Design and Implementation of a Module Generator for Low Power Multipliers

Kaihong Sun

Reg nr: LiTH-ISY-EX-3315-2003

2003-09-25

Design and Implementation of a Module Generator for Low Power Multipliers

Master's Thesis

Division of Electronics Systems
Department of Electrical Engineering
Linköping Institute of Technology
Linköping University, Sweden

By
Kaihong Sun

Reg nr: LiTH-ISY-EX-3315-2003

Supervisor: Weidong Li

Examiner: Prof. Mark Vesterbacka

Linköping, September 25, 2003

LINKÖPINGS UNIVERSITET	**Avdelning, Institution** Division, Department Division of Electronics Systems, Department of Electrical Engineering, 581 83 LINKÖPING
Datum Date	2003-09-25

Språk
Language
Svenska/Swedish
X Engelska/English

Rapporttyp
Report category
Licentiatavhandling
X Examensarbete
C-uppsats
D-uppsats

Övrig rapport

ISBN

ISRN LiTH-ISY-EX-3315-2003

Serietitel och serienummer **ISSN**
Title of series, numbering

URL för elektronisk version
http://www.ep.liu.se/exjobb/isy/2003/3315/

Titel
Title Design and Implementation of a Module Generator for Low Power Multipliers

Författare
Author Kaihong Sun

Sammanfattning
Abstract

Multiplication is an important part of real-time system applications. Various hardware parallel multipliers used in such applications have been proposed. However, when the operand sizes of the multipliers and the process technology need to be changed, the existing multipliers have to be redesigned.

From the point of library cell reuse, this master thesis work aims at developing a module generator for parallel multipliers with the help of software programs. This generator can be used to create the gate-level schematic for fixed point two's complement number multipliers. Based on the generated schematic, the entire multiplier can be implemented by small manual intervention. This feature can reduce the time of chip design.

The design phases consist of the logic, circuit and physical designs. The logic design includes gate-level schematic generation with C and SKILL programs and structural VHDL-code descriptions as well as validation. The circuit and physical design are custom in Cadence and the routing uses automatic place and route tools.

To demonstrate the design method, an 18 by 18-bit modified Booth recoded multiplier was implemented in 0.18 μm CMOS process with a supply voltage of 1.2 V and simulated using simulator (Spectre). The number of integrated transistors is 13000 and the active area is 85000 μm^2. The postlayout simulation shows the critical path with a delay of 17 ns.

Nyckelord
Keyword
Modified Booth Encoding, Low Power, Multipliers, Module Generator.

ABSTRACT

Multiplication is an important part of real-time system applications. Various hardware parallel multipliers used in such applications have been proposed. However, when the operand sizes of the multipliers and the process technology need to be changed, the existing multipliers have to be redesigned.

From the point of library cell reuse, this master thesis work aims at developing a module generator for parallel multipliers with the help of software programs. This generator can be used to create the gate-level schematic for fixed point two’s complement number multipliers. Based on the generated schematic, the entire multiplier can be implemented by minimizing manual intervention. This feature can reduce the time of chip design.

The design phases consist of the logic, circuit and physical designs. The logic design includes gate-level schematic generation with C and SKILL programs and structural VHDL-code descriptions as well as validation. The circuit and physical design are custom in Cadence and the routing uses automatic place and route tools.

To demonstrate the design method, an 18 by 18-bit modified Booth recoded multiplier was implemented in 0.18 µm CMOS process with a supply voltage of 1.2 V and simulated using simulator (Spectre). The number of integrated transistors is 13000 and the active area is 85000 µm^2. The post-layout simulation shows the critical path with a delay of 17 ns.

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor Weidong Li and examiner Professor Mark Vesterbacka for giving me the opportunity of this interesting work, especially Weidong Li for his support and guidance during this master thesis work. I also would like to acknowledge Emil Hjalmarson for his help and providing the useful material of Skill programmable language. Thanks also to the staff, Electronics systems at Linkoping University, for their support in many aspects.

I would sincerely grateful to my classmates, friends and many others for their support and invaluable help, not only during this master thesis work, but also during the past two years.

Finally, I would like to express my extreme gratitude to my wife Fengling and my son Lei for their encouragement and endless support in every aspect of my life.

Table of Contents

List of Figures

List of Tables

Abbreviations

BE	**B**ooth **E**ncoding
CLA	**C**arry **L**ook **A**head
CPL	**C**omplementary **P**ass-transistor **L**ogic
LSB	**L**east **S**ignificant **B**it
MBA	**M**odified **B**ooth **A**lgorithm
MBE	**M**odified **B**ooth **E**ncoding
Mcand	**M**ultiplicand
MSB	**M**ost **S**ignificant **B**it
PP	**P**artial **P**roduct
PPG	**P**artial **P**roduct **G**enerator
PPCT	**P**artial **P**roduct **C**ompression **T**ree
PPRT	**P**artial **P**roduct **R**eduction **T**ree
PPSB	**P**artial **P**roduct **S**ign **B**it
Prod	**P**roduct
RCA	**R**ipple **C**arry **A**dder
TG	**T**ransmission **G**ate
VMA	**V**ector **M**erging **A**dder

Chapter 1 Introduction

Low power high performance multipliers have become a basic building block in computations especially in digital signal processing. For most of the applications, multiplication operations take a significant part of time delay, area cost, and power consumption. Therefore, many techniques and design methodologies have been proposed to improve the speed and power dissipation of the multipliers. Most of the designs are targeted at a specific technology and require redesign for a new process technology. To speed up the chip design, a module generator for implementation of parallel multipliers with different sizes is presented in this thesis.

1.1 Purpose

The aim with this thesis work is to develop a module generator for fixed-point parallel multipliers. The delay, area and power have also been taken into considerations. The multiplier should be able to multiply two n-bit two's complement numbers and produce a 2n-bit product. Using such a method, the basic library cells can be reused, which results in a less time of the chip designs. To demonstrate the design method, an 18 by 18 bit parallel multipliers is designed.

1.2 Design Specifications

The design specifications for the parallel multiplier include the general requirements for designing the parallel multipliers and special requirements for implementing the 18 by 18 bit multiplier. Both of them are described as follows.

General Requirements

Multiplicand:	n-bit two's complement number.
Multiplier:	n-bit two's complement number.
Product:	2n-bit two's complement number.
Supply voltage:	1.2 V.
Rise/fall time:	500 ps.
Target performance:	Minimum area and power consumption under the required delay.

Special Requirements

Multiplicand:	18-bit two's complement number.
Multiplier:	18-bit two's complement number.
Product:	36-bit two's complement number.
Supply voltage:	1.2 V.
Rise/fall time:	500 ps.
Target performance:	Minimum area and power consumption under the operating frequency of 25.6 MHz.

In addition, the design and implementation should also satisfy the following further requirements.

1. The logic design and functional validation shall be performed in UNIX C and the Modelsim from Mentor Graphics for VHDL simulation.
2. The gate-level schematic shall be generated according to the required word-length.
3. All the transistor sizes shall be parameteriable.
4. The library for all the transistors in schematic view shall be analogLib and the DK_hcmos8d for the layout.
5. The design and implementation shall be carried out in 0.18 µm CMOS process technology.

1.3 Reading guidelines

This thesis consists of six chapters. The rest of the chapters are organized as follows.

Chapter 2 gives an overview of the theoretical algorithms on parallel multipliers, such as encoding and sign extension schemes.

Chapter 3 briefly presents the power reduction techniques that are related to the design and implementation of parallel multipliers.

Chapter 4 contains the description of the overall architecture as well as the major functional units of the parallel multiplier. In addition, the partial product reduction tree topologies are also described in this section.

Chapter 5 focuses on the design of the module generator and the implementation of the 18 by 18 bit MBE multiplier. Three design phases, that is, logic, circuit and physical designs, have been represented in details.

Chapter 6 summarizes the results and comes to the conclusions from the master thesis work. Moreover, some suggestions on the future possible improvements are discussed in this chapter.

This page is left blank on purpose.

Chapter 2 Encoding Schemes

This chapter briefly describes the methods for generating partial products. The major encoding schemes used for multipliers will be introduced, and their advantages and disadvantages will also be discussed. In order to introduce the concept of the encoding for the multiplication operation, let us start with an overview of the multiplication process.

2.1 Multiplication Process

The simplest multiplication operation is to directly calculate the product of two numbers by hand. This procedure can be divided into three steps: partial product generation, partial product reduction and the final addition.

To further specify the operation process, let us calculate the product of two two's complement numbers, for example, $1101_{two}(-3_{ten})$ and $0101_{two}(5_{ten})$, when computing the product by hand, which can be described according to figure 2.1.

$$\begin{array}{rrl}
 & 1\,1\,0\,1 & \text{Multiplicand} \\
\times & 0\,1\,0\,1 & \text{Multiplier} \\
\hline
 & \boldsymbol{1\,1\,1\,1}\,1\,1\,0\,1 & \text{PP1} \\
 & \boldsymbol{0\,0\,0}\,0\,0\,0\,0 & \text{PP2} \\
 & \boldsymbol{1\,1}\,1\,1\,0\,1 & \text{PP3} \\
+ & \boldsymbol{0}\,0\,0\,0\,0 & \text{PP4} \\
\hline
 & \boxed{1}\,1\,1\,1\,1\,0\,0\,0\,1 = -15 & \text{Product}
\end{array}$$

discard this bit

Figure 2.1 Multiplication calculation by hand

The bold italic digits are the sign extension bits of the partial products. The first operand is called the multiplicand and the second the multiplier. The intermediate products are called partial products and the final result is called the product. However, the multiplication process, when this method is directly mapped to hardware, is shown in figure 2.2.

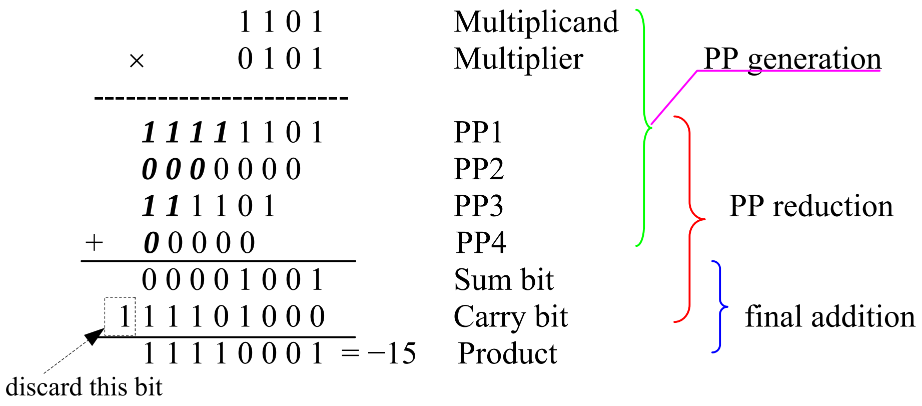


Figure 2.2 Multiplication operation in hardware

As can been seen in the figures, the multiplication operation in hardware consists of PP generation, PP reduction and final addition steps. The two rows before the product are called sum and carry bits. The operation of this method is to take one of the multiplier bits at a time from right to left, multiplying the multiplicand by the single bit of the multiplier and shifting the intermediate product one position to the left of the earlier intermediate products. All the bits of the partial products in each column are added to obtain two bits: sum and carry. Finally, the sum and carry bits in each column have to be summed.

Similarly, for the multiplication of an *n*-bit multiplicand and an *m*-bit multiplier, a product with $n + m$ bits long and *m* partial products can be generated.

The method shown in figure 2.2 is also called a non-Booth encoding scheme. Its advantages and drawbacks will be discussed in next section.

2.2 Non-Booth encoding

Using the non-Booth encoding method for partial product generation, the multiplier bits are examined sequentially starting from LSB to MSB. If the multiplier bit is one, the partial product is simply the multiplicand. Otherwise, the partial product is zero. Each new partial product is shifted one bit position to the left. Each partial product can be produced by just using a row of two-input AND gates. The number of partial products generated equals the size of the multiplier bits.

The advantage of this method is that the partial product circuit is simple and easy to implement. Therefore, this scheme is suitable for the implementation of small multipliers.

The drawback is that the method is not able to efficiently handle the sign extension and it generates a number of partial products as many as the number of bits of the multiplier, which results in many adders needed so that the area and power consumption increase. This method is not applicable for large multipliers.

2.3 Booth Encoding

The Booth encoding, or Booth algorithm, was proposed by Andrew D. Booth in 1951 [1]. This method can be used to multiply two two's complement number without the sign bit extension.

The operation of Booth encoding consists of two major steps [2]: the first one is to take one bit of the multiplier, and then to decide whether to add the multiplicand according to the current and previous bits of the multiplier. This encoding scheme is serial, which means that the different value of the 2 bits (current and previous bits) corresponds to the different operations. The serial encoding scheme is usually applied in serial multipliers. The operation procedure can be described with the following table.

00:	no arithmetic operation.
01:	adding the multiplicand to the left half of the product.
10:	subtracting the multiplicand from the left half of the product.
11:	no arithmetic operation.

The second step is to shift the product right one bit.

For example, let us consider the multiplication of two two's complement number $0110_{two}(6_{ten})$ and $1011_{two}(-5_{ten})$ = $11100010_{two}(-30_{ten})$. The operation is illustrated in Figure 2.3.

Itera-Tion	Multi-plicand	Step	Product
0	0110	Initial values	0000 1011 0
1	0110	10 => Prod = Prod − Mcand	1010 1011 0
	0110	Shift right product	1101 0101 1
2	0110	11 => no operation	1101 0101 1
	0110	Shift right product	1110 1010 1
3	0110	01 => Prod = Prod + Mcand	0100 1010 1
	0110	Shift right product	0010 0101 0
4	0110	10 => Prod = Prod − Mcand	1100 0101 0
	0110	Shift right product	***1110 0010*** 1

Note: The circled bits are used to determine the operation for the next step.

Figure 2.3 Booth encoding with negative multiplier

2.4 Modified Booth Encoding

The modified Booth encoding (MBE), or modified Booth's algorithm (MBA), was proposed by O. L. Macsorley in 1961 [3]. The encoding method is widely used to generate the partial products for implementation of large parallel multipliers, which adopts the parallel encoding scheme. The basic principle for the modified Booth encoding can be described as follows.

Let us consider the multiplication of two fixed-point two's complement numbers, *X* and *Y*, where *X* is the multiplier and *Y* is the multiplicand, both of them have *n* bits, and the *X* can be expressed by

$$\begin{aligned} X &= -X_{n-1}2^{n-1} + \sum_{i=0}^{i=n-2} X_i 2^i , \\ &= \sum_{i=0}^{i=n/2-1} (-2X_{2i+1} + X_{2i} + X_{2i-1}) \cdot 2^{2i} , \end{aligned}$$

$$= \sum_{i=0}^{i=n/2-1} d_i \cdot 2^{2i} ,$$
$$= \sum_{i=0}^{i=n/2-1} d_i \cdot 4^i \qquad (2\text{-}1)$$

Using this notation, the multiplication of *X* and *Y* is given by

$$YX = \sum_{i=0}^{(n/2)-1} d_i \cdot 4^i \cdot Y ,$$
$$= \sum_{i=0}^{(n/2)-1} P_i \cdot 4^i \qquad (2\text{-}2)$$

In this way, the bits of the multiplier are partitioned into sub-strings by the 3 adjacent bits and each sub-string group ($X_{2i+1}, X_{2i}, X_{2i-1}$) corresponds to one of the value in the set {−2, −1, 0, +1, +2}[30]. This means that the each three adjacent bits of the multiplier can generate a single encoding digit, which is called the modified Booth recoding digit (d_i) [5], as shown in table 2.1. Each MBE blocks can work in parallel, therefore, all the partial product bits are generated simultaneously. The parallel encoding scheme is suitable for parallel multipliers.

Table 2.1 Modified Booth encoder truth table

X_{2i+1}	X_{2i}	X_{2i-1}	d_i
0	0	0	0
0	0	1	+1
0	1	0	+1
0	1	1	+2
1	0	0	−2
1	0	1	−1
1	1	0	−1
1	1	1	0

The number of bits for the multiplier, *X*, must be even. Otherwise, the sign bit of *X* should be extended. For the $n \times m$ multiplication, using the modified Booth encoding $m/2$ partial products are produced or $(m+1)/2$ partial products if *m* is odd. Obviously, from the equation (2-2), the partial product, P_{i+1}, should be shifted two positions to the left of the partial product, P_i, due to the P_i is multiplied by 4^i.

The operation for Y times X can be summarized in figure 2.4.

d_i	Operation on mcand (Y)
0	$0*Y$: 0 => Prod
+1	$+1*Y$: mcand => Prod
+2	$+2*Y$: one shift to the left for macnd = > Prod
−1	$-1*Y$: inverted mcand & added 1 to the LSB
−2	$-2*Y$: one shift to the left for macnd, then inverted mcand & added 1 to the LSB

Figure 2.4 Partial product selections by using MBE

This operation can also be illustrated graphically. For example, an 8×8 bit MBE multiplier with $X = 10011101_{two}(-99_{ten})$, $Y = 01101101_{two}(109_{ten})$, $n = 8$, is shown in figure 2.5. The binary numbers in parentheses are the generated sign bits of the partial products.

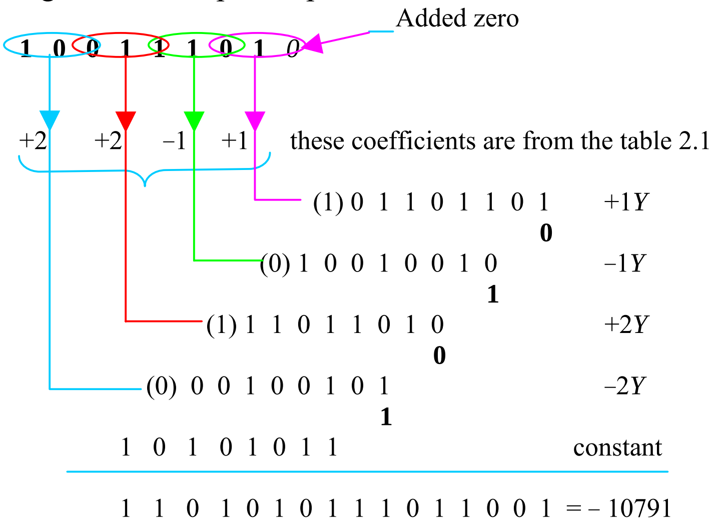


Figure 2.5 An example for an 8×8 bit MBE multiplier

The advantage of using MBE is that it can reduce the number of partial products by 50%, which results in about half of the adders reduced compared to the non-Booth encoding, and the consumed power is also decreased. This encoding method is applicable for parallel multipliers with the input operands of equal to or greater than 16-bit.

