UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Contemporary data path design optimization

Permalink

https://escholarship.org/uc/item/6xv8j2nx

Author

Liu, Jianhua

Publication Date

2006

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA, SAN DIEGO

Contemporary Data Path Design Optimization

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in
Computer Science

by

Jianhua Liu

Committee in charge:

Professor Chung-Kuan Cheng, Chair
Professor Paul Chau
Professor Fan Chung Graham
Professor Russell Impagliazzo
Professor Tajana Rosing

2006

.

Copyright
Jianhua Liu, 2006
All rights reserved.

The dissertation of Jianhua Liu is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2006

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Professor Chung-Kuan Cheng for his support and for his confidence in me and my work. I would also like to thank Professor John Lillis of University of Illinois at Chicago, Professor David Harris of Harvey Mudd College, Professor Borivoje Nikolic of Berkeley and Mr. Mike Hutton of Altera. Their advices on my research work are also very important to me.

I wish to thank my dissertation committee members, Professor Russell Impagliazzo, Professor Fan Chung Graham, Professor Paul Chau and Professor Tajana Rosing for technical discussions and their advices and reviews of this dissertation.

I am grateful to all the graduate students in the UCSD VLSI CAD group for making the group a friendly and fun place to work. Among them, special thanks to Shuo Zhou, Rui Shi, He Peng, Haikun Zhu, Yi Zhu, Ling Zhang and many others.

Finally my special thanks go to my parents and my girl friend, Sai Ma, for their support during my education and for their understanding and tolerance during the last couple of years.

Chapter V has been submitted for publication of the material as it appears in ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays 2006, Liu, Jianhua; Zhu, Haikun; Cheng, Chung-Kuan. The dissertation author was the primary investigator and single author of this paper.

VITA

1999	B.E. in Computer Science and Technology Tsinghua University, Beijing, P.R. China
2001	M.S. in Computer Science and Technology Tsinghua University, Beijing, P.R. China
2006	Ph.D. in Computer Science University of California, San Diego

PUBLICATIONS

Jianhua Liu, Haikun Zhu, C.K. Cheng, An Iterative Division Algorithm for FPGAs, ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays 2006

Jianhua Liu, Michael Cheng, C.K. Cheng, John MacDonald, N.C. Chou, Peter Suaris, Fast Adders in Modern FPGAs, poster session of ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays 2004

Jianhua Liu, Shuo Zhou, Haikun Zhu, C.K. Cheng, An Algorithmic Approach for Generic Parallel Adders, IEEE/ACM International Conference on Computer Aided Design 2003

Jianhua Liu, Shuo Zhou, Haikun Zhu, K.T. Tseng, C.K. Cheng, Optimal Parallel-Prefix Adders using a Dynamic Programming Algorithm, International Workshop on Logic and Synthesis 2003

FIELDS OF STUDY

Major Field: Computer Science
Studies in VLSI CAD
Professor Chung-Kuan Cheng

ABSTRACT OF THE DISSERTATION

Contemporary Data Path Design Optimization

by

Jianhua Liu
Doctor of Philosophy in Computer Science
University of California, San Diego, 2006
Professor Chung-Kuan Cheng, Chair

As the core of most digital computing systems, data-path design is essential to determine the whole system performance. In the past decades many advanced methodologies and technologies have been proposed to optimize data-path designs. Nowadays new design requirements are emerging with the technology development:

- Low power design. Power consumption becomes more critical issue than performance in modern data-path designs, especially for mobile device applications.
- Extremely high performance design for micro processors. With the shrink of feature size and the increase of clock frequency, extremely high performance data-path components operated on multiple giga-hertz clock are required in micro processor designs.
- High performance low cost design for ASIC. Application-specific design constraints, such as area/power budget and non-uniform signal required times, must be satisfied.

Inspired by these requirements, we propose two optimization techniques to efficiently minimize power consumption and achieve timing/area/power tradeoff for specific applications.

1) Binary addition is the most widely used fundamental operations. Various applications require different adder designs for high speed, small area or low power consumption. Since parallel prefix adders provide great flexibility to satisfy a specific application, we propose an integer linear programming method to build optimal prefix adders, which counts gate and wire capacitances in the timing and power models. Furthermore the proposed method can handle nonuniform arrival time and required time on each bit position. Therefore, a realistic minimal-power prefix adder can be found with arbitrary timing and area constraints.

2) Division is a fundamental but expensive arithmetic operation. We analyze the computation efforts from memory, arithmetic functions and iterations in division operation and propose a hybrid algorithm which employs Prescaling, Series expansion and Taylor expansion (PST) algorithms together. The proposed algorithm boosts very-high radix division by efficiently estimating the reciprocal of divisor, and achieves outstanding performance-cost tradeoff. Optimizations of the basic PST algorithm are also developed to improve the performance further. The proposed algorithm is suitable for both ASIC and FPGA applications with high-performance division units.

These research works optimize data-path designs in different levels from algorithm to logical/physical synthesis. Unlike the previous works, both approaches proposed in the dissertation explore the design space in terms of timing, area and power consumption.

I

Introduction

I.A Introduction

Computer hardware has experienced the most dramatic improvement in capabilities and costs in its short 50-years. Nowadays computing devices are part of everyday applications. They are not only the microprocessors in computers but also digital signal processors (DSP) and other application-specific integrated circuits (ASIC). With the increasing demand from various applications and the rapid advances in integration technologies, large-scale computing system (VLSI) design becomes to a very promising but challenging topic.

Fig. I.1 shows the typical VLSI design flow. It starts from a behavior description. The behavior description is then transformed to a structure description which includes functional modules, such as registers and arithmetic logic units (ALU), and their interconnections. The geometrical layout on the chip is determined afterwards. Individual functional modules are implemented as leaf cells. Physical design including detail placement and routing is performed next. Although the design flow has been described in linear fashion for simplicity, in reality there are many iterations back and forth, especially between any two neighboring steps. Both top-down and bottom-up approaches have to be combined. It is very important to feed forward low-level information to higher levels (bottom up) as

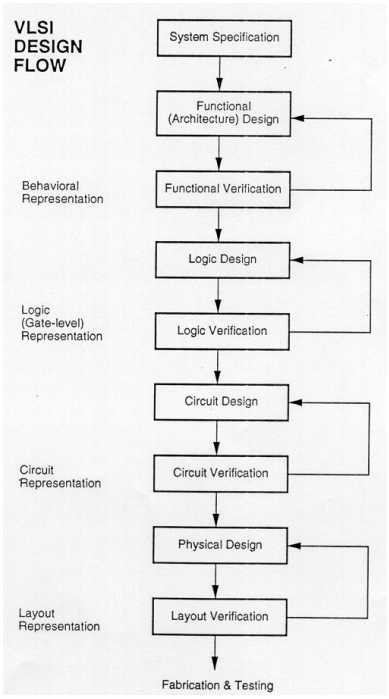

Figure I.1 VLSI Design Flow

early as possible.

Each computing system can be functionally partitioned to data-path and accessorial control logic. Data-path contains data storage components and data processing components. In most computing systems, data-path dominates the whole system performance and cost. For example, Fig. I.2 demonstrate the layout of Intel Pentium microprocessor. Around 80% area is occupied by data-path components. Therefore data-path design, especially the arithmetic logic unit design is extremely important.

I.B Computing Arithmetic

Computing algorithms are various for different applications. Some algorithms are complicated such as Discrete Cosine Transformation (DCT) or COordinate Rotation DIgital Computer (CORDIC). However almost all the algorithms are composed by four fundamental arithmetic operations: addition, subtraction,

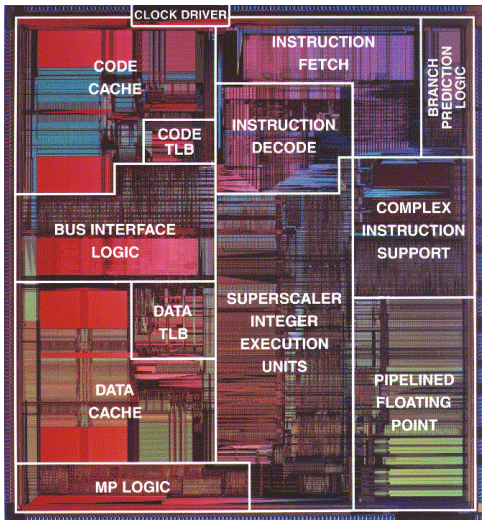

Figure I.2 Layout of Pentium Microprocessor

multiplication and division. IEEE standard 754 [1] defines formats for representing numbers with the four arithmetic operations that operate on these values, and is followed by many microprocessor implementations.

Binary addition is the most fundamental operation. It is widely used in more complex operations like multiplication and division. Also it is the base of subtraction, increamentation and magnitude comparison. Therefore the performance of binary adder is very critical. However the binary addition problem involves an slow carry-propagation step, the evaluation time of which is dependent on the operand word length. The efficient implementation of the addition operation in an integrated circuit is a key problem in VLSI design.

Multiplication can be considered as multiple-operands addition. It can take advantage of the efficient carry-save addition instead of carry-propagation. Modern multiplier architectures use Wallace trees [2] to add the partial products together in a single cycle. The performance of the Wallace tree implementation is sometimes improved by Booth encoding [3] [4] one of the two multiplicands, which reduces the number of partial products that must be summed.

Division is the inverse of multiplication. Compared with the three other fundamental operations, division is much more expensive in terms of both timing

and area. It is due to the interleaving dependency between partial quotients and partial remainder. Existing division algorithms fall into two main categories: high-radix division and very-high-radix division. Today high-radix division becomes the performance bottleneck of modern microprocessors. Therefore, how to implement very-high-radix division and reduce the hardware overhead turns to an important research problem.

I.C Challenges on Data-path Design

The technology development of VLSI is mainly driven by the steady decrease of feature size. The scaling effect makes the transistor density increasing exponentially. It validates the Moore's law, number of transistors per integrated circuit doubles every two years, as shown in Fig. I.3. However the scaling effect also bring several challenges on data-path design.

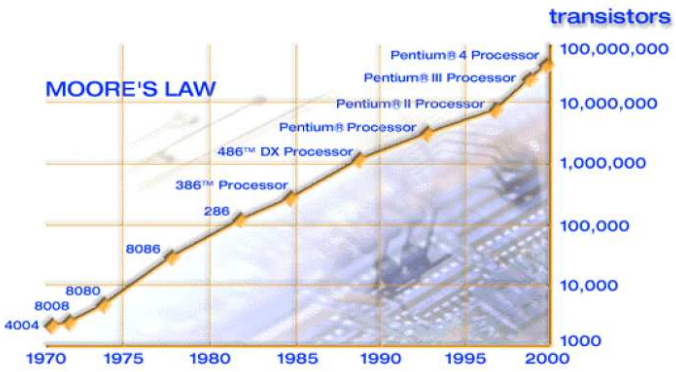

Figure I.3 Moore's Law

1. Power Consumption. With the increasing transistor density and clock frequency, power consumption is also enlarged exponentially. Fig. I.4 shows the power consumption change with feature size shrinking on Intel microprocessors. Nowadays power consumption, instead of performance, becomes the most critical concern in many data-path designs, especially for mobile device applications. Accordingly power optimizations are required on every

design step. For data-path design, algorithm level and logic synthesis level optimization algorithms need to be revised or developed to minimize power.

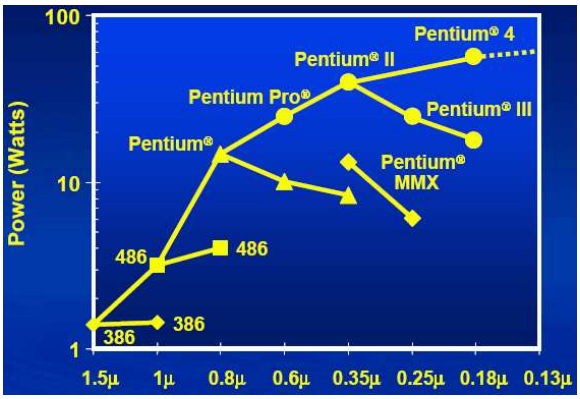

Figure I.4 Scaling Effect on Power Consumption

2. Physical Synthesis. Gate delay is used to dominate the system performance. However with the decrease of feature size, interconnect delay becomes more and more significant. When feature size is smaller than 0.25um, interconnect delay is comparable, and in some cases much greater than gate delay, as shown in Fig. I.5. This shift enlarges the gap between logical design and physical design. To compensate the gap, physical information must be considered in logic synthesis step. Therefore, physical synthesis is necessary, to merge logic synthesis with physical design.

3. Advanced CAD tools. High transistor density enables ultra large scale integrated circuits. It makes the design complexity extremely huge. On the other hand, the demand of ASIC designs is rising fast. Manual design is not capable to produce high quality designs in short period. The burden on efficient CAD tools to support the high-performance and large scale data-path design is bigger than ever.

We will addressed on these challenges in the dissertation.

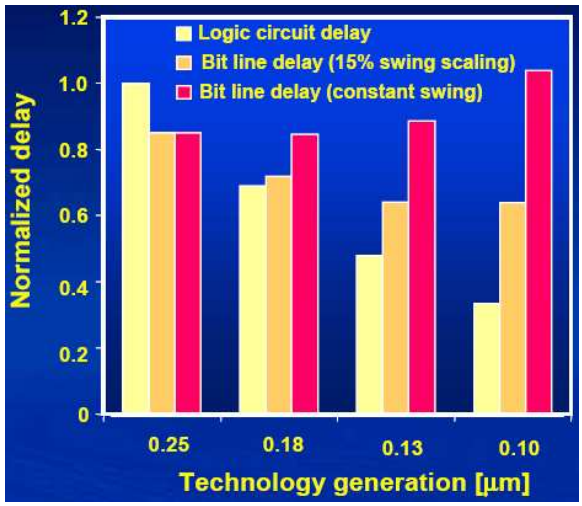

Figure I.5 Scaling Effect on Gate and Wire Delay

I.D Dissertation Overview

Inspired by the challenges appearing on data-path, we propose a power efficient algorithm for the high performance division and a power-oriented physical synthesis approach on binary addition.

Chapter II introduces the basic addition principles and structures. The existing binary adder structures including ripple-carry, carry-skip, carry-select adders and parallel prefix adder. Among these adders, parallel prefix addition provides the maximal flexibility. Several classical regular and irregular prefix structures are also discussed in this chapter.

Chapter III first presents the comprehensive timing/area/power model used in the proposed method, which involves gate capacitance as well as wire capacitance based on physical layout. An Integer Linear Programming (ILP) approach is then proposed to optimize power consumption for prefix adders and provide good design flexibility.

Chapter IV introduces the previous division algorithms including non-restoring division, SRT division, Prescaling division in high-radix division category, and Taylor expansion division, series expansion division in very-high-radix division

category. Furthermore, we will propose the concepts of iteration effort, memory effort and arithmetic effort, and use them in the analysis of previous algorithms.

In Chapter V, we present an hybrid division algorithm which combines Prescaling, Series expansion and Taylor expansion algorithm together. It is so called PST division. The proposed algorithm efficiently utilizes the strength of iteration effort, memory effort and arithmetic effort to achieve high performance and low power. It can be applied on both ASIC and FPGA applications. Especially for FPGA, it can fully take advantage of built-in memory and multiplier blocks.

The final chapter concludes with a brief summary of the contributions made in this dissertation.

II

Binary Addition Background

This chapter introduces the definition of binary addition and popular binary adders. Especially we will address on prefix adders following the concept of prefix addition. Several classical prefix structure will be reviewed.

II.A Binary Addition Basic

Binary addition is the simplest arithmetic operation. The binary addition problem is defined as follows: given an n-bit augend A, an n-bit addend B, and a 1-bit carry-in c_0, generate the n-bit sum S and the 1-bit carry-out c_n. Suppose $A = a_{n-1} \dots a_1 a_0$ and $B = b_{n-1} \dots b_1 b_0$, we define s_i and c_i as:

$$s_i = a_i \oplus b_i \oplus c_i \tag{II.1}$$

$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i \tag{II.2}$$

Following the definition of binary addition, ripple-carry adder composes a combinational circuit using n full-adders connected in series. Each full adder performs single-bit addition according to the previous equations. Fig.II.1 shows the structures of full-adder and ripple-carry adder.

The critical path of ripple carry adder is from the first bit through the carry propagate chain to the last bit. Therefore, the timing complexity of ripple-carry adder is n. The shortage of ripple-carry adder is its poor performance. The

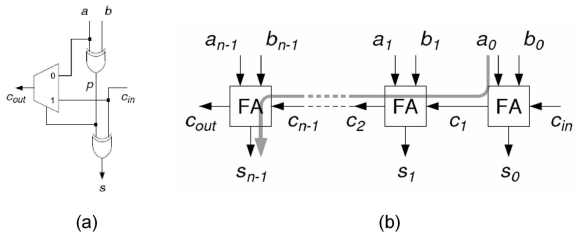

Figure II.1 (a)Full Adder (b) Ripple-Carry Adder

Manchester carry chain [5] is a circuit-optimized implementation of a ripple-carry adder. By using serially connected high-speed pass transistor, a Manchester carry chain reduces the carry propagation delay to n transistors.

To reduce the logic depth further, the carry-skip and carry-select adders were developed as two group-based binary adders accelerating the carry propagation across each group. The carry-skip adder [6] computes a group propagate signal. The carry-out of a group can be directly calculated from the product of the carry-in and the group propagate signal, as shown in Fig.II.2. Hence, the logic depth of an optimal carry-skip adder [7] decreases to $O(\sqrt{n})$.

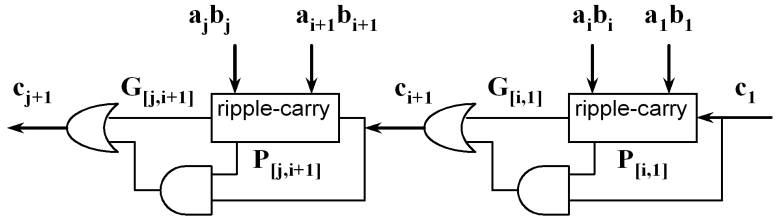

Figure II.2 Carry-Skip Adder

Different from carry-skip adders, the concept behind carry-select adders [8] is that two parallel additions on a group are performed for the carry-in equal to 0 and 1. When the true carry-in is obtained, the correct result is selected, as shown in Fig.II.3. The conditional-sum addition [9] introduces a binary selection tree to determine the complete sum. The tree topology helps to reduce the logic depth to $O(\log n)$.

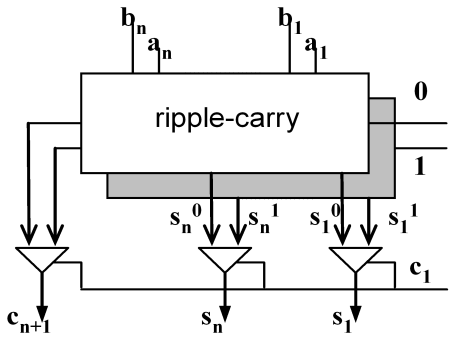

Figure II.3 Carry-Select Adder

II.B Parallel Prefix Addition

The carry lookahead adder [10]is probably the best-known adder implementation. It was proposed in 1956, and formulated as parallel prefix computation [11] in 1980. The formula provides a simple expression as well as great flexibility.

