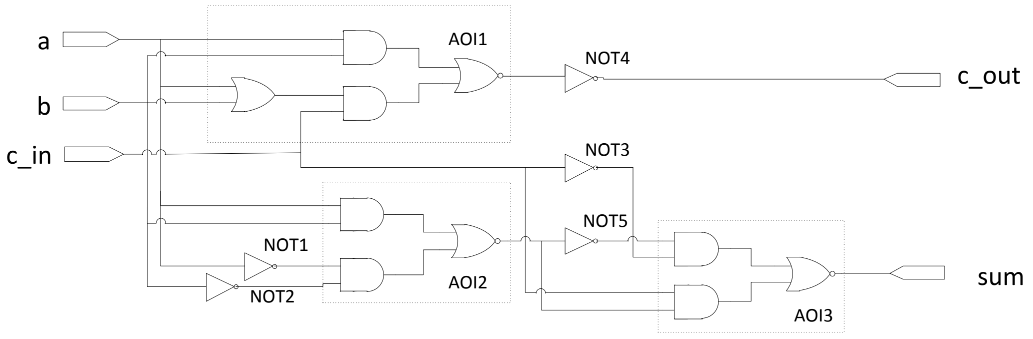


Figure 3.13 Schematic view of an accurate (3:2) counters

The corresponding schematic of an accurate (3:2) counter is shown in Figure 3.13. With the aim of evaluating the delay of the (3:2) counter cell, the basic cells are placed as Figure 3.14. The top grey bar denotes VDD while the bottom bar denotes the ground.

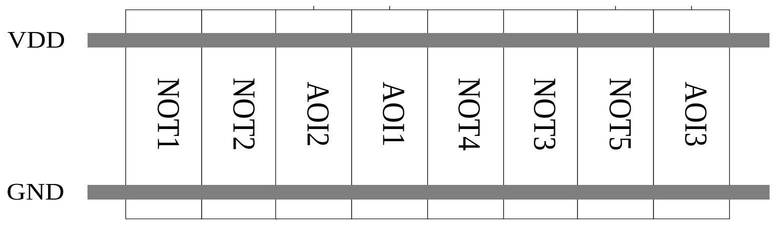


Figure 3.14 Layout placement of the accurate (3:2) counter in Figure 3.13

The logical effort for the accurate (3:2) counters is in Appendix E. The worst delay of the accurate (3:2) counter is about 224 ps.

3.2.3 Accurate (5:3) Compressors

The conventional (5:3) compressors consist of two series accurate (4:2) counters. However, the delay of conventional (5:3) compressors is extremely large since four XOR logic gates are on the critical path.

An optimized (5:3) compressors are introduced in [4] (Chang, Gu and Zhang, 2004) and [24] (Radhakrishnan and Preethy, 2004). Their method of accelerating the speed is to make two XOR gates operate in parallel. Therefore the delays of the critical path can be reduced to three-XOR-stage delays. Recall that the basic operation of conventional (5:3) compressors is:

$$\text{SUM} = A \oplus B \oplus C \oplus D \oplus \text{CIN} \tag{3-35}$$
$$\text{COUT} = A\#B\#C \tag{3-36}$$
$$\text{CARRY} = (A \oplus B \oplus C)\#D\#\text{CIN} \tag{3-37}$$

The modified equations for sum and carry are as following:

$$\text{SUM} = \overline{\overline{\text{A} \oplus \text{B} \oplus \text{C} \oplus \text{D} \oplus \text{CIN}}} \quad [3\text{-}38]$$

$$\text{COUT} = \overline{\overline{(\text{A} + \text{B}) \cdot (\text{C} + \text{D})}} \quad [3\text{-}39]$$

$$\text{CARRY} = \overline{\left((\text{A} \oplus \text{B} \oplus \text{C} \oplus \text{D}) + \overline{\text{CIN}}\right) \cdot (\overline{\text{A} \oplus \text{B} \oplus \text{C} \oplus \text{D}} + \overline{(A \cdot B) + (C \cdot D)})} \quad [3\text{-}40]$$

The corresponding schematic, layout placement can be seen in Figure 3.15 and Figure 3.16 respectively. More

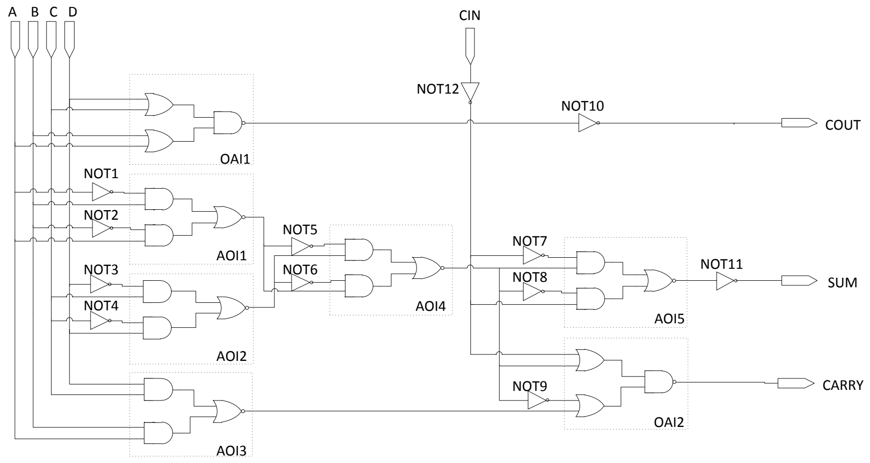


Figure 3.15 The schematic of an accurate (5:3) compressor

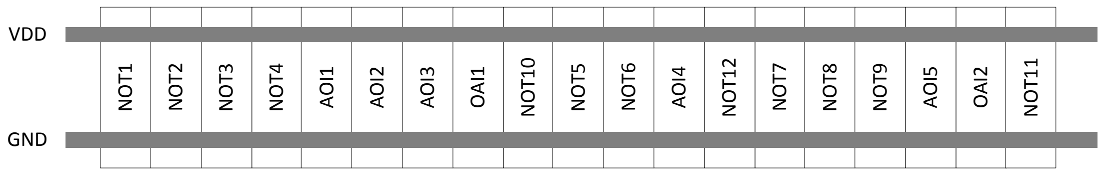


Figure 3.16 Layout placement of the accurate (5:3) compressor in Figure 3.15

3.2.4 Performance Comparison

Cell family	**Error rate**	**SUM delay(ps)**	**Internal carry delay(ps)**	**CARRY delay(ps)**	**MOSFET quantity**	**Area (um^2)**
full_adder	0%	224	N/A	98	42	29.4
COM53	0%	374	136	366	80	69.4
COM42_H_1	6.25%	246.7	N/A	132	58	49.4
COM42_H_2	6.25%	255	N/A	249.9	70	60.0
COM42_M_1	25%	163.84	N/A	130	48	40.0
COM42_M_2	25%	160	N/A	134	40	40.0
COM42_L_1	31.25%	179	N/A	130.6	56	40.0
COM42_L_2	31.25%	179	N/A	130.6	56	40.0
COM42_L_3	31.25%	163.84	N/A	130	48	40.0

Table 3.13 The performance of compressor family
* The width of transistors is 100 nanometers
* Cell naming: COM[inputs - output]_[accuracy level]_[cell type]
Example: COM42_H_2 -- (4:2) compressor cell, high-accuracy, type 2

Table 3.13 shows the performance of the compressors used in this project. All of these cell designs could be found in Appendix B-E. Though the approximate (4:2) compressors have occasional errors, their speed is faster or roughly equal to that of full adders. Among the (4:2) compressors, COM42_M_2 achieves the fastest speed, the least CMOS quantity, and the smallest size since it has a simpler CARRY logic. In the entire cell, CARRY delays are hidden by the SUM delays so that the critical path of trees is along series SUM logic circuits.

3.2 Topology Design

As explained in 2.3, the topologies applied in this project are trees since trees are very fast structures for summing partial products (Flynn and Oberman, 2001[10]).

In general, multiplier topologies define the methods of connections and thus they seriously affect the speed, power dissipation and area, as well as error rate. This section analyzes tree structures according to their allocation schemes. A new structure, which has a more compact allocation, will be introduced in 3.3.2. In Chapter 4, various compressors will be applied in the structures with different allocation schemes, which are aimed to explore the relationships among the speed, error rate and topologies.

3.3.1 Allocation Schemes

Allocation schemes closely related to the efficiency use of compressors. According to the density of integration, trees can be divided into three types as shown in the Figure 3.17.