However, the modified Booth encoding is not suitable for implementing smaller multipliers due to the extra hardware overhead for MBE encoder and the complex circuit of the partial product generator.

2.5 Other Encoding

Besides the non-Booth and Modified Booth encoding, higher radix Booth encoding such as radix-8 can be also used to generate partial products. Radix-8 Booth encoding method is also called the Booth 3 scheme [32]. Using the Booth 3 encoding scheme, the multiplier is divided into overlapping groups of 4 bits in parallel. Each partial product can be selected from the set of the multiplicand Y $\{0, \pm Y, \pm 2Y, \pm 3Y, \pm 4Y\}$ [32].

The advantage of this encoding method is that it can further reduce the partial products to $(n + 1)/3$. But the drawback is obviously the complexity of the partial product selection logic and the Booth encoders as well as the generation of the $\pm 3Y$ multiple. In this thesis work, this method will not be discussed in detail.

Another encoding scheme for generation of partial products is to use smaller multipliers. For instance, an 8×8 bit multiplier can be constructed with four 4×4 bit multipliers and two adders [4], as shown in Figure 2.6.

The non-Booth encoding scheme can be used to partition and distribute the two 8-bit numbers to the four 4×4 multipliers. The four 4×4 smaller multipliers could be implemented by non-Booth encoding method, and their partial product generator is simply two-input AND gates. The four 8-bit products produced can be added by using two adders.

In general, this encoding is not efficient compared to other encoding schemes implemented in current process technology [32].

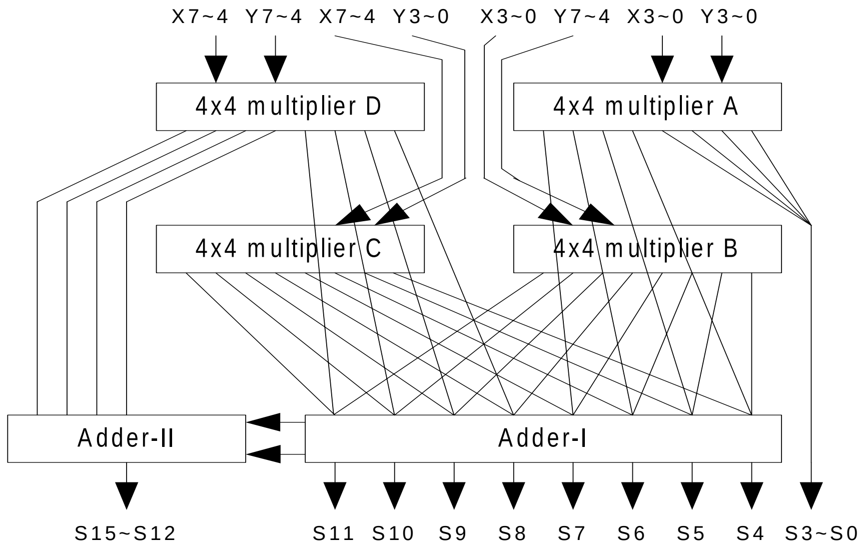


Figure 2.6 An 8×8 bit multiplier based on smaller multipliers

2.6 Sign Extension Schemes

The multiplication and addition operations for two's complement numbers have to handle the sign bits, as shown in figure 2.2. The addition of the extended sign bits for each partial product results in additional cost. To reduce the cost of the sign extension, several extension schemes have been proposed, as described in [28].

In the following section, the basic principle of sign extension and one method used for sign extension in this thesis will be introduced.

2.6.1 Basic Concept of Sign Extension

The two's complement is a special case of radix complement for binary numbers in which the radix equals to two. For instance, a $k+1$ bit number A can be represented in two's complement as

$$A = -a_k \cdot 2^k + \sum_{i=0}^{k-1} a_i \cdot 2^i \tag{2-3}$$

where the a^k is the sign bit. *A* is positive when a^k equals to zero, while *A* is negative when a^k is 1.

If the sign bit of a two's complement number *A* is extended by *S* bits, then *A* should include three parts [29], the original MSB, the extension of the sign bit by *S* bits and the number's value. In this case, the *A* is rewritten by

$$A = -a_k \cdot 2^{k+S} + \sum_{i=k}^{k+S-1} a_k \cdot 2^i + \sum_{i=0}^{k-1} a_i \cdot 2^i \qquad (2\text{-}4)$$

When defining A_{ext} as the sign bit plus the S extended bits, the A_{ext} can also be presented using two's complement format with a length of $S+1$ and bit significances from 2^k to 2^{k+S}. The A_{ext} can be expressed as

$$\begin{aligned} A_{ext} &= -a_k \cdot 2^{k+S} + \sum_{i=k}^{k+S-1} a_k \cdot 2^i \\ &= -a_k \cdot 2^{k+S} + a_k \cdot \sum_{i=k}^{k+S-1} 2^i \\ &= a_k(-2^{k+S} + 2^{k+S} - 2^k) \\ &= -a_k 2^k, \quad \text{with} \sum_{i=k}^{n-1} 2^i = 2^n - 2^k. \qquad (2\text{-}5) \end{aligned}$$

From the above derivation, it is clear that the sign for the number with sign extension is the same as the original one. Therefore, the positive two's complement numbers actually have an infinite number of 0s on the left, whereas the negative ones have an infinite number of 1s. In order to fit the width of the hardware, sign extension can be used to restore some of the hidden sign bits.

2.6.2 Conventional Sign Extension

Conventional sign extension is similar to the method used to calculate the multiplication by hand [28]. This method can be used to add the partial products sequentially. This means that the first row of partial products is summed to second row and the result is added to third row and so on. In this way, sign extension is only performed from one row to the next. Furthermore, the sign is encoded into the carry and sum of the MSB of the intermediate addition results. Therefore, the carry and sum of MSB should be extended to the next row.

This method is not efficient for low power design since the full adder on the most significant position in each row has one more fan-out than the rest of the adders. Another efficient method that is called sign generate is described in the following section.

2.6.3 Sign Generate Sign Extension

The sign generate scheme [5] is an efficient method to reduce the length of each partial product. This sign extension scheme assumes that all the partial products are negative. Based on such an assumption, for an *n* by *m* multiplier, the sum of all sign extensions can be precalculated as

$$\begin{aligned} Signs &= \sum_{i=0}^{(m/2)-1} ((-1)2^n)4^i , \\ &= 2^n(-1)(\frac{2^m-1}{3}) \ . \end{aligned} \qquad (2\text{-}6)$$

The equation (2-6) shows the relationship that can be interpreted as a fixed number, $(-1)\left[(2^m-1)/3\right]$, which should be added to the Nth binary position of the partial product leftwards. This number expressed in binary form is equal to 1010101…01011, where there are exactly $m/2-1$ zeros. If the partial product generated is positive, its sign bit should be simply replace by a one to suppress the effect of the previous assumption. This technique can be summarized as follows.

1. Inverting the sign bit of each partial product, and placing it into the *N*th binary position.
2. Adding one to the left of each partial product.
3. Adding one in *N*th bit column.

The operation of the one addition can be implemented by using increment adders. Therefore, no extra adders for adding these constant 1s are required using this method. The advantage of the sign generate method is that it does not only reduce the area, power consumption, but also speed up the multiplication. The following example illustrates an 8 by 8 multiplier using this method together with the modified Booth encoding [28]. In this case, the sign of the final result can be expressed as

$$S = S_0\sum_{i=8}^{15} 2^i + (S_1\sum_{i=8}^{13} 2^i)2^2 + (S_2\sum_{i=8}^{11} 2^i)2^4 + (S_3\sum_{i=8}^{9} 2^i)2^6 \qquad (2\text{-}7)$$

where S_i is the sign bit of the partial product in the i^{th} row. By using the following two equations

$$\sum_{i=j}^{n-1} 2^i = 2^n - 2^j \qquad (2\text{-}8)$$

$$S_i = 1 - \overline{S_i} \qquad (2\text{-}9)$$

then *S* becomes

$$S = \overline{S}_0 2^8 + \overline{S}_1 2^{10} + \overline{S}_2 2^{12} + \overline{S}_3 2^{14} + 2^9 + 2^{11} + 2^{13} + 2^{15} + 2^8 . \qquad (2\text{-}10)$$

Equation 2-10 indicates that the sign of the final result can be calculated directly according to the partial products. Figure 2-7 shows the partial product diagram with the sign generate method, in which *T* is the one's complement of the sign and *C* is the correction constant for the negative partial products. Another example is shown in figure 2.5.

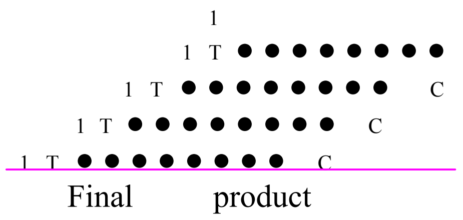


Figure 2.7 Partial product diagram with the sign generate scheme

2.7 Summary

The aim of this chapter was to give an overview of the methods for generating the partial products. It started with the introduction of the multiplication process. Several encoding schemes have been described and their advantages and drawbacks have also been discussed.

The Non-Booth encoding method generates the same number of partial products as the number of bits of the multiplier. It is suitable for implementing the smaller multipliers due to the simple realization of the partial product generator and no need to use an encoding circuit.

The original Booth encoding performs the encoding serially. The serial encoding scheme is usually employed in bit-serial multipliers.

The modified Booth encoding performs the encoding in parallel, which is widely used to generate the partial products of the large parallel multipliers. In general, this method is not applied to implement the multipliers with a word length less than 16 bits.

Higher radix Booth encoding also performs the encoding in parallel, which can further reduce the number of partial products, but it uses a more complex circuit for the Booth encoder.

A small multiplier can also be used to construct large multipliers. However, it is not an efficient method compared to other encoding schemes in current implementation technology.

Chapter 3 Power Reduction Techniques

Reducing power consumption has become an important issue in digital circuit design, especially for high performance portable devices. Many power reduction techniques have also been proposed from the system level down to the circuit level. In this section, some of these techniques, which are related to the design for parallel multiplier, will be presented.

3.1 Sources of power Dissipation

The sources of power dissipation in digital CMOS circuits are composed of the following parts: switching power, short-circuit power, and leakage power, which are expressed in the following equation

$$P_{total} = \overbrace{\underbrace{\alpha_{0-1} \cdot C_L \cdot V_{dd}^2 \cdot f_{clk}}_{\text{Switching power}} + \underbrace{I_{SC} \cdot V_{dd}}_{\text{Short-circuit power}}}^{\text{Dynamic Power}} + \overbrace{\underbrace{I_{leakage} \cdot V_{dd}}_{\text{Leakage Power}}}^{\text{Static Power}} \tag{3-1}$$

The first term stands for the switching power, which is the power required to charge/discharge the circuit nodes. α_{0-1} is the node switching activity factor of the circuit, which is the average number of the node making a power consuming transition per clock cycle. C_L is the load capacitance, V_{dd} is the supply voltage, and f_{clk} is the clock frequency. The switching power consumption is the dominating component in digital circuits, and it can be reduced by minimizing any one or several of α_{0-1}, $C_{L,}$ V_{dd}, and f_{clk} under the required performance.

The second term in equation (3-1) represents the short-circuit power consumption due to short-circuit current. The short-circuit current in complementary CMOS circuit arises when both the pull-up network and the pull-down network are turned on at the same time during the transitions. The amount of I_{sc} is proportional to the rising and falling time of the input signals, transistor sizes and the output load capacitance [6]. Hence, the longer the transition time for the input signals, the larger the short-circuit current which results in more power consumed. The short-circuit power consumption can be lowered by optimal transistor sizing and input reordering transistors [7].

The total average short-circuit current can be minimized by designing with equal input and output edge times [8]. In this way, the power consumed by the short-circuit currents is less than 10% of the total dynamic power. In particular, when the supply voltage is lowered to be below the sum of the thresholds of the transistors, the short-circuit currents can be eliminated.

The third term in equation (3-1) refers to the leakage power dissipation due to the leakage current. Though one and only one of the pull-up and pull-down networks in a static CMOS circuit is conducting in steady state, there still is a small leakage current which flows through the reverse-biased diode junctions of the transistors between the diffusion regions and the substrate [9]. Another source of the leakage current is potentially the subthreshold current of the transistors. Both sources of leakage caused the static power dissipation which constitutes a small fraction of the overall power dissipation in current technologies. However, with the progress of the technology scaling, the subthreshold leakage currents will become a larger component in total power dissipation. The leakage current depends strongly on the technology, and it can be reduced by applying some techniques such as multithreshold voltage CMOS technology [10] etc.

3.2 Supply Voltage Scaling

The most effective method to reduce the power consumption is scaling the supply voltage, as indicated by equation (3-1). Reducing the supply voltage can significantly reduce the power dissipation that is a quadratic function of the operating voltage. This is illustrated in figure 3.1, which shows the power consumption as a function of V_{dd} for a 4-bit carry look-ahead adder in 0.18 μm process technology. The power consumption dependence on supply voltage for various logic functions and logic styles has been described in [11].

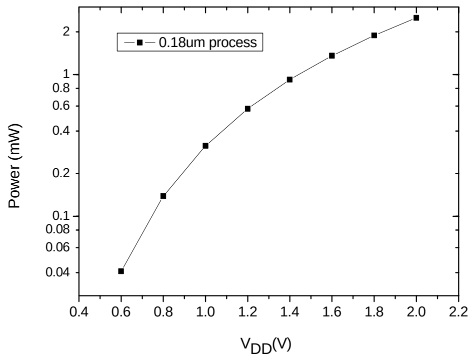


Figure 3.1 Power consumption for a 4-bit CLA as a function of V_{dd}

However, reducing the supply voltage also increases the delay. The relationship between V_{dd} and the delay, T_d, can be expressed [8] by

$$T_d = \frac{C_L \times V_{dd}}{I} = \frac{C_L \times V_{dd}}{\frac{\mu C_{ox}}{2}(W/L)(V_{dd} - V_t)^2} \qquad (3\text{-}2)$$

From the equation (3-2), when V_{dd} approaches the threshold voltage, V_t, the delay increases drastically, as shown in figure 3.2.

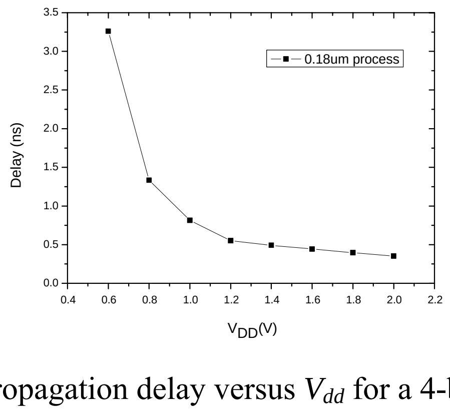


Figure 3.2 Propagation delay versus V_{dd} for a 4-bit CLA adder

Obviously, using this method causes the performance loss on the speed. In order to compensate for the loss in throughput at low supply voltages, several techniques can be applied such as parallel and pipelined architectures as well as modifying the threshold voltage of the devices [8].

3.3 Reducing Effective Capacitance

When the performance loss in throughput due to lowering the supply voltage is not acceptable, reducing the effective capacitance can also obtain low power consumption in CMOS circuits. The effective capacitance is defined by the product of the physical capacitance and the switching activity, which is shown as

$$C_{effective} = \alpha_{0-1} C_L$$

where α_{0-1} is the node transition activity factor, and C_L is the load capacitance which refers to physical capacitance. The switching power consumption can be rewritten as

$$P_{switching} = C_{effective} V_{dd}^2 f_{clk}$$

From the above equation, reducing the switching power consumption can be achieved by minimizing both of the physical capacitance and the switching activity.

3.3.1 Physical Capacitance Reduction

The physical capacitance can be reduced through selecting the appropriate circuit style and optimizing the transistor sizes.

Effects of Circuit Styles

The different circuit and logic styles result in different gate and diffusion capacitance of the transistors in a combinational logic circuit. Some of the circuit styles can substantially reduced the physical capacitance and is good for low-power operation. Figure 3.3 shows the relationship between the power-delay products of an 8-bit adder that was implemented in 2 μm CMOS technology with different circuit styles and the corresponding propagation delays [9].

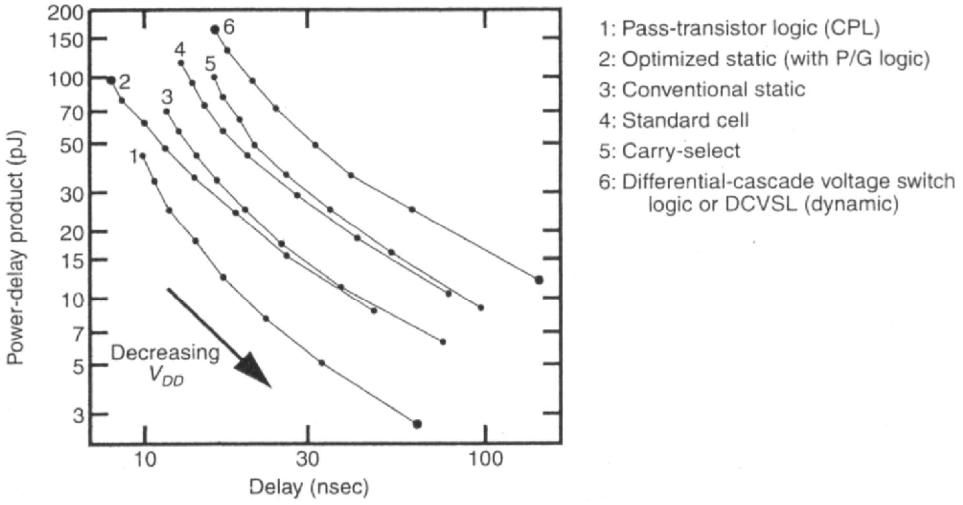


Figure 3.3 Power-delay product versus delay for an 8-bit adder

As shown in Figure 3.3, the adder that was implemented by using complementary pass transistor logic (CPL) is about twice as fast as the conventional static CMOS. This is due to that CPL improves the performance of the circuit with a lower input capacitance and reduced voltage swing. Moreover, a CPL logic circuit consumes less power than a static CMOS one, for instance, the power saving for a CPL adder is about 30% compared to a conventional static CMOS adder [12]. This improvement is mainly due to the reduction in capacitance.