II.B.1 Prefix Computation

Ladner and Fischer defined the prefix problem as: Let $\bullet$ be an associative operation on n inputs $x_1, \ldots, x_n$, to compute each of products $x_1 \bullet x_2 \bullet \ldots \bullet x_k, 1 \leq k \leq n$. In the application of binary addition, the input of prefix computation is a group of binary vectors with two domains g_i (*generate*) and p_i(*propagate*):

$$g_i = \begin{cases} c_0, & if \text{ i=0} \\ a_i b_i, & otherwise \end{cases} \tag{II.3}$$

$$p_i = \begin{cases} 0, & if \text{ i=0} \\ a_i \oplus b_i, & otherwise \end{cases} \tag{II.4}$$

(Pre-processing)

If g_i equals 1, a carry is generated at bit i; otherwise if p_i equals 1, a carry is propagated through bit i. By prefix computation, the concept of generate and

propagate can be extended to multiple bits. We define $G[i:k]$ and $P[i:k]$ $(i \geq k)$ as:

$$G_{[i:k]} = \begin{cases} g_i, & if \text{ i=k} \\ G_{[i:k]} + P_{[i:j]}G_{[j-1:k]}, & otherwise \end{cases} \quad \text{(II.5)}$$

$$P_{[i:k]} = \begin{cases} p_i, & if \text{ i=k} \\ P_{[i:j]}P_{[j-1:k]}, & otherwise \end{cases} \quad \text{(II.6)}$$

(Prefix computation)

To simplify the representation, we continue to use the same operator to denote the prefix computation on (G, P):

$$(G, P)_{[i:k]} = (G, P)_{[i:j]} \bullet (G, P)_{[j-1:k]} \quad \text{(II.7)}$$

The width of the (G, P) term is calculated by $i - k + 1$. For final outputs, s_i and c_i can be generated from G and P:

$$c_i = G_{[i:0]} \quad \text{(II.8)}$$

$$s_i = p_i \oplus c_{i-1} \quad \text{(II.9)}$$

(Post-processing)

Since pre-processing and post-processing have constant delay, prefix computation becomes the core of prefix adders and dominates the performance. A visual representation of prefix computation structures is to use directed acyclic graphs. Fig.II.4 shows some basic structures in [11].

For (G, P) computation in binary addition problem, it has two important properties:

- **Property 1:** (G, P) **computation is associative.** That is

$$\begin{aligned} (G, P)_{[i:k]} &= (G, P)_{[i:j]} \bullet (G, P)_{[j-1:k]} \\ &= (G, P)_{[i:l]} \bullet (G, P)_{[l-1:k]}, \quad i \geq l, j > k \end{aligned} \quad \text{(II.10)}$$

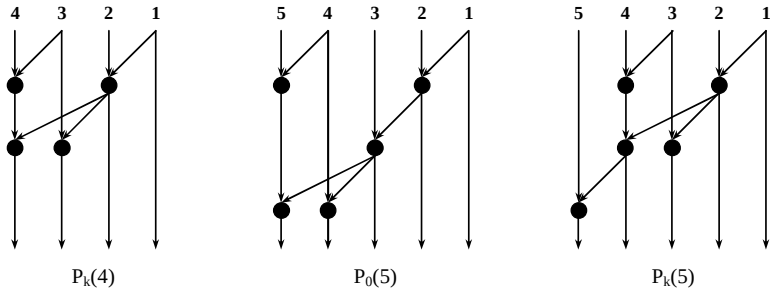

Figure II.4 Directed Acyclic Graphs of Prefix Structures

- **Property 2:** (G, P) **computation is idempotent.** That is

$$\begin{aligned}(G,P)_{[i:k]} &= (G,P)_{[i:j]} \bullet (G,P)_{[j-1:k]} \\ &= (G,P)_{[i:j]} \bullet (G,P)_{[l:k]}, \quad i \geq l > j-1 \geq k \end{aligned} \tag{II.11}$$

These two properties limit the design space of parallel prefix adders. That is the solution space of (G, P) computation covers every tree-like structures defined under bit width n.

II.B.2 Regular Prefix Adders

Ladner and Fischer proposed an algorithm to construct a prefix adder after the definition of the prefix problem. Given the bit width n and depth requirement $T \geq \lceil \log n \rceil = M$, the algorithm constructs a prefix structure recursively. Assuming $C = T - M$, which is referred as extra depth, Fig.II.5 presents the recurrence:

Ladner and Fischer's method not only provides a possible way to achieve minimal depth, but also establishes a depth-area tradeoff for the first time. The upper bound of area is:

$$A_{P_C(n)} < 2(1 + \frac{1}{2^C})n - 2, n \geq 1 \tag{II.12}$$

Their ideas of recursively divide-and-conquer, half-half bipartition and odd-even bipartition are very inspiring. Many following works applied similar ideas.

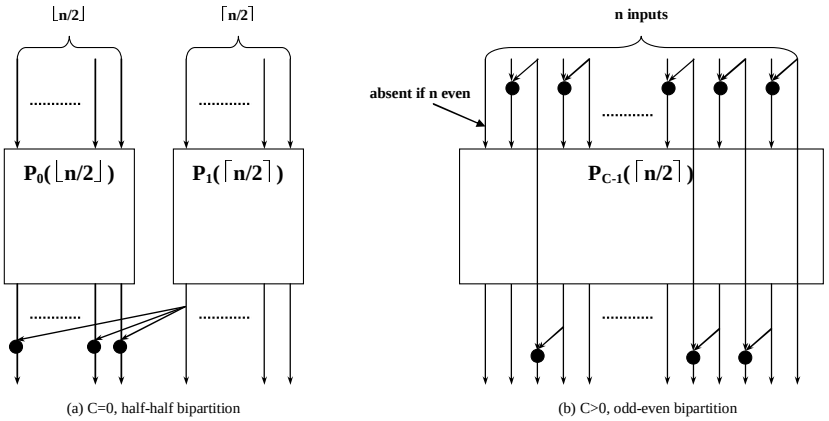

Figure II.5 The Construction of $P_C(n)$

The Kogge-Stone adder [12] is another classic prefix structure that reaches minimal depth of $\log n$. However, it utilizes a different idea instead of half-half bipartition. In each level l, The Kogge-Stone adder constructs all $(G, P)s$ as wide as possible. Therefore, any (G, P) with width equal to or smaller than 2^l will be achieved. After $\log n$ levels, the prefix structure generates all needed outputs. A 8-bit Kogge-Stone structure is shown in Fig.II.6(a). One shortage of the Kogge-stone adder is a large area overhead, which is $n * \log n - n + 1$. It comes from the extremely greedy algorithm. The algorithm can be understood as calculating $(G, P)s$ for every bit position separately, and no share between them. However, a positive effect is that the fanout of each node is limited to 2. In contrast, the max fanout increases exponentially in half-half bipartition algorithms.

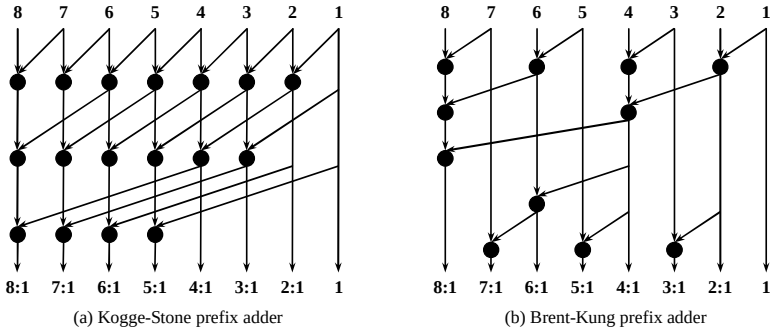

Figure II.6 Kogge-Stone and Brent-Kung Prefix Adders

The Brent-Kung adder [13] is an area-effective adder proposed in 1982. It builds a complete binary tree for the most significant bit first, and then adds at most one (G, P) addition for each unvalued bits. Thus the area is reduced to $2n - \log n - 2$, and the depth is kept within the same order, which is $2 * \log n$. Fig.II.6(b) shows the diagram representations of an 8-bit Brent-Kung adder. If looking at the structure carefully, we can find that the Brent-Kung adder is a special case of Ladner-Fischer structures when the extra depth C equals $\log n$. In other words, Brent-Kung structures are built from the pure odd-even bipartition algorithm.

Very similar to the tradeoff in Ladner-Fischer structures, a depth-area [14] tradeoff exists between the Kogge-Stone and Brent-Kung adders. It is a combination of odd-even bipartition and greedy algorithms. The odd-even bipartition algorithm takes advantage of the extra depth to reduce the problem to $\frac{2^M}{2^C}$, and the greedy algorithm builds the core with minimal depth of $M - C$. Fig.II.7 shows a 16 bits prefix structure with depth 6. It spends two more levels compared with the minimal depth Kogge-Stone adder, but reduces the area from 49 to 27. In general, the area is bounded by $2^{M-C} * (2^{C+1} + M - C - 3) + 2^C - C$. And the max fanout is limited by the extra depth C.

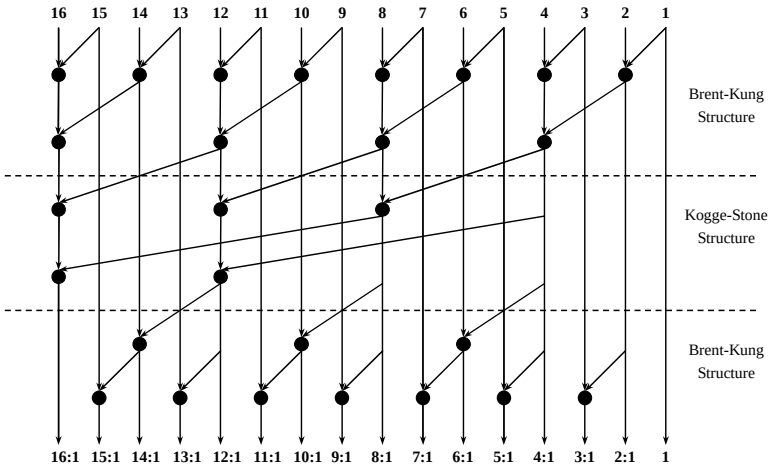

Figure II.7 Tradeoff between Kogge-Stone and Brent-Kung Adders

All algorithms introduced so far have depth of $O(\log n)$ with various area

and fanout overheads. These constructive methods are very effective when the signal arrival profile is uniform. However, their shortage is that they are not flexible enough to explore the whole solution space and find an optimized solution for a specific application.

II.B.3 Irregular Prefix Adders

John Fishburn [15] demonstrated a process to convert a ripple-carry adder into a carry-lookahead adder, which gives a first view of the solution space of prefix computation structure. The original ripple prefix structure has n depth. It corresponds to a ripple-carry adder in topology, as shown in fig.II.8.

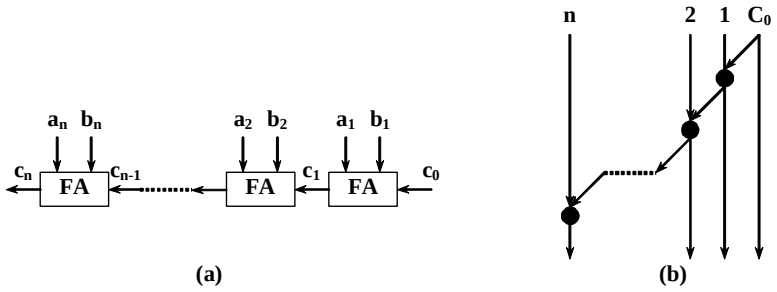

Figure II.8 Transform to Ripple-Carry Adder

A simple local operation is performed iteratively to reduce the logic depth. The operation called peephole transformation is shown in Fig.**??**. The overall algorithm is in a greedy fashion: peephole transformation is iteratively performed on the current critical path, until depth requirement is satisfied. The final structure is very close to a Ladner-Fischer adder. Fig.**??** illustrates a resulting structure with depth of $2 \log n$.

Fishburn's work shows an optimize procedure to decrease depth according to the current situation. More importantly, a powerful tool, peephole transformation, is introduced to explore the solution space. However, its greedy heuristic algorithm limits the scope of solution searching and cannot guarantee to find the optimal solution. Furthermore, although Fishburn showed that ripple-carry adder can be covered by prefix structure topologically, he didn't explicitly indicate the

positions of other conventional adders in the solution space.

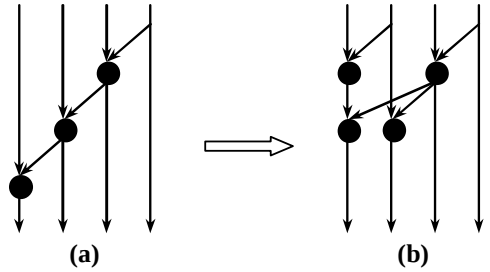

Figure II.9 Peephole Transformation

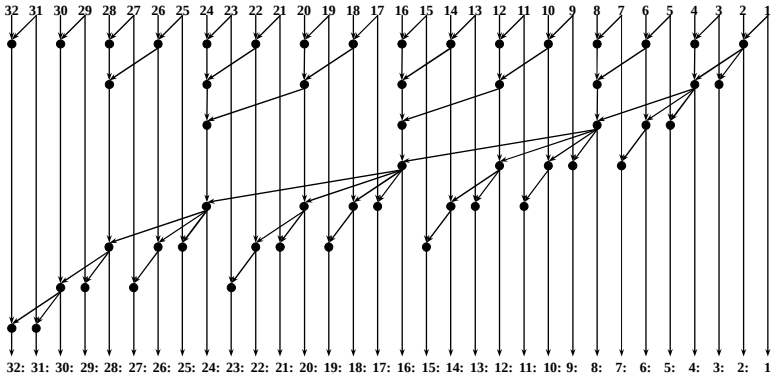

Figure II.10 Fishburn's Result

Reto Zimmermann [16] completed Fishburn's work on 1996. Zimmermann presented the solution space of prefix structure covers all the tree structure including some conventional adders: ripple-carry adders, carry-select adders, and conditional sum adders (also known as Sklansky adders). Fig.II.11 shows the prefix structures corresponding to a carry-select adder, a multi-level carry-select adder and a conditional sum adder.

Zimmermann furthermore extended the manipulating operation by adding the inverse of peephole optimization. These two operation plus a non-heuristic algorithm can fully probe all feasible tree structure as well as potential area-time tradeoffs. The algorithm compresses a prefix graph to obtain a faster implementation, and then expands the prefix graph to reduce area. This approach benefits from the high flexibility, and get good results for many specific environments, such

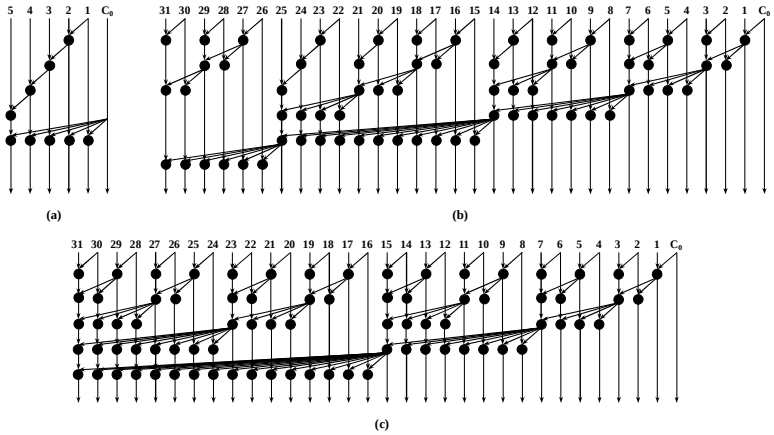

Figure II.11 (a)Carry-Select (b)Multi-Level Carry-Select and (c) Sklansky

as non-uniform signal arrival profile and non-uniform output requirement profile. The only weakness is that the algorithm doesn't guarantee the resulting structure can achieve minimal delay under non-uniform signal arrival profile. The expansion procedure to reduce area is a kind of greedy algorithm, which converts most significant bits into ripple structure, as shown in Fig.II.12. The algorithm is effective, but may not reach the minimal area. The minimal area for arbitrary bit width and required logic depth is still unknown.

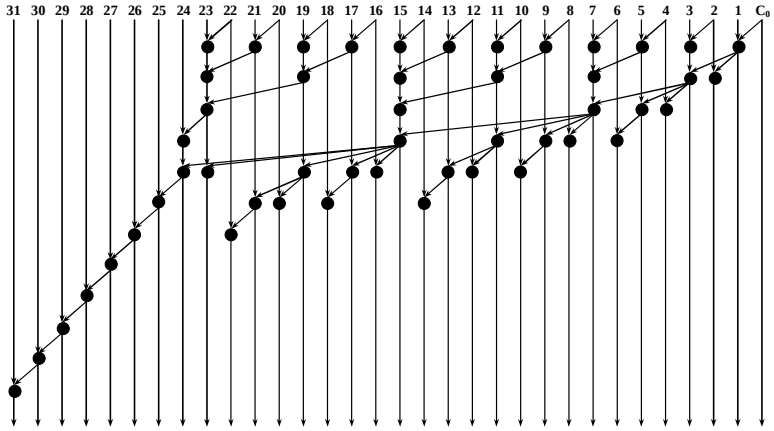

Figure II.12 Greedy Expansion

II.C Binary Addition Summary

Among existing binary adders, parallel prefix adder provide the maximal flexibility to fit various design requirements, from high performance to low cost. Parallel prefix adder can cover ripple-carry adder, carry-skip adder and carry-select adder in term of topology. Prefix adders can be classified into two categories: regular structure and irregular structure. In regular structure category, the Kogge-Stone adder and the Brent-Kung adder have the lower bound of logic depth and the lower bound of area respectively. In practice, Kogge-Stone adder is considered as the fastest regular prefix adder. However, regular structure cannot handle non-uniform input arrival times or output required times. In this case, irregular structure can be produced by manipulating a prefix structure with local operations. The method proposed by Fishburn and Zimmermann is very effective in practical ASIC designs.

Although the design space of prefix adder seems to have been well-studied, the common timing/area model in these previous works is still idealistic. The concept of logic levels is used to estimate timing and even input arrival time. However, the real delay is not proportional to the number of logic levels. It highly depends on gate size and total load capacitance including both gate capacitance and wire capacitance. With technology development, wire load capacitance becomes more and more significant, and makes more impact on the idealistic model. For area, using the total number of elements as the estimation of area does not considered the bit-slice placement for data-path. The width of a prefix adder is decided by the operand bit-width and the depth is actually limited by the maximal number of elements in each bit-slice. Furthermore, power consumption becomes an important design issue which is not studied in previous works. These changes demand a comprehensive timing/area/power model and a new methodology for prefix adder synthesis, to reduce the gap between high-level synthesis and back-end design. Therefore, the new model should incorporate physical placement to accurately es-

timate area and wire capacitance, count both gate and wire capacitance in timing and power estimation, and consider static power and dynamic power with activity probability. On the other hand, a methodology based on the new model should have the capabilities to handle non-uniform input arrival times and output required times and even perform gate sizing and buffer insertion.

III

ILP on Prefix Addition

In this chapter, we will propose an Integer Linear Programming method to construct minimal power prefix adder with comprehensive area/timing/power model. The ILP method is also extended to support gate sizing, buffer insertion and hierarchical design.

III.A Area/Timing/Power Model

In the new area model, we will consider physical placement instead of using logical structure as physical layout. Based on the physical placement, both gate capacitance and wire capacitance can be well estimated. It enables the accurate linear timing model as well as the accurate power model including both dynamic and static power consumption.