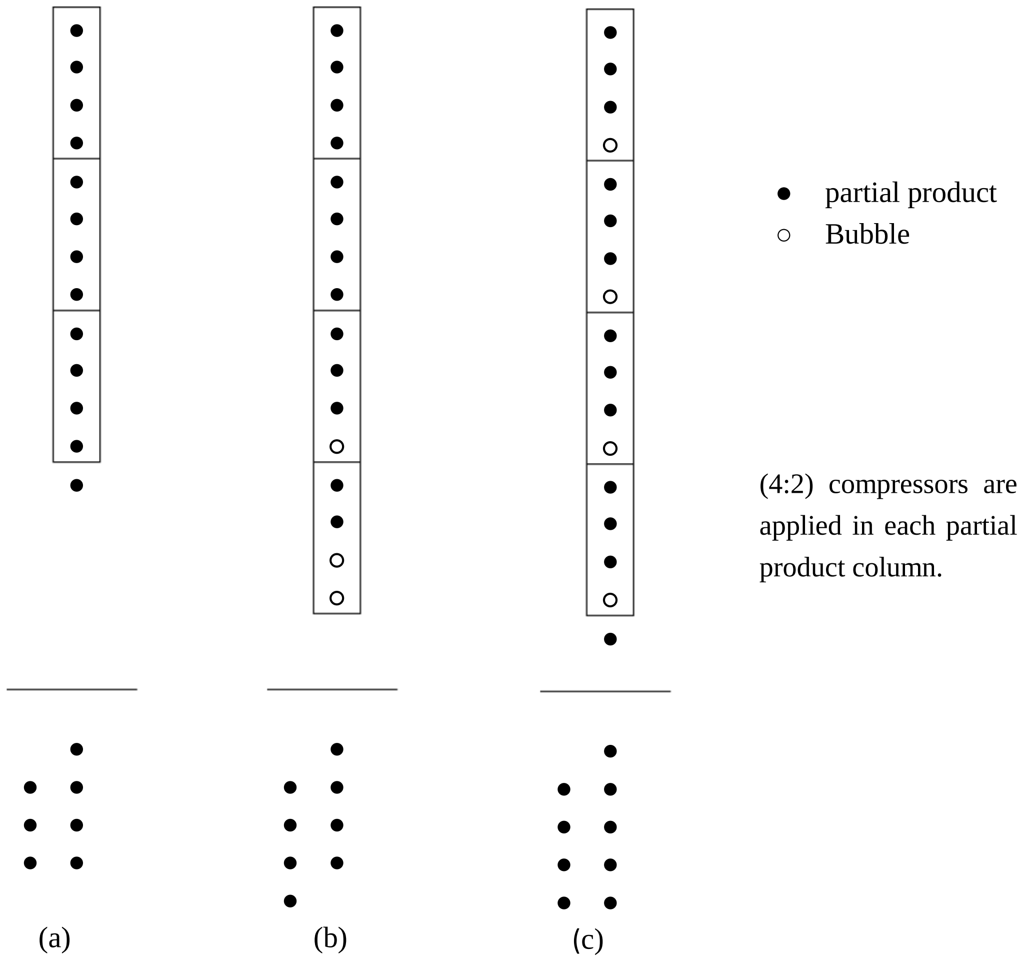

● partial product
○ Bubble
(4:2) compressors are applied in each partial product column.
(a)
(b)
(c)

Figure 3.17 Allocation schemes: (a) Tight (b) Medium (c) Loose

When an input of compressors is not connected to the ground or any unavailable terminal, we say the input have a bubble in this dissertation.

According to the bubble locations, the three allocation schemes can be defined as following:

- Tight: No bubble exists in the partial product column. The residue partial product bits will be calculated in the next reduction stage.
- Medium: Most of the compressors are in tight like style. In other words, only a small number of compressors have bubbles in the reduction stages.
- Loose: Bubbles exist in all compressors which are located in the partial product column.

Obviously, tight scheme have several advantages in single partial product column:

1. minimum number of compressors are used in partial product reduction stage, which is able to reduce the area and power dissipation.
2. minimum number of SUM bits are generated. Fewer bits may reduce the reduction stages of the current partial product column.
3. minimum number of CARRY bits are generated. Fewer carries out may reduce the reduction stage of the subsequent partial product column

However, when tight scheme is applied in the whole tree structure, the speed of the multiplier will be extremely slow since more partial product reduction stages are required. Figure 3.18 shows two 8 bit by 8 bit tree multipliers. One applies the medium scheme while the other uses the tight scheme. Though the number of compressors used in tight-style based tree is slightly smaller than that used in medium tree, tight-style tree have much more reduction stages along the critical path. The irregular allocation of tight scheme brings difficulty in the use of (5:3) compressors, which normally achieve better performance than that of two series full adders.

Loose method certainly has the worst performance in speed, power dissipation and area, because the efficient use of compressors will cause more SUM dots and CARRY dots generated in reduction stage. At last the delays will be a very large number.

Figure 3.19 shows the relationship among the bubble quantity, compressor usage rate and the number of partial product reduction stages. The compressor usage rate goes high if the multiplier can fully use the inputs. It is worth pointing out that the number of partial product reduction stages will be small only on the condition that there is a medium quantity of bubbles in the compressors.

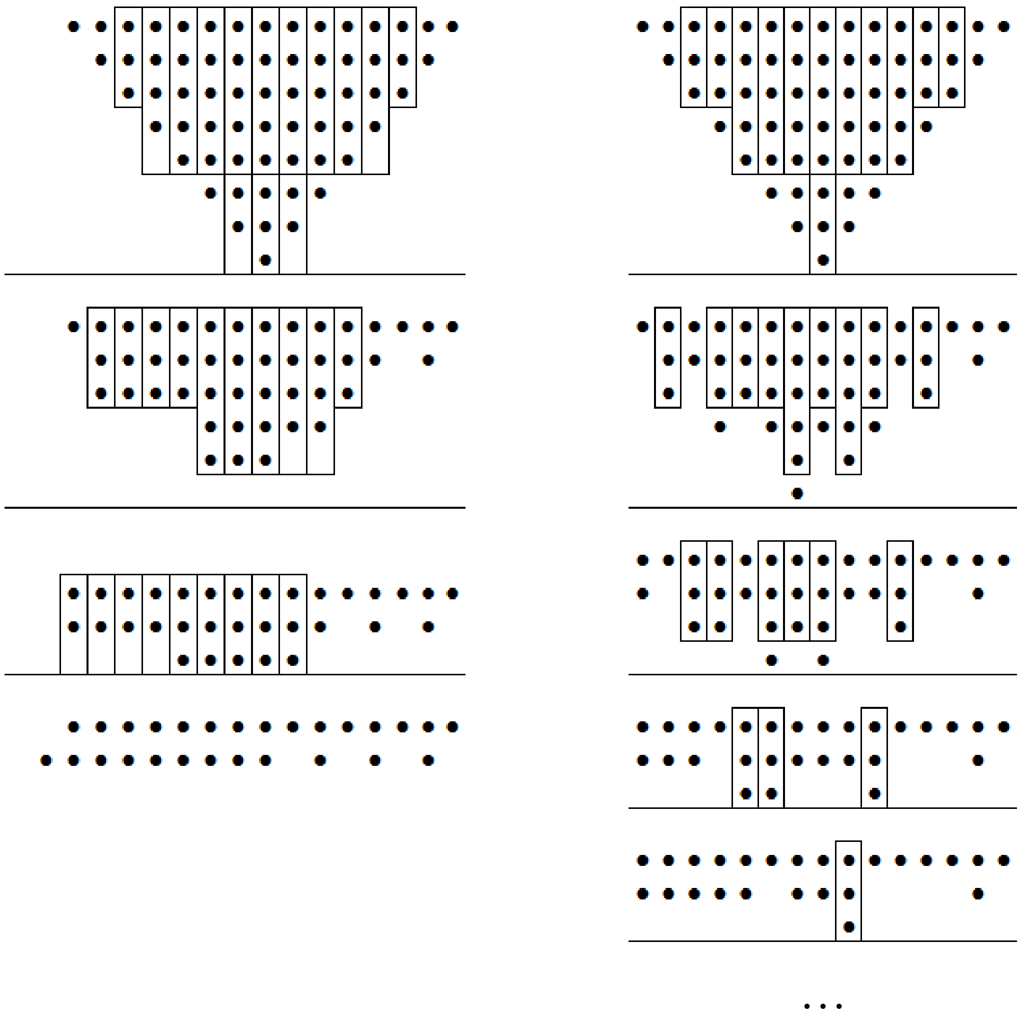


Figure 3.18 (a) A medium scheme used in an 8 bit by 8 bit tree multiplier

(b) A tight scheme used in an 8 bit by 8 bit tree multiplier

(the multiplier uses accurate (3:2) and (5:3) compressors in partial product reduction stage)

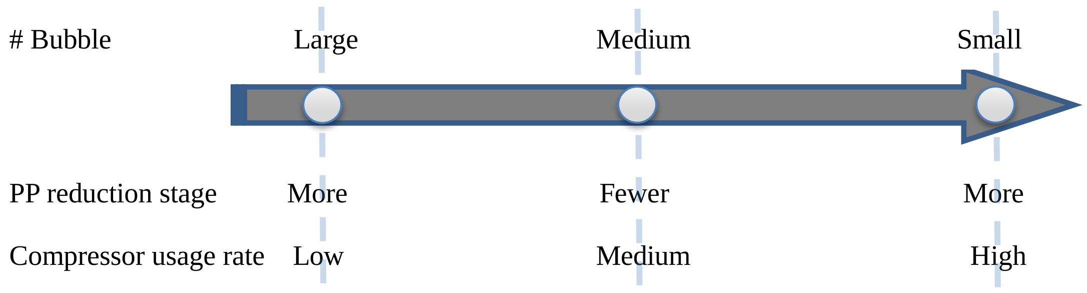


Figure 3.19 The relationship among the bubble quantity, compressor usage rate and the number of partial product reduction stages

3.3.2 Bottom-up Tree

Usually, the power can be measured by the transistor quantity. In (4:2) compressor based multiplier designs, the multipliers which require more power are the ones consist of more compressors. It is easy to understand the compressor usage is relative to the number of compressors used in a tree multiplier. Recall what is explained in 3.3.1, bubble allocation schemes will seriously affect the number of PP (partial product) reduction stages. Thus, a method to balance the speed and power consumption is to find a certain number of bubbles in a tree.