The performance of a full adder implemented with different circuit styles, such as conventional CMOS, transmission gate CMOS, CPL without output swing restoration, CPL with minimum size PMOS restoration transistors (LCPL2), CPL and TG combination (CPL-TG), has been compared in [13]. This comparison reveals that the circuit styles impact dramatically on the delay and power dissipation of the circuit. The compared results indicate that the CPL-TG provides the lowest power delay product, and the LCPL2 has the second lowest power delay product. Both of them are the best suited for low-power high-performance applications such as adders and multipliers.

Transistor Sizing

The capacitive load that originates from transistor capacitance and interconnect wiring can be reduced by optimizing transistor sizes whenever possible and reasonable. In general, increasing the transistor sizes results in a large (dis)charging current and simultaneously increases the parasitic capacitance. On the other hand, reducing the transistor sizes will result in decreasing input capacitance that may be the load capacitance for other gates and lowering the speed of the circuit. Thus, the objective of transistor sizing is to obtain the minimum power dissipation under given performance requirements.

In order to explain how to make transistor sizing, let us consider a static inverter driving a load capacitance being composed of an intrinsic (diffusion) and an extrinsic (wiring and fan out) capacitances. When the total load capacitance to the gate output is dominated by the diffusion capacitance, the smallest possible sizes of the transistors should be used for obtaining the lowest power consumption. Otherwise, if the load capacitance is dominated by the extrinsic component, the power consumption first begins to decrease with increasing transistor sizes and then starts to increase. An optimal sizing factor that corresponds to the minimum power consumption can be found [8].

3.3.2 Switching Activity Reduction

The dynamic power consumption of a circuit is strongly related to the switching activity of the circuit. The node switching activity in the circuit is predominantly determined by the architectural and register transfer level [14]. At the circuit level, one main consideration for low-power designs is the choice of the static or dynamic logic styles. The dynamic logic gates are clocked, and undergo the precharge and evaluation phases, which are suitable for high-speed applications at the expense of high power dissipation [14]. Whereas the static CMOS is the best choice for low-power high-speed implementation of dedicated circuit applications like multipliers [14].

The switching activity can be reduced by many means such as reordering input signals, no bus-sharing technique, and minimizing the glitching activity of the static circuits etc.

Minimizing Glitching Activity

Glitches, or dynamic hazards, are unwanted signal transitions which occur before the signal settles to its intended value. Glitches can be generated and propagated in both data path and control parts of the circuits. Figure 3.4 illustrates the glitching behaviour for a 4-bit ripple carry adder which was implemented in static CMOS.

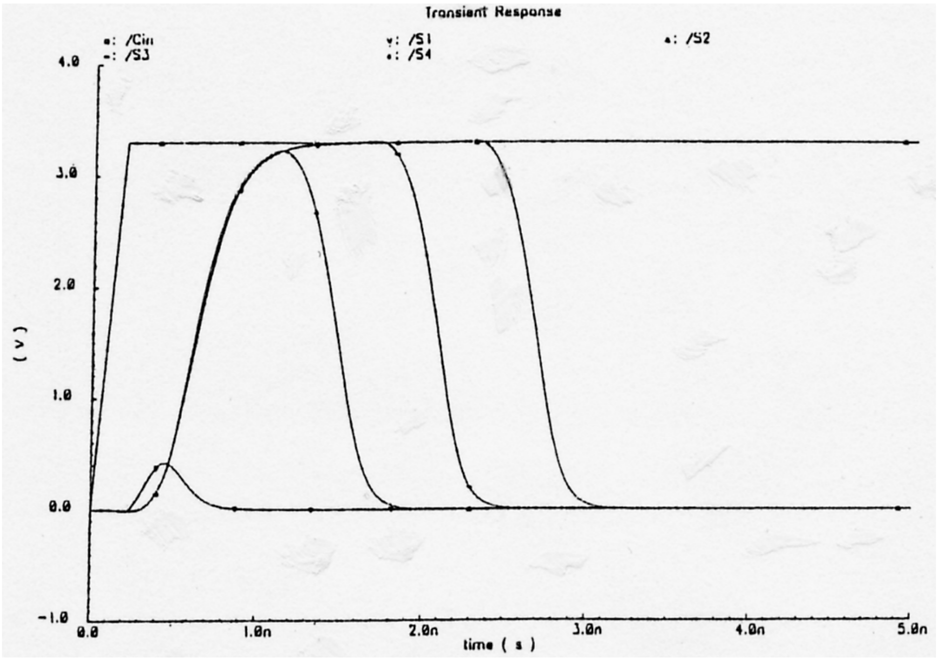


Figure 3.4 Glitching behaviour for a 4-bit RCA

The simulation result from the circuit simulator (Spectre) was obtained under the following conditions. All input bits of A_i and C_{in} go up from zero to one, and all the input bits of B_i are set to zero. As shown in the figure, spurious transitions appear at the sum bits of S_i due to the finite propagation delays of the intermediate carry signals. The spurious transitions consume extra power compared to the glitch-free scenarios. The number of spurious transitions in a circuit depends on the logic depth, input patterns, and intermediate carry signal states etc.

In some arithmetic circuits such as adders and multipliers, the glitches may result in a large portion of the switching power dissipation. For example, in a non-pipelined 16 by 16 bit array multiplier, 75% of the switching power consumption is due to glitches [15].

The glitching activity in static circuit designs can be minimized by selecting structures with balanced signal paths and reduced logic depth. The tree structures can be applied to implement a circuit with both of the balanced signal paths and less logic depth, while the chain structures are quite the contrary. A good example in figure 3.5 illustrates the choice of the tree or chain structures. In the chained implementation shown in figure 3.5(a), the second adder computes twice and the third adder computes three times per cycle due to the finite propagation delay through the previous adders. By contrast, the logic depth in the tree case has been reduced from three to two and the signal paths are more balanced. Thus, the switched capacitance (effective capacitance) for the chained case is a factor of 1.5 larger than in the tree [8].

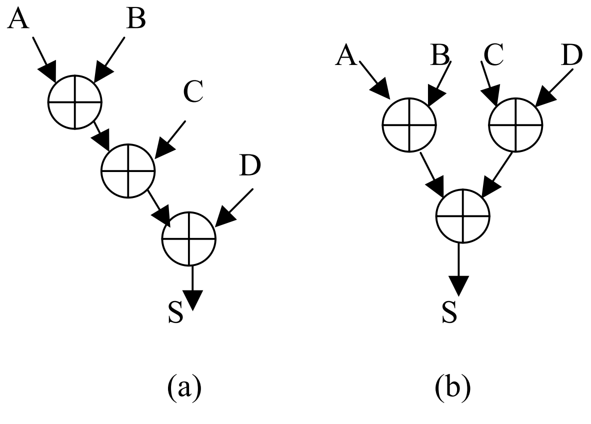


Figure 3.5 Tree versus chain structures

Another possible approach to eliminate the spurious transitions is to use dynamic logic circuits instead of static logic, since any node in dynamic logic circuits can only undergo at most one transition per clock cycle.

3.4 Summary

This chapter briefly described some of the power reduction techniques that are related to the arithmetic circuit designs such as the adder and multiplier. In some arithmetic circuits, the major portion of the switching power consumption is due to glitches. The glitching activity can be minimized by selecting structures with balanced signal paths and reduced logic depth. Furthermore, both supply voltage scaling and reduction of effective capacitance are the important means to lower the power consumption.

This page is left blank on purpose.

Chapter 4 Multiplier Architecture

To meet the various demands of multiplication-based arithmetic operations, many classes of multipliers such as bit-serial multipliers, digit-serial multipliers, and parallel multipliers have been developed. However, for high-speed applications, the parallel multiplier is one of the best solutions.

In general, the architecture of a parallel multiplier consists of the following parts: partial product generator (PPG), partial product reduction tree (PPRT), and final addition. Each part can be implemented by using various architectural choices. Figure 4.1 shows the architecture of the parallel multiplier that has been widely applied for the large multiplier.

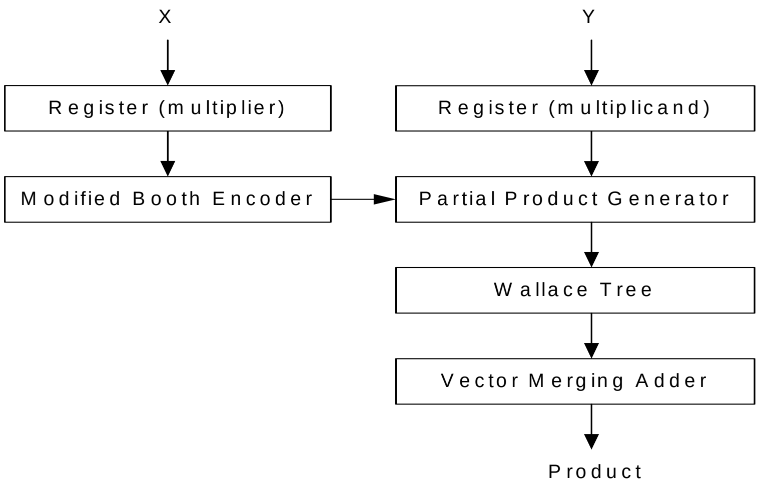


Figure 4.1 Architecture of the parallel multiplier

This architecture consists of modified Booth encoder, partial product generator, Wallace tree that is also called partial product reduction tree, and vector merging adder (VMA).

4.1 Modified Booth Encoder

When calculating fixed-point two-operand multiplication, the modified Booth (MB) encoding is often employed to produce the partial products. Usually, this method is more suitable for input operands of equal to or greater than 16-bit. Using MB encoding to generate partial products, the hardware for this section can be divided into the following three components: modified Booth encoder, partial product and sign bit generators, each component performs different logic functions.

Assuming the multiplier *X* has n bits wide and the multiplicand *Y* has m bits, for this case, *n*/2 or (*n* + 1)/2 three-input MB encoders are required. The n bit multiplier can be partitioned into overlapping groups of three bits in parallel. Each group acts as the input of one of the MB encoders. Each MB encoder generates several control signals to select one of the multiples of the multiplicand *Y* {0, ±*Y*, ±2*Y*}, the MB encoding scheme can reduce the number of partial product by 50% compared to the non-Booth encoding. The MB encoder can be implemented by using various fashions. A glitch-free MB encoder [16] at gate level is shown in Figure 4.2a.

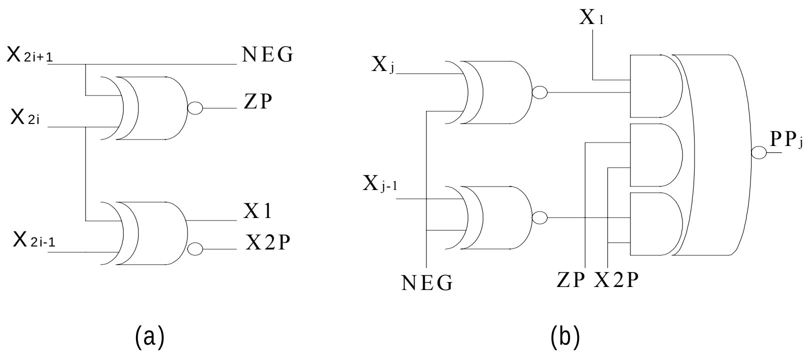
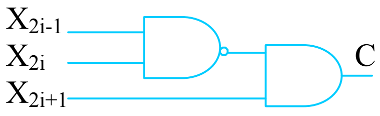


Figure 4.2: a) Glitch-free MB encoder, b) Partial product generator

The partial product generation circuit by using MB encoding is composed of complex gates, as shown in Figure 4.2b [16]. Moreover, corresponding to the operations of the negative partial products $\{-1Y, -2Y\}$, one generator should be implemented. The one generator can be controlled by the adjacent three bits of the multiplier, the circuit at gate level and the truth table are illustrated in Figure 4.3.



X_{2i+1}	X_{2i}	X_{2i-1}	C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

(a) (b)

Figure 4.3: a) One generator, b) Truth table of the one generator

The sign bits of the partial products can be obtained by using sign extension or sign generate methods. A 16×16 bit multiplier by using MB encoding scheme and sign generate is illustrated in Figure 4.4.

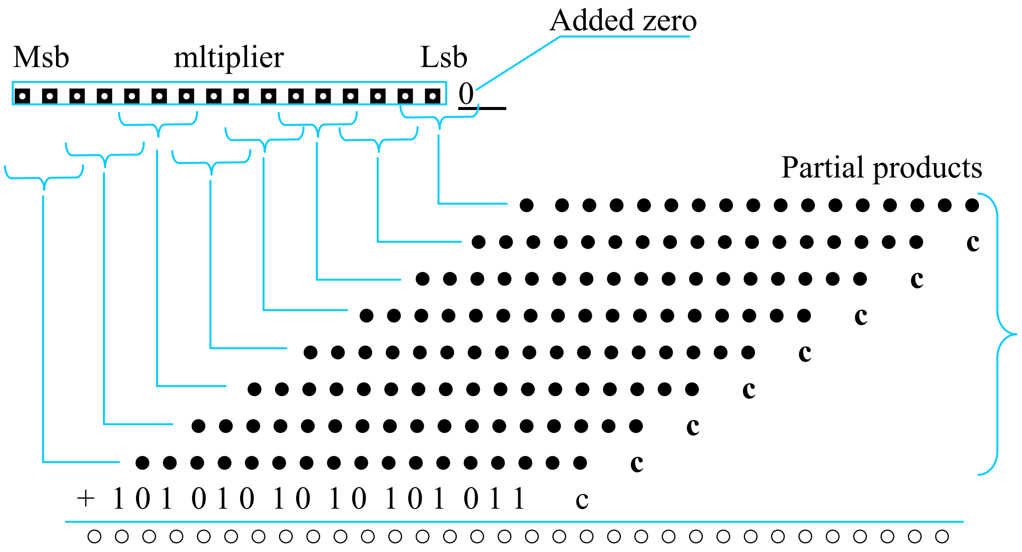


Figure 4.4 16×16 bit modified Booth encoding

4.2 Partial Product Generator

The partial products in a parallel multiplier can be generated using several encoding methods, such as non-Booth encoding, modified Booth encoding (Radix 4), higher radix Booth encoding (Radix 8), and smaller multipliers methods etc. A glitch-free partial product generator with modified Booth encoding at the gate level is shown in figure 4.2(b).

Actually, the partial products can be generated in two stages. During first stage, the modified Booth encoders generate the Booth codes for encoding the multiplicand into partial products. After that, the partial product generators read in Booth code signals and encode multiplicand producing the partial products.

4.3 Wallace Tree

The Wallace tree was proposed by C. S. Wallace in 1964 [17]. This method can be used to sum up all the bits of the partial product in each column. The summation is independent and simultaneous due to each modified Booth encoder works in parallel. It results in all bits of partial products arrive at the adder tree at the same time. Thus, the Wallace tree structure increases the speed of the multiplication by introducing parallelism.

The Wallace tree was first constructed by using 3-2 counters (carry save adders). A 3-2 counter is also called a 3-2 compressor, which has three inputs and two outputs. This counter has a maximum of two XOR delays. The Wallace tree uses 3-2 counters to sum up all the partial products with the same weight, and produce two bits, one is the carry bit with the weight of $n + 1$ and the other is the sum bit with the weight of n.

In order to sum up N partial products to two bits, this operation requires about $\lceil \log_{3/2}(N/2) \rceil$ levels of the 3-2 counters [31]. For example, if the maximal number of the partial product in a column is 7 bits, three levels are required, yielding the Wallace tree with 3-2 counters in Figure 4.5.

The Wallace tree with 3-2 counters is irregular in structure and is difficult to layout due to the irregular interconnections.

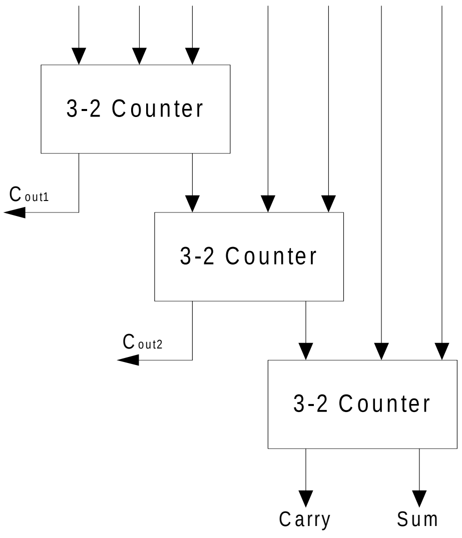


Figure 4.5 Wallace tree with 3-2 counters

4.4 4-2 Compressors

A more regular partial product reduction tree based on a binary tree can be obtained with 4-2 compressors. 4-2 compressors can be used to reduce the number of partial products by one half. This method was first proposed by A. Weinberger, and improved by V. G. Oklobdzija and D. Villeger [18]. A 4-2 compressor can be built by using two 3-2 counters (full adder based) in cascade, as described in Figure 4.6.

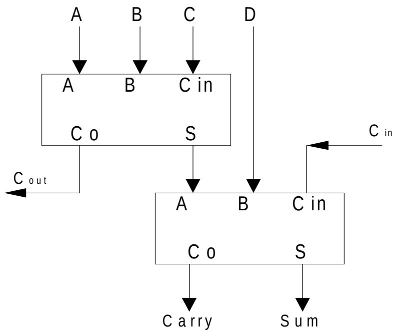


Figure 4.6 4-2 compressor built with two 3-2 counters

As described in Fig. 4.6, a 4-2 compressor has five inputs and three outputs. The five inputs and sum output have the same weight, whereas the outputs of Cout and Carry have one greater binary bit weight. In addition, the output of the Cout does not have to be a function of the Cin input, so that the carry propagation is avoided. By this implementation, the sum, intermediate carry and carry output signals can be expressed with

$$Sum = \left[\left[(A \oplus B) \oplus C\right] \oplus D\right] \oplus Cin$$

$$Cout = A \cdot B + A \cdot C + B \cdot C$$

$$Carry = \left[(A \oplus B) \oplus C\right] \cdot (D + Cin) + D \cdot Cin$$

The 4-2 compressor was constructed as described above and denoted the conventional approach. It has a critical path which contains a maximum of four XOR delays [19]. But this 4-2 compressor has more regular structure and suitable to layout than the 3-2 compressors. The truth table of the 4-2 compressor is shown in Table 4.1.