III.A.1 Area Model

As a datapath component, prefix adder will keep the bit-slice structure in the placement, which is consistent with the logical structure. However, with each column each GP adder can take advantage of any empty space in the same bit-slice. When all the empty bubbles are below the GP adders in one bit-slice, it's called "compact placement". Fig.III.1 shows the compact placement of the 8-bit Brent-Kung adder.

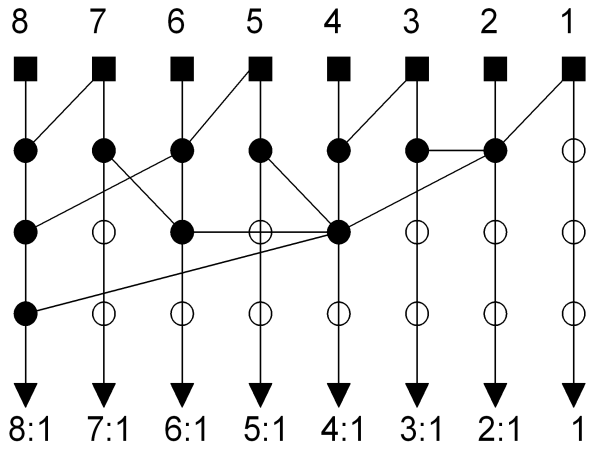

Figure III.1 Compact Placement of the 8-bit Brent-Kung Adder

It can be observed that the physical depth required to hold a prefix adder can be smaller than the logical depth. The lower bound of physical depth is the max number of GP adders in one bit-slice. Known the physical depth, denoted as m, the physical area is formulated to $n \times m$.

$$Area = n \times m \tag{III.1}$$

III.A.2 Timing Model

We adopt a linear timing model following the concept of logical effort. Fig.III.2 shows the structure and logical effort of an inverting CMOS GP adder in [17].

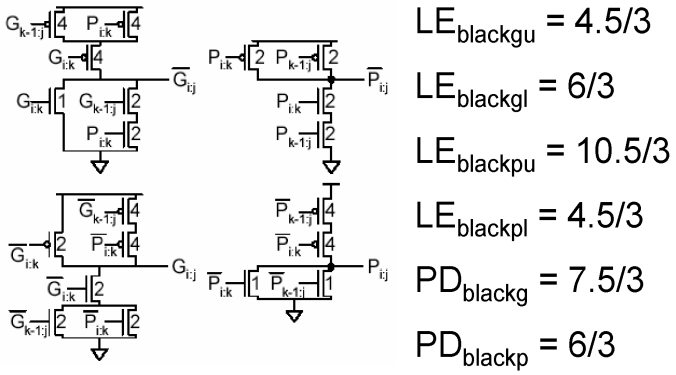

Figure III.2 Structure and Logical Effort of Inverting CMOS GP Adder

Assuming the G path is always slower than the P path, the previous model can be simplified to a two-inputs one-output format. The logical effort formulation is written as:

$$LE_l = 10.5/3 = 1.5 \quad \text{(III.2)}$$

$$LE_r = 6/3 = 2 \quad \text{(III.3)}$$

$$PD = 7.5/3 = 2.5 \quad \text{(III.4)}$$

Given the logical effort and the parasitic delay, the gate delay depends on the ratio of output load and input capacitance. When gate size is uniform, the gate delay can be calculated from load capacitance. The result unit is $\frac{1}{5}$ FO4 delay, denoted as $\frac{1}{5}D_{FO4}$. Accordingly, the timing model of an GP adder is shown as follows:

$$Delay_l = 1.5 \cdot Cload + 2.5 \quad \text{(III.5)}$$

$$Delay_r = 2 \cdot Cload + 2.5 \quad \text{(III.6)}$$

In terms of load capacitance, it is composed of gate capacitance and wire capacitance. Gate capacitance can be derived directly from the number of fanouts, when every GP adder has unit input capacitance. Wire capacitance is linear to the wire length, which depends on the physical placement. For given physical placement, the total wire length driven by a GP adder is estimated by the number of columns and rows occupied by the bounding box around all the fanouts. It matches with the daisy-chain interconnect structure. Hence the wire capacitance can be calculated as the total wire length scaled by a scaling factor. We pick 0.5 as the scaling factor for current technology. Note that all the parameters can be various for different technology .

III.A.3 Power Model

Power consumption of CMOS circuit includes two parts: (a) dynamic power consumption (b) static power consumption. The dynamic power consumption is mainly due to the charge and discharge of capacitance. Hence the activity probability of each GP adder is an essential factor in dynamic power estimation. Vanichayobon [18] analyzes the activity of prefix adder and proposes a power consumption model, shown as follows, where d is the logical depth of the prefix network and w_i is the number GP adders in level i.

$$\sum_{i=1}^{d} i \cdot w_i \tag{III.7}$$

The equation demonstrates the switching probability of each node in level i. To incorporate the other key factor, load capacitance, the equation (III.7) is revised to equation (III.8), where C_i is the total load capacitance in level i.

$$\sum_{i=1}^{d} i \cdot C_i \tag{III.8}$$

Nowadays static power consumption grows fast with the technology development. It is independent from switching activities but proportional to the number of GP adders when gate size is uniform. To make the static power consumption comparable with dynamic power, we use 3 times the total number of GP adders as the estimate. The result unit is $\frac{1}{4}$ FO4 switching power consumption, denoted as $\frac{1}{4}P_{FO4}$. Thus the total power consumption is calculated by:

$$Power = \sum_{i=1}^{d} i \cdot C_i + 3.0 \cdot \sum_{i=1}^{d} w_i \tag{III.9}$$

III.B Basic ILP Formulation

In this section we present the ILP formulations to describe the prefix adder problem according to the area/timing/power models proposed in the pre-

vious section. The first subsection demonstrates the representations of a prefix adder and its physical placement, followed by GP property constraints described in subsection 2. Subsection 3 characterizes the calculation of load capacitance for each GP adder. Based on the load capacitance, timing constraints and power optimization objective are displayed in the following subsections.

III.B.1 Structural Constraints and Physical Placement

The ILP representation of a prefix adder is quite straightforward to match with the logic view of a prefix adder. It describes GP adders and interconnections in the $n \times d$ array, where n is the operand bit-width and d is the logical depth. The bit-slice structure is kept for each column: the left fanin of each GP adder is always from some one above in the same column, and the right fanin is connected from the top-right quadrant. Also, at least one input is from the previous logical level. This bit-slice property is referred as **Bit-slice rule** in the ILP constraints.

To guarantee the described structure is a feasible prefix network, the GP property and GP operation rules must be satisfied. That is, each GP adder should cover a certain segment which is determined by the left and right inputs. In additional, the two inputs must cover two adjacent segments. The primary output at column i should cover the certain segment [i,1]. This **GP rule** will be formulated in the ILP constraints.

So far the aforementioned ILP formulation has defined all the feasible prefix networks for the given bit-width n and logical depth d. The next step is to place a prefix network in a physical $n \times m$ array, where m is the physical depth. As mentioned in the area model, all GP adders in one logical column are placed in the same physical column. The only constraint is that no two GP adders are placed on the same physical position. (**Overlap rule**)

According to the previous discussion, we define the following variables:

- $x_{(i,j)} \in \{0, 1\}$: 1 if and only if an GP adder is in the i^{th} bit and j^{th} level [1],

[1] We call this column i and row j, or simply position (i,j) later on

for every (i, j) in the $n \times d$ array.

- $w^L_{(i,j,h)} \in \{0,1\}$: 1 if and only if there is a left fanin wire to position (i,j) from position (i,h), and $h < j$.
- $w^R_{(i,j,k,l)} \in \{0,1\}$: 1 if and only if there is a right fanin wire to position (i,j) from position (k,l), and $k < i, l < j$.
- $w^Z_{(i,j)} \in \{0,1\}$: 1 if and only if the output of the GP adder (i,j) connects to the primary output in column i.
- $y^L_{(i,j)}, y^R_{(i,j)} \in [1,n]$: the left and right segment bounds of GP adder (i,j). That means the GP adder in position (i,j) covers the segment $[y^L_{(i,j)}{:}y^R_{(i,j)}]$.
- $p_{(i,j)} \in [0,m]$: the physical level of GP adder (i,j). Then its physical position will be $(i,p_{(i,j)})$.

The following constraints correspond with the **Bit-slice rule**, **GP rule** and **Overlap rule**:

Bit-slice rule:

$$\sum_h w^L_{(i,j,h)} = x_{(i,j)} \qquad \forall (i,j)\ i > h \tag{III.10}$$

$$\sum_{(k,l)} w^R_{(i,j,k,l)} = x_{(i,j)} \qquad \forall (i,j)\ i > k \& j > l \tag{III.11}$$

$$w^L_{(i,j,j-1)} + \sum_k w^R_{(i,j,k,j-1)} \geq x_{(i,j)} \qquad \forall (i,j) \tag{III.12}$$

$$\sum_j w^Z_{(i,j)} = 1 \qquad \forall i \tag{III.13}$$

GP rule:

$$y^L_{(i,j)} = y^L_{(i,h)} \qquad if\ w^L_{(i,j,h)} = 1 \tag{III.14}$$

$$y^R_{(i,j)} = y^R_{(k,l)} \qquad if\ w^R_{(i,j,k,l)} = 1 \tag{III.15}$$

$$y^R_{(i,h)} = y^L_{(k,l)} + 1 \qquad if\ w^L_{(i,j,h)} = 1\ and\ w^R_{(i,j,k,l)} = 1 \tag{III.16}$$

$$y^L_{(i,j)} = i \qquad if\ w^Z_{(i,j)} = 1 \tag{III.17}$$

$$y^R_{(i,j)} = 1 \qquad if\ w^Z_{(i,j)} = 1 \tag{III.18}$$

Overlap rule:

$$p_{(i,j)} \neq p_{(i,h)} \quad \forall i \ j \neq h \tag{III.19}$$

A problem in the GP rule is that these constraints are conditional constraints, which cannot be handled by ILP solver directly. They have to be transformed to strict linear constraints. Equation (III.20) and (III.21) shows the linear format of equation (III.14).

$$y^L_{(i,j)} \geq y^L_{(i,h)} - n \cdot (1 - w^L_{(i,j,h)}) \tag{III.20}$$

$$y^L_{(i,j)} \leq y^L_{(i,h)} + n \cdot (1 - w^L_{(i,j,h)}) \tag{III.21}$$

It can be observed that when $w^L_{(i,j,h)} = 1$, $y^L_{(i,j)}$ is restricted to $y^L_{(i,h)}$, otherwise these constraints are cancelled by the product term. It is important to notice that these constraints may not be effective until variable w^L is integerized. We call them "pseudo-linear" constraints. Pseudo-linear constraint is harmful to the performance of ILP solver, because it will reduce the chance to cut off infeasible solutions before every integer variable has been exactly determined.

III.B.2 Capacitance Constraints

Load capacitance is the essential parameter for both timing and power estimation. It includes gate load capacitance and wire load capacitance. Gate load capacitance depends on all the logical connections from a GP adder. Assume each GP adder has a unit input capacitance, the gate load capacitance can be calculated by the number of fanouts from a GP adder. (**Gate rule**)

The wire capacitance highly depends on the physical positions of each GP adders. As mentioned in timing model, we use the half perimeter of the bounding box covering all fanouts as the measure of wire length, as shown in Fig.III.3.

The height/width of the bounding box is the max vertical/horizontal distance of each fanout. Based on the known wire length, wire capacitance is the product of a scaling factor and the wire length. We pick 0.5 as the scaling factor to

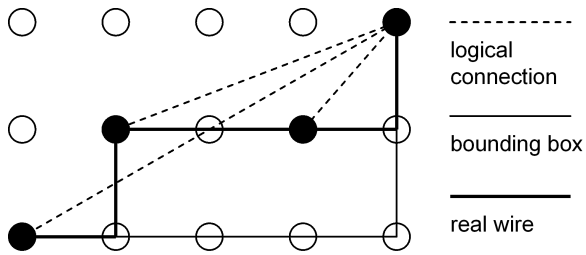

Figure III.3 Wire Length Estimation

represent the ratio of unit-length wire capacitance to unit gate capacitance. (**Wire rule**)

Following variables are defined for capacitance calculation:

- $c^G_{(i,j)} \in [0, Cmax]$: The gate load capacitance on GP adder (i,j). And $Cmax$ is a large constant representing the max load capacitance value.
- $c^{WV}_{(i,j)} \in [0, Cmax]$: The wire load capacitance from the vertical height of the fanout bounding box.
- $c^{WH}_{(i,j)} \in [0, Cmax]$: The wire load capacitance from the horizontal width of the fanout bounding box.
- $c_{(i,j)} \in [0, Cmax]$: The total load capacitance on GP adder (i,j).

The **Gate rule** and **Wire rule** in ILP formulation list as follows:

Gate rule:

$$c^G_{(i,j)} = \sum_h w^L_{(i,j,h)} + \sum_{(k,l)} w^R_{(i,j,k,l)} + w^Z_{(i,j)} \quad \forall (i,j) \tag{III.22}$$

Wire rule:

$$c^{WV}_{(i,j)} \geq 0.5(p_{(i,h)} - p_{(i,j)}) \qquad if\ w^L_{(i,j,h)} = 1 \tag{III.23}$$

$$c^{WV}_{(i,j)} \geq 0.5(p_{(k,l)} - p_{(i,j)}) \qquad if\ w^R_{(i,j,k,l)} = 1 \tag{III.24}$$

$$c^{WH}_{(i,j)} \geq 0.5(k - i) \qquad if\ w^R_{(i,j,k,l)} = 1 \tag{III.25}$$

Total load capacitance:

$$c_{(i,j)} = c^G_{(i,j)} + c^{WV}_{(i,j)} + c^{WH}_{(i,j)} \quad \forall (i,j) \tag{III.26}$$

Note that the constraint of gate rule is linear, while the constraints in wire rule are all pseudo-linear. Hence the capacitance calculation is partial linear to the structure variables.

III.B.3 Timing Constraints

The timing analysis on the prefix structure is performed from top to bottom. The start points are the primary inputs with input arrival times, while the end points are the primary outputs, whose output times should be smaller than output required times. The output time of each GP adder is the max path delay from input, which depends on the left and right input arrival times and the gate delay described in the previous section. (**Output rule**) Input arrival times are obtained from the output times of two fanins. (**Input rule**)

We define following variables for input arrival times and output times:

- $t^L_{(i,j)}, t^R_{(i,j)} \in [0, Tmax]$: the left and right input arrival times of GP adder (i,j).
- $t_{(i,j)} \in [0, Tmax]$: The output time of GP adder (i,j). Tmax is a large constant.
- $t^Z_i \in [0, Tmax]$: The primary output arrival time at each bit-slice.

The **Input rule** and **Output rule** are formulated as:

Input rule:

$$t^L_{(i,j)} = t_{(i,h)} \qquad if\ w^L_{(i,j,h)} = 1 \tag{III.27}$$

$$t^R_{(i,j)} = t_{(k,l)} \qquad if\ w^R_{(i,j,k,l)} = 1 \tag{III.28}$$

$$t^Z_i = t_{(i,j)} \qquad if\ w^Z_{(i,j)} = 1 \tag{III.29}$$

Output rule:

$$t_{(i,j)} \geq t^L_{(i,j)} + 1.5 \cdot c_{(i,j)} + 2.5 \tag{III.30}$$

$$t_{(i,j)} \geq t^R_{(i,j)} + 2.0 \cdot c_{(i,j)} + 2.5 \tag{III.31}$$

$$t^Z_i \leq Output_Required_Time(i) \tag{III.32}$$

Among these timing constraints, the input arrival time constraints are pseudo-linear, and they are critical to the entire timing analysis. Conceptually the timing analysis cannot be finished until all structural variables are integerized.

III.B.4 Power Consumption Objective

Following equation (III.9), the total power consumption objective can be easily described as follow:

$$Minimize \quad \sum_{(i,j)} j \cdot c_{(i,j)} + 3.0 \cdot \sum_{(i,j)} x_{(i,j)} \tag{III.33}$$

The first term represents the dynamic power consumption and the second term corresponds to the static power consumption. Note that the static power is linear to the total number of GP adders, but the dynamic power is not entirely linear to all the structural variables. The wire load capacitance is pseudo-linear to connection variables. However, the gate load capacitance is linear to the structural variables. Overall, the linear components still dominate the total power consumption. With this property, ILP solver can efficiently search the solution space and find the optimal solutions quickly.

III.C Extended ILP Formulations

While the basic ILP formulation already supports the comprehensive area, timing and power model, there are two important logical optimization method missing in the formulation. This section will present two extended ILP formulations supporting gate sizing and buffer insertion.

III.C.1 Supporting Gate Sizing

Gate sizing is a popular logical optimization technique to improve performance. To support gate sizing in the ILP formulation, the influence on each parameters should be studied first. For a given prefix structure, gate sizing has

no effect on interconnect relations or GP property. So the structural constraints and GP property constraints keep unchanged. However it changes the input load capacitance and the driving strength of a GP adder. For example, if the size of an GP adder is enlarged x times, the input load capacitance will increase by x times too. (**Sizing-cap rule**) At the same time, the logical effort delay will decrease x times while the parasitic delay keeps the same. (**Sizing-delay rule**) Beside the effect on capacitance and timing analysis, gate sizing also increases the static power consumption linearly.

We define two sets of variables to represent the sizes of GP adders:

- $s^2_{(i,j)} \in [0,1]$: 1 if and only if the size of GP adder (i,j) is 2.
- $s^3_{(i,j)} \in [0,1]$: 1 if and only if the size of GP adder (i,j) is 3.
- $s^{CL}_{(i,j,h)} \in [0, Cmax]$: Incremental gate capacitance on each left fanin connection.
- $s^{CR}_{(i,j,k,l)} \in [0, Cmax]$: Incremental gate capacitance on each right fanin connection.
- $s^{DL}_{(i,j)} \in [0, Tmax]$: Delay improvement from the left fanin to the output for GP adder (i,j).
- $s^{DR}_{(i,j)} \in [0, Tmax]$: Delay improvement from the right fanin to the output for GP adder (i,j).

These variables can be considered to describe the incremental size of a GP adder. The actual size of GP adder (i,j) is $x_{(i,j)} + s^2_{(i,j)} + 2 \cdot s^3_{(i,j)}$. The maximum gate size is 3.