Unfortunately, a large number of medium trees could be implemented since the bubble quantity and bubble location have not been defined in medium tree design. Moreover, it is difficult to find a rule to calculate the number of bubbles in partial product column, since too many allocation schemes can be applies in tree multiplier design and various allocation schemes have various bubble locations.

Inspired by Dadda tree, a new structure, which is called bottom-up tree in this dissertation, provides a possible approach to balance the speed and power.

As described in 2.3.3, the Dadda tree applies full adders and half adders in the partial product reduction stage. The bottom-up tree, however, uses more (m,n) compressors in a multiplier design. A C code used to implement bottom-up trees in this project. In that code, two different (m,n) compressors can be applied in the multiplier design besides full adders.

Figure 3.21 shows the dot diagram for a (3:2) and (5:3) compressor based tree multiplier. It looks very different from the conventional dot diagrams since the process of analyzing is in an opposite direction. In multiplier designs, a tree structure is usually connected to a carry-propagate adder, whose inputs quantity is 2 in each bit. It indicates the height of the result of the above tree is 2 bit. In order to efficiently use the compressors, the height of the previous stage is equal to 3 in this example. Similarity, if the height of stage is 3, stage 2 will have 5-bit height. The algorithm of calculating the ideal stage height can be expressed as following:

```
ASSUME (3:2) and (p,q) compressors are used (3<p);
             the current ideal stage height is $H_{current}$
             the previous stage height is $H_{previous}$;
             variable $f = 2^{Hcurrent}-1$
IF           $1< f \leq 3$
THEN       $H_{previous} = 3$
IF           $m<f$
THEN       $H_{previous}$=round(f/p)·p+function (reminder(f/p)/3)
where function (x)=round (x)+1 when x is not integer and x>1
```

function (x)=x when m is a integer
function (x)=1 when m=1

Figure 3.20 The algorithm of calculating the stage height of a bottom-up tree.

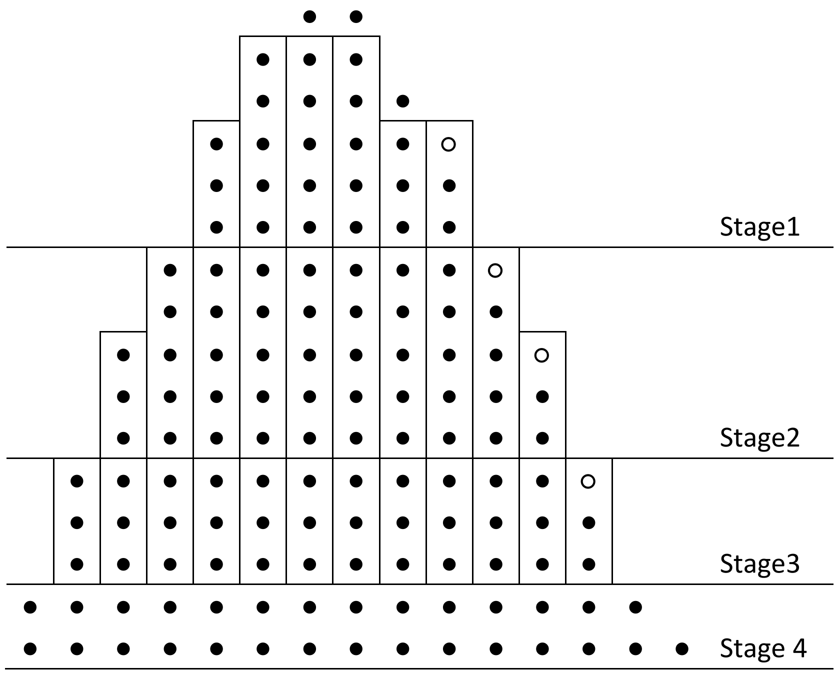


Figure 3.21 Dot diagram of a (3:2) and (5:3) compressor based tree multiplier

If the process of reduction has N stages, the ideal height of partial product reduction stages can be used to allocate the dots from stage N to 2. The allocation scheme will be extremely complex in the first stage since the remaining dots will affect the SUM and CARRY dot allocation in stage (2). The C code just applies a rough algorithm of inserting bubbles. As Figure 3.21, the ideal bottom-up tree only has bubbles in the last stage. However, the rough algorithm may fail to insert bubble in stage (2) under some special situations.

In general, the bottom-up tree is an optimized Medium style, in which the dot loads of each stage are maximized. Therefore it will achieve a faster speed than the one applies conventional top-down reduction scheme. Compared to other equivalent-speed trees, bottom-up trees use fewer compressors since only a small number of bubbles are generated in partial product reduction phase.

3.4 Verification Strategies

Since a large number of tests are required to estimate the error rate, a well-planned verification strategy is significant in the approximate multiplier design.

When planning the verification, the first question is to determine the level of granularity for the verification effort (Bergeron, 2000 [25]). The verification in this project is in unit-level. In other words, the testbenches is only used to ensure that basic functionality of multipliers is operational.

This project can be divided into two design phases. The first is module design, in which a set of compressors will be explored. The second is multiplier topology design. This dissertation classifies multipliers into three types according to the allocation schemes. Because the modules have a small number of probable results, the method of verification is to create a pre-determined input sequence to the compressor module. Figure 3.22 shows a simple testbench. It is used to check the results according to the truth table of a (4:2) compressor.

```
// Define the stimulus (top level module)
module stimulus;

// Set up variables
reg a, b,c,d;
wire sum;
wire cout;;

// Instantiate the 4-bit full adder. call it FA1_4
com42_1_1 com1 (sum,cout,a,b,c,d);

// Setup the monitoring for the signal values
initial
begin
   $monitor($time," A= %b, B=%b, C= %b, D=%b    SUM= %b, COUT= %b",
                  a, b, c,d, sum, cout);
end

// Stimulate inputs
initial
begin
a=0;b=0;c=0;d=0;
#10 a=0;b=0;c=0;d=1;
```

```
#10 a=0;b=0;c=1;d=0;
#10 a=0;b=0;c=1;d=1;
  ...
end
endmodule
```

Figure 3.22 Testbench for a (4:2) compressor

The directed test environment is suitable for the compressors that have a small number of probable results since every possible event could be checked. However, when directed tests are applied to the verification for a tree multiplier module, a number of problems related to reusability and maintenance are exposed:

- Testbench can become extremely complex.
- It is difficult to read and understand since a large number of directed test are designed in the testbench. It takes time to understand the intention of each test.
- Testbench is hard to modify only if the testbench is well understood.
- It is impossible to reuse the tests from module to system level.
- Corner cases are prone to ignore due to significant manual effort

Therefore, another strategy, which is called random verification, is normally used for verifying the functionality of system-level modules. Under a random test environment, random sequences of zero and one will be packed into the operands. This way brings dozens of pros.

- The testbench and the compressor modules are not required to be fully understood. Random tests will be generated automatically.
- Unexpected conditions or corner cases may be hit.
- Testbench is easy to read and maintain.

But practically, pure random tests are rarely used in verification, because the pure random data are usually different from the real data. A number of the generated cases are invalid.

As is well known, the best way of verifying the modules is using real data in tests. The approximate multiplier design can be used for value speculation and DSP operation and thus the real data can be considered to be benchmarks used for arithmetic and media operations. Kelly, Phillips and Al-Sarawi (2010[26]) provided the cumulative distribution of multiplication operands in benchmark in their paper. Unfortunately, these benchmark programs are not available in our lab.

In order to mimic the distribution of data in testbench, constrained random tests are applied in verification. In this project, SystemVerilog is used as the hardware language since it provide a good support to the constrained random stimulus generation, Figure 3.23 shows the main part of the testbench. In the class Rand, the random variables are defined first. Then constraints are created to define the range of random data. Note that the constraint expression does not use begin…end. In SystemVerilog syntax, curly braces are for declarative expression while begin…end statements are for procedural expression. In the main program, Function assert() is used to wake up random data generation..

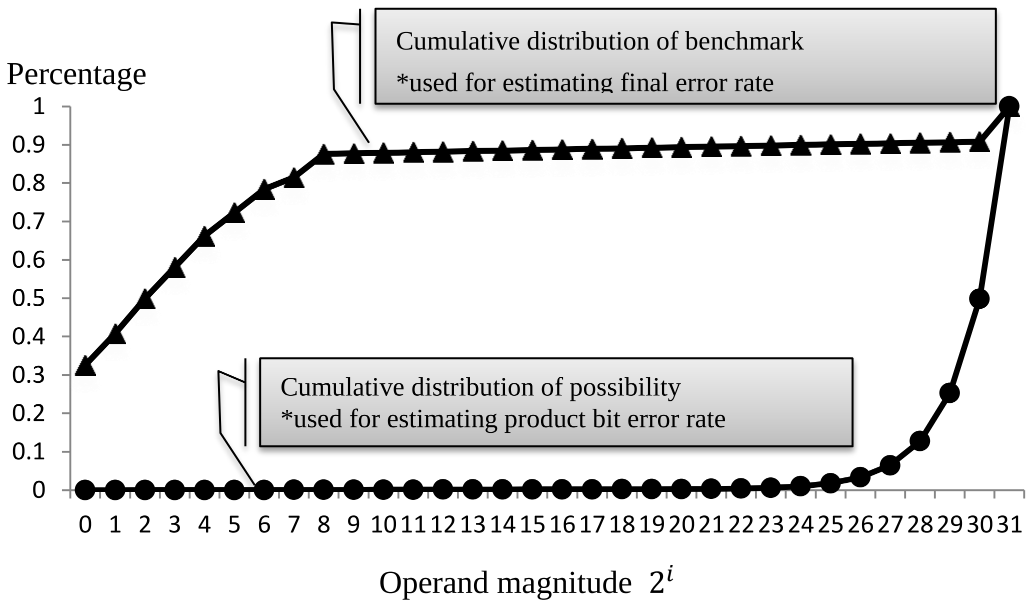


Figure 3.23 Cumulative distribution of multiplication operands in benchmark (Adapted from : Kelly, Phillips and Al-Sarawi, 2010[26])

```
class Rand;
   int a=1;
   rand bit [32:0][31:0] inA,inB;
   rand bit inC;
   constraint c1 {      // the constraints on the random variables
   inA[1] >= 0;inA[1] < 2;
   inB[1] >= 0;inB[1] < 2;
                  …
}
endclass

program automatic STI (main_bus.soft bus);
assert(r1.randomize());
bus.X<=r1.inA[y];
bus.Y<=r1.inB[y];
bus.CIN<=r1.inC;
…
endprogram
```

Figure 3.24 The main part of multiplier testbench.