Table 4.1 Truth table of the 4-2 compressor

				Cin = 0			Cin = 1		
A	B	C	D	Cout	Carry	Sum	Cout	Carry	Sum
0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0	1	0
0	0	1	0	0	0	1	0	1	0
0	0	1	1	0	1	0	0	1	1
0	1	0	0	0	0	1	0	1	0
0	1	0	1	0	1	0	0	1	1
0	1	1	0	1	0	0	1	0	1
0	1	1	1	1	0	1	1	1	0
1	0	0	0	0	0	1	0	1	0
1	0	0	1	0	1	0	0	1	1
1	0	1	0	1	0	0	1	0	1
1	0	1	1	1	0	1	1	1	0
1	1	0	0	1	0	0	1	0	1
1	1	0	1	1	0	1	1	1	0
1	1	1	0	1	0	1	1	1	0
1	1	1	1	1	1	0	1	1	1

An improved approach to build a 4-2 compressor by using pass-transistor multiplexer [20] is shown in Figure 4.7. This structure of the 4-2 compressors includes a critical path with the maximal delay of three XORs. Thus, it has higher performance than that of the full adder based design.

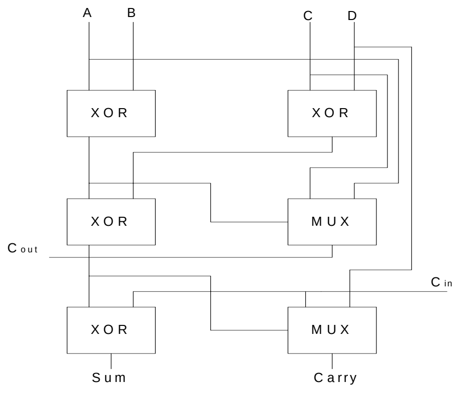


Figure 4.7 An improved structure of the 4-2 compressor

A 4-2 compressor can further reduce the logic depth. For *N* partial products with the same weight, the summation tree built with 4-2 compressors has about $\log_2 N$ levels.

4.5 Vector Merging Adders

The final unit in a parallel multiplier is a fast adder, which performs fast addition for the sum and carry bit vectors from the outputs of the PPRT. There are many different fast adders that suit parallel multipliers, such as carry look ahead, carry skip adder and carry select adder etc. In the following section, the carry look ahead adder and carry skip adder as well as the combination of them will be introduced.

4.5.1 Carry Look Ahead Adder

Carry Look Ahead (CLA) can produce carries faster due to the carry bits generated in parallel whenever inputs change. This technique uses carry bypass logic to speed up the carry propagation. In order to explain carry look ahead, two important signals, traditionally called carry generate (G_i) and carry propagate (P_i), are defined as follows.

$$G_i = A_i \cdot B_i$$

$$P_i = A_i \oplus B_i$$

The concept of the carry generation and propagation can be explained as follow. For a given stage, a carry signal is generated if G_i is true, and it propagates an input carry to its output if P_i is true.

The carry output signal can be derived from the carry generate, carry propagate and the carry-in signals, as expressed by

$$C_{i+1} = G_i + P_i \cdot C_i$$

To avoid carry ripple, the carry output C_{i+1} should be expressed by using the C_i for each stage.

Let us use this technique for the carries of a 4-bit CLA adder

$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_2 = G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)$$

$$C_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

$$C_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

The each above equation, there is a corresponding multi-input circuit. Figure 4.8 shows the block diagram of the 4-bit CLA adder.

From the figure, the CLA circuit generates the carry signals C_1, C_2, C_3, and C_4 by using the carry-in C_0 simultaneously. The adder circuits generate the sums, which is expressed by

$$S_i = C_{i-1} + A_i \oplus B_i$$

$$= C_{i-1} + P_i$$

In general, 4-bit look ahead block is used to implement an *n*-bit CLA adder with a single level. To go faster, an n-bit CLA adder can be implemented at a high level. The number of look ahead levels is $\lceil \log_r n \rceil$, where r is the maximum number of inputs per gate.

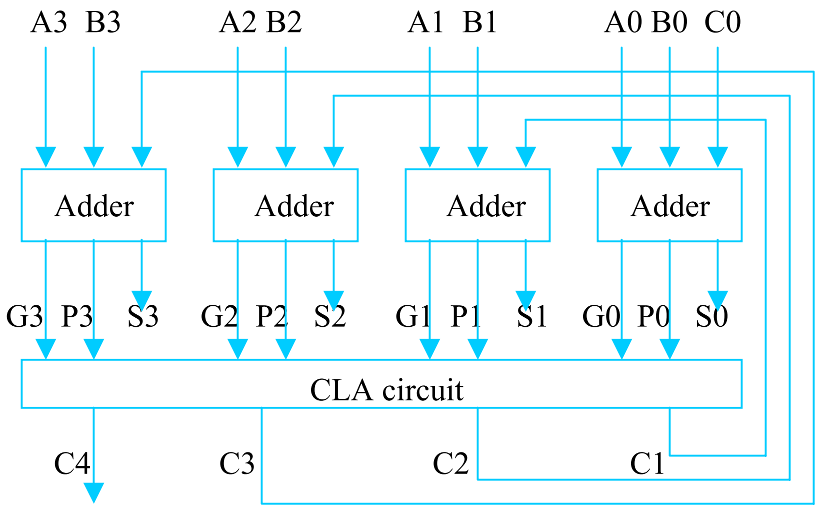


Figure 4.8 Block diagram of the 4-bit CLA adder

The delay of the CLA adder increases as the logarithm of the word size, whereas the delay of the ripple carry adder increases linearly with the word size. Thus, the addition performed by a multi-level CLA for a large word size is much faster than a ripple carry adder. For example, when we compare the number of gate delays for the critical path of two 16-bit adders, one using ripple carry and the other using two-level carry look ahead. As a result, for the 16-bit addition, carry look ahead adder is six times faster than ripple carry [2]. On the other hand, due to high complexity of carry look ahead circuit, it consumes more power than ripple carry adder.

4.5.2 Carry Skip Adder

The carry skip adder is also called a carry bypass adder. In general, a carry skip adder should be built using n-bit ripple-carry adders as basic blocks and multiplexers. Figure 4.9 shows that the block diagram of a 16-bit carry skip adder. Each basic group can be constructed using 4-bit ripple-carry adder.

Each group also generates a group propagate signal P_i which is used as the select signals. P_i can be defined as

$$P_i = p_j \cdot p_{j+1} \cdot p_{j+2} \cdot p_{j+3} \qquad (i = 1,2;\ j = 0,1,2,\ldots.\ 15\ )$$

If P_1 = 1, the carry out signal C_{out0} from the first 4-bit RCA will propagate to the incoming carry of the next 4-bit RCA. In this way, it is possible to bypass the carry out C_{out0} to the carry in of the third or fourth 4-bit RCA. While P_i = 0, the whole carry skip adder becomes a ripple carry adder.

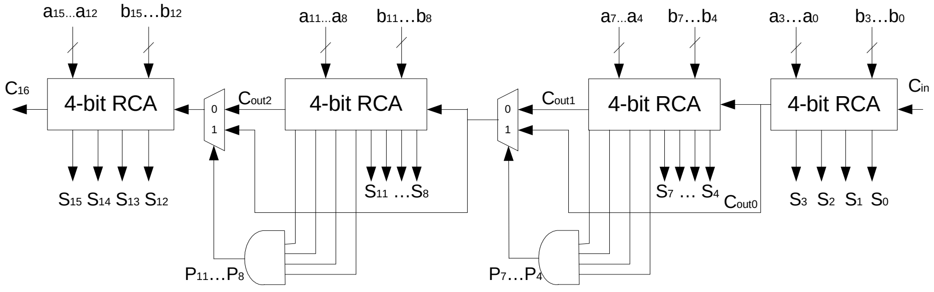


Figure 4-9 Block diagram of a 16-bit carry skip adder

The total propagate delay is linear in the number of bits *N*. Figure 4.10 shows the relationship of the propagate delay between carry skip and ripple carry adders [9]. As can be seen in the figure, for a larger adder the carry skip adder is quite faster than a ripple carry adder, while for a smaller adder the ripple carry adder should be chosen. The crossover point depends on the technology, it is normally between 4 and 8 bits.

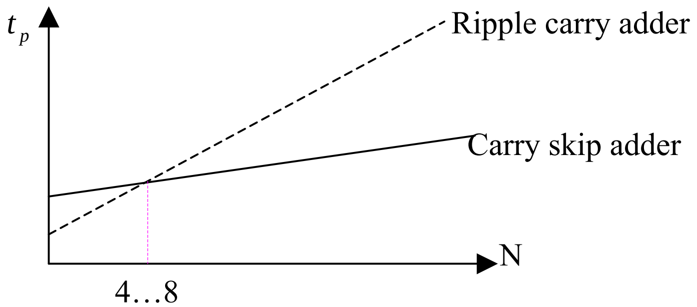


Figure 4.10 Propagation delay of the RCA versus CSKA

4.5.3 Carry Look ahead with Bypass Adder

The carry look ahead with bypass adder has the advantages of both carry look ahead and carry skip adders. The block diagram of a 16-bit carry look ahead with bypass adder is shown in figure 4.11. In this case, the bypass circuit can be implemented by using multiplexer which can be inserted between each CLA adder. The carry bypass signals BP_i is the function of the propagate signal, P_j, for each CLA. If a 4-bit carry look ahead adder used as a basic block constructs the 16-bit, the BP_i can be defined as

$$BP_i = p_j \cdot p_{j+1} \cdot p_{j+2} \cdot p_{j+3} \qquad (i = 1,2,3;\ \ j = 0,1,2,\dots.\ 15\ )$$

The critical path for the CLA with bypass adder could be the first CLA block, three multiplexers and the final CLA block. This method is more suitable for larger adders.

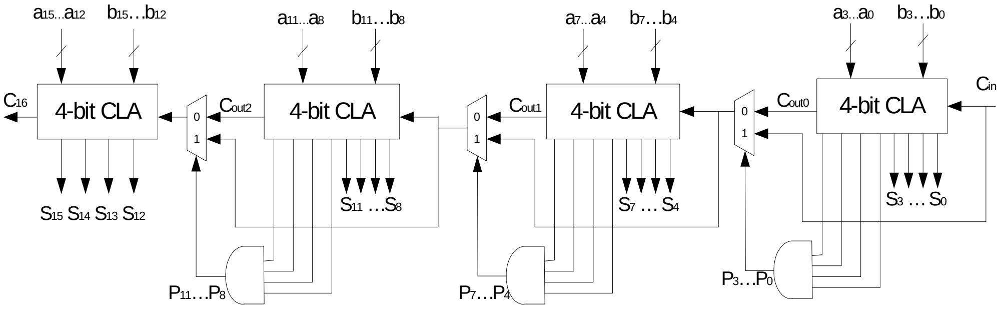


Figure 4-11 Block diagram of a 16-bit CLA with bypass adder

4.6 Partial Product Reduction Tree Topologies

After the partial products are generated, the partial product matrix must be summed up in each column to obtain the final product. To solve this problem, several techniques have been proposed such as the Wallace tree, Carry-save tree, and the Wallace tree based on 4-2 compressors. These approaches are generally called partial product reduction tree (PPRT) [21] or partial product compression tree (PPCT). The PPRT performs the multioperand addition for all the generated partial products and produces the two vectors, the carry and sum.

The summation in the PPRT usually adopts counters and compressors. The counters and compressors can be connected in several different ways. This is a major consideration in the design of parallel multiplier due to different interconnection among these components, leading to different critical path delay. A special terminology, which is called multiplier topology, has been used to describe how to configure the PPRT. That is, the multiplier topology refers to the way of interconnection among bit positions in PPRT.

The multiplier topology has two important points to keep in mind, one is the minimum number of wires needed to interconnect the counters or compressors, and the other is the number of counter or compressor delays. The topologies for configuring the PPRT can be roughly classified into regular and irregular structures.

4.6.1 Regular Topologies

Use of regular topology is a common method in custom digital circuit design, especially for the design of parallel multipliers. Regular topology takes into account a trade-off between performance optimisation and design effort. Regularity makes the generation of the structure possible to program. Regular topology can be realized in array and tree structures.

4.6.1.1 Array Structures

To introduce array structures in a PPRT, let us consider the multiple of two unsigned 4-bit binary numbers. If we apply manual computational method and directly map the operations into hardware, the resulting structure is called an array multiplier. In this case, there is a one-to-one correspondence between the partial products in the multiplier and the ones in the manual method. The adder tree with an array structure is shown in Figure 4.12.

The array topology in figure 4.12 is implemented as ripple carry structure. This array method is also called linear or simple array. There are more than two almost identical critical paths in this array.

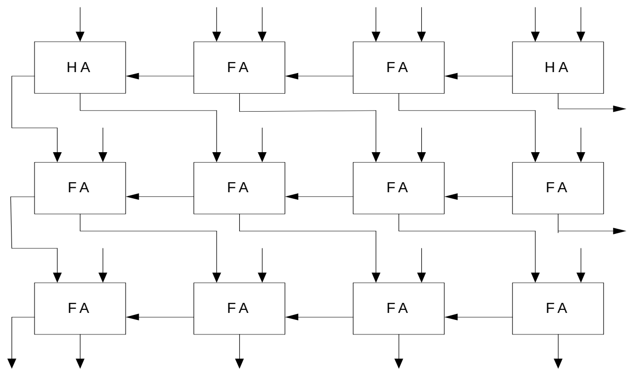


Figure 4.12 4×4 ripple carry array multiplier

The optimization of the performance can only be achieved with careful transistor sizing, which is time-consuming. Another more efficient array structure can be obtained by using carry save adders. The PPRT without the final adder is presented in figure 4.13.

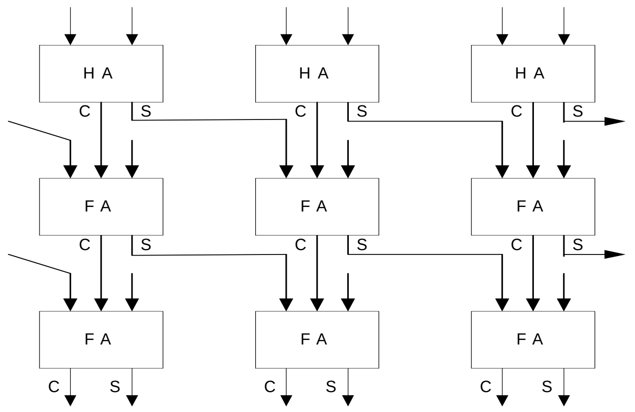


Figure 4.13 Rectangular floorplan of 4×4 carry save multiplier

In this way, the array just includes only one worst-case critical path. The number of counter delays is two full adder plus one half adder, while the number of wiring tracks per bit channel is 3.

Obviously, the addition of partial products can be performed with (*m*, *n*) counters and 4-2 compressors in an array topology. For example, a 54×54-

bit full parallel multiplier [22] adopted 4-2 compressors to construct the PPRT, as shown in Figure 4.14.

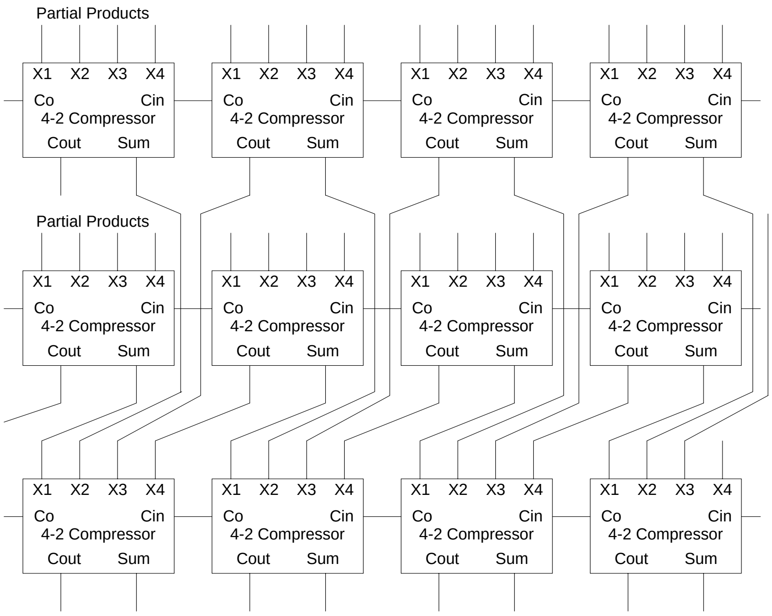


Figure 4.14 Array topology using 4-2 compressors

For this special case, the 4-2 compressors in four levels are needed to add 27 partial products. The compressors can sum up the partial products concurrently. Though the carry out *Co* signal is connected to next 4-2 compressor's carry-in signal, *Cin*, as can be seen in the figure, the *Co* is never produced by *Cin*. Moreover, the 4-2 compressors have 3 XOR gate propagations at the critical path, while two 3-2 counters in carry-save adder tree have 4 XOR gate propagations. Hence, the array topology used 4-2 compressors can speed up the summing computation compared to an ordinary carry save array structure.

In summary, the principle of array topology is to minimize the width of wiring channel per bit position at the expense of counter delays. Comparatively, array topologies have poor speed performance and power wastage due to spurious transitions. But this method has a regular structure and is easy to layout, therefore it still is a common approach to partial product reduction.

4.6.1.2 Tree Structures

Trees, similar to the array mentioned above, are also constructed using counters and compressors, but the tree structure is substantially faster than the array topology due to the optimized depth at the expense of the width of a tree [31]. The width of a tree also refers to the number of wires per bit position in the tree.

Trees are either regular ones or irregular. The width of regular tree is a known function of the number of partial products, while the width of irregular tree is determined by design layout. The regular trees contain binary tree, balanced-delay tree, and overturned staircase tree etc. In this section, a brief overview of the binary and balanced-delay trees will be given.

Binary Tree

The binary tree [23] has a regular structure. This tree is implemented using 4-2 compressors. 4-2 compressors are also called (5,3) counters due to the components with 5 inputs and 3 outputs. The 4-2 compressors can be combined using two (3,2) counters serially or implemented using the method shown in figure 4.7. The latter has three XORs delay in the critical path, whereas the former with four XORs delay.

The binary tree topology with 4-2 compressors is similar to the complete binary tree with *n* nodes. When the *n* nodes in the complete binary tree are replaced using 4-2 compressors, the resulting structure becomes the binary tree topology that can reduce *n* partial products to two bits. The bit slice of a 16-bit binary tree is shown in figure 4.15 [31] on the next page.