The **Sizing-cap rule** and **Sizing-delay rule** list as follows:

Sizing-cap rule:

$$s^{CL}_{(i,j,h)} = s^2_{(i,j)} + 2 \cdot s^3_{(i,j)} \qquad if\ w^L_{(i,j,h)} = 1 \qquad \text{(III.34)}$$

$$s^{CR}_{(i,j,k,l)} = s^2_{(i,j)} + 2 \cdot s^3_{(i,j)} \qquad if\ w^R_{(i,j,k,l)} = 1 \qquad \text{(III.35)}$$

$$\tilde{c}_{(i,j)} = c_{(i,j)} + \sum_h s^{CL}_{(i,j,h)} + \sum_{(k,l)} s^{CR}_{(i,j,k,l)} \qquad \forall (i,j) \qquad \text{(III.36)}$$

Sizing-delay rule:

$$s^{DL}_{(i,j)} \leq 0.75\tilde{c}_{(i,j)} \qquad if\ s^2_{(i,j)} = 1 \qquad \text{(III.37)}$$

$$s^{DL}_{(i,j)} \leq 1.0\tilde{c}_{(i,j)} \qquad if\ s^3_{(i,j)} = 1 \qquad \text{(III.38)}$$

$$s^{DR}_{(i,j)} \leq 1.0\tilde{c}_{(i,j)} \qquad if\ s^2_{(i,j)} = 1 \qquad \text{(III.39)}$$

$$s^{DR}_{(i,j)} \leq 1.3\tilde{c}_{(i,j)} \qquad if\ s^3_{(i,j)} = 1 \qquad \text{(III.40)}$$

$$\tilde{t}_{(i,j)} \geq t^L_{(i,j)} + 1.5 \cdot \tilde{c}_{(i,j)} + 2.5 - s^{DL}_{(i,j)} \qquad \forall (i,j) \qquad \text{(III.41)}$$

$$\tilde{t}_{(i,j)} \geq t^R_{(i,j)} + 2.0 \cdot \tilde{c}_{(i,j)} + 2.5 - s^{DR}_{(i,j)} \qquad \forall (i,j) \qquad \text{(III.42)}$$

Finally the revised power objective with gate sizing becomes to:

$$Minimize \quad \sum_{(i,j)} j \cdot \tilde{c}_{(i,j)} + 3.0 \cdot \sum_{(i,j)} (x_{(i,j)} + s^2_{(i,j)} + 2 \cdot s^3_{(i,j)}) \qquad \text{(III.43)}$$

III.C.2 Supporting Buffer Insertion

The extension to support another powerful synthesis technique, buffer insertion, is more complicated than handling gate sizing. Because the buffer insertion will compromise the consistency between logical and physical connections, and consequently change the constraints on load capacitance, timing analysis and power estimation. In this case, it is not enough to present the logical structure only by structural variables. A complete explicit physical view of each solution is necessary. On the other hand, logical structure is essential to provide linear constraints on the objective function. So the extension supporting buffer insertion will operate on both logical and physical structural variables.

A prefix structure is completely represented in both logical view and physical view, as shown in Fig.III.4. There is no buffer in the logical view. It

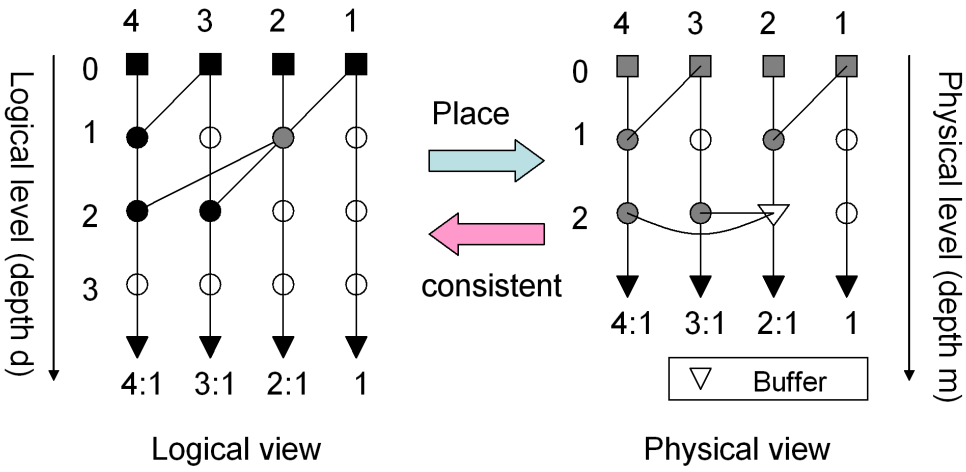

Figure III.4 Logical View and Physical View with Buffers

keeps the pure prefix network with logical interconnections. The logical network is placed in the physical view, and each logical interconnection can be physically implemented through one or multiple buffers. The physical view must be consistent with the logical view.

There are mainly two steps to maintain the consistency between the logical and physical views. The matching information must be recorded. That is, where each GP adder in logical view is placed in physical view, and for each GP adder or buffer in physical view, which GP adder in logical view it represents for. (**Matching rule**) Note that due to the appearance of buffer, a GP adder in logical view may be represented by one GP adder and multiple buffers in physical view. Known the matching information, every physical interconnect can be checked if it represents a valid logical interconnect. (**Interconnect rule**)

To express the physical view and keep the matching information, we define the following variables:

- $x'_{(i',j')} \in \{0,1\}$: 1 if and only if an GP adder is placed in the i'^{th} bit and j'^{th} level in physical view, for every (i', j') in the $n \times m$ array.
- $b'_{(i',j')} \in \{0,1\}$: 1 if and only if a buffer is placed in the i'^{th} bit and j'^{th} level in physical view, for every (i', j') in the $n \times m$ array.

- $w'^{L}_{(i',j',h')} \in \{0,1\}$: 1 if and only if there is a physical connection from position (i',h') to the left fanin of position (i',j').

- $w'^{R}_{(i',j',k',l')} \in \{0,1\}$: 1 if and only if there is a physical connection from position (k',l') to the right fanin of position (i',j').

- $w'^{B}_{(i',j',k',l')} \in \{0,1\}$: 1 if and only if there is a physical connection from position (k',l') to the buffer fanin of position (i',j').

- $w'^{Z}_{(i',j')} \in \{0,1\}$: 1 if and only if the output of the GP adder (i',j') in the physical view connects to the primary output in column i'.

- $p_{(i,j,i',j')} \in \{0,1\}$: 1 if and only if the GP adder (i,j) in logical view is placed on the position (i',j') in physical view.

- $q_{(i',j')} \in ([1,n],[0,m])$: the logical coordinate of the GP adder represented by the component at physical position (i',j'). For example, if $p_{(i,j,i',j')} = 1$, then $q_{(i',j')} = (i,j)$.

The **Matching rule** and **Interconnect rule** are formulated based on the variables:

Matching rule:

$$\sum_{(i',j')} p_{(i,j,i',j')} = x_{(i,j)} \qquad \forall(i,j) \tag{III.44}$$

$$x'_{(i',j')} = \sum_{(i,j)} p_{(i,j,i',j')} \qquad \forall(i',j') \tag{III.45}$$

$$q_{(i',j')} = (i,j) \qquad if\ \ p_{(i,j,i',j')} = 1 \tag{III.46}$$

$$q_{(i',j')} = q_{(k',l')} \qquad if\ \ w'^{B}_{(i',j',k',l')} = 1 \tag{III.47}$$

Interconnect rule:

$$(w^{L}_{(i,j,h)} = 1)\&(q_{(i',j')} = (i,j))\&(q_{(i',h')} = (i,h)) \qquad if\ \ w'^{L}_{(i',j',h')} = 1 \tag{III.48}$$

$$(w^{R}_{(i,j,k,l)} = 1)\&(q_{(i',j')} = (i,j))\&(q_{(k',l')} = (k,l)) \qquad if\ \ w'^{R}_{(i',j',k',l')} = 1 \tag{III.49}$$

$$(w^{Z}_{(i,j)} = 1)\&(q_{(i',j')} = (i,j)) \qquad if\ \ w'^{Z}_{(i',j')} = 1 \tag{III.50}$$

The previous variables and constraints provide two consistent logical and physical views. The GP property constraints can be applied on the logical view only. The capacitance and timing calculation can be performed on the physical view. Only two more timing constraints are added for buffer:

$$t^B_{(i',j')} = t_{(k',l')} \qquad if \;\; w'^B_{(i',j',k',l')} = 1 \tag{III.51}$$

$$t_{(i',j')} = t^B_{(i',j')} + c_{(i',j')} + 1 \qquad if \;\; b'_{(i',j')} = 1 \tag{III.52}$$

III.D Experimental Results

We implement the ILP formulation of the optimum prefix adders and solve the problem by CPLEX 9.1 with various timing and area configurations. For uniform signal arrival/required time profile, the entire 8-bit prefix adder design space is explored. We then apply this method to non-uniform signal arrival time applications. Also, for high-bit-width applications, the ILP method can be employed in a hierarchical design methodology.

III.D.1 Uniform Input Arrival Time

To fully demonstrate the tradeoff between timing, power and area for 8-bit prefix adder design, we test every timing point with different area settings. The first starting point is the slowest but smallest structure: ripple carry structure. Then the timing requirement gradually becomes smaller, which implies tighter timing constraint, until no more feasible solution can be found. At each time point different physical depths are applied as area constraints. However, if they produce the same result, we only record the result associated with the smallest area constraint. In addition, all of the basic ILP, the extension supporting gate sizing and the extension supporting buffer insertion are tried. Again, if they produce the same solution, it is counted as no gate sizing. The timing and power results are normalized to FO4 delay (D_{FO4}) and FO4 switching power (P_{FO4}).

Table III.1 Optimum Prefix adders

Method	Timing (D_{FO4})	Depth	Power (P_{FO4})	CPU Time (s)
ILP	10.0	1	20.1	0.31
ILP	10.0	2	17.5	124
ILP (sizing)	9.0	1	25.6	2.83
ILP	9.0	2	17.5	83.4
ILP (sizing)	8.6	1	27.6	1.28
ILP	8.6	2	17.5	93.2
Brent-Kung	7.8	3	19.9	-
ILP	7.6	2	18.0	112
ILP	7.0	2	18.6	99.6
Sklansky	6.8	3	20.8	-
Kogge-Stone	6.2	3	29.0	-
ILP	6.0	2	20.9	259
ILP	5.6	2	22.9	45.7
ILP (sizing)	5.6	2	21.6	756
ILP	5.6	3	21.9	1237
ILP (sizing)	5.0	2	23.6	1208
ILP	5.0	3	25.6	4563
ILP	4.6	3	26.1	7439
ILP (sizing)	4.2	3	27.9	9654
ILP (sizing)	4.0	4	36.4	20211

Note that the number of logical levels is not a parameter any more. Instead, it is adjusted by the ILP algorithm automatically. Table.III.1 shows the ILP results and three classic regular prefix adders, and Fig.III.5 demonstrates the optimum prefix adders in the design space corresponding to the data in the table. All the data in the table are based on the area/timing/power model in Section A.

According to the data, there are several observations:

- The ripple carry adder with the minimal area can be faster by gate sizing, but the power overhead is huge. On the other hand, physical depth 2 and 3 provide great flexibility for 8-bit prefix adders. Fig.III.8 shows the minimal power prefix adder, and it is 14% faster than ripple carry adder. When timing requirement is loose, gate sizing is not necessary. With the increase of timing

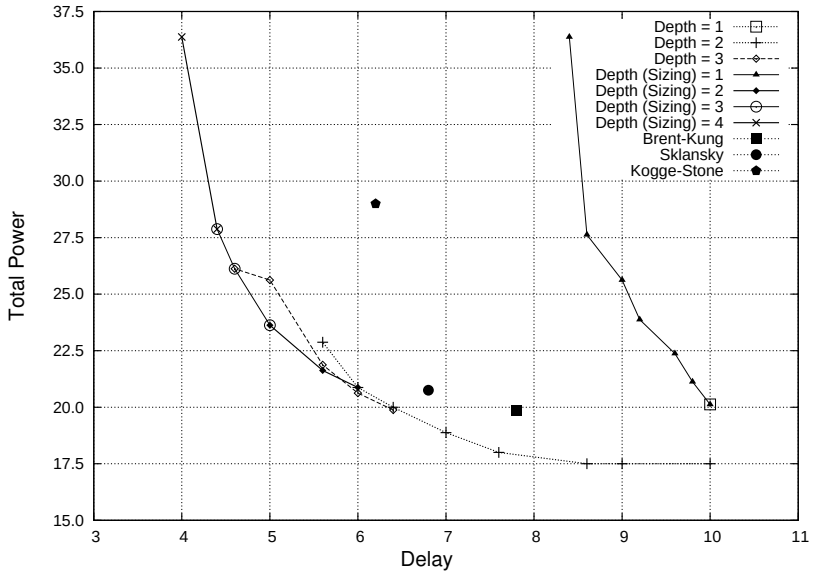

Figure III.5 Optimum Timing-Power Curves in the Design Space

requirement, either gate sizing or larger area will help to meet the timing constraint and reduce power. For extremely high performance adders, both gate sizing and large area are needed, while the power consumption increases sharply.

- None of the three classic prefix adders is optimal in terms of either area or power consumption for the given timing constraints. The Sklansky and Kogge-stone adders shown in Fig.III.6 and Fig.III.7, are usually considered as the fastest adders, but they still have more than 35% gap to the fastest one. Fig.III.9 to Fig.III.13 presents three fastest prefix adders with physical depth 2, 3 and 4 respectively. They all have 4 logical levels.

- Big gate size is not very helpful for 8-bit prefix adder. Although the max gate size allowed by the program is 3, only size 2 has appeared in the solutions. Also there is no buffer insertion in all 8-bit optimal prefix adders with uniform input arrival and output required time. One possible reason is that for 8-bit prefix addition, load capacitance is not big enough to take advantage of buffer insertion.

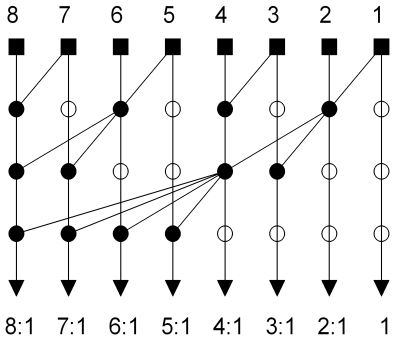

Figure III.6 Sklansky Adder

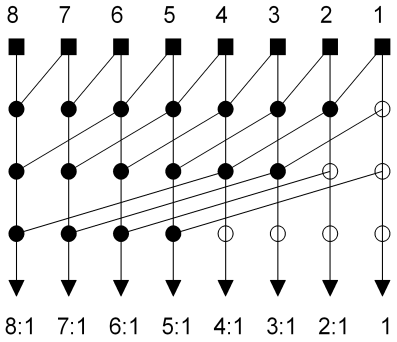

Figure III.7 Kogge-Stone Adder

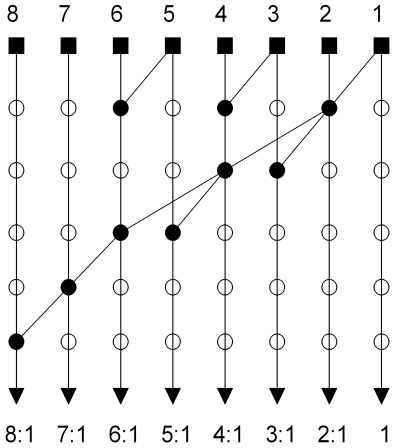

Figure III.8 Minimal Power Adder

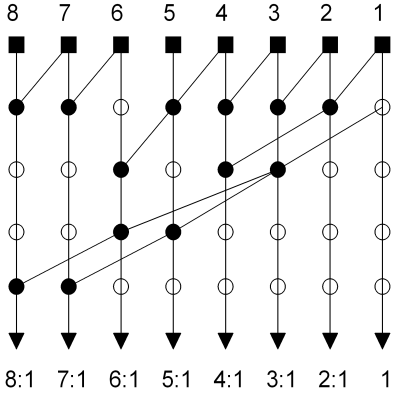

Figure III.9 Fastest Adder (Depth:2)

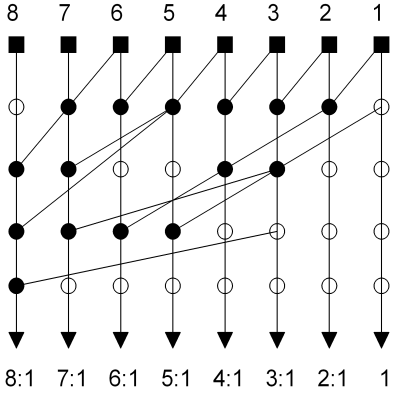

Figure III.10 Fastest Adder (Depth:3)

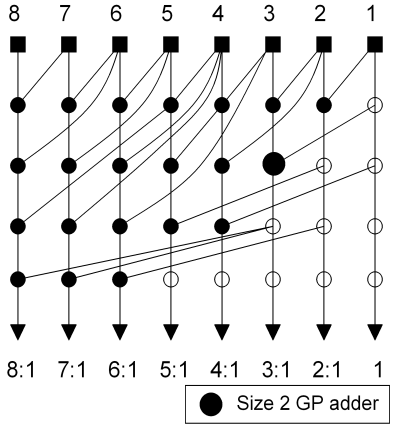

Figure III.11 Fastest Adder (Depth:4)

- The CPU time of the ILP solver is the main drawback of the proposed method. It raises quickly with the increasing timing requirement. The timing analysis is defined by pseudo-linear constraints. Therefore when timing constraint is too tight, ILP solver cannot efficiently verify the feasibility of each variable assignments. So the proposed method is more suitable for power optimization problem with moderate timing requirement.

III.D.2 Non-uniform Arrival and Required Times

Besides uniform signal arrival profile, some applications need non-uniform signal arrival/required times. Binary Multiplier is an example. A binary adder is used as final adder to sum up two partial product reduced from partial products reduction tree. The middle bits arrive later than most and least significant bits.

Table III.2 Non-uniform Arrival/Required Time Cases

Case	Power	Depth
Increasing arrival time	20.8	3
Decreasing arrival time	25.1	3
Convex arrival time	21.6	2

Here we demonstrate the optimum prefix adders for three representative arrival time profiles: increasing, decreasing and convex. Fig.III.12 to Fig.III.14 illustrate the three test cases solved by the ILP method. The numbers attached in the square brackets at the inputs and outputs depict the input arrival times and actual output arrival times. Table III.2 lists the physical depth and power consumption of each case. These three cases have various structures, which shows the flexibility of the proposed method. Note that there is still no buffer insertion in the optimal solutions, although we did see buffers in some intermediate results.

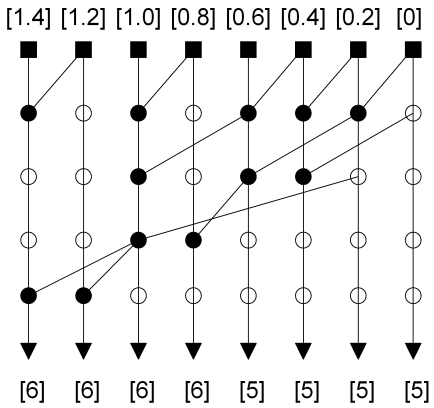

Figure III.12 Increasing Input Arrival Time

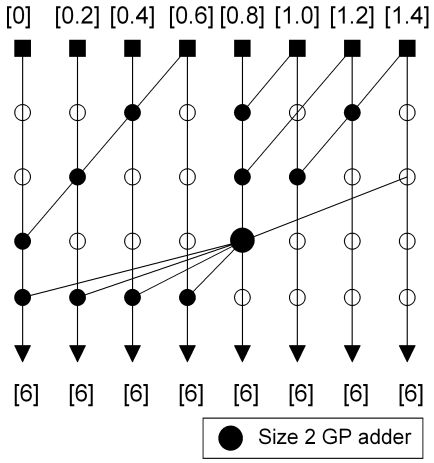

Figure III.13 Decreasing Input Arrival Time

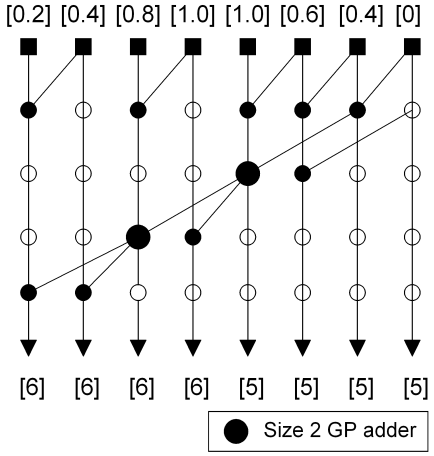

Figure III.14 Convex Input Arrival Time

III.D.3 Hierarchical Design

The previous experiments have shown the advantage of ILP method on 8-bit prefix addition applications. For high-bit-width applications, the ILP method can be applied in a hierarchical design methodology. There are two reasons to use hierarchical design methodology instead of pure ILP method. The first reason is that data-path design favors global regular structures. Global irregular structure increases the difficulty on detail routing and compromises the circuit reliability. The second reason is that ILP method is not scalable. The computation load of ILP solver increase exponentially with the operand bit-width.