Chapter 4: EXPERIMENT RESULTS AND DISCUSSION

The previous chapters illustrate a set of compressors as well as some popular tree topologies. The performance of all compressors used in this project is shown in section 3.3.4. All of the schemes, which apply different compressor cells in different 32 by 32 bit multiplier topologies, have been tried. It is very interesting to see how these approximate compressors affect the speed, power dissipation, and error rate. Several experiment findings as well as approximate multiplier examples are described in this chapter. The approximate multipliers trade off their accuracy against faster speed, lower power consumption and smaller area. Certainly, the aim of approximation is to win higher performance with fewer errors.

4.1 Conclusion I

No obvious relationship between the accuracy of approximate compressors and the accuracy of approximate multipliers.

Tree topology	Compressors	Error rate	Estimation rate
Conventional Medium style tree multiplier	**COM42_H_1**	12%	13%
	COM42_H_2	12%	13%
	COM42_M_1	100%	∞
	COM42_M_2	100%	∞
	COM42_L_1	65%	185%
	COM42_L_2	59%	144%
	COM42_L_3	45%	82%
Bottom up Medium style tree Multiplier	**COM42_H_1**	11%	12%
	COM42_H_2	11%	12%
	COM42_M_1	100%	∞
	COM42_M_2	100%	∞
	COM42_L_1	33%	49%
	COM42_L_2	32%	47%
	COM42_L_3	21%	27%

Table 4.1 Estimation results of (4:2) compressor based tree multipliers

According to Table 4.1, though medium-accuracy compressors have an acceptable error rate, a large number of errors will occur when these compressors are used to build up a whole tree multiplier. That is mainly caused by the uneven distribution of 1 and 0. Recall the testbench is able to generate constrained random tests according to the distribution of multiplication operands in benchmark and most of the data are from 0 to 500 (decimal). The exact addition 0+0=0 becomes extremely significant in approximate multipliers since most of the addition results from bit 9 to bit 31 are 0.

The error rate of bottom-up trees looks slightly better than that of conventional trees. That is because accurate compressors are applied in 18 partial product columns of the bottom-up tree. This scheme balances the speed of every column. Moreover, it will improve the error rate of multipliers.

4.2 Conclusion II

The number of bubbles may affect the accuracy of approximate mutlipliers.

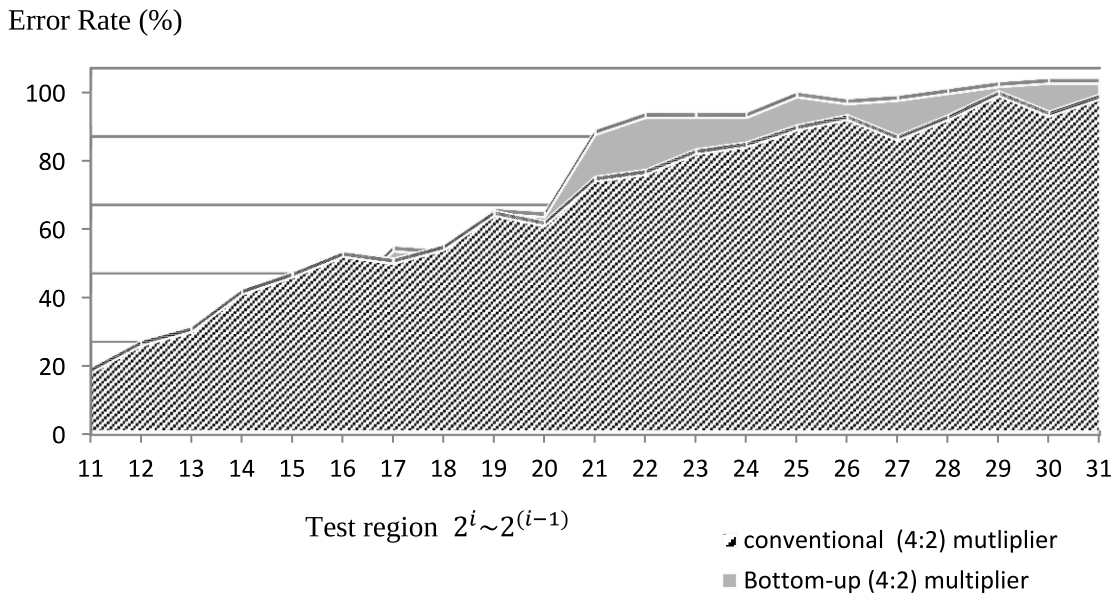


Figure 4.1 Comparison of the error rate between conventional (4:2) multipliers and bottom-up (4:2) multipliers

In order to analyze the accuracy in different data region, the testbench has divided the tested data into 31 regions:

Region 1: 0≤data<2

Region 2: 2≤data<4

…

Region n: 2^{n-1}≤data<2^{n} (n≥2)

Since the bottom-up trees in this project applies accurate compressors in the first 9 partial product columns and all of the results which are less than 512 are exact, Figure

4.1 ignores test region 1 - 10. Obviously, the error rate of the bottom-up tree multiplier is much higher from region 20.

As explained in section 3.3.2, the bottom-up tree is usually faster and has a upper-medium tight structure. Thus, fewer bubbles can be found in this structure.

Unfortunately, bubbles may be good things to approximate multipliers though they will reduce the efficiency of partial product reduction process. As well known, the common error of (4:2) compressors will occur when all of the inputs are set to 1. Assume a bubble is inserted to any input of a (4:2) module. The approximate compressor will become exact. That's the reason why bubbles may increase the accuracy of approximate multipliers. But in fact, the relationship between bubbles and error rate is related to the hardware structure of compressors. It is not safe to say that more bubbles will kill more errors.

4.3 Conclusion III

The error rate of each partial product column is related to the number of approximate compressors.

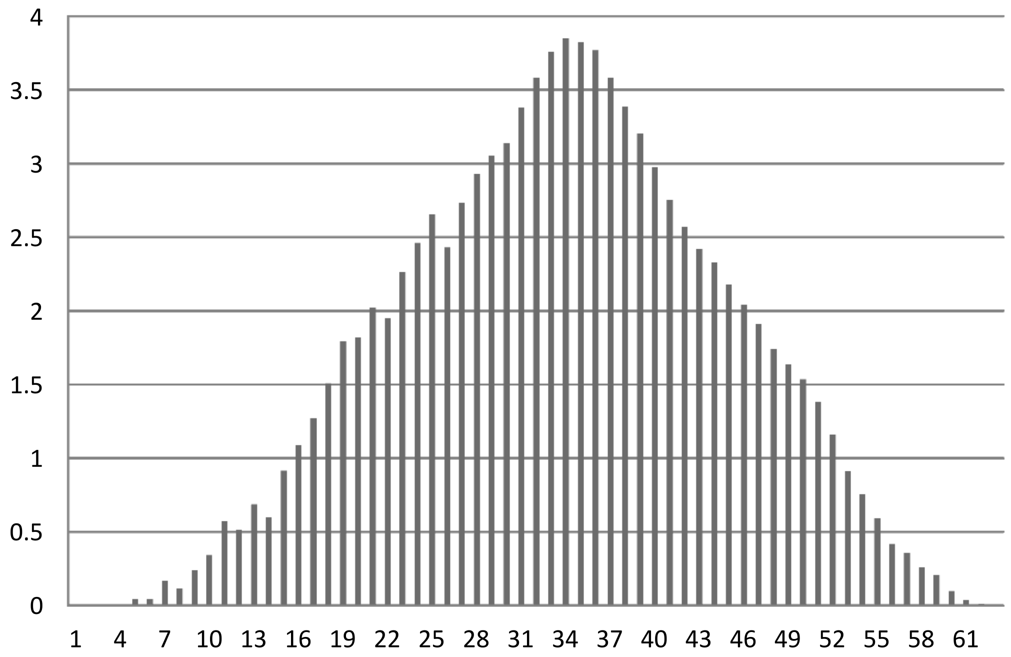


Figure 4.2 Error rate of each product bit

* The result is from (4:2) compressor based conventional tree structure

Generally, the errors in partial product column can be classified into two types. The first is called absolute error, which is caused by the approximate SUM logic along the path. The second is termed relative error. The cause of relative errors is much more complex. The errors from approximate CARRY logic will seriously affect the addition result of subsequent column. SUM logic, however, will also bring relative errors since the current wrong addition result will affect the CARRY results of next partial product reduction stage.