The 4-2 compressors can reduce the partial products in the 2-to-1 compression ratios. Therefore, to reduce *n* partial products to two, the $\lceil \log_2(n/2) \rceil$ levels of 4-2 compressors are needed [32].

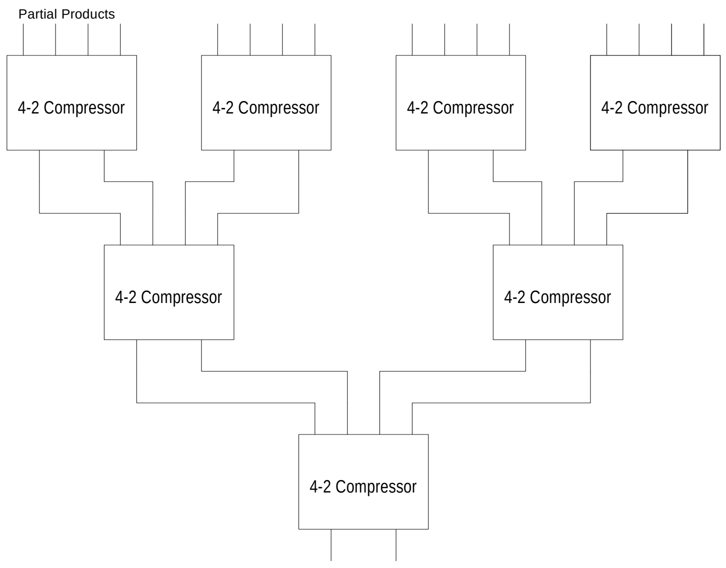


Figure 4.15 A binary tree topology with 4-2 compressors

Balanced Delay Tree

The balanced delay tree was proposed by D.Zuras and W.McAllister (ZM) [24] in 1986. This tree has regular tree structure. It can be recursively defined by a tree body and a chain. The tree body and chain consist of 3-2 counters, the delay in each chain is the same as one of the tree. Figure 4.16 illustrates a balanced delay tree [31], which reduces 18 partial products to two. As can be seen in the figure, the series 3-2 counters increase in the lateral direction as 1, 1, 3, 5,.. . The tree constructed in this way is called a balanced delay tree of type 1. This tree can reduce the *N* partial products in $O(2\sqrt{N})$ 3-2 counters levels [31].

4.6.2 Irregular Topologies

The first tree for partial product reduction was the Wallace tree [17], which was an irregular tree originally built using 3-2 counters. This topology can speed up the multiplication time for large multipliers compared to the carry save structure. On the other hand, it is hard to design and layout due to the irregularity and complexity. Thus, the Wallace tree is only applied to the designs whose performance is critical and design effort is of secondary consideration.

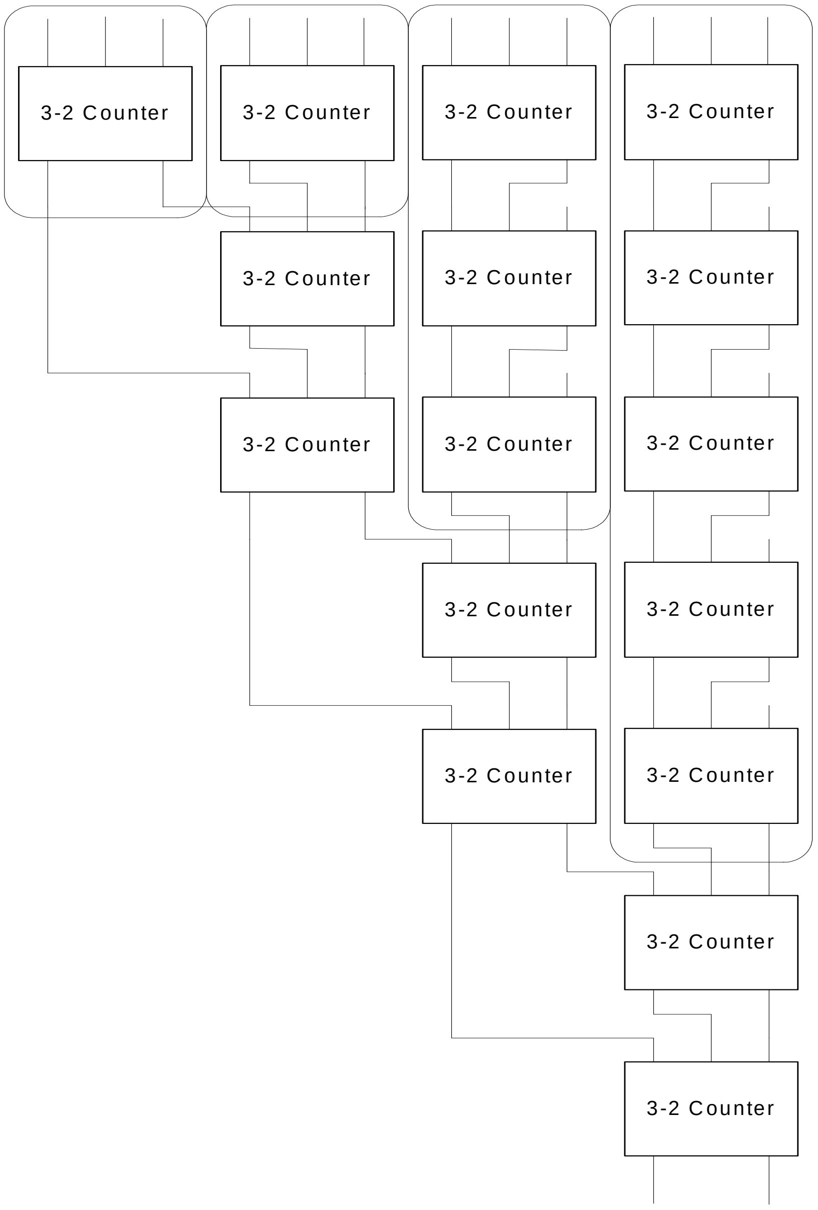


Figure 4.16 A balanced delay tree topology using 3-2 counters

4.7 Summary

In this chapter, the most commonly used architecture for implementation of a parallel multiplier was presented, and the partial product reduction tree topologies were also described. The topologies can be divided into either regular or irregular. The regular topology can be realized in array and tree structures, which takes into account a trade-off between performance optimization and design effort. The irregular topology is applied to the designs whose performance is critical and the design effort is of secondary consideration.

Chapter 5 Implementation

Various design and implementation techniques for parallel multipliers have been proposed. However, most of these techniques can only be applied for that of fixed word-length multipliers. This means that when the word-length of the multipliers or the process technology is changed, the whole circuits need to be redesigned. To solve this problem, this chapter describes the design procedure of the parallel multipliers with variable sizes through the implementation of an 18×18 bit multiplier. This design consists of the logic, circuit and physical stages. The meet-in-the-middle design methodology will be used in this thesis. For this custom design, the speed, area and power trade-off of the multipliers should be taken into account. Furthermore, some choices for the design from the architecture to the circuit styles will be made later in the following sections.

5.1 Architecture Selection

Selecting the architecture is very important before starting the detailed design, since the performance such as speed, area, and power dissipation strongly depends on the architecture of the multiplier. The architecture should be low-power, high-performance, and easy to implement. The choice of the architecture should include the encoding scheme, sign extension method, partial product reduction structure and the final addition. The chosen architecture is shown in figure 4.1.

For the selected architecture, the modified Booth encoding is used for this design due to MBE can reduce the number of the partial products by a factor of 50%, which results in a higher speed and a lower power dissipation than a non-Booth encoding scheme. Also, the glitch-free partial product generator is employed, whose gate-level circuit is shown in figure 4.2(b). In addition, the Wallace tree with 4-2 compressors is applied to the partial product reduction tree. The tree structure in the static style can reduce the spurious transitions as mentioned in section 3.3.2 and increase the speed of summing by means of increased parallelism as well as result in a much more regular structure. Furthermore, the carry look-ahead adders that uses block of 4 bits are made use of the final addition. These are the most common choice and more suitable for the cell-based design of the multiplier with different operand sizes.

5.2 Design Methodology

For a custom ASIC design, several design methodologies such as bottom-up, top-down and meet-in-the-middle have been proposed. These methods can be applied to deal with the design complexities and reduce the design time. Which methodology to select depends on the system requirements, the complexity, the cost and the available time. In this thesis, the meet-in-the-middle design method will be used to develop the module generator for parallel multipliers. By using this method, the logic and circuit designs could be started independently and simultaneously, but the results for the logic and circuit designs should be exactly same. The basic principle of this method is that the specification process is essentially a top-down method to validate the functional correctness, while the design of the building blocks for the circuits starts from the bottom level. Thus, the advantages of this methodology are that the library cells can be reused, and the circuit design phase can be shorted through using program creating the structural schematic. Because the cell-based design of a large parallel multiplier could include hundreds of building blocks and the interconnections among these blocks could be up to thousands. In such a case, the structural schematic of a manually generated multiplier is hard to ensure correctness for.

5.3 Design Flow

The previous section presents the architecture and the design methodology employed in the multiplier. In this section, the design flow used to realize the above architecture is presented, as shown in figure 5.1.

The design process based on the design flow starts from the specification that defines the functionality and performance targets according to the user’s requirements. Based on the specification, the architecture, including the encoding scheme, the PPRT structure, and the vector merging adder should be chosen. The modified Booth encoding method and its application in multiplier have been proved in [25, 30]. In general, the design is partitioned into more basic blocks so that each block can be reused and analysed individually. After this, each cell symbol is created in Cadence, however, the cell symbols at this moment are combined by using only empty rectangles with the input and output pin names, the circuit inside is going to be designed in circuit design stage. Also, the preliminary floorplan should be created through estimating the size and complexity of the each block. Based on the above design steps, the logic and circuit design can be started. When logic and circuit designs are complete, the physical design may begin. In the subsequent subsections, these three design stages will be presented in more detail.

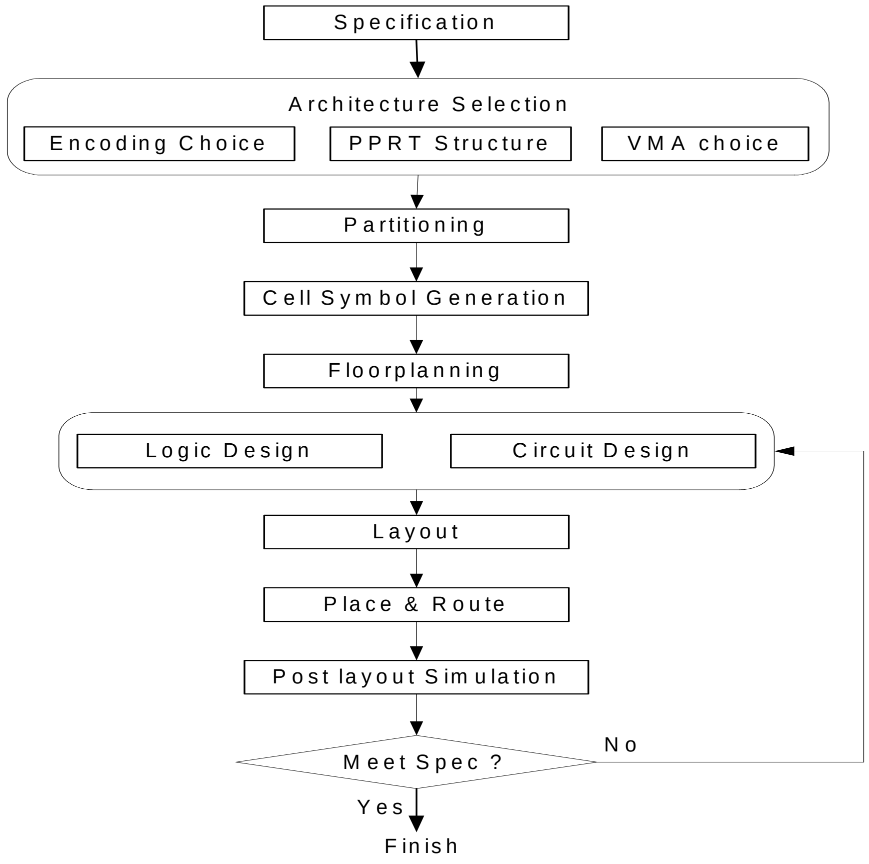


Figure 5.1 Design flow

5.4 Logic Design

As shown in figure 5.1, the logic design begins with creating the VHDL-code for each basic block when the design partition is finished. For the logic design, there are several tasks to be performed, which will be described in the following sections.

5.4.1 Cell Models

Each cell model is created in VHDL, these functional models are then validated by using a corresponding testbench. The design environment is the Modelsim from Mentor Graphics for VHDL simulation, the text editor that used for VHDL editing, Vcom for VHDL compiling, and Vsim for VHDL simulation.

5.4.2 Creating schematic generation files

In order to create the structural schematic, three files used to define the necessary information need to be created by using a dedicated C program that is written in UNIX C.

The first file named BKins.txt is used to define the cell names and instance names as well as their coordinates.

The second file called BKpin.txt is created to describe the input/output pin definition and connection to the corresponding cells. This file includes the input/output pin names that will be connected to the bus, instance names and the pin names on the instance. The relationship among them is that the input/output pins from a bus connect to the pins on the instances.

The third file called BKnet.txt defines the interconnections between different logic blocks. This file consists of instance names and their pin names. There are two instance names and two corresponding pin names on each row in this file, and the two pins are interconnected.

5.4.3 Gate-Level Schematic Generation

When the three files are created and the cell models written in VHDL are validated, the gate-level schematic for the entire multiplier is ready to be generated by using another existing program that was written in the SKILL programmable language. This method has been proved to be feasible for generating some part of the multiplier. However, to create the entire schematic of the multiplier also needs manual intervention due to the complexity. Since this project includes a lot of design tasks, there was one part of the C program that was not completed during this thesis work. This means that the C program still needs to be modified in the future work. Of course, this schematic can also be generated manually. The schematic for the 18×18-bit parallel multiplier is shown in figure 5.2.

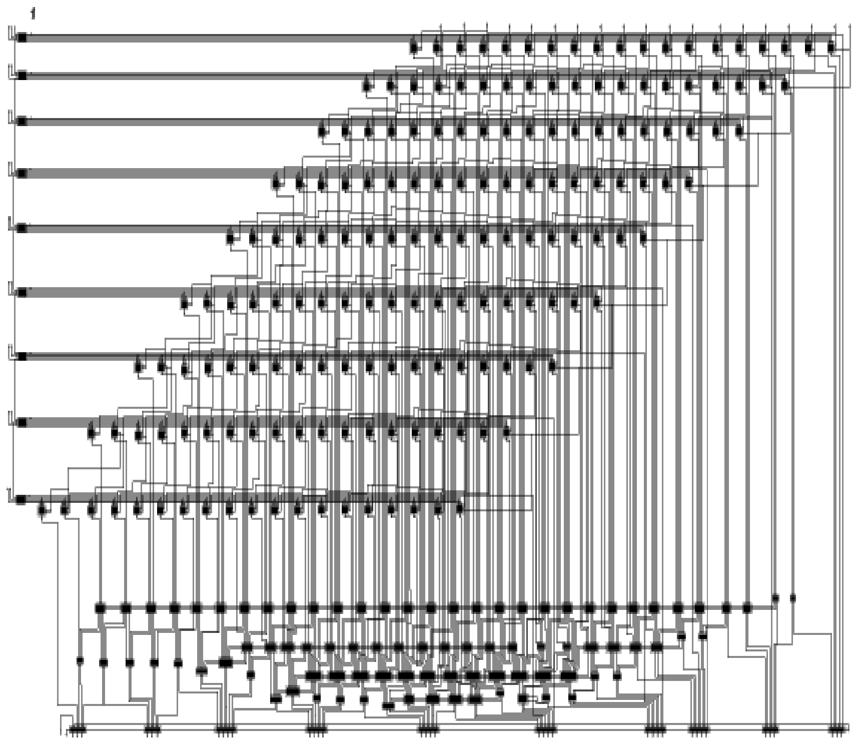


Figure 5.2 Schematic for the 18×18-bit multiplier

5.4.4 Structural VHDL Generation

Once the gate-level schematic of the whole multiplier is generated, the structural VHDL of the multiplier can be created using another existing program that was also written in SKILL. This structural VHDL-code describes the hierarchical design that reduces the size and complexity of the schematic. This structural VHDL-code for the 18x18-bit multiplier is more than 1400 lines, which would be very hard to generate manually. Moreover, this hierarchical structure includes 31 basic blocks whose VHDL code has been created manually and each cell has been validated through simulation using an individual test bench.

5.4.5 Structural VHDL-Code Validation

After the structural VHDL-code is automatically generated, the VHDL-code is validated using a test bench. The test bench has been created during this thesis work and works well. It can be reused after specifying the word length. A random test method has been used for this validation. 20000 test vectors have been generated randomly. These test vectors are used to drive the model under test and the responses have been compared to the expected responses that have been generated using the conventional multiplication method.

5.5 Circuit Design

When the functionality of the multiplier is validated through the structural VHDL simulation, the circuit design can be started. Actually, the circuit design can be also started together with the logic design if there are more designers. The circuit design of a multiplier can be accomplished manually or automatically, which one to be used depends on the required specification. In this thesis, the design for each building block at the low-level is performed by the custom method.

5.5.1 Custom versus Automatic Designs

There are two styles of circuit design at the low-level: custom and automatic.

The automatic circuit design uses synthesis tools to select circuit topologies and gate sizes. It takes much less time than custom design in which both the schematic is drawn and the paths are optimised manually. On the other hand, the drawback is that the results in this way strongly depend on the synthesis tools, and it is usually restricted to a fixed library of static CMOS cells and generates slower circuits compared to the designs by a skilled engineer.

The custom design gives the designers more flexibility to generate building blocks at transistor level or select the predefined cells from a given library. The design effort is to obtain the better performance. In a custom design, many design decisions such as logic styles and circuit topology choices should be made before starting the design. The custom design is a good method to understand the basic principle of the ASIC design. Therefore, the custom design approach is adopted in this thesis.