ILP is applied to design a 64-bit two level hierarchical prefix adder. Sparse tree structure [19] is selected as global structure, and each 8-bit prefix block is generated by ILP method. Fig.III.15 demonstrates the hierarchy structure.

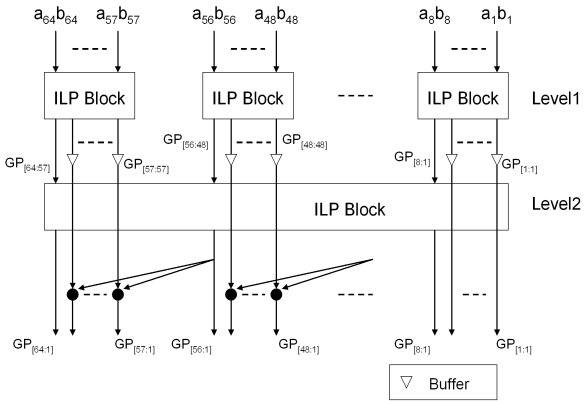

Figure III.15 64-bit Hierarchy Prefix Adder

In both hierarchy levels, each prefix block is 8-bit. The boundary timing requirement can be negotiated between the two levels. We build 64-bit hierarchical prefix adder for various timing requirement and compare them with 64-bit Kogge-Stone, Brent-Kung and Sklansky adders. Table.III.3 shows the results in terms of delay and power.

The Hierarchical ILP method not only achieves at least 20% power saving compared with 64-bit Brent-Kung and Sklansky adders, but also reach the same

Table III.3 64-bit Prefix Adders

	Timing	Power
ILP Hierarchy	28	369
Brent-Kung	27	473
ILP Hierarchy	26	370
ILP Hierarchy	24	373
ILP Hierarchy	22	375
ILP Hierarchy	20	379
ILP Hierarchy	18	386
Sklansky	17	492
ILP Hierarchy	16	402
ILP Hierarchy	15	416
Kogge-Stone	15	3032
ILP Hierarchy	14	473

performance as 64-bit Kogge-Stone adder. Fig. III.16 and III.17 demonstrate the two level physical structures in the fastest 64-bit ILP adder. The level-1 structure has fast paths from inputs to the MSB output (critical path), but save power for other bits (non-critical path). The level-2 network utilizes gate-sizing to improve the drive strength for large fanouts. They all show the flexibility of the ILP method.

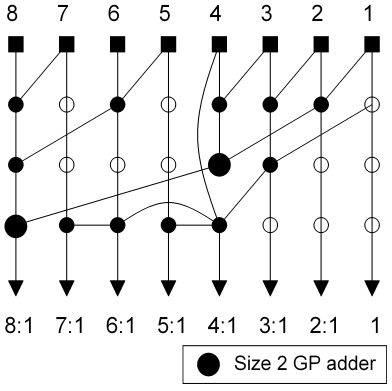

Figure III.16 Hierarchical ILP (level 1)

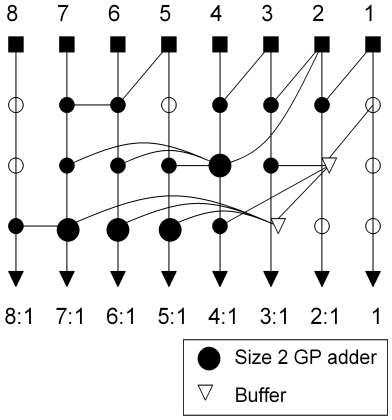

Figure III.17 Hierarchical ILP (level 2)

III.E Summary

In this chapter, we propose an ILP method to solve minimal power prefix adders on a comprehensive area/timing/power model. Based on physical placement information, both gate and wire load capacitances are involved in timing/power model. Furthermore, the power model counts both static power consumption and dynamic power consumption with activity probability. This method can handle non-uniform input arrival times and output required times for each application. The extension of the method can even support gate sizing and buffer insertion. The experiments demonstrate the complete 8-bit prefix adder solution space. It also shows the optimum area, timing and power tradeoff curves which outperform previous classic structures. For high-bit-width applications, hierarchical ILP method can generate high-performance prefix adder with low power consumption.

IV

Division Background

This chapter briefly introduces the definition of division and existing division algorithms. We also propose the concept of iteration effort, memory effort and arithmetic effort to analysis the performance and the cost of existing algorithms.

IV.A Division Basic

In mathematics, division is an arithmetic operation which is the inverse of multiplication, and sometimes it can be interpreted as repeated subtraction. Specifically, if Q times B equals A, written:

$$Q \times B = A \tag{IV.1}$$

where B is not zero, then A divided by B equals Q, written:

$$\frac{A}{B} = Q \tag{IV.2}$$

In the above expression, A is called the dividend (numerator), B the divisor (denominator) and Q the quotient.

IV.A.1 Binary Division Definition

In binary number system, the accuracy of every number is limited by its bit width. In most cases, the quotient cannot be guaranteed to be exactly

expressed by a limited-width binary number, unless a remainder, an amount "left over", is also acknowledged. We define the N-bit binary division as follow. Assume N is the data width of two operands:

$$A = 0.a_1a_2\ldots a_N \quad \text{(IV.3)}$$

$$B = 0.b_1b_2\ldots b_N \quad (\frac{1}{2} \leq B < 1) and (0 \leq A < B) \quad \text{(IV.4)}$$

The quotient and remainder are two N-bit numbers defined as follow:

$$Q = 0.q_1q_2\ldots q_N \quad \text{(IV.5)}$$

$$R = 2^{-N} \times 0.r_1r_2\ldots r_N \quad \text{(IV.6)}$$

$$A = Q \times B + R \quad (R < 2^{-N} \times B) \quad \text{(IV.7)}$$

In few applications, only the remainder part is useful, it is called modulo operation. In most normal division applications, the value of remainder is not important. The remainder is optional for IEEE floating point standard. However, the nullity of the remainder is required by the rounding step. The IEEE standard requires the use of 3 extra bits of less significance than the mantissa bits implied in the representation. These extra bits are guard bit, round bit and sticky bit. The guard and round bits are just 2 extra bits of precision that are used in calculations. The sticky bit is an indication of what could be in less significant bits that are not kept. In binary division, the sticky bit is decided by the nullity of the remainder. Based on the guard, round and sticky bits, four rounding methods are supported in the IEEE standard.

In this dissertation, we focus on the calculation of the quotient and keep the nullity property of the remainder.

IV.A.2 Fundamental Division Algorithm

The fundamental division algorithm involves three simple operations, comparison, subtraction and shifting. One quotient bit is generated in each iteration. If the remainder is greater than divisor B, the quotient bit is set to 1 and

subtracted B from the remainder. Otherwise, the remainder is simply shifted to the left.

$$q_i = \begin{cases} 1 & if\ 2 \times R_{i-1} \geq B \\ 0 & if\ 2 \times R_{i-1} < B \end{cases} \quad \text{(IV.8)}$$

$$R_i = 2 \times R_{i-1} - q_i \times B \quad \text{(IV.9)}$$

The following figure IV.1 shows how to decide what to set for the quotient bit and what is the next value of the remainder. This process repeats N times to get the N-bit final quotient.

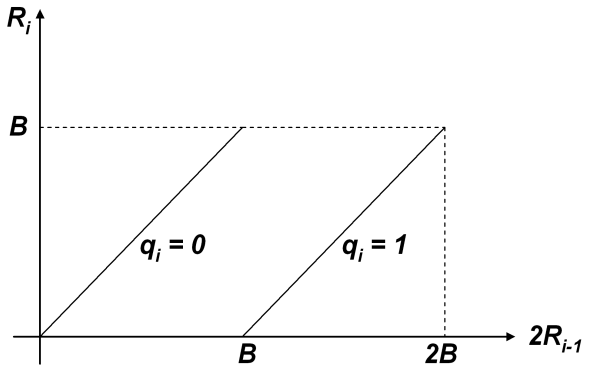

Figure IV.1 Quotient Digit Selection of Restoring Division

The fundamental algorithm can be directly implemented as restoring division algorithm. The comparison is completed by subtracting B from $2 \times R_{i-1}$. If the result is positive, it is the value of R_i. Otherwise, the previous remainder R_{i-1} need to be restored and shifted left as R_i. Advantage of restoring algorithm is that we never have to due with negative numbers. The disadvantage is that we have to subtract at every step and on average we have to restore half of the operations.

An alternate scheme for the fundamental algorithm is the non-restoring division. To avoid the restoring step, the quotient bit is either 1 or -1. Therefore, negative remainder is allowed and the restoring step is unnecessary. When the remainder is negative, it will be compensated in the next iteration of division by adding to the remainder instead of subtracting from it. Equations (IV.10) and

(IV.11) and Fig.IV.2 shows the non-restoring algorithm.

$$q_i = \begin{cases} 1 & if\ 2 \times R_{i-1} \geq 0 \\ -1 & if\ 2 \times R_{i-1} < 0 \end{cases} \quad \text{(IV.10)}$$

$$R_i = 2 \times R_{i-1} - q_i \times B \quad \text{(IV.11)}$$

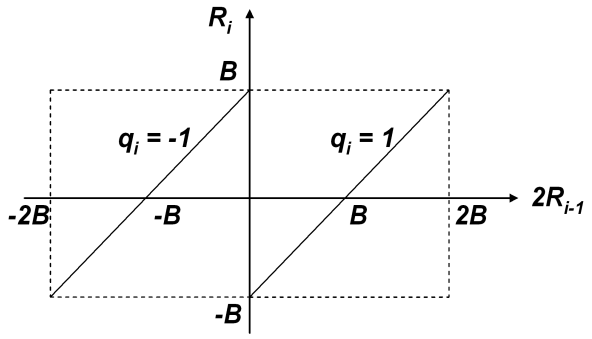

Figure IV.2 Quotient Digit Selection of Restoring Division

Advantage of non-restoring algorithm is that we do not have to restore. However, There are negative quotient bits and remainders and may need to do correction at the last step if the remainder is negative. And either a subtraction or addition still has to be done in each iteration.

IV.A.3 Iteration Effort

From the previous subsection, we can see that a division unit can be very compact. One addition/subtraction unit is enough. However, it takes N iterations to finish one division. Between these iterations, partial remainders are calculated to connect adjacent iterations. We call the computation load on these partial remainders as "Iteration Effort". The computation load is measured by the number of partial product bits (PPB). In each iteration, one N-bit number, the product of one quotient bit with the N-bit divisor, is added/subtracted. Therefore, the total iteration effort is N^2 PPBs. With the pure iteration effort, the fundamental division algorithms need N iteration cycles to finish one division operation.

The advantage of iteration effort is that hardware can be reused in every iteration. Obviously, the drawback is the long delay to complete the N iterations. For high performance division units, the N iterations are not practical. Other computing effort is needed to calculate multiple quotient bits each time and reduce the total number of iterations.

IV.B High-Radix Division Algorithms

High-radix division algorithms are also known as digit recurrence divisions (or slow divisions). Different from the fundamental division, high-radix division algorithms obtain multiple quotient bits in each iteration. The most popular high-radix division algorithms are the SRT (Sweeny, Robertson and Tocher) division [20] [21] and prescaling division [22] [23].

IV.B.1 SRT Division

SRT division is similar to Non-Restoring division, but it uses a lookup table based on the dividend and the divisor to determine each quotient digit (may include multiple quotient bits). The Intel Pentium processor's infamous divider bug was caused by an incorrectly coded lookup table [24].

The most basic SRT algorithm is SRT radix-2 division. It is an improvement over non-restoring division by allowing 0 to be a quotient digit for which no add/subtract operation is needed, as shown in equation (IV.12) and (IV.13) and Fig.IV.3.

$$q_i = \begin{cases} 1 & if\ 2 \times R_{i-1} \geq 1 \\ 0 & if\ -1 \leq 2 \times R_{i-1} < 1 \\ -1 & if\ 2 \times R_{i-1} < -1 \end{cases} \quad \text{(IV.12)}$$

$$R_i = 2 \times R_{i-1} - q_i \times B \quad \text{(IV.13)}$$

Advantage of the SRT algorithm is that the remainder is compared with 1 instead of the divisor B. Comparison with 1 is just a 1-bit comparison. The

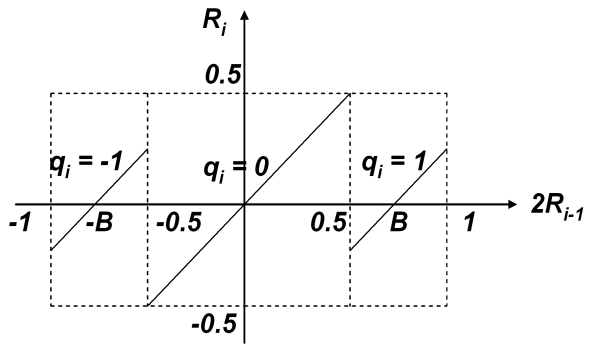

Figure IV.3 Quotient Digit Selection of SRT Radix-2 Division

improvement is very important to higher radix SRT divisions. Because It implies that the quotient digit can be decided by the most significant digits of divisor B and partial remainder R_i.

To get multiple quotient bits in one iteration, the most direct way is to use look-up table with precalculated quotient bits in it. SRT radix-4 division fetches 2-bit quotient digit from a look-up table based on the first 5 bits of divisor (including the fixed 1 on the most significant bit) and the first 7 bits of partial remainder (including the sign bit). The quotient digit ranges from -2 to 2, as shown in Fig.IV.4. After the quotient digit is known, the next remainder is calculated by equation (IV.14).

$$R_i = 4 \times R_{i-1} - Q_i \times B \tag{IV.14}$$

Note that each quotient digit constants two bits, and can guarantee that the next remainder is two-order smaller than the current remainder in magnitude. Therefore, SRT radix-4 division only needs $N/2$ iterations to complete one division operation. Higher radix SRT divisions are also achievable, but they need larger look-up table.

IV.B.2 Prescaling Division

Another popular division algorithm is prescaling method, also known as Svoboda-Tung division. The basic idea is that when the divisor closes to 1, the remainder will close to the quotient. Instead of looking for the partial quotient

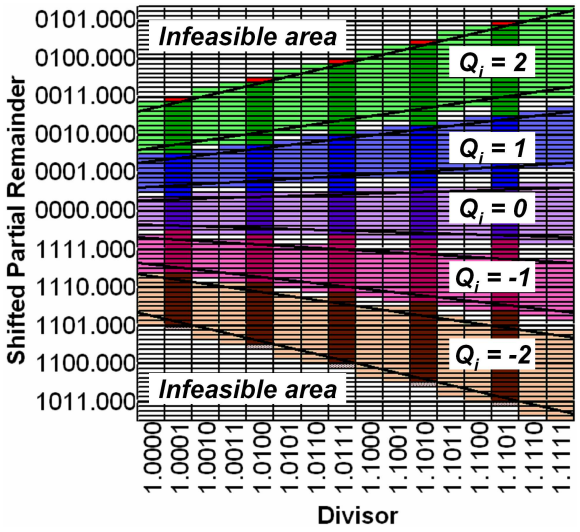

Figure IV.4 Quotient Digit Selection of SRT Radix-4 Division

directly, prescaling acquires the estimated reciprocal of the divisor $E \approx 1/B$ from memory. The estimation E scales the original operand A and B to A' and B', and keeps the same quotient. The partial quotient is derived directly from the scaled partial remainder. The prescaling step is described as follow.

$$E \approx \frac{1}{B} \tag{IV.15}$$

$$\frac{A}{B} = \frac{A \times E}{B \times E} = \frac{A'}{B'} \tag{IV.16}$$

$$Q_i \approx R'_{i-1} \tag{IV.17}$$

When the scaling factor E has two bits, the partial remainder calculation can be expressed by equation (IV.18).

$$R'_i = 4 \times R'_{i-1} - Q_i \times B' \tag{IV.18}$$

The main difference between SRT division and prescaling division is that the look-up table for SRT division stores quotient digits directly while the prescaling look-up table only contains the reciprocal of divisor. Hence the prescaling division needs less memory for the look-up table than SRT division. The overhead is the two multiplications in prescaling step. Another important fact is that re-

mainders are also scaled by prescaling. Therefore the original remainder is hard to be recovered. Fortunately, it doesn't change the nullity property of remainder.

IV.B.3 Memory Effort

In the SRT and Prescaling divisions, memory plays an important role to calculate partial quotient in each iteration. We call this computation effort from look-up table as "memory effort", and measure it by the bits number in the table. In general, to get M bits partial quotient in one iteration, SRT division needs $2^{2M} \times M$ memory bits while prescaling division requires $2^M \times M$ memory bits. Although prescaling division has less memory requirement, it introduces two extra multiplications in the prescaling step. The extra computation effort will be explain in the next section.

Memory effort reduces the number of iterations by M times by increasing the bit-width of partial quotient to M. And the operation speed on memory is relatively fast when memory size is small. It can achieve remarkable performance improvement on division operation. However, the exponential increasing cost limits the speedup factor M. When memory size is too large, both the delay and area are unaffordable.

Note that the iteration effort on the remainder calculation keeps to be N^2 PPBs.

IV.C Very-High-Radix Division Algorithms

Very-high-radix division algorithms are also known as functional iteration divisions (or fast divisions). Very-high-radix division algorithms can produce more quotient bits than high-radix division algorithms can afford in each iteration. It is achieved by using arithmetic functions in calculation of partial quotients. The existing very-high-radix division algorithms are the Taylor expansion division algorithm [25] [26] and series expansion division algorithm [27] [28].

IV.C.1 Taylor Expansion Division

Taylor expansion is a powerful tool to get an estimation of a function and the error can be controlled by the number of expansion terms. Then Taylor expansion can be applied to division function. The accuracy of partial quotient, also the bit-width of partial quotient, is decided by the initial estimation and the order of Taylor expansion. Taylor expansion algorithm is formulated as follow:

$$B = B_h + B_l \tag{IV.19}$$

$$E = \frac{1}{B_h} - B_l \times (\frac{1}{B_h})^2 + B_l^2 \times (\frac{1}{B_h})^3 ... \tag{IV.20}$$

$$Q_i = R_{i-1} \times E \tag{IV.21}$$

In the equations, B_h denotes the first M bits of B and B_l denotes the rest lower significant part. E is an estimation of $1/B$ based on Taylor expansion.