Figure 4.2 can roughly describe the error rate of each product bit. The errors in Figure 4.2 include relative errors and absolute errors. It can be seen that the bit 35-37 achieve highest error rate, where a large number of approximate compressors are used. On the contrary, the columns, which consist of fewer approximate compressors, have a higher accuracy. Thus, we can say that there is a close relationship between the number of approximate compressors and error rate. But the error rate is seriously affected by the distribution of tested data and the compressors in different reduction stage and different column have different accuracy. The calculation of the error rate in tree-level is not a simple mathematical operation.

4.4 Conclusion IV

Thee multipliers can achieve an acceptable error rate if only a small number of low-accuracy are used. COM42_M modules are not suitable for building up multipliers.

According to Table 4.1, the multipliers can get an accuracy of 88% if high-accuracy compressors are used in a whole tree. However, the error rate of low-accuracy compressor based tree multiplier is extremely low.

Though low-accuracy compressors are not suitable for building up a whole tree, the multipliers can get an equivalent perfect accuracy when low-accuracy compressors are partly used. There are two ways of using low-accuracy compressors. The first approach is to use these faster cells in the middle and use accurate cells near MSB (Most Significant Bit) and LSB (Least Significant Bit). It is possible since the speed of multiplier columns is uneven. However, this method may only slightly improve the accuracy. A large number of series approximate compressors are still located in the critical path. The second approach is based on Conclusion V. Approximate compressors can be allocated in the stages which have less effort on error rate. Though this may decrease the speed, the accuracy may be improved a lot.

As explained in 4.1, the operation 0+0=0 is important in multipliers. So COM42_M modules cannot be used in any tree since it will fail in the case that all inputs are set to 0.

4.5 Conclusion V

Errors in earlier stage in a tree give greater inaccuracy to final addition result.

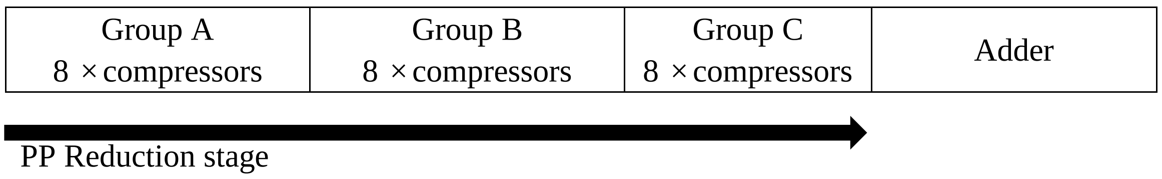


Figure 4.3 The critical path of a tree multiplier

In order to see how absolute errors in different stages affect the final result, an experiment is designed as shown in Table 4.2. Approximate compressors are only used in the Group A,B and C, which are located in the critical path as shown in Figure 4.3. Other compressors in the tree are all accurate.

Test	Group A	Group B	Group C	Error quantity
1	Approximate compressors	Accurate compressors	Accurate compressors	1046
2	Accurate compressors	Approximate compressors	Accurate compressors	110
3	Accurate compressors	Accurate compressors	Approximate compressors	3

Table 4.2 Tests for evaluating the errors in different reduction stages

According to Table 4.2, obviously, the 8 compressors in early stage bring more errors to the final results. That is because the errors in early stage will affect the SUM and CARRY results in subsequent stages.

Based on above conclusion, a merged compressor based tree can be built up as:
high-accuracy compressors are used in the first reduction stage;
COM42_L cells are used in the subsequent stages.

According to the simulation results, the accuracy is also up to 88%.

Chapter 5: CONCLUSION AND FUTRURE WORK

Approximation circuits provide an approach of accelerating the speed by reducing the accuracy. This project is aimed to find the possible ways of building up tree multipliers by approximate (4:2) compressors.

Seven different approximate (4:2) compressors have been designed in this project. Generally, these compressors can be classified into three levels according to their accuracy. As shown in Table 3.13 the faster compressors usually have small size and low accuracy since they have simplified logic.

Bubbles are considered to be a factor related to the accuracy. Multiplier topologies will seriously affect the error rate since different tree structure has different number of bubbles. When approximate compressors are applied to the tree multipliers, the accuracy of the multiplier does not obviously depend on the accuracy of compressors, but the structure.

Table 5.1 compares the performance of approximate multipliers with that of conventional Wallace tree. The approximate multipliers can be 32% faster while the accuracy decreases 12%. The merged compressor based tree is even faster. It is 46% faster than Wallace tree.

Tree	Speed (ns)	MOSFET quantity	Error rate (%)	Area (u$\boldsymbol{m^2}$)
Conventional Wallace tree	1792	39690	0	55566
Conventional (4:2) com based tree[1]	1233.5	28212	12	49787.136
Bottom-up (4:2) com based tree[1]	1210.8	28486	11	47268.144
Bottom-up Merged com based tree[2]	962.22	26806	12	47268.144

Table 5.1 Performance comparison of tree multipliers

*the compressor is COM42_H_1

*Merged compressors are COM42_H_1 and COM42_L_3

*The results are by hand calculation. The interconnect delays in structure are ignored.

In fact, more explorations can be made in approximate multiplier design. The error rate is related to the probability of correctness of compressors in different partial product reduction stage. The better we know the probability of correctness, the easier to build up higher-accuracy approximate multipliers.

Another approach of reducing the power of approximate multipliers is to use control logic circuits, in which input signals can pass over XOR gates in specified conditions. It is possible since most of the operations in benchmark are 0+0=0. The power consumption will drop when a number of XOR gates are ignored.

Bibliography

[1] Lu, S. L., “Speeding up processing with approximation circuits,” Computer 37(3), pp.67–73, 2004

[2] Prasad, K. and Parhi, K.K., “Low-power 4-2 and 5-2 compressors,” in Proc. of the 35th Asilomar Conf. on Signals, Systems and Computers, vol. 1, 2001, pp. 129–133

[3] Margala, M. and Durdle, N.G., “Low-power low-voltage 4-2 compressors for VLSI applications,” in Proc. IEEE Alessandro Volta Memorial Workshop Low-Power Design, 1999, pp. 84–90

[4] Chang, C. H., Gu, J.M., and Zhang, M.Y., “Ultra low-voltage low-power CMOS 4-2 and 5-2 compressors for fast arithmetic circuits,” in IEEE Transactions on Circuits and Systems-I Regular Papers., VOL. 51, NO. 10, October 2004

[5] SanGregory, S. L., Brothers, C., Gallagher, D. and Siferd, R., “A fast, low-power logarithm approximation with CMOS VLSI implementation,” Circuits and Systems, , 388 - 391 vol. 1, 1999

[6] Phillips, B.J., Kelly D.R., and Ng, B.W., “Estimating adders for a low density parity check decoder”, In Franklin T. Luk, editor, Advanced Signal Processing Algorithms, Architectures, and Implementations XVI, volume 6313 of Proc. SPIE, San Diego, Ca, USA, pages 631302-1-9, September 2006. SPIE

[7] Koren, I., Koren, Y., and Oomman, B.G., “Saturating counters: application and design alternatives”, Computer Arithmetic, 2003. Proceedings. 16th IEEE Symposium on 228 – 235 June 2003

[8] Oklobdzija, V., "High-Speed VLSI Arithmetic Units: Adders and Multipliers", in "Design of High-Performance Microprocessor Circuits", Book Chapter, Book edited by A. Chandrakasan, IEEE Press, 2000

[9] Stine, J. E. ,”Digital Computer Arithmetic Datapath Design Using Verilog HDL”, Kluwer Academic Publishers, Norwell, MA, 2004

[10] Flynn, M.J. and Oberman, S.S., “Advanced Computer Arithmetic Design”, Wiley, 2001, pp43-99

[11] Colo, G., Bone, N., and Winterrowd, P., “Multiplier Evolution: A Family of Multiplier VLSI Implementations”, The Computer Journal, 2008

[12] Bickerstaff, K., Schulte, M. J., and Swartzlander, E. E., “Analysis of column compression multipliers”, In Proceedings of the 15th Symposium on Computer Arithmetic, pages 33–39, 2001

[13] Dandapat, A., Ghosal, S., Sarkar, P., and Mukhopadhyay, D., “A 1.2-ns16×16-Bit Binary Multiplier Using High Speed Compressors”, International Journal of Electrical, Computer, and Systems Engineering 4:3 2010

[14] Kwon O., Nowka. K, and Swartzlander, E. E., “ A 16 bit by 16 bit MAC design suing fast 5:3 compressor cells”, Journal of VLSI signal processing 31,77-89, 2002

[15] Wallace, C., “A suggestion for a fast multiplier”, IEEE Trans, Electronic Computers, Feb, 1964, pp. 14-17

[16] Weste N.and Harris D., “CMOS VLSI Design: A Circuits and Systems Perspective", Fourth Edition 2010, Addison Wesley

[17] Hennessy, J. L. and Patterson, D. A., “Computer Architecture: A Quantitative

Approach", 3rd Edition, Morgan Kaufmann Publishers, pp. H1-H69, 2002

[18] Ciletti, Michael D., "Advanced Digital Design with the Verilog HDL", Pearson Education, pp663, 2008

[19] Eriksson, H, "Efficient Implementation and Analysis of CMOS Arithmetic Circuits", the University of California at Berkeley, pp.35-61, 2003

[20] Bickerstaff, K.C, "Optimization of Column Compression Multipliers", University of Texas at Austin, pp.4-27, 2007

[21] Sutherland, I.E., Sproull, B.F. and Harris, D. L., "Logical Effort: Designing Fast CMOS Circuits", Morgan Kaufmann Publishers, pp1-93, 1998

[22] Zimmermann, R., Fichtner, W., "Low-power logic styles: CMOS versus pass-transistor logic", IEEE Journal of Solid-State Circuits, Vol. 32, No.7, pp.1079-1090, 1997

[23] A. P. Chandrakasan and R. W. Brodersen, "Low Power Digital CMOS Design", Norwell, MA: Kluwer, 1995.