5.5.2 Logic Style Considerations

Implementation for the partitioned blocks at circuit level can be done in static logic or dynamic logic circuits. Both of them have advantages and disadvantages. The one to be used depends on the primary requirement such as ease of design, robustness, area, delay, or power consumption etc. As opposed to dynamic logic gates, static logic gates have the advantages of being robust and they are more amenable to supply voltage scaling. Thus, for low-power design, the static approach is attractive for the implementation of the arithmetic circuits such as adders and multipliers. In this design, several static logic styles, e.g. static CMOS, transmission gate (TG), and complementary pass-transistor logic (CPL) are considered.

Static CMOS

A static CMOS gate is composed of two complementary networks. One is the pull-up (PUN) and the other is the pull-down (PDN). The PUN consists of PMOS devices, whereas the PDN is constructed using NMOS transistors. This choice is due to the strong one generated by the PMOS and the strong zero produced by the NMOS. In steady state, one and only one of the networks turns on. This implies the output node is always a low-impedance node and hence there is no static power consumption. The static CMOS has high noise margins due to its V_{OH} and V_{OL} equal to V_{dd} and GND, respectively. Its robustness against transistor sizing and voltage scaling allows the operation at low voltages. Therefore, it is the best choice for implementation of a low-power, low voltage combinational circuits due to its single-rail property that saves the routing resources, and its robustness which is an important issue in submicron VLSI design [14].

The partial product generator and one generator are implemented by using static CMOS gates.

Transmission Gate

The transmission gate (TG) based on pass-transistors is composed of an NMOS and a PMOS device in parallel. It acts as a bidirectional switch controlled by the gate signals that are a pair of complementary signals. The TG logic is one of the ratioless logic gates. Therefore, if the external load capacitance is not the dominating factor in the devices, the minimum transistor sizes should be used as mentioned in section 3.3.1, which results in less area and power dissipation. The TG is characterized by no threshold loss and an almost constant resistance (RC-equivalent network) that implies that long cascades of the TG should be avoided. The drawback of this circuit is that the control signal and its complement have to be generated, and a static buffer at the output needs to be used to improve the switching speed for the gates with large load capacitance.

The TG can be applied to efficiently build some complex gates such as XOR/XNOR, which is shown in figure 5.3. The implementation of such a gate requires a less number of transistors when compared to other logic styles implementing this gate. For instance, a new CMOS XOR circuit based on pass-transistors uses only six transistors to produce both an XOR and its complementary XNOR functions with full voltage-swing and negligible static power dissipation [26], which is shown in figure 5.3(b). The conventional implementation of an XOR gate using CMOS would require about 12 transistors.

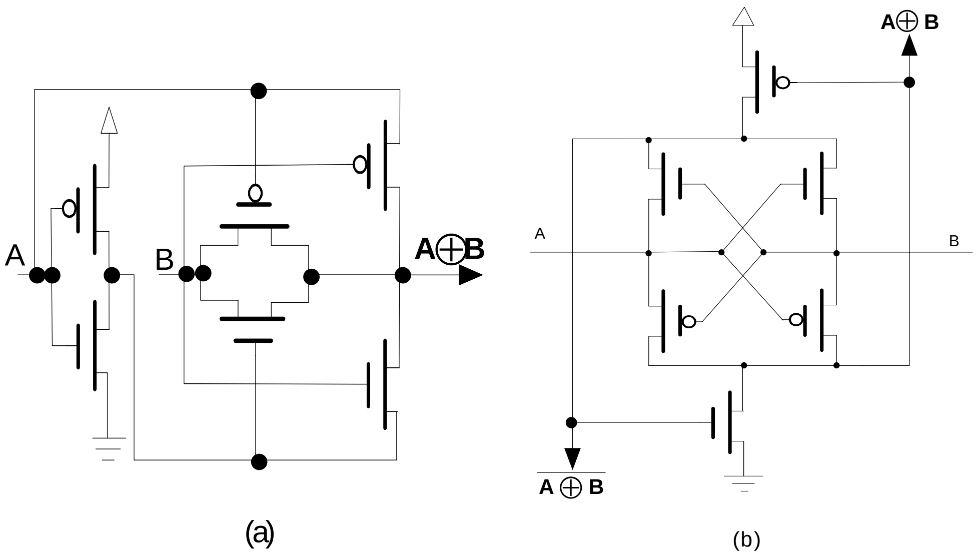


Figure 5.3 Pass-transistor XOR/XNOR circuits

Since the logic functions for modified Booth encoder, and one-bit adders using propagate and carry in signals for CLA, are simple XOR and XNOR operations, it is an efficient method to implement these blocks with minimum area and power using TG.

Complementary Pass-Transistor Logic

The typical CPL gate is constructed by using two NMOS logic networks for dual-rail, two small pull-up PMOS transistors for swing restoration, and two output static CMOS inverters for the complementary output signals. A number of CPL gates, such as AND/NAND, OR/NOR, and XOR/XNOR, can be implemented using a small number of transistors, as shown in figure 5.4 [9].

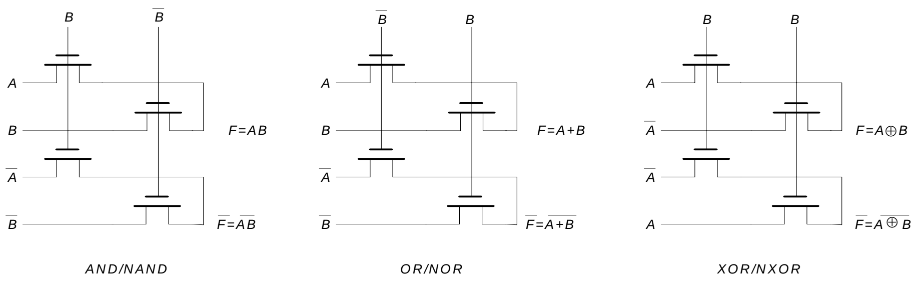


Figure 5.4 Typical complementary pass-transistor logic gates

The advantages of the CPL are the small input capacitance that results in lower power and faster operation, and the intermediate low-swing node which contributes to lowering the power dissipation as well as the fast differential stage due to the cross-coupled PMOS pull-up transistors. The differential characteristic means that complementary input and output signals are always available which eliminate the need for extra inverters.

The drawbacks are a larger short-circuit current due to the differential stage which consumes extra power, and high wiring overhead due to the dual-rail signals. However, the CPL gates still are good choices for low-power high performance digital circuit applications [14] in the current process technology.

The 4-2 compressors, full adders, half adders and increment adders in the partial product reduction tree are implemented by using CPL gates with the logic functions AND, OR, XOR and their inversions.

5.5.3 Leaf Cell Design

This section describes the design of the leaf cells for the multiplier. The design is done with bottom-up method. Each block is built based on the partition as described in section 5.3 and the chosen logic styles described in section 5.5.2.

The design procedure for each cell includes the description, the design, the implementation, the layout and the simulation, and so forth.

The partitioned basic blocks for the multiplier contain the modified Booth encoder, one-generator, partial product generator, 4-2 compressor, full adder, half adder, increment adder, 3-bit and 4-bit carry look ahead adders etc. The design and implementation for each leaf cells will be presented in subsequent subsection.

5.5.3.1 Design Environment

The design and implementation for the multiplier will be performed using Cadence electronic design package. The process technology used is 0.18 µm CMOS that is a modern technology. This process has some special features, such as six metal layers available, thinner wire, minimum area requirement, smaller transistor length with minimal value of 0.18 µm and the minimum width with the value of 0.28 µm, and so on.

5.5.3.2 Transistor Sizing Criteria

To satisfy the performance constraints, the transistor sizing is very important for a custom design. This is a tedious and subtle process due to the choice of the transistor dimensions having a major impact on the area, performance, and power dissipation of a circuit. For this reason, some transistor sizing strategies have been proposed as shown in section 3.3.1. During the implementation of each cell, the transistor sizing, and the transistor resizing after the postlayout simulation, if necessary, can be accomplished according to the criteria below, if not otherwise specified.

1. The minimum length with the value of 0.18 µm for both NMOS and PMOS is used.
2. The rise and fall time of a gate is less than 1 ns.
3. The width of NMOS is determined according to the load capacitance to be driven. Since the transistor size increases with the increase in the load driven by it for a given transition time at the output signal of a gate, the actual capacitance can only be back annotated after the layout. Therefore, the trial and error method is used.
4. The ratio between NMOS and PMOS is taken by a factor of 2.

5. N cascaded transistors need N times of the width of the only one transistor.

5.5.3.3 Layout Requirements

When implementing a cell layout in Cadence, in addition to following the given design rules, several constraints also need to be specified as follows.

1. To simplify the routing, all building blocks have to be adjusted to have the same height between V_{dd} and GND.
2. The "align" layer should be used as a boundary for each layout block. The "align" rectangle is very suitable as a reference when aligning a number of blocks during the routing.
3. The layout styles for each cell should be as similar as possible.
4. The pin name for each cell should be defined by using the "dot" shape.

5.5.3.4 Implementing the Modified Booth Encoder

Description

The modified Booth encoding (MBE) technique has been widely applied for partial product reduction in parallel multiplier implementations. The MBE using the Radix-4 encoding scheme can reduce the number of partial products by half. The MBE can also be defined and implemented in many ways.

The conventional implementations for the MBE have a large fan out which results in a slow implementation and extra power consumption. Some techniques used to improve the MBE in both speed and power aspects have been proposed. One of them is a glitch-free MBE recoding scheme at gate level [27] which was implemented by using XOR/XNOR gates at transistor level [16]. It used a 6-transistor CMOS XOR circuit with complementary outputs. This MBE encoder works well in 0.35 µm process at a supply voltage of 3 V. However, the same structure did not work in the same process at 1.5 V. For this reason, a new glitch-free MBE encoder based on the principle proposed in [27], which is also implemented by using TG XOR structure, has been proposed in this thesis. This new MBE encoder works well in both 0.35 μm and 0.18 μm process at a supply voltage of 1.5 V and 1.2 V, respectively. The truth table for the new MBE encoder is the same as the one presented in [27], which is given in table 5.1.

Table 5.1 Truth table of the modified Booth encoder

B_{2i+1}	B_{2i}	B_{2i-1}	d_i	Inv_ COMP	SHIFT	Inv_ SHIFT	ZP	ONE	PPG	PPn
0	0	0	0	0	1	0	1	0	0	1
0	0	1	+1	0	0	1	1	0	A_1	$\overline{A_1}$
0	1	0	+1	0	0	1	0	0	A_1	$\overline{A_1}$
0	1	1	+2	0	1	0	0	0	A_0	$\overline{A_0}$
1	0	0	–2	1	1	0	0	1	$\overline{A_0}$	A_0
1	0	1	–1	1	0	1	0	1	$\overline{A_1}$	A_1
1	1	0	–1	1	0	1	1	1	$\overline{A_1}$	A_1
1	1	1	0	1	1	0	1	0	0	1

From the table 5.1, the modified Booth encoder functions, inv_COMP, SHIFT, inv_SHIFT, ZP, ONE, PPG, and PPn, are derived as follow.

$$\text{inv\_COMP} = B_{2i+1}$$

$$\text{SHIFT} = \overline{B_{2i} \oplus B_{2i-1}}$$

$$\text{inv\_SHIFT} = \overline{\text{SHIFT}}$$

$$\text{ZP} = \overline{B_{2i} \oplus B_{2i+1}}$$

$$\text{ONE} = B_{2i+1} \cdot (\overline{B_{2i} \cdot B_{2i-1}})$$

$$\text{PPG} = \overline{\overline{\text{SHIFT}} \cdot (\overline{A_1 \oplus \text{inv\_COMP})}} \cdot \overline{\text{SHIFT} \cdot ((\overline{A_0 \oplus \text{inv\_COMP}}) + \text{ZP})}$$

$$\text{PPn} = \overline{\text{PPG}_{N-1}}$$

where B_{2i+1}, B_{2i} , B_{2i-1} represent the adjacent 3 bits of the multiplier, B, and SHIFT denotes whether a shift is required, inv_COMP indicates whether the partial product is negative or positive, ONE is the correction constant required to generate a negative partial product, PPG is the partial product, PPn is the one's complement of the sign, d_i is the recoding digit and A_1, A_0 indicate the adjacent 2 bits of the multiplicand.

This new MBE encoder can be used to achieve the possible equal path for all output signals, which also is very compact due to just two more transistors used compared to the implementation in [16].

The Implementation

The transistor level schematic for the MBE encoder, including the one-generator, is shown in figure 5.5.

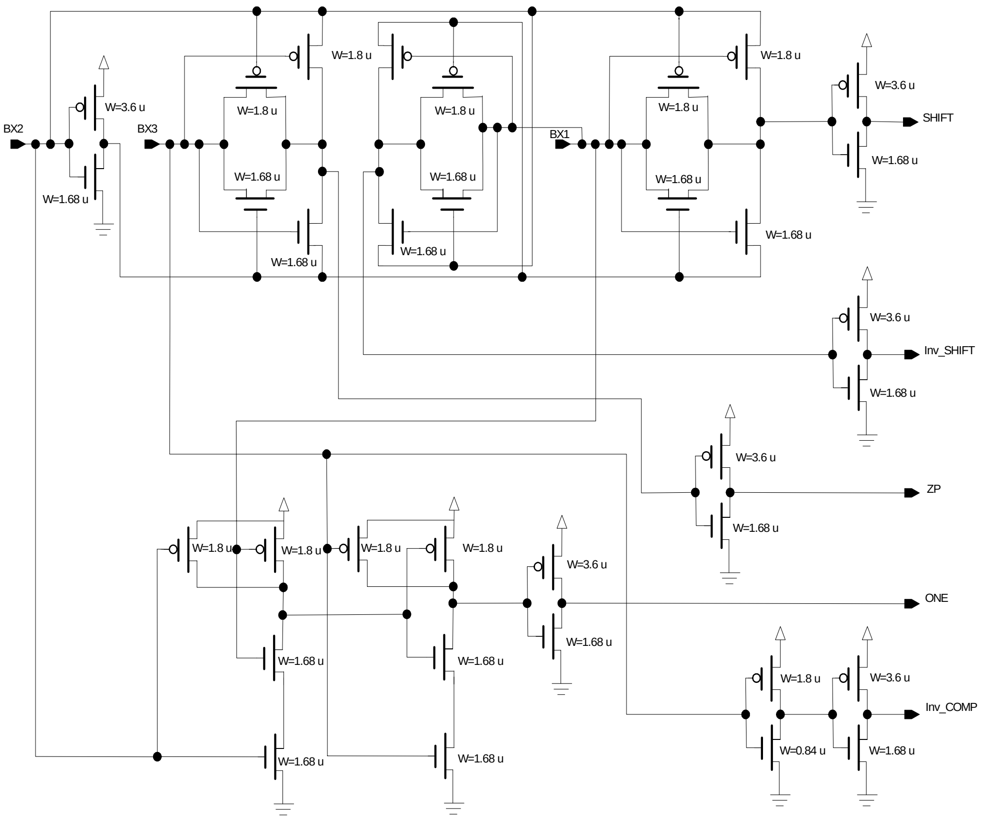


Figure 5.5 Schematic of the MBE encoder

The Layout

Figure 5.6 shows the layout for the MBE encoder and the one-generator with buffers. For the layout, two metal layers, mental1 and metal2, have been used for local interconnections in the cell. This block integrates 34 transistors which contains the MBE encoder, output buffers, input inverter and one-generator. It has the area of 222 μm^2.

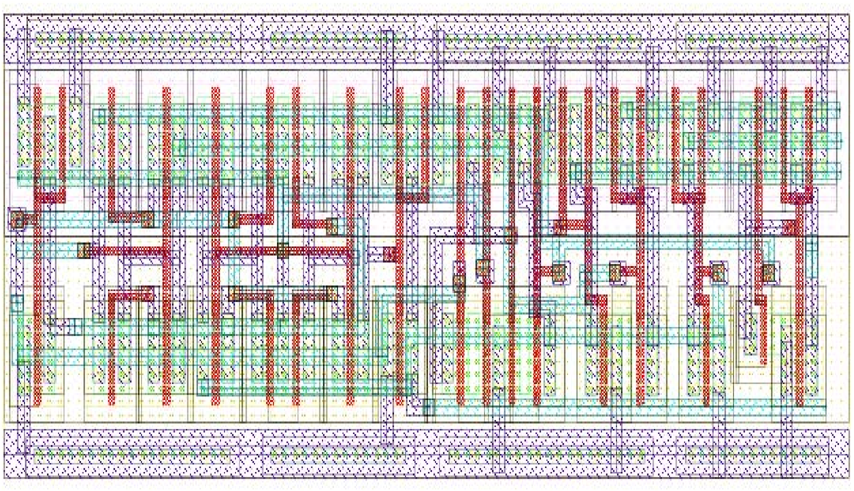


Figure 5.6 Layout of the MBE encoder

When the cell layout is created and has passed the DRC (design rule check) check, the extracted view has to be generated, which translates the layout into a netlist with the parasitics. Afterwards, the LVS (layout versus schematic) can be done. If no discrepancy could be found between the extracted netlist and the schematic netlist, the analog extracted view can be created, which is similar to the extracted view, however, it is more suitable for the simulation.

The Simulation

When the analog extracted view is produced, it is possible to start the simulation for the cell by using the simulation tool that is called “Affirma Analog Circuit Design Environment”. Before running the simulation, a cellview called configuration has to be built, which includes all the cellviews needed for creating the netlist. After this, the simulation with parasitics can be started.

The MBE encoder and one-generator were validated to be correct after many simulations. The resulting transistor sizes have been determined according to the simulation results and the transistor sizing criteria described in section 5.5.3.2, which is also shown in figure 5.5. Since this cell has been simulated together with the PP generator, their combined propagation delay and power dissipation will be given in the following subsection 5.5.3.6.

5.5.3.5 Implementing One-Generator

The one-generator is used to create a binary one during the operation of the MBE encoding. When the operation is –Y or –2Y, a binary one has to be added to the LSB of the partial product. The gate level schematic and the truth table have been described in figure 4.3. The transistor level schematic and the layout have been combined in the MBE encoder as shown in figure 5.5 and 5.6.

5.5.3.6 Implementing PP Generator

Description

A glitch-free partial product generator based on a glitch-free MBE encoder proposed in [27] can be used to reduce the energy dissipation by about 30% due to eliminating the spurious transitions. Such a structure has been implemented at transistor level by using 0.35 μm process at 3.3 V supply voltage [16]. This partial product generator consists of three stages, where the first stage is the XNOR operation, which was implemented using XOR/XNOR gates [26]. To make sure that this logic circuit is working correctly in the 0.18 μm process at a lower supply voltage of 1.2 V, another alternative using TG XNOR gates is employed to replace this stage, whose transistor level schematic is illustrated in figure 5.7 on the next page.