It can be proved that for $1st$-order Taylor expansion, the required precision of E is $2M$ bits to get $2M$ bits partial quotient in each iteration. Therefore the first order term in equation (IV.20) involves an M by M multiplication, and equation (IV.21) is a $2M$ by $2M$ multiplication. In practice, $(\frac{1}{B_h})^k$ is derived from memory. For instance, $1st$-order Taylor expansion needs one $2^M \times 2M$ bits table for $\frac{1}{B_h}$ and a $2^M \times M$ bits table for $(\frac{1}{B_h})^2$.

With the multiple look-up tables and multiplications, Taylor expansion division can speedup division by a large factor and achieve high performance division. It has been applied to AMD K7 microprocessor [29].

IV.C.2 Series Expansion Division

Series expansion is a special case of Taylor expansion. Instead of B_h, series expansion expands $1/B$ at 1. The series expansion division algorithm is

described as follow:

$$B = 1 - X \tag{IV.22}$$

$$E = 1 + X + X^2 + X^3... \tag{IV.23}$$

$$= (1 + X)(1 + X^2)(1 + X^4)...$$

$$Q_i = R_{i-1} \times E \tag{IV.24}$$

According the previous equations, the series expansion does not need any look-up table. It can be a memory-free algorithm. However, the partial quotient precision is quite dependent on the magnitude of X. Assume $X < 2^{-M}$, then the $1st$-order series expansion can produce $2M$-bit partial quotient and introduce a $2M$ by M multiplication. High order series expansion division algorithm can also achieve high performance division as Taylor expansion divisions. This algorithm has been used in IBM RISC System/6000 microprocessor [30].

IV.C.3 Arithmetic Effort

Recall the prescaling division algorithm in the previous section and the very-high radix division algorithms in this section, we found that these extra polynomial functions can reduce the memory requirement or increase the precision of partial quotient or both. The total extra multiplication load on every partial quotient computation is called "arithmetic effort". It is also measured by the number of partial product bits. The arithmetic efforts of prescaling, $1st - order$ Taylor expansion and $1st - order$ series expansion divisions are $2MN$, $2MN + M^2$ and MN respectively. Note that equations (IV.20) and (IV.23) only perform once for a given divisor.

In general, arithmetic effort can boost the bit-width of partial quotient by multiple times, while the computation load is polynomial. This property enables the high performance division units in modern microprocessors.

IV.D Division Algorithms Summary

Existing division algorithms can be revised from the view of iteration effort, memory effort and arithmetic effort. Iteration effort limits the lower bound of computation effort involved in one division. However it takes long time, N iterations, to finish one operation, as fundamental division algorithms. To speedup the operation, memory effort and arithmetic effort can help to increase the precision of each partial quotient and reduce the number of iterations. Memory effort is efficient to extend the partial quotient bit-width by a certain number, but the exponentially increasing load limits the speedup factor. Memory effort can only achieve high-radix divisions like SRT division and prescaling division. Arithmetic effort is the most sophisticated effort among the three kinds of computation effort. It can multiple the partial quotient bit-width with polynomial overhead.

The following table compares the previous division algorithms in terms of iteration effort, memory effort, arithmetic 3 effort and the number of iterations in one division. The iteration effort and arithmetic effort are measured by the number of PPBs, while the memory effort is gauged by the memory size. For the $1st$-order

Table IV.1 Comparison on Computation Efforts

Algorithm	Iteration Effort (PPBs)	Memory Effort (Bits)	Arithmetic Effort (PPBs)	#Iterations
Pencil-paper	N^2	-	-	N
SRT	N^2	$2^{2M} \times M$	-	N/M
Prescaing	N^2	$2^M \times M$	$2MN$	N/M
0-order Taylor Exp.	N^2	$2^M \times M$	MN	N/M
$1st$-order Taylor Exp.	N^2	$3 \times 2^M \times M$	$2MN+M^2$	$N/2M$
$1st$-order Series Exp.	N^2	-	MN	$N/2M$

series expansion in the table, it is assumed that $B = 1 - X$ and $X < 2^{-M}$.

Based on the previous discussion, the whole solution space of division algorithms can be considered as a triangle with the three computation efforts as vertexes. Fig.IV.5 demonstrates the solution space and positions of the previous

algorithms. The iteration effort corner presents the minimal cost, while the edge between memory effort and arithmetic effort leads to high performance algorithms. A smaller triangle near memory effort corner is forbidden by the memory wall. The exponential cost is unaffordable in this region.

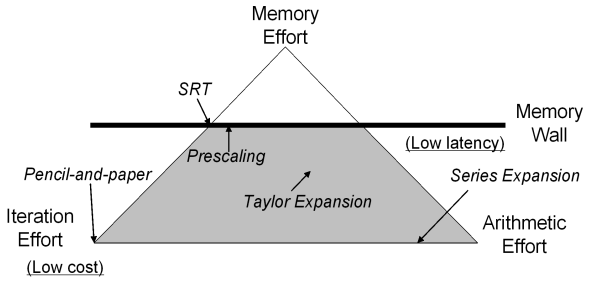

Figure IV.5 Solution Space of Division Algorithms

The empty space near the top-right corner in the gray area shows the absence of a powerful division algorithm which can fully utilize both memory effort and arithmetic effort to achieve high-performance divisions.

V

PST Division Algorithm

In this chapter, we will propose a new division algorithm which combines prescaling, series expansion and 0-order Taylor expansion together to fully utilize the advantage of memory effort, arithmetic effort and iteration effort.

V.A Notations

Some common notations used in this chapter are defined as follow:

- A: The original N-bit dividend.
- B: The original N-bit divisor.
- E_0: The estimation of $1/B$.
- A_i: The scaled dividend after the ith scaling step.
- B_i: The scaled divisor after the ith scaling step.
- E_i: The estimation of $1/B_i$.
- $\tilde{Q}_j$: The partial quotient in the jth iteration.
- Q_j: The accumulated quotient after the jth iteration.
- R_j: The remainder after the jth iteration.

- Q: The N-bit final quotient.
- R: The scaled final remainder.

V.B The Proposed Division Algorithm

The computation efforts analysis shows that series expansion algorithm is quite efficient if the divisor B is close to 1. This feature can be enabled by another algorithm, prescaling, powered by memory effort. Therefore the basic idea of the proposed algorithm is to combine prescaling and series expansion to boost the accuracy of each partial quotient, and finish one division in few iterations.

V.B.1 Basic PST Division

The basic PST algorithm is described as follow:

1. The initial scaling factor E_0 is obtained from a look-up table. The value is the reciprocal of the up-rounded divisor B and then truncated at the $M+1$ bit. Because E_0 is an estimation of $1/B$, the product of E_0 and B is close to 1, i.e. it has M bits guaranteed leading $1'$s.

$$B_{[M+2]} = 0.b_1 b_2 \dots b_{M+2} 111 \dots \tag{V.1}$$

$$E_0 = trunc(1/B_{[M+2]})_{M+1} = 1.e_1^0 e_2^0 \dots e_{M+1}^0 \tag{V.2}$$

   Considering the normalized divisor $B \geq 1/2$, b_1 is always '1'. Therefore the size of the look-up table is 2^{M+1} words and $M+1$ bits per word.

2. The dividend A and the divisor B are scaled by the scaling factor E_0 simultaneously.

$$\begin{cases} A_1 = A \times E_0 = 0.a_1^1 a_2^1 \dots a_{N+M+1}^1 \\ B_1 = B \times E_0 = 0.11 \dots 1 b_{M+1}^1 b_{M+2}^1 \dots b_{N+M+1}^1 \end{cases} \tag{V.3}$$

3. The estimation E_1 of $1/B_1$ is the reverse of B_1 truncated at the $2M$ bit.

$$\begin{aligned} E_1 &= trunc(\bar{B_1})_{2M} \\ &= 1.00\ldots0\bar{b^1}_{M+1}\bar{b^1}_{M+2}\ldots\bar{b^1}_{2M} \end{aligned} \tag{V.4}$$

4. The partial quotient $\tilde{Q_j}$ is calculated from the product of the truncated previous remainder R_{j-1} and E_1, and then is truncated at the $2M$ bit. Initially $R_0 = A_1$.

$$\begin{aligned} \tilde{Q_j} &= trunc(trunc(R_{j-1})_{2M} \times E_1)_{2M} \\ &= 0.\tilde{q^j}_1\tilde{q^j}_2\ldots\tilde{q^j}_{2M} \end{aligned} \tag{V.5}$$

5. The partial quotient is added to the previous quotient Q_j to approach the final result. The product of $\tilde{Q_j}$ and B_1 is subtracted from R_{j-1}. The new remainder R_j has $2M-2$ guaranteed leading $0'$s to be shifted out. We will prove this claim in the next section. If $(2M-2)\times j < N$, repeat the previews step. Assume J is the total number of the iterations, then $J = \lceil \frac{N}{2M-2} \rceil$.

$$\begin{cases} Q_j &= Q_{j-1} + 2^{-(2M-2)\times(j-1)} \times \tilde{Q_j} \\ R_j &= 2^{2M-2} \times (R_{j-1} - B_1 \times \tilde{Q_j}) \end{cases} \tag{V.6}$$

6. To get the final quotient Q, the corresponding remainder R needs to be recovered. The extra tail of Q_J beyond N bits , denoted as Qt, is removed. Correspondingly, R is calculated by adding the product of Qt and B_1 to R_J. If R is greater than or equal to B_1, Q should increase one at the least significant bit.

$$\begin{cases} Q &= trunc(Q_J)_N \\ Qt &= 0.00\ldots0q^J_{N+1}q^J_{N+2}\ldots \end{cases} \tag{V.7}$$

$$R = R_J + Qt \times B_1 \tag{V.8}$$

$$\begin{cases} Q &= Q + 2^{-N} \\ R &= R - B_1 \end{cases} \quad if\ R \geq B_1 \tag{V.9}$$

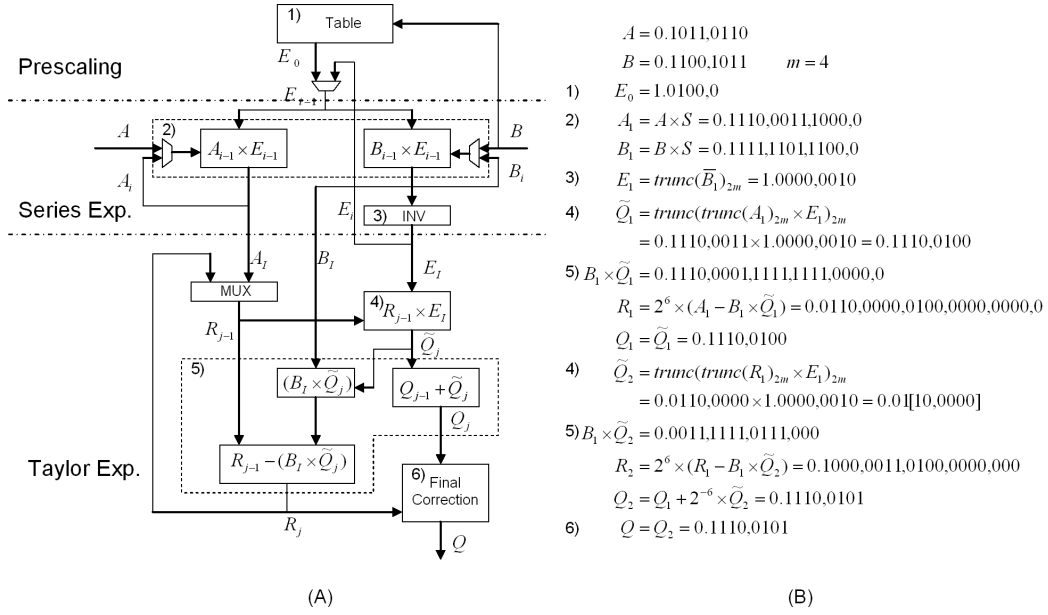

Figure V.1 (A) PST Algorithm Architecture (B) An 8-bit example

Step 6) can be considered as a computation overhead from generating extra quotient bits more than N. This computation load is mainly on the multiplication of Qt and B_1. This operation is avoidable. Qt can be directly removed from the last partial quotient $\tilde{Q}_J$. Therefore the correction step is simply adding one at the least significant bit of Q if necessary.

$$\tilde{Q}_J = trunc(\tilde{Q}_J)_{2M+N-(2M-2)\times J} \tag{V.10}$$

$$\begin{cases} Q &= Q_J = Q_{J-1} + 2^{-(2M-2)\times(J-1)} \times \tilde{Q}_J \\ R &= R_J = 2^{2M-2} \times (R_{J-1} - B_1 \times \tilde{Q}_J) \end{cases} \tag{V.11}$$

$$\begin{cases} Q &= Q + 2^{-N} \\ R &= R - B_1 \end{cases} \quad if\ R \geq B_1 \tag{V.12}$$

Fig. V.1(B) demonstrates an 8-bit example. Dividend A and divisor B are two 8-bit numbers. According to the higher 6 bits of B, we lookup a 5-bit scaling factor E_0. After prescaling, the scaled divisor B_1 has 4 bits guaranteed leading 1s. Step(3) calculates an accurate estimation of B_1 inverse, E_1, by simply inverting the higher 9 bits of B_1, including the 0 before the radix point and four

fixed 1s after the radix point. Based on E_1, the first 8-bit partial quotient $\tilde{Q}_1$ and partial remainder R_1 are obtained. There are six guaranteed leading 0s in the partial remainder, so it can be shifted left by 6 bits. In the second iteration, partial quotient $\tilde{Q}_2$ is truncated to 2 bits, because the lower part exceeds the precision requirement. The final quotient Q is the summation of $\tilde{Q}_1$ and $\tilde{Q}_2$, and the final remainder R equals R_2. Because $R < B_1$, correction step is unnecessary in this case.

V.B.2 Correctness Proof of the Basic PST Algorithm

The correctness proof essentially is an analysis of errors introduced in each steps. Each partial quotient is limited within an error range. Therefore, every intermediate remainders have a fixed number of guaranteed leading zeros, which implies that the quotient is converging to the correct result.

The proof contains three parts. The first part is the error analysis for prescaling. The second part is for the error introduced in series expansion. The last part is the leading zeros in each intermediate remainder.

1. For the prescaling factor look-up table in step 1), if both the data and address bit-widthes are m, then B_1 has $m-1$ guaranteed leading $1'$s. That is to say $(1 - B_1) < 2^{-m+1}$.

   Proof : There are two parts of error in the estimation of $1/B$, E_0. The first part of error δ_1 comes from using $B_{[m+1]}$ instead of B, and the second part δ_2 is introduced from the finite precision.

$$E_0 = \frac{1}{B} - \delta_1 - \delta_2 \tag{V.13}$$

$$\begin{aligned} 1 - B_1 &= 1 - B \times E_0 = 1 - B \times (\frac{1}{B} - \delta_1 - \delta_2) \\ &= B \times (\delta_1 + \delta_2) \end{aligned} \tag{V.14}$$

   Note that both δ_1 and δ_2 are positive. Their error analysis is the same the

analysis of the basic method in [26]. It has been proved that:

$$0 < \delta_1 < \frac{1.1}{2^{m+1} \times B_{m+1}^2} \tag{V.15}$$

$$0 < \delta_2 < \frac{1}{2^m} \tag{V.16}$$

Applying these inequations to (V.14), we have:

$$\begin{aligned} (V.14) &= \frac{1.1 \times B}{2^{m+1} \times B_{[m+1]}^2} + \frac{B}{2^m} \\ &< \frac{1.1}{2^{m+1} \times B} + \frac{B}{2^m} \end{aligned} \tag{V.17}$$

The maximum value of (V.17) happens when $B = 1/2$.

$$(V.17) \leq \frac{1.1}{2^m} + \frac{0.5}{2^m} < \frac{1}{2^{m-1}} \tag{V.18}$$

Therefore B_1 has $m-1$ guaranteed leading $1'$s. ■

In the basic PST algorithm, m is set to $M+1$. So B_1 has M leading $1'$s.

2. If B_i has m leading $1'$s, then the product of B_i and E_i has $2m-1$ leading $1'$s. Although this value may not be really calculated in the algorithm, it shows the accuracy of E_i as an estimation of $1/B_i$.

   Proof : Assume $B_i = 1 - X$ and the error introduced by the truncate operation is δ_1.

$$\begin{aligned} B_i \times E_i &= (1 - X) \times (1 + X - \delta_1) \\ &= 1 - X^2 - \delta_1 + \delta_1 \times X \end{aligned} \tag{V.19}$$

   Applying $0 < \delta_1 \leq 2^{-2m}$ and $X = 1 - B_i < 2^{-m}$ known from that B_i has m leading $1'$s, we have:

$$(V.19) > 1 - 2^{-2m} - 2^{-2m} = 1 - 2^{-(2m-1)} \tag{V.20}$$

   So $1 > B_i \times E_i > 1 - 2^{-(2m-1)}$. ■

   Apply to the basic PST algorithm, the product of B_1 and E_1 has $2M-1$ leading $1'$s.

3. Assume the product of B_I and E_I has m leading 1′s. If both R_{j-1} and $\tilde{Q}_j$ are truncated at $m+1$ bits, then the intermediate remainder R_j has $m-1$ leading 0′s.

   Proof : Assume δ_1 and δ_2 are the errors introduced by truncations on R_{j-1} and $\tilde{Q}_j$. $0 < \delta_1, \delta_2 < 2^{-m-1}$.

$$\begin{aligned} R_j &= R_{j-1} - B_I \times \tilde{Q}_j \\ &= R_{j-1} - B_I \times ((R_{j-1} - \delta_1) \times E_I - \delta_2) \\ &= R_{j-1} - B_I \times R_{j-1} \times E_I + B_I \times \delta_1 \times E_I + B_I \times \delta_2 \\ &= R_{j-1} \times (1 - B_I \times E_I) + B_I \times E_I \times \delta_1 + B_I \times \delta_2 \end{aligned} \tag{V.21}$$

   Known from the assumption, $(1 - B_I \times E_I) < 2^{-m}$.

$$\begin{aligned} (V.21) &< R_{j-1} \times 2^{-m} + B_I \times E_I \times \delta_1 + B_I \times \delta_2 \\ &< 2^{-m} + \delta_1 + \delta_2 \\ &< 2^{-m} + 2^{-m-1} + 2^{-m-1} = 2^{-m+1} \end{aligned} \tag{V.22}$$

   It indicates that R_j has $m-1$ leading 0′s. ■

   For the basic PST algorithm, R_j has $2M-2$ leading 0′s.

From the previous analysis, it's clear that the estimation errors are well controlled and the correctness of the PST algorithm is guaranteed.