[24] Radhakrishnan, D., Preethy, A.P., "Low Power CMOS Pass Logic 4-2 Compressor for High-Speed Multiplication", Proc.43 rd IEEE Midwest Symp.on Circuits and Systems, Lansing MI, Aug8-11, 2000

[25] Bergeron, J., "Writing testbenches: verification of HDL models", Kluwer Academic Publishers, pp.1-81, 2000

[26] Kelly, D.R., Phillips, B.J. and Al-Sarawi, S., "Approximate signed binary integer multipliers for arithmetic data value speculation", University of Adelaide, 2010

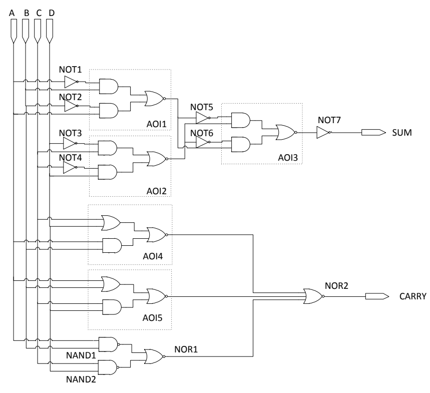
Appendix A: Cell Library

All values are for single-stick cells.

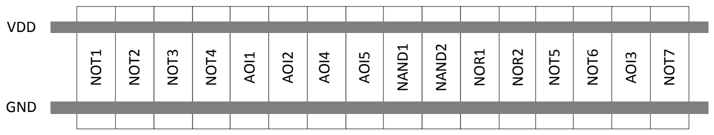
Cell	Equation	Width
NOT	z = !a	2
NAND2	z = !(a&b)	3
NAND3	z = !(a&b&c)	4
NOR2	z = !(a|b)	3
NOR3	z = !(a|b|c)	4
AOI21	z = !(a&b|c)	4
OAI21	z = !((a|b)&c)	4
AOI22	z = !(a&b|c&d)	5
OAI22	z = !((a|b)&(c|d))	5
AOI31	z = !(a&b&c|d)	5
OAI31	z = !((a|b|c)&d)	5
OAI211	z = !((a|b)&c&d)	5
AOI211	z = !(a&b|c|d)	5
MIN	z = !(a&b | c&(a|b))	6
XOR2	z = a⊕b	7
XNOR2	z = !(a⊕b)	7
MUX0n	z = sel&!a + !sel&b	7
MUX1n	z = sel&a + !sel&!b	7
D-type ff	…	12*

Appendix B: Design for High-accuracy approximate (4:2) compressor

Compressor (4:2) _H_2



CD \ AB	00	01	11	10
00	0	1	2	1
01	1	2	3	2
11	2	3	0	3
10	1	2	3	2



Logical Effort calculation

cell	Fanout load	Effort load	Parasitic delay
NOT1	AOI1+track	6/3+6/20=2.3	3/3=1
NOT2	AOI1+track	6/3+4/20=2.2	3/3=1
NOT3	AOI2+track	6/3+6/20=2.3	3/3=1
NOT4	AOI2+track	6/3+4/20=2.2	3/3=1
AOI1	AOI3+NOT5+track	(6+3)/3+25/20=4.25	12/3=4
AOI2	AOI3+NOT6+track	(6+3)/3+21/20=4.05	12/3=4
AOI4	NOR2+track	7/3+13/20=2.98	22/3
AOI5	NOR2+track	7/3+9/20=2.78	22/3
NAND1	NOR1+track	5/3+3/20=1.82	6/3=2
NAND2	NOR1+track	5/3+0/20=1.67	6/3=2
NOR1	NOR2+track	8/3+0/20=2.67	9/3=3
NOR2	NOT2+AOI1+AOI4+AOI5+NAND1+track	(3+6+8+7+3)/3+30/20=10.5	19/3
NOT5	AOI3+track	6/3+2/20=2.1	3/3=1
NOT6	AOI3+track	6/3+0/20=2	3/3=1
AOI3	NOT7+track	3/3+0/20=1	12/3=4
NOT7	NOT1+AOI1+AOI4+AOI5+NAND1+track	(3+6+8+7+3)/3+20/20=10	3/3=1

1 assume cout is connected to input b　　*2 assume sum is connected to input a*

Delay calculation in the critical path:

$F=\prod g\cdot b=2.3\times4.25\times2.1\times1\times10=205.28$

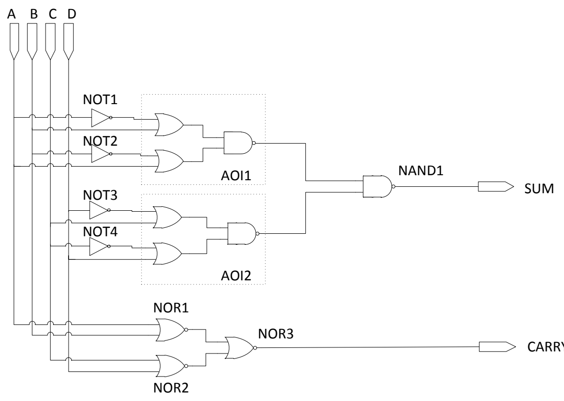
$N=\log_4 205.28=3.84\approx3$

$\alpha=205.28^{\frac{1}{5}}=2.90$

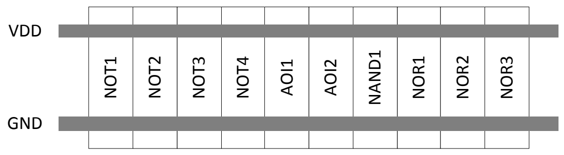
$D=N\cdot\alpha+\sum p=5\times2.90+1+4+1+4+1=25.5$delay units $\approx$ 5.10 FO4 delays=255ps

Appendix C: Design for Medium-accuracy approximate (4:2) compressor

Compressor (4:2)_M_2



CD \ AB	00	01	11	10
00	1	1	1	1
01	1	2	3	2
11	1	3	3	3
10	1	2	3	2



Logical Effort calculation

cell	Fanout load	Effort load	Parasitic delay
NOT1	AOI1+track	6/3+6/20=2.3	3/3=1
NOT2	AOI1+track	6/3+4/20=2.2	3/3=1
NOT3	AOI2+track	6/3+6/20=2.3	3/3=1
NOT4	AOI2+track	6/3+4/20=2.2	3/3=1
AOI1	NAND1+track	4/3+4/20=1.53	12/3=4
AOI2	NAND1+track	4/3+0/20=1.33	12/3=4
NAND1	NOT1+AOI1+NOR1+NOR2+track	(3+6+5+5)/3+31/20=7.88	6/3=2
NOR1	NOR3+track	5/3+3/20=1.82	9/3=3
NOR2	NOR3+track	5/3+0/20=1.67	9/3=3
NOR3	NOT2+AOI1+NOR1+NOR2+track	(3+6+5+5)/3+22/20=7.43	9/3=3

1 assume cout is connected to input b   *2 assume sum is connected to input a*

Delay calculation in the critical path:

$F=\prod g \cdot b=2.3*1.53*7.88=27.73$

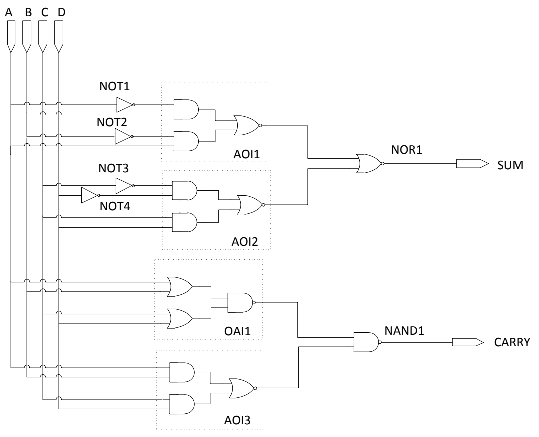
$N=\log_4 27.73=2.4\approx 2$

$\alpha=27.73^{\frac{1}{3}}=3.0$

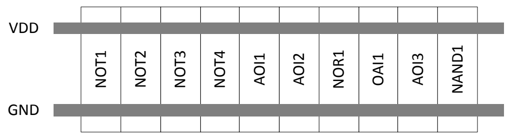
$D=N\cdot\alpha+\sum p= 3\times 3.0+1+4+2=16$ delay units $\approx$ 3.2 FO4 delays =160ps