The truth table has been given in table 5.1. In this table, PPG represents the partial product, and PPn stands for the generated sign bit.

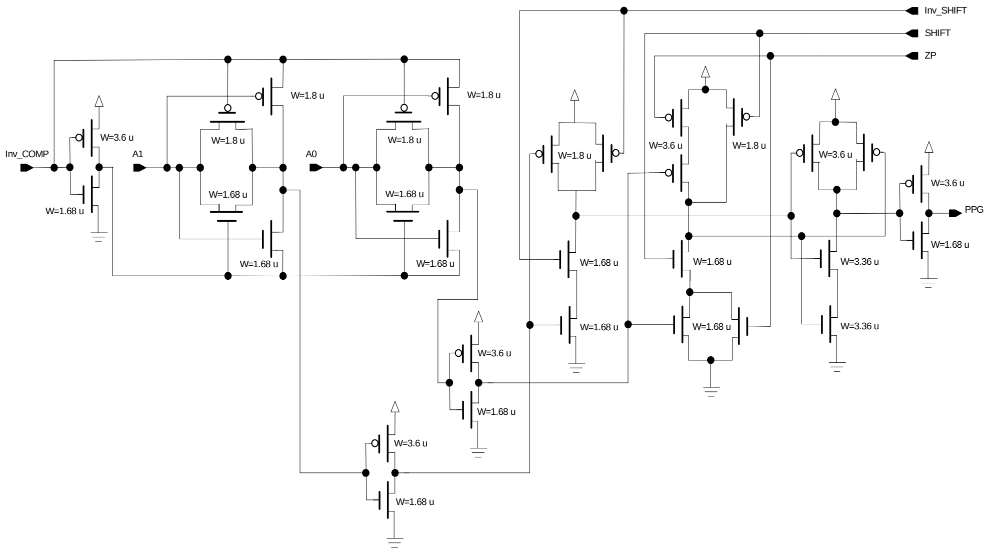


Figure 5.7 Schematic of the partial product generator

The Layout

The layout of the partial product generator is illustrated in figure 5.8. For this block, the number of transistors, including the inverters, is 30, and the area is 210 μm^2.

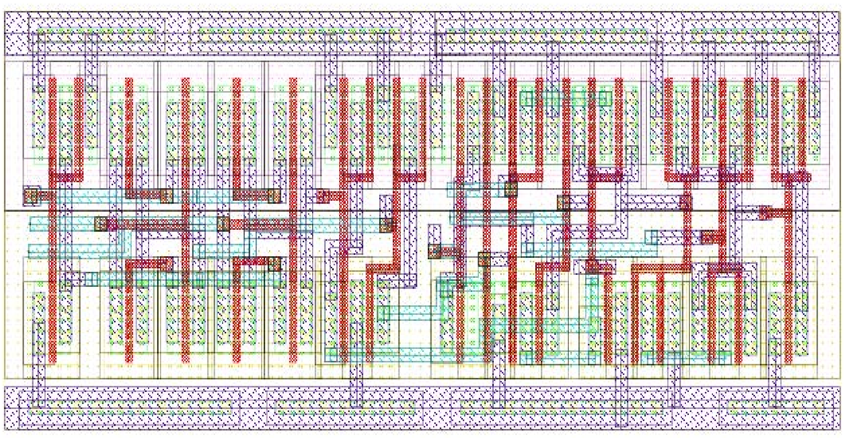


Figure 5.8 Layout of the partial product generator

The Simulation

When the configuration view is available, the simulation of the extracted net list with parasitic capacitance can be done. The simulation results indicate that the logic function is correct. After simulating several times, the transistors have been sized as shown in figure 5.7. The maximum propagation delay and the average power dissipation, including the modified Booth encoder, for the input patterns of $B_{2i+1}, B_{2i}, B_{2i-1} = 000 \sim 111$ and $A_{j+1}, A_j = 11$, are 1.5 ns and 0.16 mW.

5.5.3.7 Implementing Sign Generator

Description

Sign-extension is an important aspect to implement a high speed and low power parallel multiplier, since the sign extension has a direct impact on the power consumption and the performance for the multiplier. Several sign extension techniques have been proposed as discussed in section 2.6. In this thesis, the sign-generate scheme has been chosen for generating partial products. This scheme can not only simplify the sign bit implementation, but also reduce the power dissipation due to that the extra "one" can be merged into the adder tree by using an increment adder that will be described later in this chapter.

The sign-generate scheme can be realized by using a block called sign generator. The truth table has been given in table 5.1. From the point of the simplification, this block can be implemented based on the partial product generator in which the signal A_{i-1} can be replaced by using A_i and adds an extra inverter at the output. In this way, the layout can almost reuse the layout of the partial product generator completely, and the block delay and power dissipation only have a slight increase due to the extra inverter. For this reason, this cell will not be presented in detail.

5.5.3.8 Implementing Adder Cells

Description

The adder cells consist of a full adder, half adder and increment adder. They are used to construct the partial product reduction tree together with 4-2 compressors that will be described later. These adders are implemented by using CPL. In general, a CPL gate is composed of an NMOS pass transistor logic network, a CMOS output inverter, two small pull-up PMOS transistor, and either single rail or dual rail inputs/outputs.

When the inputs for the adder cells and compressors come from the partial product generators, the single rail for the adder cells or compressors in the first level of the PPRT has to be used. In the second and further levels, the dual rail for the adder cells or compressors should be used. On the other hand, when the outputs from adder cells and the compressors feed into the final adder, the single rail has to be used.

The pull-up PMOS transistors are used for swing restoration, which can decrease the static power consumption. As long as the pull-up function can be realized, the width for the PMOS transistors can be taken to be as small as possible.

The Notation

The adder cells or compressors used for constructing the partial product reduction tree (PPRT) will be implemented by CPL. The differential characteristic of CPL leads to that the adder cells or compressors on the different positions in the PPRT could have different input and output styles (single rail or dual rail). For this reason, the adder cells and compressors are identified by defining a group of characters, such a special group of characters consists of the adder name, input and output styles which is used to specify whether single rail or dual rail is to be used. This group of characters is divided into three fields by the underscore "_". These fields can be explained as follows.

1. The first field represents the adder name that comprises one of the following three kinds of adders.

FA: full adder, HA: half adder, and IA: increment adder.

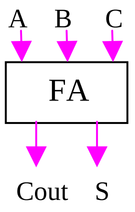
2. After an underscore, "_", the second field tells that the inputs are either single rail or dual rail. "s" stands for single rail, while "d" refers to dual rail.
3. There is another underscore, "_", after the second field. The third field indicates that the outputs are either single rail or dual rail.

For example:
FA_sss_dd, represents a full adder with three single rail inputs and two dual rail outputs where one is sum and the other is carry out.

The Logic Functions

The logic functions and the corresponding truth tables for the adder cells are illustrated in figure 5.9.

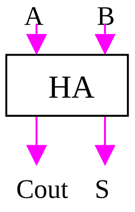
Full Adder



$$\begin{cases} S = A \oplus B \oplus C \\ Cout = AB + AC + BC \end{cases}$$

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

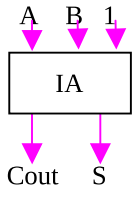
Half Adder



$$\begin{cases} S = A \oplus B \\ Cout = AB \end{cases}$$

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

Increment Adder



$$\begin{cases} S = \overline{A \oplus B} \\ Cout = A+B \end{cases}$$

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

Figure 5.9 Logic functions and truth tables for the adder cells

The Implementation

The full adder, half adder and increment adder, have been implemented using CPL. Figure 5.10 shows the schematic of the full adder.

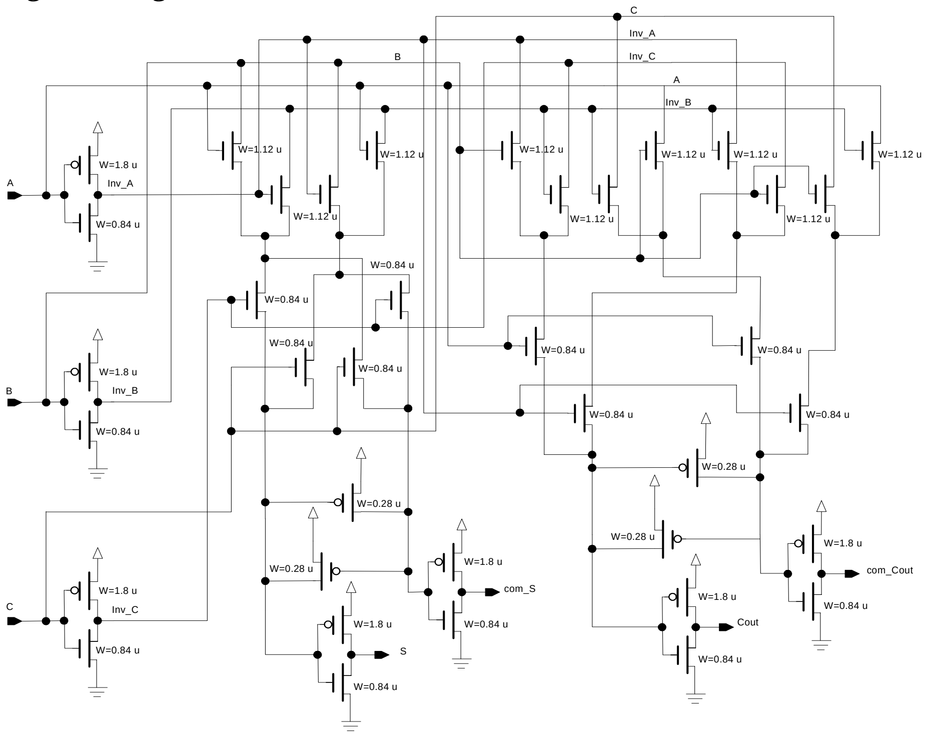


Figure 5.10 Schematic of the full adder

Based on these three kinds of adders, more than thirty basic cells have been constructed in both circuit and layout levels, which are used for the partial product reduction tree as compressors.

The Layout

Figure 5.11 on the next page shows the layout for the full adder. The area and the number of transistors for the full adder, half adder and increment adder will be given in table 5.2.

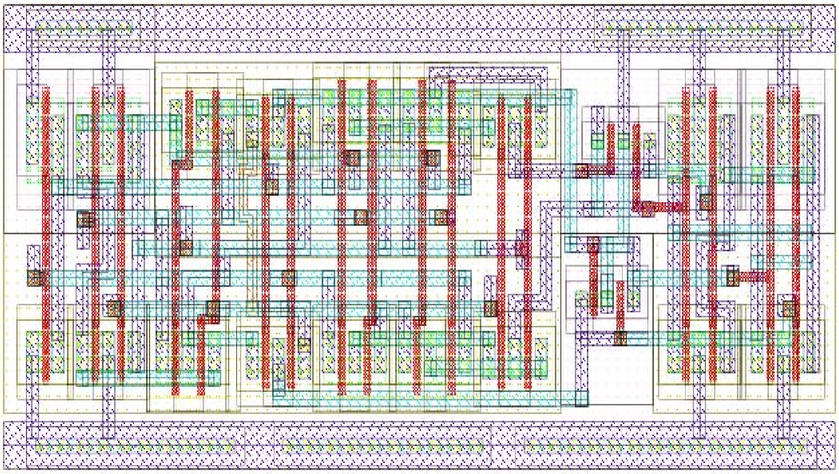


Figure 5.11 Layout of the full adder

The Simulation

The simulations for the adder cells have been done individually. The block delay and the power dissipation for each cell are given in table 5.2.

Table 5.2 Features for the adder cells (at 1.2 V, 25.6 MHz)

Cell Name	No. of trans	Area μm^2	Worst case delay (ps)	Average power Dissipation (μW)	Input style	Output style
FA	38	219	796	37	sss	dd
HA	24	151	457	25	ss	dd
IA	24	148	512	34	ss	dd

5.5.3.9 Implementing 4-2 compressors

The 4-2 compressors can be constructed by using two full adders that has been implemented according to the description above. The truth table and hierarchical schematic have been given in table 4.1 and figure 4.6. From the figure, the 4-2 compressors adds four partial products and a carry in signal, and then generates a sum signal and two carry signals (Co and Cout), where the Co is independent of the input Cin. In order to avoid using "zero" or "one" as the input for a 4-2 compressor, the second full adder in a 4-2 compressor can be replaced by using a half adder or an increment adder. Therefore, the 4-2 compressors can be built using different cell combinations as shown below.

FA_FA: two full adders used.
FA_HA: one full adder and one half adder used.
FA_IA: one full adder and one increment adder used.
IA_HA: one increment adder and one half adder used.

The Notation

Similarly, the 4-2 compressors can also be identified using the notation presented for the adders. The difference for the 4-2 compressors is that the group of the character has been divided into five fields, as explained as follows:

1. The first field consists of two adder's names, which are used to realize the compressors. There is an underscore, "_", between the two adder's names.
2. The second and third fields are the same as that of the adder cells.
3. The fourth field indicates that the intermediate carry out signal is either single rail or dual rail.
4. The fifth field stands for that the carry in signal is either single rail or dual rail.

For example:
FA_IA_ddd_dd_d_d, specifies a 4-2 compressor combined with a full adder and an increment adder. Such a 4-2 compressor has three dual rail input signals and one hidden input of "one", two dual rail outputs, one dual rail intermediate carry out signal, and one dual rail carry in signal.

The Layout

The layouts for the 4-2 compressors built with different adder cells have been done.

The Simulation

The post layout simulations for the 4-2 compressors designed in this thesis work have also been done. Some transistors in the compressors have been resized according to the simulation results. The propagation delay and the power dissipation can be obtained through simulation. For instance, the cell, FA_FA_ssss_dd_d_d, has 1.6 ns maximum delay and consumes 0.075 mW average power.

5.5.3.10 Implementing the Vector-Merging Adders

The vector-merging adder, or final adder, can be implemented using carry skip or carry look-ahead adders. In this thesis work, the vector-merging adders with the word-length more than 6 bits are constructed by using 3-bit and 4-bit CLA adders.

Figure 5.12 on the next page shows a 4-bit CMOS carry look-ahead generator circuit [9]. Such a circuit can simultaneously generate the carry signals due to that the generate signals G_i and the propagate signals P_i have been created in parallel. However, the drawback of this carry look-ahead circuit is that the delay for each bit will increase with increasing the number of bits. For this reason, another alternative has been proposed in this thesis, as can be seen in figure 5.13, where the difference is that the third and forth bit structures directly use the second carry out signal. In this way, the area of the circuits can be reduced substantially keeping the same speed. Moreover, the 4-bit carry look-ahead circuit becomes very simple and regular, which makes the layout easy. And the proposed structure in this thesis can be used to construct the carry look-ahead circuit block with a bit number of greater than 4 bits.

The sum generator has been implemented by using transmission gates since the truth and complement of the propagate signal were available, as shown in figure 5.13. This proposed structure in the thesis is a good choice for the sum generator due to only four transistors are used.

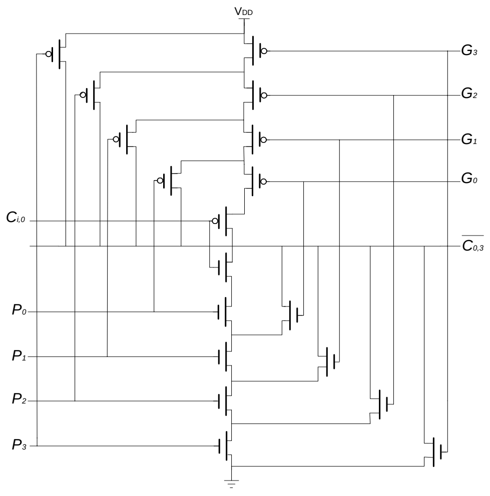


Figure 5.12 Schematic diagram of a 4-bit look-ahead adder

The Layout

The circuit structure for the 3-bit CLA is similar to that of the 4-bit carry look-ahead adders. Thus, figure 5.14 only shows the layout of the 4-bit CLA. The area and the number of transistors for the 3-bit and 4-bit CLA adders will be given in table 5.3.