V.B.3 Advanced PST Algorithm

We now describe a faster PST algorithm with multiple series expansion iterations. Instead of using 1-order series expansion in the basic PST algorithm, we apply high order series expansion in the advanced PST algorithm to boost the precision further. Assume $B_i = 1 - X$, then the 1-order series expansions of $1/B_i$

can be expressed as follow:

$$\begin{cases} B_i & = 1 - X \\ E_i & = 1 + X \approx \frac{1}{B_i} \end{cases} \tag{V.23}$$

If E_i is multiplied back to B_i, we get B_{i+1} and then E_{i+1}:

$$\begin{cases} B_{i+1} & = B_i \times E_i = 1 - X^2 \\ E_{i+1} & = 2 - B_{i+1} = 1 + X^2 \end{cases} \tag{V.24}$$

The product of E_i and E_{i+1} has the following formula, which is the 3-order series expansion of $1/B_i$.

$$E_i \times E_{i+1} = (1 + X)(1 + X^2) = 1 + X + X^2 + X^3 \tag{V.25}$$

Equation(V.25) demonstrates that two 1-order series expansion iterations have equivalent precision as one 3-order series expansion. 1-order series expansion can be iterated by I times. The number of precise bits of B_i inverse is doubled in each iteration. The algorithm detail is listed as follow:

1. The initial scaling factor E_0 is obtained from a look-up table. The value is the reciprocal of the up-rounded divisor B and then truncated at the $M+1$ bit. Because E_0 is an estimation of $1/B$, the product of E_0 and B is close to 1, i.e. it has M bits guaranteed leading $1'$s.

$$B_{[M+2]} = 0.b_1 b_2 \ldots b_{M+2} 111 \ldots \tag{V.26}$$

$$E_0 = trunc(1/B_{[M+2]})_{M+1} = 1.e_1^0 e_2^0 \ldots e_{M+1}^0 \tag{V.27}$$

   This step is the same as the first step in basic PST algorithm.

2. Assume $A_0 = A$ and $B_0 = B$. The current dividend A_i and the divisor B_i are calculated by scaling the previous dividend and divisor by E_{i-1} simulta-

neously.

$$\begin{cases} A_i & = A_{i-1} \times E_{i-1} \\ & = 0.a_1^i a_2^i \ldots a^i_{N+(2^i-1)M+\frac{3i-i^2}{2}} \\ B_i & = B_{i-1} \times E_{i-1} \\ & = 0.11 \ldots 1 b^i_{2^{i-1}M+2-i} \ldots b^i_{N+(2^i-1)M+\frac{3i-i^2}{2}} \end{cases} \tag{V.28}$$

3. The estimation of $1/B_i$, E_i, is the reverse of B_i truncated at the $2^iM+2-2i$ bit. If $i < I$, repeat the previous step.

$$\begin{aligned} E_i &= trunc(\bar{B}_i)_{2^iM+2-2i} \\ &= 1.00 \ldots 0 \bar{b^i}_{2^{i-1}M+2-i} \ldots \bar{b^i}_{2^iM+2-2i} \end{aligned} \tag{V.29}$$

4. The partial quotient $\tilde{Q}_j$ is calculated from the product of the truncated previous remainder R_{j-1} and E_I, and then is truncated at the $2^IM+2-2I$ bit. Initially $R_0 = A_I$.

$$\begin{aligned} \tilde{Q}_j &= trunc(trunc(R_{j-1})_{2^IM+2-2I} \times E_I)_{2^IM+2-2I} \\ &= 0.\tilde{q^j}_1 \tilde{q^j}_2 \ldots \tilde{q^j}_{2^IM+2-2I} \end{aligned} \tag{V.30}$$

5. The partial quotient is added to the previous quotient Q_j to approach the final result. The product of $\tilde{Q}_j$ and B_I is subtracted from R_{j-1}. The new remainder R_j has $2^IM - 2I$ guaranteed leading $0'$s to be shifted out. We will prove this claim in the next section. If $(2^IM - 2I) \times j < N$, repeat the previous step. Assume J is the total number of the iterations, then $J = \lceil \frac{N}{2^IM-2I} \rceil$.

$$\begin{cases} Q_j & = Q_{j-1} + 2^{-(2^IM-2I)\times(j-1)} \times \tilde{Q}_j \\ R_j & = 2^{2^IM-2I} \times (R_{j-1} - B_I \times \tilde{Q}_j) \end{cases} \tag{V.31}$$

6. The algorithm finishes when j is equal to J. The final correction step is the same as the corresponding step in the basic PST algorithm.

Compared with the basic PST algorithm, advanced PST algorithm amplifies the arithmetic effort of series expansion and increase the precision of partial quotient quadratically. It makes the advanced PST algorithm applicable to extremely high performance applications.

V.B.4 Correctness Proof of the Advanced PST Algorithm

The correctness of the advanced PST algorithm is based on the following three theories:

1. For the prescaling factor look-up table in step 1), if both the data and address bit-widthes are m, then B_1 has $m-1$ guaranteed leading $1'$s.
   This claim has been proved in B.2.1. So B_1 has M leading $1'$s in the advanced PST algorithm.

2. For every $1 \leq i \leq I$, B_i has $2^{i-1}(M-1)+1$ guaranteed leading $1'$s.
   Proof : According to the previous claim, B_1 has M leading $1'$s, which satisfies the theory. Assume B_{i-1} has $2^(i-2)(M-1)+1$ leading $1'$s. According to B.2.2, the number of leading $1'$s in $B_i = B_{i-1} \times E_{i-1}$ is

   $$2 \times (2^{i-2}(M-1)+1) - 1 = 2^{i-1}(M-1)+1$$

   Therefore this claim holds for every $1 \leq i \leq I$. ■
   When $i = I$, B_I has $2^{I-1}(M-1)+1$ leading $1'$s.

3. The intermediate remainder R_j has $2^I(M-1)$ leading $0'$s.
   Proof : The previous claim shows that B_I has $2^{I-1}(M-1)+1$ leading $1'$s. According to B.2.3, the number of leading $0'$s in R_j is

   $$2 \times (2^{I-1}(M-1)+1) - 2 = 2^I(M-1)\blacksquare \qquad \text{(V.32)}$$

Compared with the basic PST algorithm, the advanced PST algorithm can dramatically reduce the number of iterations.

V.C Parallel PST Division

While the two multiplications in the series expansion step can be parallelized easily, it is difficult to parallelize the three sequential operations in the 0-order Taylor expansion step by a traditional way. The partial quotient is calculated first, and then multiplies with the divisor B_I. The product is subtracted from the previous remainder finally. Fig. V.2 shows the data dependency in the basic PST algorithm. Each operation requires the previous result.

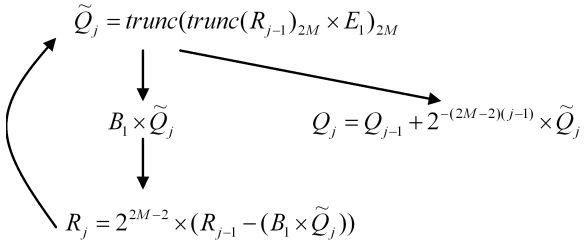

Figure V.2 Data Dependency in PST Division

We propose two methods based on the properties of data to parallelize the operations. To simplify the representation, we discuss the basic PST algorithm here. The parallelization on the advanced PST algorithm is similar.

V.C.1 Method 1

The first method is based on the property of E_1. $E_1 = 1 + X$, and $X < 2^{-M}$. The calculation for the partial quotient $\tilde{Q}_j = trunc(R_{j-1})_{2M} \times (1 + X)$ has this formula:

$$\begin{aligned} trunc(R_{j-1})_{2M} &: 0.r_0^{j-1} \ldots r_M^{j-1} r_{M+1}^{j-1} \ldots r_{2M}^{j-1} \\ trunc(R_{j-1})_{2M} \times X &: 0.0 \ldots 0 z_{M+1} \ldots z_{2M} \ldots \end{aligned} \tag{V.33}$$

From this formula, the computation for partial quotient can be rewritten as

$$\begin{aligned} [R_{j-1}]_{Mh} &= trunc(R_{j-1})_M \\ &= 0.r_0^{j-1} r_1^{j-1} \dots r_M^{j-1} \end{aligned} \tag{V.34}$$

$$\begin{aligned} [R_{j-1}]_{Ml} &= trunc(R_{j-1})_{2M} - [R_{j-1}]_{Mh} \\ &= 0.00 \dots 0 r_{M+1}^{j-1} r_{M+2}^{j-1} \dots r_{2M}^{j-1} \end{aligned} \tag{V.35}$$

$$[\tilde{Q}_j]_h = [R_{j-1}]_{Mh} \tag{V.36}$$

$$[\tilde{Q}_j]_l = trunc([R_{j-1}]_{Ml} + trunc(R_{j-1})_{2M} \times X)_{2M} \tag{V.37}$$

$$\tilde{Q}_j = [\tilde{Q}_j]_h + [\tilde{Q}_j]_l \tag{V.38}$$

The bit width of the first term $[\tilde{Q}_j]_h$ is M. The second term $[\tilde{Q}_j]_l$ has the bit width of $M+1$, and the magnitude is smaller than $2^{-(M-2)}$. Hence the product of $\tilde{Q}_j$ and B_1 can be divided to two terms with slight computation overhead.

$$\alpha_h = B_1 \times [\tilde{Q}_j]_h \tag{V.39}$$

$$\alpha_l = B_1 \times [\tilde{Q}_j]_l \tag{V.40}$$

$$R_i = R_{i-1} - \alpha_h - \alpha_l \tag{V.41}$$

Following this formula, the first term α_h is calculated directly from $[R_{j-1}]_{Mh}$ independent from the calculation of the second term α_l, as Fig. V.3 presented.

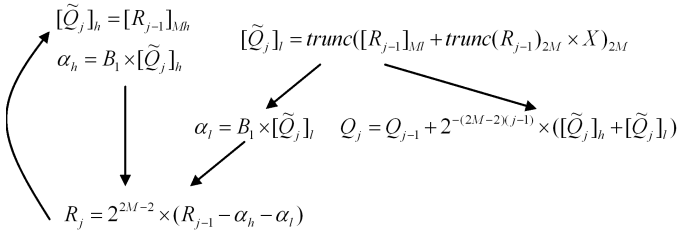

Figure V.3 Parallelize the Operations: Method 1

V.C.2 Method 2

The second method parallelizes operations between iterations. In the loop shown in Fig. V.2, B_1 cannot be truncated to minimize computation load,

because a small error of a intermediate remainder R_j in an early iteration will stay there and become to a crucial problem in some later iterations. However, the computation of partial quotient $\tilde{Q}_j$ does not need the accurate value of the previous remainder R_{j-1}. From the analysis of the third claim in Section II.C, we know that an error smaller than 2^{-2M} on R_{j-1} is acceptable for partial quotient computation. Therefore an optimization idea is to calculate an acceptable value of a remainder based on a truncated B_1, and a correction term is computed simultaneously to compensate the error in the remainder later. Therefore, the next partial quotient computation can start earlier before the exact remainder is derived.

B_1 is separated into two parts, $[B_1]_h$ and $[B_1]_l$ as following:

$$[B_1]_h = (1 - 0.00\ldots0\overline{b^1}_{M+1}\overline{b^1}_{M+2}\ldots\overline{b^1}_{4M}) \quad \text{(V.42)}$$

$$[B_1]_l = 0.00\ldots0\overline{b^1}_{4M+1}\overline{b^1}_{4M+2}\ldots\overline{b^1}_{N+M+1} + ls1^1 \quad \text{(V.43)}$$

It's not hard to verify $B_1 = [B_1]_h - [B_1]_l$. The computation of the current remainder is correspondingly changed to:

$$R_j = R_{j-1} - [B_1]_h \times \tilde{Q}_j + [B_1]_l \times \tilde{Q}_j \quad \text{(V.44)}$$

The last term is put to the correction term C. We define an estimation of R_j as R'. Hence the real remainder R_j equals the estimation R' plus the correction term C.

$$R' = R_{j-1} - [B_1]_h \times \tilde{Q}_j \quad \text{(V.45)}$$

$$C = [B_1]_l \times \tilde{Q}_j < 2^{-4M} \quad \text{(V.46)}$$

$$R_j = R' + C \quad \text{(V.47)}$$

R' is used to replace R_i for the partial quotient computation in the next iteration, and R_j must be ready before the next R' calculation. Fig. V.4 displays two unrolled iterations and data dependency relation of this optimization.

The correctness of the method 2 is proved as follow. After substitute R_j with R' in the partial quotient computation, the equation V.21 can be rewritten

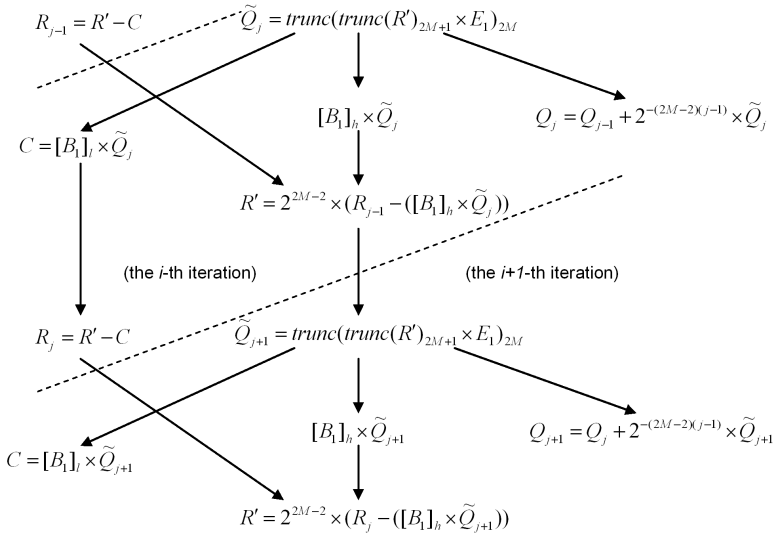

Figure V.4 Parallelize the Operations: Method 2

as:

$$\begin{aligned} R_{j+1} &= R_j - B_1 \times \tilde{Q}_{j+1} \\ &= R_j - B_1 \times ((R' - \delta_1) \times E_1 - \delta_2) \\ &= R_j - B_1 \times ((R_j - C \times 2^{-(2M-2)} - \delta_1) \times E_1 - \delta_2) \end{aligned} \tag{V.48}$$

Note that the introduction of the error term C does not really introduce any error on the computation of each intermediate remainder R_j, because C is added back before R_{j+1} is calculated in the following iteration. It only affects the computation of $\tilde{Q}_i$. Therefore according to the proof of claim 3), the algorithm is correct as long as the total error on R_j is bounded by 2^{-2M}. Known from the equation V.46, $C < 2^{-4M}$. If the truncate error δ_1 reduces to $2^{-(2M+1)}$, the total error on R_j is still less than 2^{-2M}. Hence the correctness is proved.■

The two methods split partial quotient $\tilde{Q}_j$ and scaled divisor B_1 respectively. They can be applied to the PST algorithm simultaneously. Fig. V.5 demonstrates the optimized computing architecture. Compared with the original structure in Fig. V.2, the optimized architecture divides the product $B_1 \times \tilde{Q}_j$ into

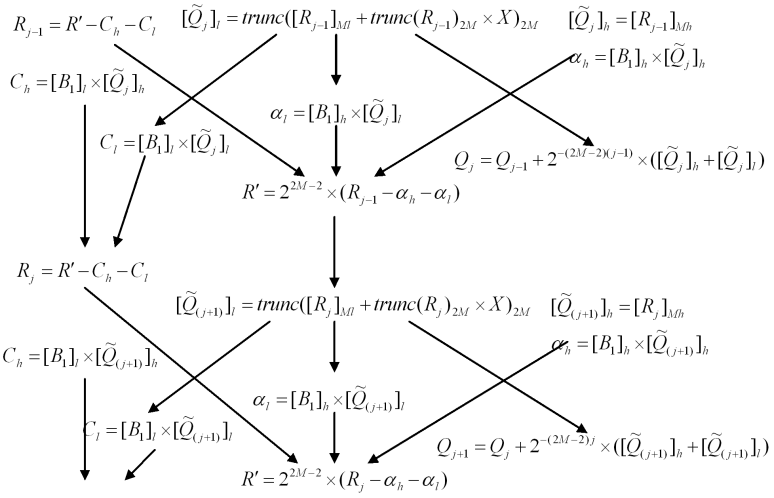

Figure V.5 Parallel PST Division Algorithm

four terms, and keeps only one term $[B_1]_h \times [\tilde{Q}_j]_l$ in the critical path. Therefore the $2M$ by $2M$ multiplication delay is reduced to an M by M multiplication delay.

V.D Evaluations and Implementations

To test the soundness of the proposed PST algorithm, we compare the PST algorithm with other existing algorithms by numerical analysis in terms of delay, area and power consumption. Furthermore, we also implement the PST algorithm on both ASIC and FPGA devices and demonstrate the benefit on real design.

V.D.1 Numerical Analysis

The analysis on the computation effort and the number of iterations gives a general idea on the tradeoff between performance and cost. However, to fully understand the relation among operation delay, quantitative analysis is necessary.

First of all, timing, area and power model should be set up. In the existing division algorithms, there are only three operations: table look-up, multiplication and binary addition. Therefore we need only three modules for them: Read-Only

Memory (ROM), Partial Product Reduction Array (PPRA), and Carry-Propagate Adder (CPA). Note that a multiplier is composed by a partial product reduction array and a carry-propagate adder.

We use Parameterized technology independent models for these modules, as used in [31]. These models only depends on the bit-width of each operands but are independent from the technology and detail design. This method can characterize the performance and cost of an algorithm before the algorithm is implemented in detail. The models for ROM, PPRA and CPA list as following:

- **ROM**: The timing and area of a memory block is determined by its address bit-width and word bit-width. For n-bit address and m-bit word, the total bits number in the memory is $2^n \times m$. And the delay is proportional to the address bit-width n.

- **PPRA**: Partial product reduction array also has two parameters, the bit-with of multiplicand n and the bit-width of multiplier m. Because it is a matrix multiplication, the total area is proportional to $n \times M$. Taking advantage of carry-save addition, the delay only depends on the number of bits in one bit slice. And carry-save addition can be preformed in a tree structure, then the delay of a PPRA is $O(\lg(m))$.

- **CPA**: Carry-propagate addition has only one parameter, the bit-width of two operands n. Usually ripple-carry adder is too slow to satisfy timing requirement. Therefore we assume prefix adders is applied for all carry-propagate additions here. Due to the tree structure of prefix adder, both the delay and the area are proportional to $\lg(n)$.

Table V.1 shows the delay and area models of ROM, PPRA and CPA. All the numbers in the table are normalized to the unit of full adder (FA).

Now the power model of each module is still missing. A rough estimation of power consumption to assume that the power consumption is linear the to area. However, this estimation does not consider dynamic power. For example, in one

Table V.1 Estimate of Delays and Areas of Basic Modules $[t_{FA}, A_{FA}]$

Module	Delay $[t_{FA}]$	Area $[A_{FA}]$
2^n by m Table	$n-2$	$1.3\times(2^{n-4}+1)\times\frac{m}{n}$
n by m PPRA	$1.5\times\lg(m)-1.5$	$1.15\times n\times m+0.7\times n$
n-bit CPA	$0.5\times\lg(n)+1.5$	$0.3\times(\lg(n)+2)\times n$

PST division, the look-up table and prescaling multiplications are only performed once, but the multiplications and additions in Taylor expansion iterations will run multiple times. Therefore, we revised the power model as the product of area and operating frequency on a module.

Based on the timing/area/power model, Fig.V.6 and Fig.V.7 demonstrate the delay-area and delay-power tradeoffs achieved by various division algorithms for 64-bit divisions. In general, larger look-up table leads to higher performance and larger area. When the table size reaches the limit, larger table size won't help to improve the performance further. At this time, arithmetic effort is effective to boost the performance further. An interesting observation is that the power consumption is not monotonous to the decreasing delay. It is because we considered dynamic power consumption in the power model. Low area leads to low precision on partial quotient, and then leads to more iterations.

Fig.V.8 and Fig.V.9 show the delay-area and delay-power tradeoffs for 128-bit divisions. In both 64-bit and 128-bit cases, PST algorithms have the best delay-area and delay-power tradeoffs. PST division can achieve very high performance division with relatively low power consumption.