Appendix D: Design for Low-accuracy approximate (4:2) compressor

Compressor (4:2)_L_2



CD \ AB	00	01	11	10
00	0	1	2	1
01	0	2	2	2
11	2	3	2	3
10	0	2	2	2



Logical Effort calculation

cell	Fanout load	Effort load	Parasitic delay
NOT1	AOI1+track	6/3+6/20=2.3	3/3=1
NOT2	AOI1+track	6/3+4/20=2.2	3/3=1
NOT3	AOI2+track	6/3+6/20=2.3	3/3=1
NOT4	AOI2+track	6/3+4/20=2.2	3/3=1
AOI1	NOR1+track	5/3+4/20=1.87	12/3=4
AOI2	NOR1+track	5/3+0/20=1.67	12/3=4
NOR1	NOT1+AOI1+OAI1+AOI3+track	(3+6+6+6)/3+34/20=8.7	9/3=3
OAI1	NAND1+track	4/3+4/20=1.53	12/3=4
AOI3	NAND1+track	4/3+0/20=1.33	12/3=4
NAND1	NOT2+AOI1+OAI1+AOI3+track	(3+6+6+6)/3+23/20=8.15	6/3=2

1 assume cout is connected to input b   *2 assume sum is connected to input a*

Delay calculation in the critical path:

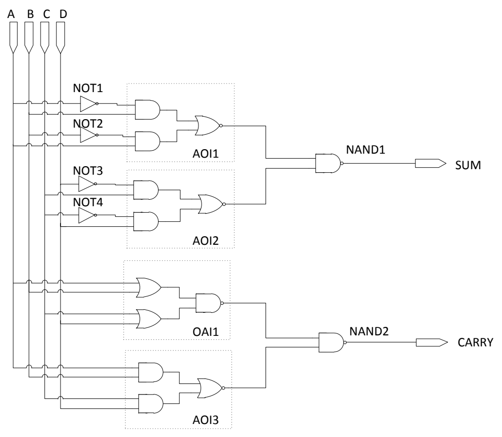
$F=\prod g\cdot b=2.3*1.87*8.7=37.42$

$N=\log_4 37.42=2.6\approx 2$

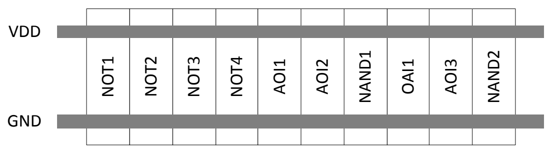
$\alpha=37.42^{\frac{1}{3}}=3.3$

$D=N\cdot\alpha+\sum p=3\times 3.3+1+4+3=17.9$ delay units $\approx$ 3.58 FO4 delays = 179 ps

Compressor (4:2)_L_3



CD \ AB	00	01	11	10
00	0	1	2	1
01	1	3	3	3
11	2	3	2	3
10	1	3	3	3



Logical Effort calculation

cell	Fanout load	Effort load	Parasitic delay
NOT1	AOI1+track	6/3+6/20=2.3	3/3=1
NOT2	AOI1+track	6/3+4/20=2.2	3/3=1
NOT3	AOI2+track	6/3+6/20=2.3	3/3=1
NOT4	AOI2+track	6/3+4/20=2.2	3/3=1
AOI1	NAND1+track	4/3+4/20=1.53	12/3=4
AOI2	NAND1+track	4/3+0/20=1.33	12/3=4
NAND1	NOT1+AOI1+OAI1+AOI3+track	(3+6+6+6)/3+34/20=8.7	6/3=2
OAI1	NAND2+track	4/3+4/20=1.53	12/3=4
AOI3	NAND2+track	4/3+0/20=1.33	12/3=4
NAND2	NOT2+AOI1+OAI1+AOI3+track	(3+6+6+6)/3+23/20=8.15	6/3=2

1 assume cout is connected to input b　　*2 assume sum is connected to input a*

Delay calculation in the critical path:

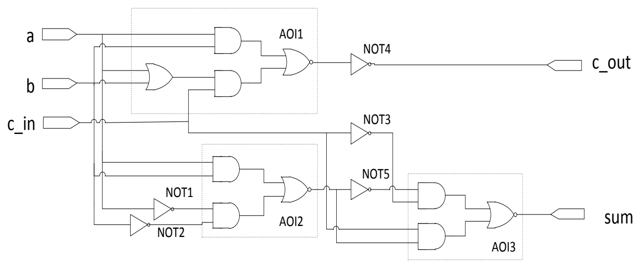
$F=\prod g\cdot b=2.3*1.53*8.7=30.61$

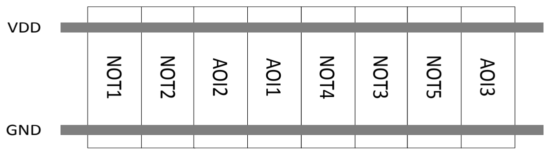
$N=\log_4 30.61=2.47\approx 2$

$\alpha=30.61^{\frac{1}{3}}=3.1$

$D=N\cdot\alpha+\sum p=3\times 3.1+1+4+2=16.4$ delay units $\approx$ 3.28 FO4 delays=163.84ps

Appendix E: Design for accurate (3:2) counter





Logical Effort calculation

CMOS cell	Fanout load	Effort load	Parasitic delay
AOI1	NOT4+track	3/3+0/20=1	12/3=4
AOI2	NOT5+AOI3+track	(3+6)/3+10/20=4	12/3=4
AOI3	NOT1+AOI1+AOI2+track[1]	(3+6+6)/3+8/20=5.4	12/3=4
NOT1	AOI2+track	6/3+2/20=2.1	3/3=1
NOT2	AOI2+track	6/3+0/20=2	3/3=1
NOT3	AOI3+track	6/3+2/20=2.1	3/3=1
NOT4	AOI1+NOT3+AOI3+track[2]	(6+3+6)/3+16/20=5.8	3/3=1
NOT5	AOI3+track	6/3+0/20=2	3/3=1

1 assume c_out is connected to input c　　*2 assume sum is connected to input a*

Delay calculation in the critical path:

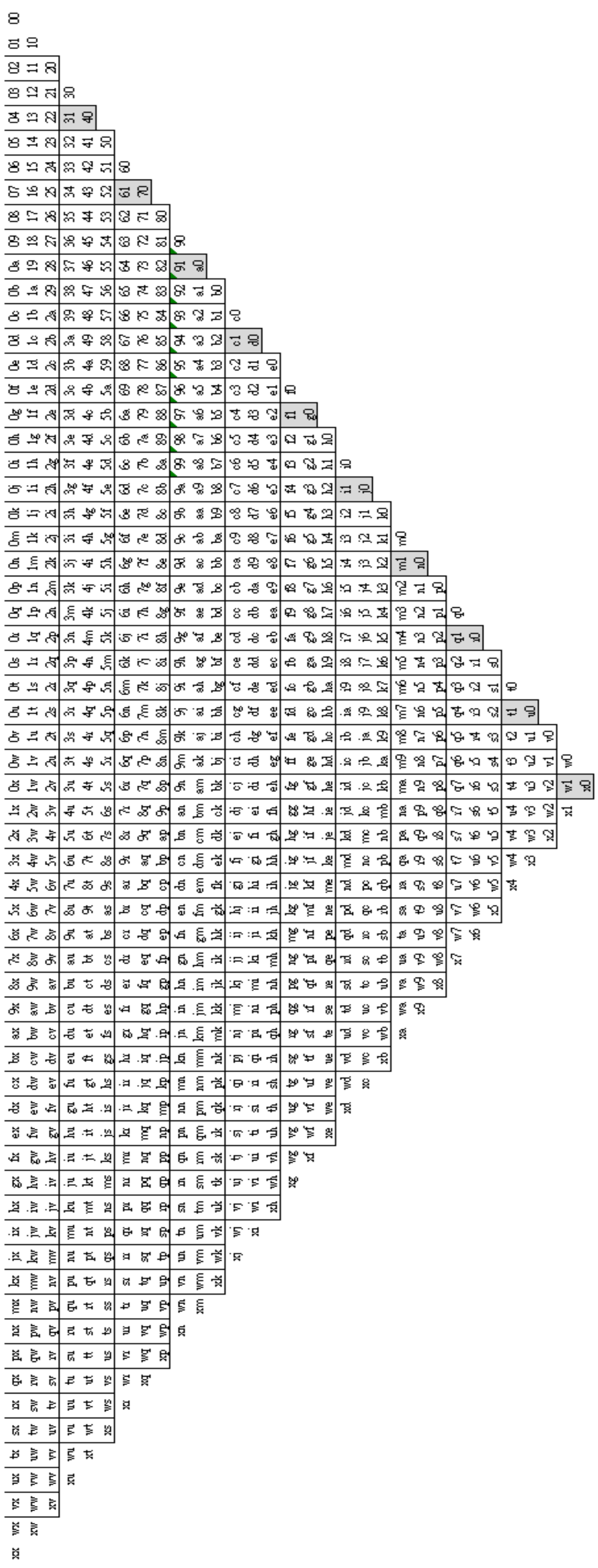
$F=\prod g \cdot b=2.1\times4\times2\times5.4=90.7$