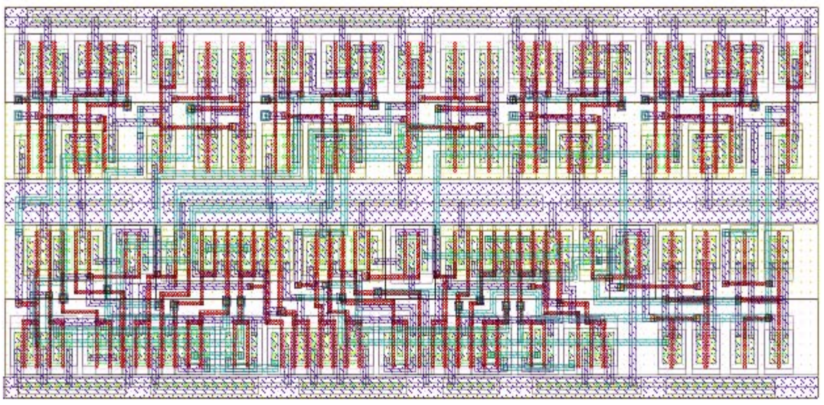


Figure 5.14 Layout of the 4-bit CLA

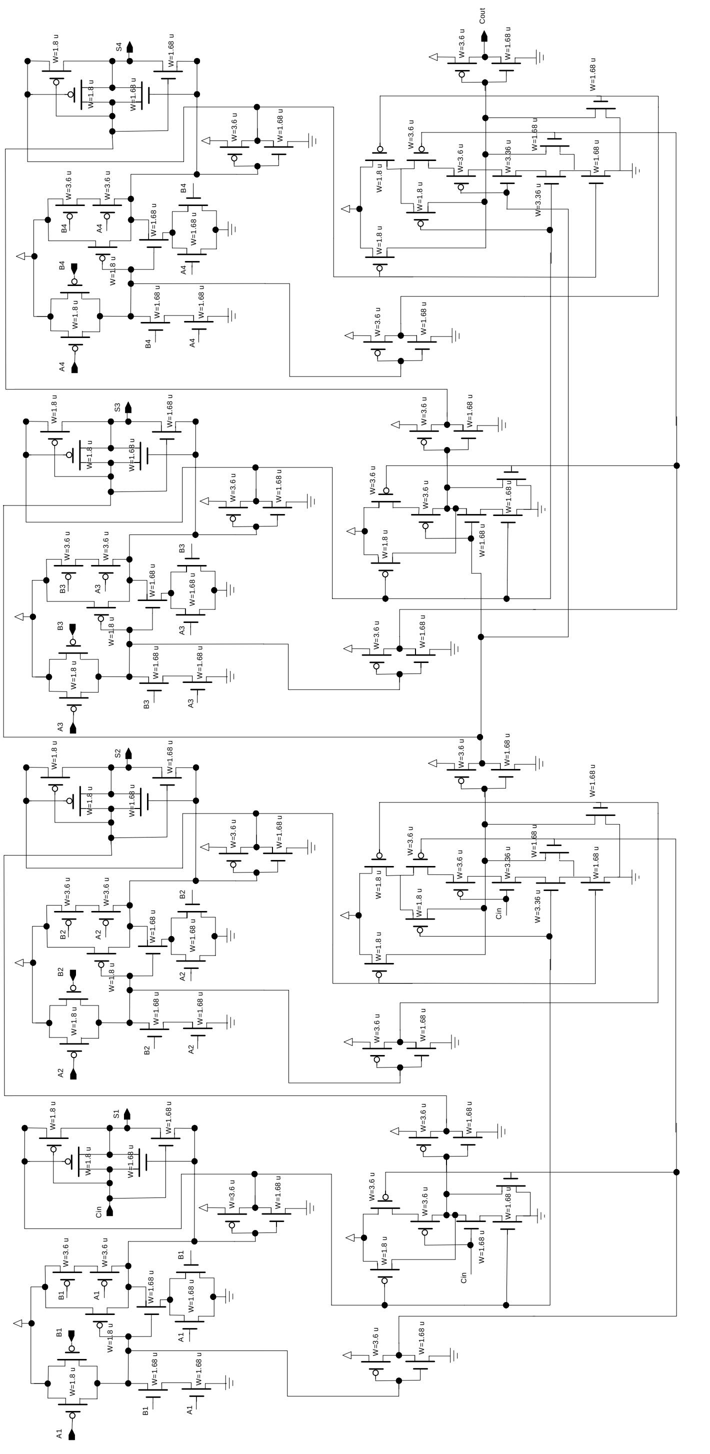


Figure 5.13 The proposed schematic of the 4-bit CLA

The Simulation

The simulations for the 3-bit and 4-bit CLA have been done. The delay and the power dissipation for both of cells under the worst case input pattern of $A_i = 0, B_i = 1,$ and $C_{in} = 1$ are given in table 5.3.

Table 5.3 Features of the vector merging adders (at 1.2 V, 25.6 MHz)

Cell Name	No.of trans	Area µm^2	Worst case delay (ps)	Power Dissipation (mW)
3-bit CLA	82	650	818	0.13
4-bit CLA	112	862	957	0.18

5.5.4 Cell-Level Schematic

The cell-level schematic for the 18×18-bit multiplier has been created. It looks like the gate-level schematic as shown in figure 5.2. However, the difference is that the cell level schematic has a hierarchical structure and the low level circuits have been available. Also, the functionality has been validated through the prelayout simulations.

5.6 Routing

When the cell-level schematic has been created and the layout for each leaf cell has been done, it is possible to start the routing that can be performed by using the automatic design tool called Cadence's ICcraftman Routing Tool.

The routing can be done according to the following steps.

1. Set the rules filling out the layer form.
2. Translate the design from the layout window to Virtuoso custom Router.
3. Import cells to be needed for constructing the whole multiplier. This step can be done using programs. Firstly, one program used reads each cell schematic view in a file according to the cell's instance names and coordinates. After this, another used C program sorts the file based on the magnitude of the coordinates for each instance.

Finally, another used SKILL program reads out each instance from the file and imports them to the router window.

4. Determine the location of all the cells and assign the interconnect areas within the flexible block, so that the total area used can be minimized.
5. Connect the power supply wires manually, metal2 may be used for GND, while metal3 for Vdd.
6. Add I/O pins. This can be performed by using a program that picks the I/O pins from the schematic view and then adds them to the corresponding points in the router window.
7. After design rule checking, if there is no design rule violations, the routing can automatically be performed using Cadence's ICcraftmen Routing Tool.
8. After the routing, the DRC check has to be done, and any design rule violations have to be fixed.
9. After the routing is complete, the exact length and position of interconnects for each cell have been determined, and the parasitic capacitance and resistance related to each interconnect can be calculated. This is done through a process called extraction.
10. Cadence provides two major design check functions, one is DRC, and the other is LVS. After the extraction, an electrical schematic is extracted from the physical layout. This schematic can be used to compare to the net list that has been generated from the cell-level schematic. Any discrepancy and design rule violation should be fixed.
11. After doing LVS check, the analog extraction can be performed, which can be used for simulation. By this time, the routing for the entire multiplier has been finished.

Following the above steps, the final layout of the 18×18 bit multiplier can be obtained, as shown in figure 5.15. The active area is 85000 μm^2.

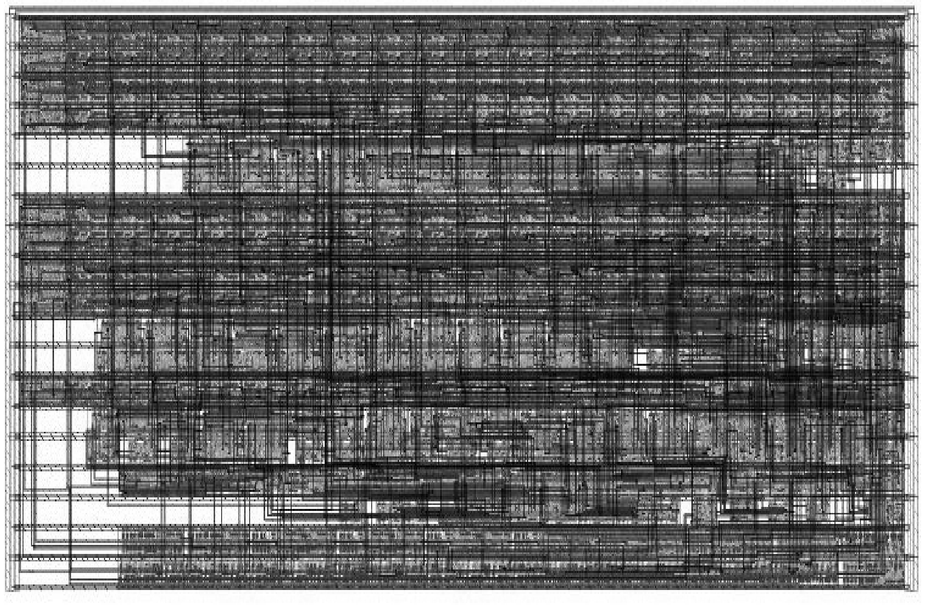


Figure 5.15 Final layout of the 18×18 bit multiplier

5.7 Simulation Strategy

After creating a configuration view, the full simulation of the multiplier has been tried. However, the full simulation for a complex circuit such as the 18×18-bit multiplier is not possible in Cadence. For this reason, the simplified simulation method proposed in [13] can be used to estimate the performance. The basic idea is to replace most of the cells by their equivalent input capacitance.

Based on the above idea, we only analyse the critical path of the 18×18-bit multiplier, but we use the actual input capacitances that can be back annotated from the extracted view of the final layout. The critical path, corresponding to the seventeenth weight, is shown in figure 5.16.

The cells on the critical path include the MBE encoder, PP generator, three levels of 4-2 compressors and six groups of the CLA adders from bit 14 to the position of bit 36. These cells can be divided into two sections, the first part consists of the MBE encoder, PP generator, three levels of 4-2 compressors, and the second part is the six groups of the CLA adders. The delay can be measured through the simulation with parasitics at 1.2 V. As a result, the former has a maximum delay of 13 ns, and the latter has the worst case delay of 4 ns. Therefore, the total critical path delay is 17 ns.

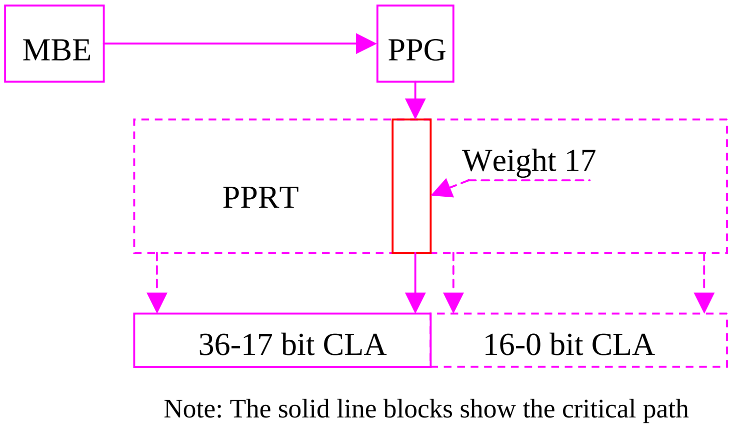


Figure 5.16 Critical path for the 18×18 multiplier

5.8 Summary

In this chapter, the design of a module generator for low power parallel multipliers was described in detail and the implementation of an 18 by 18 bit multiplier has also been performed successfully.

This module generator can be used both to redesign an existing multiplier when the process needs to be changed from one to another and to redesign an existing multiplier when the word length needs to be changed. Using this method for both cases, lots of design tasks could be saved.

In the former case, the logic design, circuit drawings and their validation can be reused, and the routing could be simplified by using two programs for automatically importing all the cells and adding the pins.

In the latter case, for the logic design, the existing cell's VHDL descriptions and their test benches as well as the structural test bench of the entire multiplier can be reused. Moreover, the structural schematic of the multiplier can be generated using a program with minimum manual intervention and the structural VHDL-code can also be created automatically using a program. The circuit design and the routing can be simplified as mentioned in the former case.

For the 4-bit CLA design, two new structures have been proposed in this thesis. One is for the 1-bit adder and the other is used for the carry look ahead circuit, which have been described in details in subsection 5.5.3.10.

This page is left blank on purpose.

Chapter 6 Conclusion

This chapter summarizes the results and gives the conclusions from the master thesis work. Also, some suggestions and the future possible improvements will be discussed.

6.1 Conclusions

The aim of this thesis is to design a module generator for the parallel multiplier and implement an 18×18 bit multiplier in 0.18 µm CMOS process. The generator could be used to automatically create the gate-level schematic for a fixed point two’s complement number parallel multiplier. Based on the generated schematic, the entire multiplier can be implemented with small manual intervention. This feature can reduce the time of chip design and is suited the case when the operand sizes and the process technology need to be changed.

To satisfy the demand on the changeable sizes, a number of basic library cells based on the different logic and circuit styles should be built. However, during this thesis work it is impossible to build so many basic cells. For implementing a larger multiplier, more basic library cells could need to be created.

This thesis work is relevant to the software programming skills, arithmetic algorithms, logic styles, topologies and power reduction techniques etc. The logic family styles have a large effect on the performance of the multiplier. Therefore, it is important to understand their advantages and drawbacks at the beginning of the design. The algorithm selection for implementing a parallel multiplier is also important, the modified Booth algorithm is the best choice for the multiplier with more than 16 bit sizes.

Moreover, to implement a low power high performance multiplier, all the aspects in the multiplier have to be optimized. Such a thesis work can provide a better understand for multiplier algorithms, logic styles and ASIC design technologies.

During this thesis the module generator (except one C program that still needs to be modified) has been developed and the method has been used to implement an 18×18 bit multiplier successfully.

The resulting 18×18 bit multiplier works correctly and the critical path delay satisfies the specification on the speed, but the full simulation for the whole multiplier was not performed due to Cadence inability to deal with such a complex circuit. For this reason, the total power consumption is not given.

6.2 Comments on the Project

The thesis project has been very challenging since it includes both hardware and software designs. Such a project would be too large to be finished under a required 20 weeks for one master student. For the hardware design, there are lot of tasks to be done. For example, more than thirty cells in circuit and layout levels have been designed both in 0.35 μm and 0.18 μm process technologies, which took much time for the functional validation, transistor sizing, layout and post layout simulation. In addition, many design decisions had to be made, such as what encoding scheme should be chosen? What logic and circuit styles should be used for different cells? However, the software design was even more challenging due to the complexity and the requirements on both aspects of the programming skill background and the comprehensive principle of the parallel multipliers. Therefore, this project should preferably be taken by two master students, one is responsible for the hardware design and the other is in charge of the software design.

6.3 Future Improvements

Some considerations should be done to further improve the module generator of the multiplier in future work.

1. The full simulation for the entire multiplier should be done, if possible.

2. The C program used to generate the gate-level schematic also needs to be modified for the section of the partial product reduction tree.

3. The total power dissipation should be measured using NanoSim or similar tools.

REFERENCES

[1] A. D. Booth, "A signed binary multiplication technique," Quarterly J. Mechan. Appl. Math., vol. IV, 1951.

[2] J. L. Hennessy and D. A. Patterson, "Computer organization & Design: The hardware/Software Interface," Second Edition, Morgan Kaufmann, 1998.

[3] O. L. MacSORLEY, "High-Speed Arithmetic in Binary Computers," Proc. IRE, Jan. 1961.

[4] R. Lin et al., "A Novel Self-Repairable Parallel Multiplier Architecture, Design and Test," IEEE Proc. On Asia-Pacific Conference, Aug. 2002.

[5] J. Fadavi-Ardekani, "$M \times N$ Booth Encoded Multiplier Generator Using Optimized Wallace Trees," IEEE, Trans, Very Large Scale Integration (VLSI) Systems, vol. 1. No.2, 1993.

[6] H. J. M. Veendrick, "Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits," IEEE Journal of Solid-State Circuits, Vol 19, Aug., 1984.

[7] M. Borah et al, "Minimizing power consumption of static CMOS circuits by Transistor sizing and input reordering," IEEE Proc. in VLSI Design' 95, Jan. 1995.

[8] A. P. Chandrakasan et al, "Minimizing Power consumption in Digital CMOS Circuts," IEEE Proc., Vol 83, No. 4, April, 1995.

[9] Jan M. Rabaey, "Digital Integrated Circuits: A Design perspective," Prentice Hall, Upper Saddle River, NJ, 1996.

[10] J. T. Kao et al, "Dual-Threshold Voltage Techniques for Low-Power Digital Circuits," IEEE Journal of Solid-State Circuits, Vol 35, No. 7, July, 2000.

[11] A. P. Chandrakasan et al, "Low-Power CMOS digital design," IEEE Journal of Solid-State Circuits, Vol 27, No. 4, April, 1992.

[12] K. Yano et al, “A 3.8 ns CMOS 16×16-b Multiplier Using Complementary Pass-Transistor Logic,” IEEE Journal of Solid-State Circuits, Vol 25, No. 2, April, 1990.

[13] I. S. Abu-Khater et al, “Circuit Techniques for CMOS Low-Power High-Performance Multipliers,” IEEE Journal of Solid-State Circuits, Vol 31, No. 10, October, 1996.

[14] R. Zimmermann et al, “Low-Power Logic Styles: CMOS Versus Pass-Transistor Logic,” IEEE Journal of Solid-State Circuits, Vol 32, No. 7, July, 1997.

[15] Henrik Eriksson, “Glitch-Power Analysis and Power-Efficient Design of CMOS Circuits,” Linkoping Studies in Science and Technology, Thesis No. 874, 2000. pp.10.

[16] Hwang-Cherng Chow and I. Wey, “A 3.3V 1GHz High Speed Pipelined Booth Multiplier,” IEEE Proc. Vol.5, May, 2003.

[17] C. S. Wallace, “A Suggestion for Fast Multiplier,” IEEE, Trans. Electric Computer, 1964.

[18] Pascal Bonatto et al, “Evaluation of Booth’s Algorithm for Implementation in Parallel Multipliers,” IEEE Proc. On Signals, Systems and Computers, Vol. 1, 1995.

[19] K. Prasad and K. K. Parhi, “Low-Power 4-2 and 5-2 Compressors,” IEEE Proc. Vol. 1, Nov. 2001.

[20] N. Ohkubo et al., “A 4.4 ns CMOS 54×54-b Multiplier Using Pass-Transistor Multiplexer,” IEEE, Journal of Solid-State Circuits, vol. 30, No. 3, March 1995.

[21] Wen-Chang Yeh et al., “High-Speed Booth Encoded Parallel Multiplier Design,” IEEE Trans. On Computers, vol. 49, No. 7, July 2000.

[22] J. Mori et al., “A 10-ns 54×54-b Parallel Structured Full Array Multiplier with 0.5 um CMOS Technology,” IEEE, Journal of Solid-State Circuits, Vol. 26, No. 4, April, 1991.

[23] S. Bhattacharya and Wei-Tek Tsai, “Area Efficient Binary Tree Layout,” Proc. IEEE 1991 First Lakes Symposium in VLSI, March 1991.

[24] D. Zuras and W. McAllister, “Balanced Delay Trees and Combinational Division in VLSI,” IEEE Journal of Solid-State Circuits, Vol SC-21, No.5, October, 1986.

[25] H. Sam et al, “A Generalized Multibit Recoding of Two’s Complement Binary Numbers and Its Proof with Application in Multiplier Implementations,” IEEE Transactions on Computers, Vol. 39, No.8, August, 1990.

[26] M. Vesterbacka, “A New Six-transistor CMOS XOR Circuit with Complementary Output,” IEEE Proc. On Circuits and Systems, Vol. 2, 8-11 Aug. 1999.

[27] R. Fried , “Minimizing Energy Dissipation in High-Speed Multipliers,” IEEE Proc. On Low Power Electrics and Design, Aug. 1997.

[28] Edwin de Angel et al, “Low Power parallel multipliers”, IEEE Proc. On VLSI Signal Processing, 30, Oct. 1, Nov. 1996.

[29] O.Salomon et al, “General Algorithms for a Simplified Addition of 2’s Complement Numbers,” IEEE Journal of Solid-State Circuits, vol. 30, No. 7, July 1995.

[30] L. P. Rubinfield, “A Proof of the Modified Booth’s Algorithm for Multiplication,” IEEE, Trans. Computers, Oct. 1975.

[31] M. J. Flynn, “EE486 lecture 9: Multiply,” Winter 00-01, Stanford University.

[32] Hesham Abdulaziz Al-Twaijry, “Area and Performance Optimized CMOS Multipliers,” Ph. D. Thesis, Stanford University, August 1997.