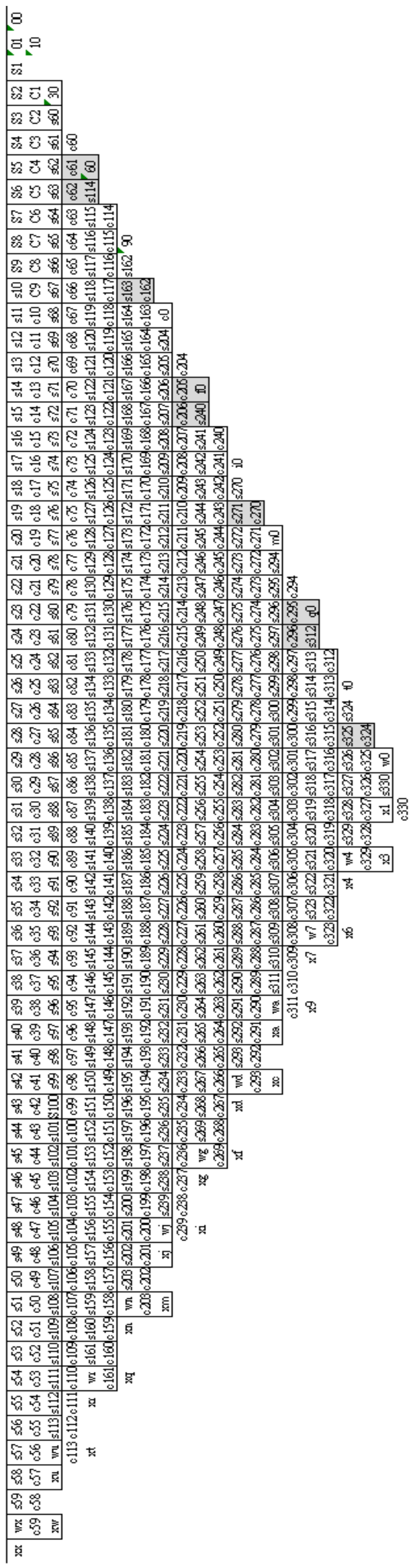
$N=\log_4 90.7=3.3$

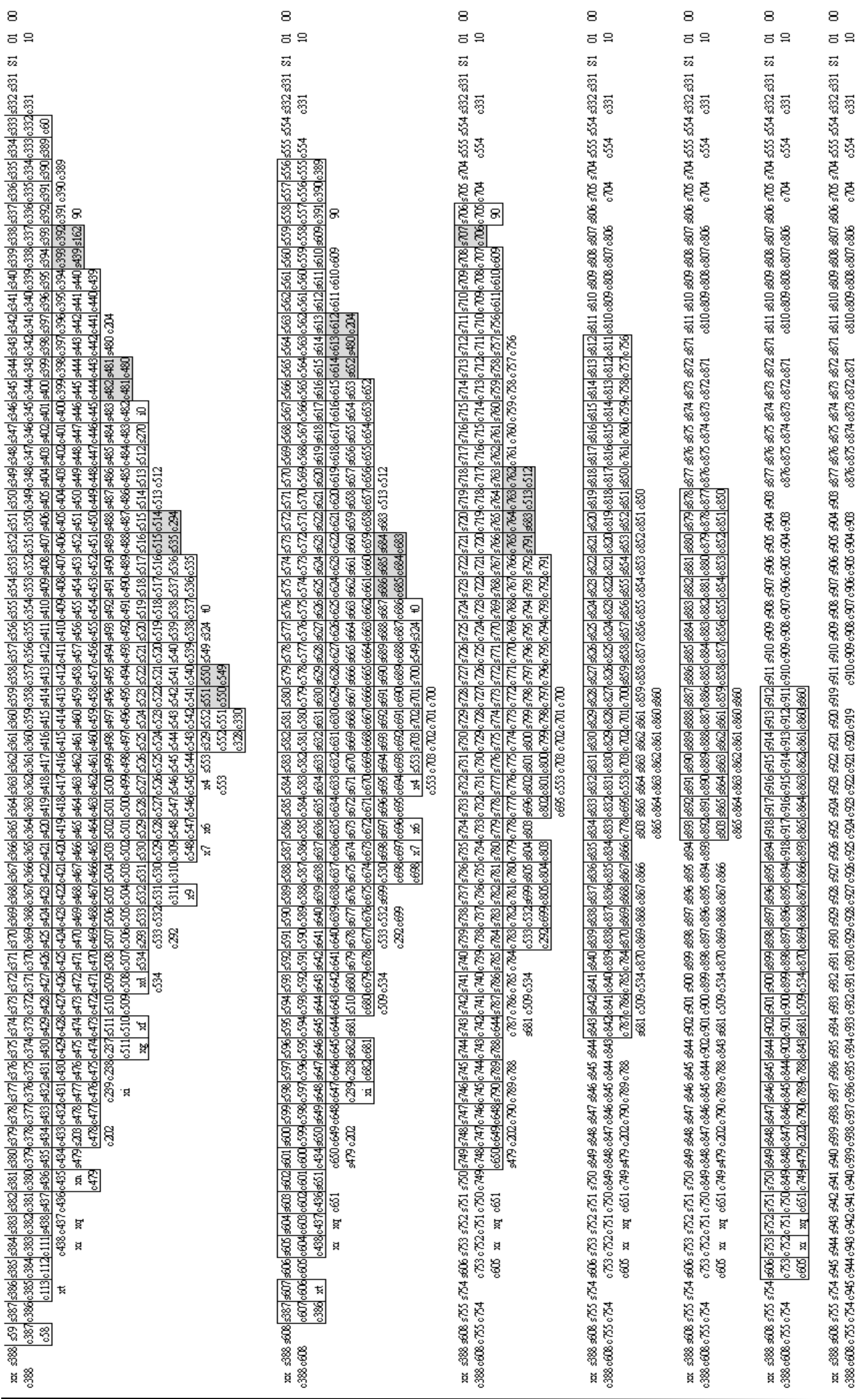
$\alpha=90.7^{\frac{1}{4}}=3.1$

$D=N\cdot\alpha+\sum p=4\times3.1+1+4+1+4=22.4$ delay units $\approx$ 4.48 FO4 delays=224 ps

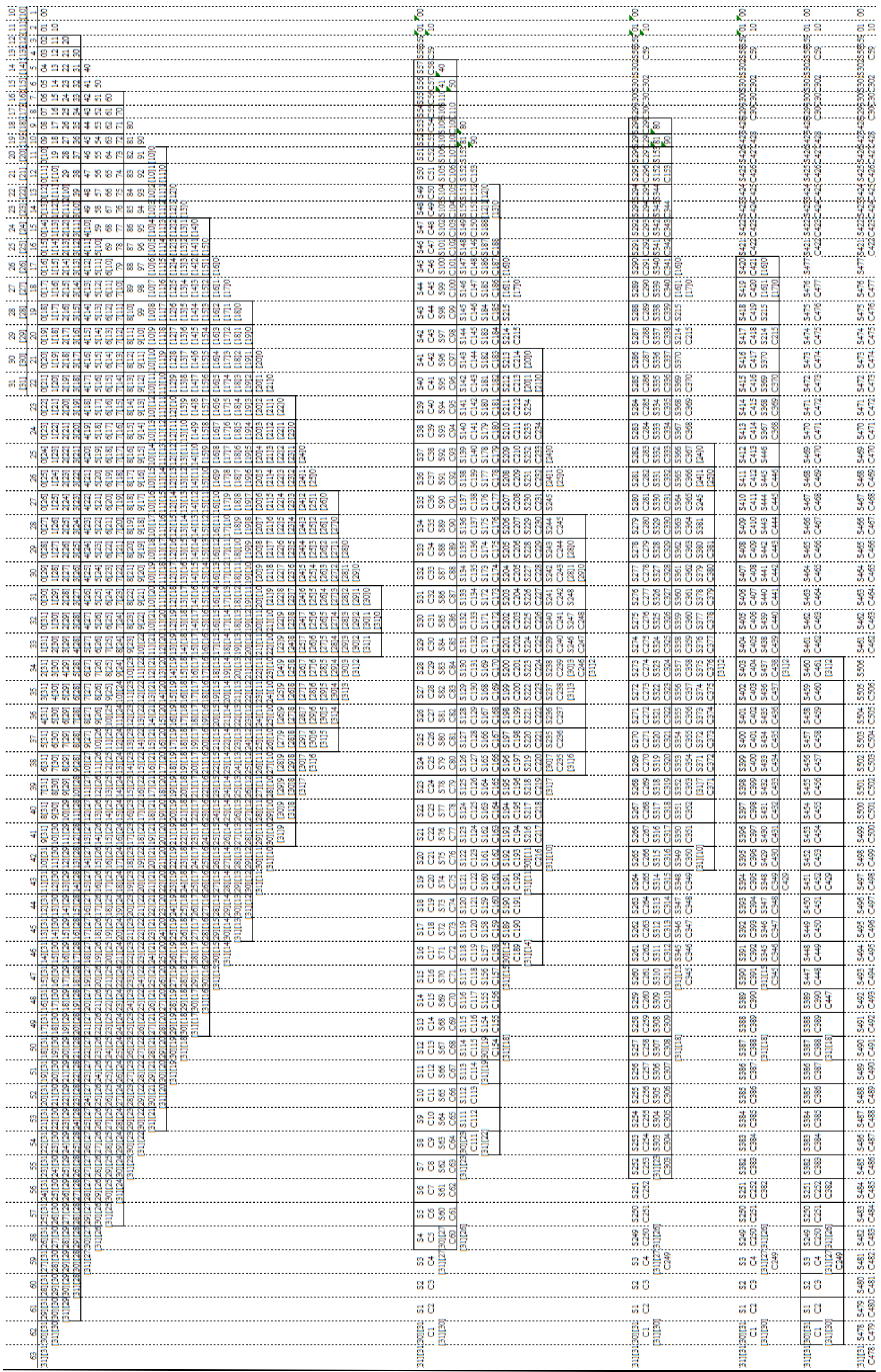
Appendix F: 32×32 bit Wallace tree







Appendix G: 32×32 bit Convectional (4:2) compressor based tree



Appendix H: 32×32 bit Bottom-up (4:2) compressor based tree