V.D.2 ASIC Implementations

To confirm the estimates, we implement algorithms of 0-order Taylor expansion, $1st$-order Taylor expansion, prescaling, and PST division for 128-bit division. It follows a typical ASIC design flow. Each algorithm is described as a behavior description in Verilog code. The behavior description is then synthesized by Synopsys Design Compiler [32] on the target library TSMC tcbn90ghp. Based

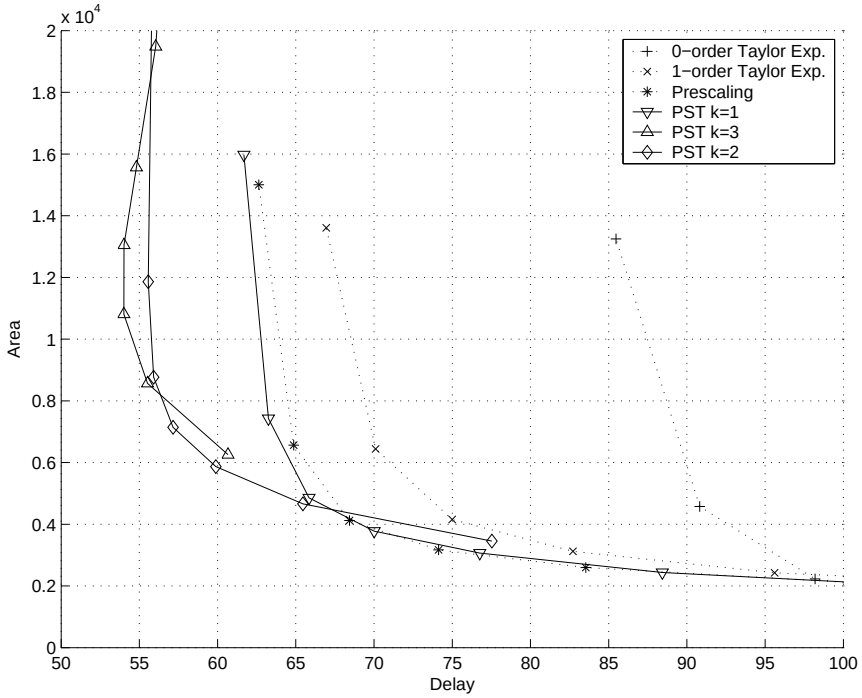

Figure V.6 Delay-Area Tradeoff (64bit)

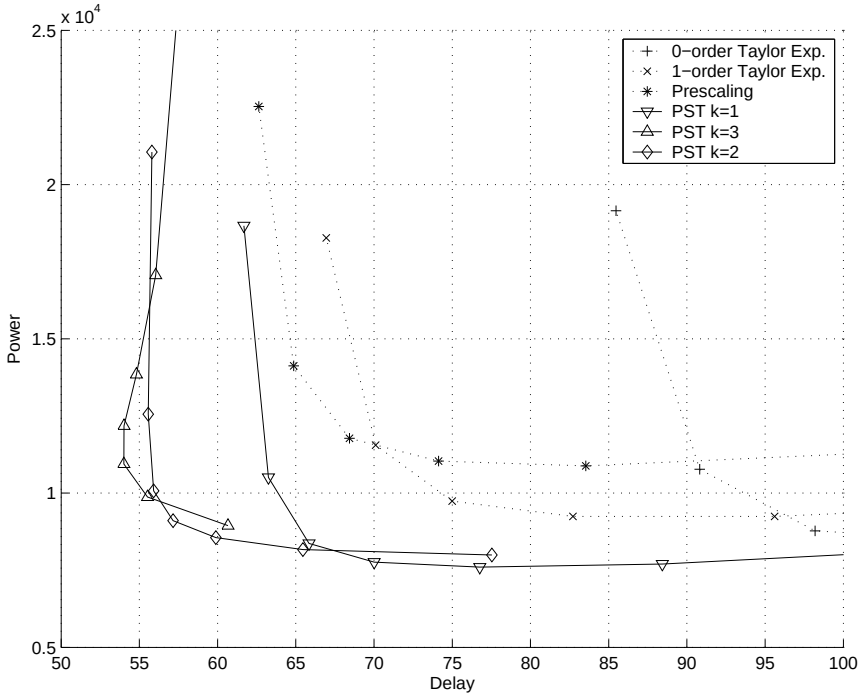

Figure V.7 Delay-Power Tradeoff (64bit)

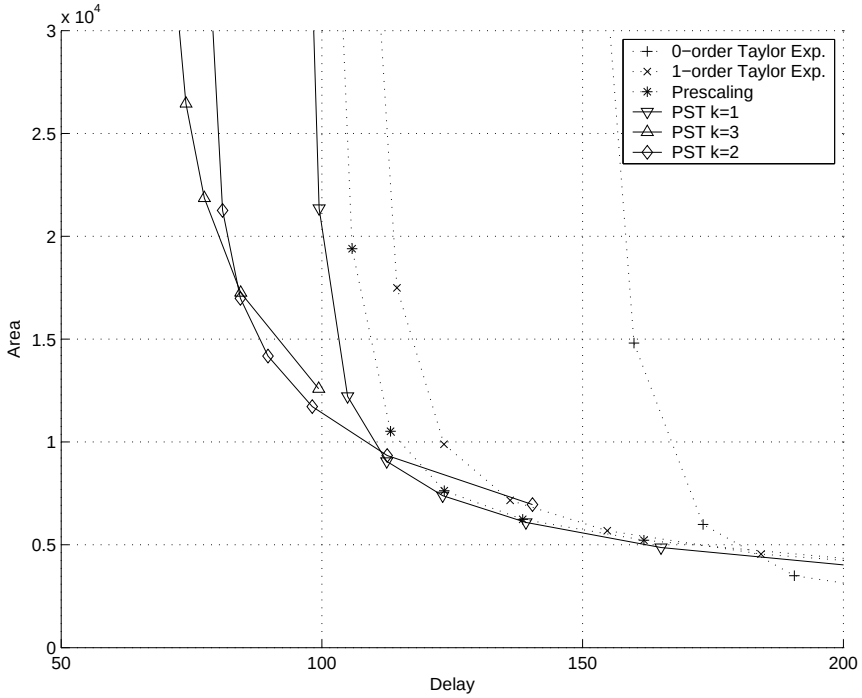

Figure V.8 Delay-Area Tradeoff (128bit)

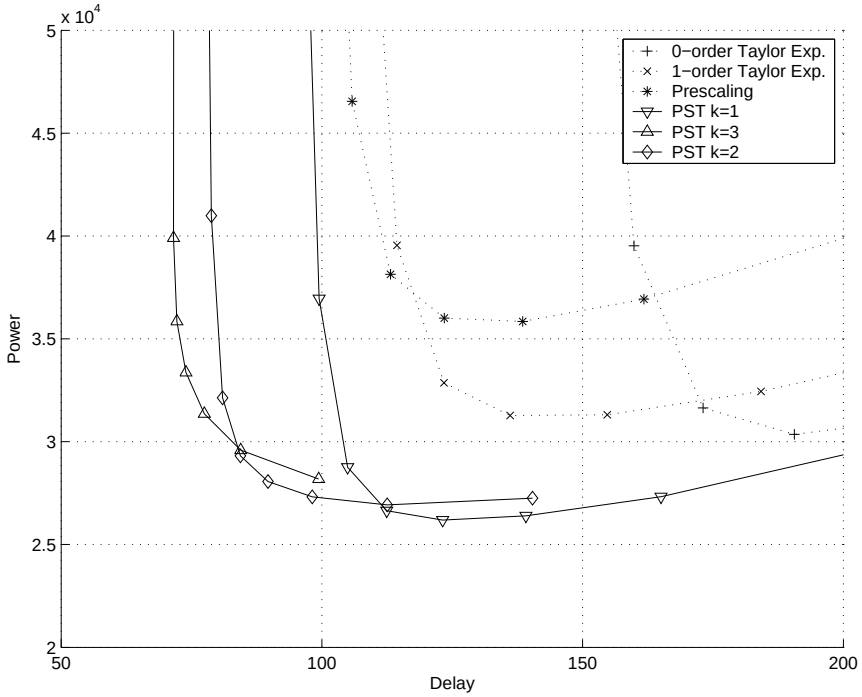

Figure V.9 Delay-Power Tradeoff (128bit)

on the synthesized net-list, Synopsys Power Compiler [33] analyzes and report then delay, area and power of the divider unit. The total execution time and power consumption for one division are calculated by multiplying delay and power to the number of iterations, which is also the number of stall cycles of the divider unit.

Table V.2 ASIC Implementations

	Delay(ns)	Area	Power(mW)
0-order Taylor Exp.			
(2K Table)	253.23	55182	455.4
(10K Table)	196.52	66827	370.6
1-order Taylor Exp.			
(3K Table)	176.67	97632	552.5
(15K Table)	145.00	122515	524.0
Prescaling			
(2K Table)	260.36	101343	779.7
(10K Table)	185.64	121701	654.5
PST (I=1)			
(2K Table)	105.71	133857	316.8
(10K Table)	94.23	161116	296.1

Table V.2 shows the implementation results on different division algorithms. The comparison of the implementation results quite meets with the estimate prediction. The PST algorithm with 10K table achieves both minimal execution time and minimal power consumption.

V.D.3 FPGA Implementations

Besides ASIC, we also implement the PST algorithm in an Altera StratixII FPGA device [34], and compare the implementation with a division IP core in terms of clock frequency, resource consumption, power consumption and throughput. We choose Altera StratixII EP2S15F484C3 as the target FPGA device, which contains 12480 ALUTs, 420KB memory and 96 DSP elements. Both the PST algorithm and the division IP core generated by MegaWizard are described in verilog format. A development environment, Altera quartusII 5.0 [35], performs the whole

design flow from logic synthesis, fitting, assembling to timing analysis and power analysis. The function correctness is verified by Modelsim-Altera. One 32-bit division is finished in 5 cycles, while the target frequency is set to 100MHz. All inputs and outputs are registered. Positive remainder is not necessary.

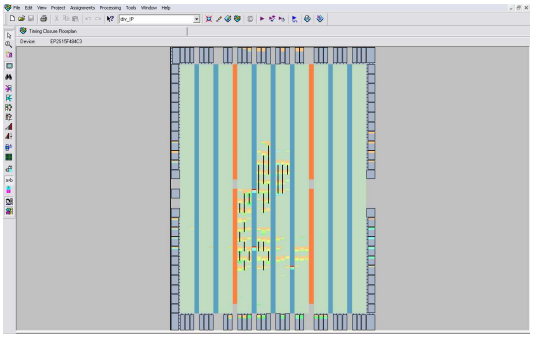

Figure V.10 IP core

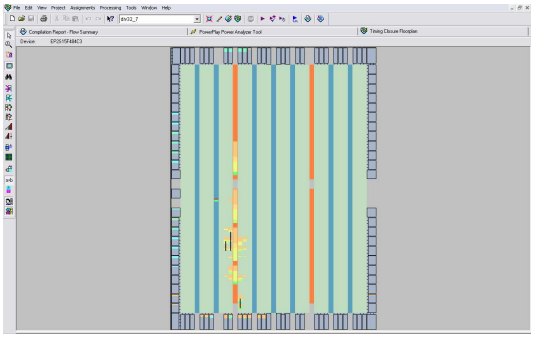

Figure V.11 PST with DSP blocks

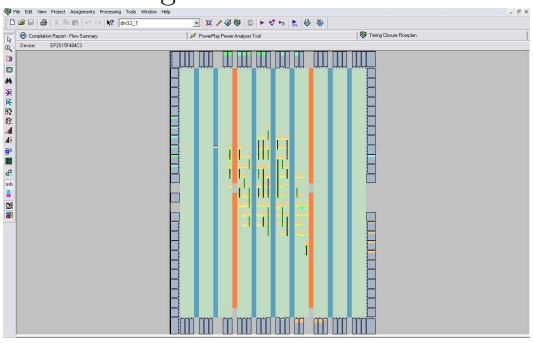

Figure V.12 PST without DSP block

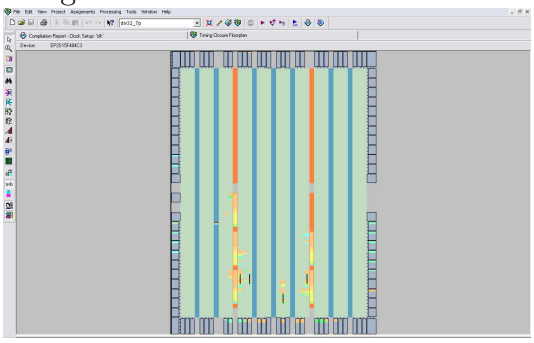

Figure V.13 PSTp with DSP blocks

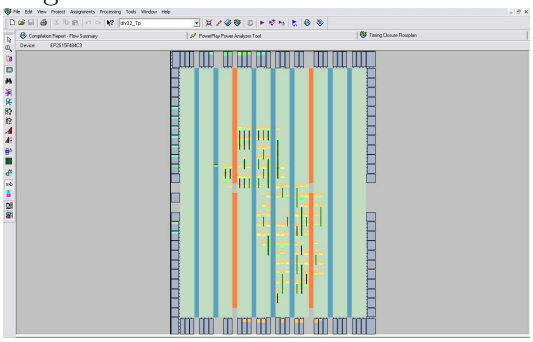

Figure V.14 PSTp without DSP block

For the parameters of PST algorithm, M is set to 6. Therefore the table size is 7×2^7, and for each Taylor expansion iteration 12 leading $0'$s can be guar-

anteed in the intermediate remainder. Hence three Taylor expansion iterations are needed to finish one operation. When two successive divisions feed to the division unit, 2 stall cycles have to be inserted between them. To avoid the stall, we unroll the PST algorithm and implement a fully pipelined version. It's called the *PSTp* division unit. Two synthesis configurations are applied for the each design. One is using DSP blocks, and the other is to avoid DSP blocks. Thus we have four versions of PST implementations: PST/PSTp division unit with DSP blocks, which uses build-in multipliers in the FPGA device, and PST/PSTp division unit without DSP block, which uses ALUTs to construct multipliers. The division IP core is a non-DSP block version by nature. Fig. V.10 to Fig. V.14 demonstrate the implementation results.

Table V.3 FPGA Implementations

	Fmax (Period)	ALUTs	Memory	DSP	Power (Dyn+Sta)	Throughput
IP core (no DSP)	50.16MHz (19.935ns)	1203	84	0	381mW (52+329)	50.16M div/s
PST (DSP)	72.8MHz (13.737ns)	213	768	28	350mW (23+327)	24.3M div/s
PST (no DSP)	73.20MHz (13.661ns)	1437	768	0	378mW (50+328)	24.4M div/s
PSTp (DSP)	74.15MHz (13.486ns)	261	768	40	344mW (17+327)	74.15M div/s
PSTp (no DSP)	76.05MHz (13.150ns)	1940	768	0	359mW (31+328)	76.05M div/s

Table V.3 summarizes the max clock frequency, resource consumption, power consumption and throughput in terms of number of division operations per second for each implementation. The comparison shows that the PST algorithm achieves about 34% delay improvement. Because both the PST unit and the IP core take 5 cycles to finish one division, the delay improvement not only increases the clock frequency but also reduces the execution timing for each operation. While the PST division unit produces one division result in every three clock cycles, the

fully pipelined PSTp division unit can avoid stall cycles. Therefore the PSTp division has the largest throughput among the compared designs. An interesting observation is that the introduction of DSP block does not help much to reduce delay. One reason is that although build-in multipliers is faster, the locations of DSP blocks limit the placement flexibility, which leads to larger wire delay. The PST algorithm also shows the advantage on dynamic power consumption. Without DSP blocks the PSTp division unit reduces the dynamic power by 38% to the division IP core. With the help of DSP elements, the PSTp division unit achieves 67% saving on power. The power estimation is based on the toggle rates statistics. The initial toggle rates and static possibilities are obtained from timing simulation, where the clock frequency is set to $50MHz$.

V.E Summary

In this Chapter, we propose a novel division algorithm, PST division, which applies prescaling and series expansion to finely combine memory effort, arithmetic effort and iteration effort together. The PST algorithm is further parallelized to improve performance without significant overhead. Based on the numerical analysis and implementation results, the PST algorithm achieve best timing-power tradeoffs among the existing division algorithms for ASIC design. In FPGA applications, The PST algorithm also shows significant delay and power reduction comparing with current division IP core on Altera StratixII FPGAs.

This chapter has been submitted for publication of the material as it appears in ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays 2006, Liu, Jianhua; Zhu, Haikun; Cheng, Chung-Kuan. The dissertation author was the primary investigator and single author of this paper.

VI

Conclusions

With the technology development and the increasing demand on VLSI, data-path design is facing the challenges from high performance and low power consumption. In this thesis two optimization methods on data design have been proposed. The research items and results of this work can be summarized as follows.

The first section of chapter III introduces a comprehensive area/power/timing model. Compared with the idealistic model used in previous works, the new model can capture the key characters of CMOS circuit, especially the effect of physical design. Based on the model, chapter III also propose an Integer Linear Programming method to build optimal prefix adder with minimal power consumption. By keeping the linear relation from decision variables to power objective, the ILP formulation can be solved efficiently. The ILP method not only can handle non-uniform input arrival and output required time, but also support gate sizing and buffer insertion. The experimental results show a great flexibility and significant power saving comparing with several classical prefix adders.

In chapter IV, we propose a new method to analyze the computation effort in division algorithm. Computation effort in division is partitioned to iteration effort, memory effort and arithmetic effort. Iteration effort provides the lower bound of computation effort of division and keeps constant for given bit-width.

Memory effort can increase the precision of partial quotient by current number of bits. But memory effort is limited by memory wall due to its exponential cost. Arithmetic effort can multiple the bit-width of partial quotient with polynomial cost.

Chapter V presents a hybrid division algorithm which combines prescaling, series expansion and 0-order Taylor expansion together. The proposed PST algorithm fully utilize the advantages of memory effort, arithmetic effort and iteration effort, and achieves optimal timing-power tradeoff among existing division algorithms. This algorithm is not only applicable to ASIC division unit design, but also very suitable for modern FPGA devices. The implementation on Altera StratixII demonstrates that the PST algorithm can take advantage of the built-in multipliers and memory blocks in the device, and realize high-performance and low power division unit.

Bibliography

[1] The IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754)

[2] C. Wallace, "A Suggestion for a Fast Multiplier." IEEE Trans. on Electronic Computers, vol. 13, pp. 14-17, 1964

[3] A.D. Booth, "A Signed Binary Multiplication Technique." Quarterly J. Mechanical and Applied Math., vol. 4, pp. 236-240, 1951.

[4] O.L. MacSorley, "High Speed Arithmetic in Binary Computers." Proc. IRE, vol. 49, pp. 67-91, 1961.

[5] T. Kilburn, D. Edwards, D. Aspinall, "Parallel addition in digital computers: A new fast carry circuit", Proc of IEE, vol. 106, pt. B, pp. 464-466, 1959.

[6] M. Lehman, N. Burla, "Skip Techniques for high-speed carry-propagation in binary arithmetic circuits", IRE Trans. Electron. Comput., pp.691-698, Dec 1961.

[7] V. Oklobdzija, E. Barnes, "Some optimal schemes for ALU implementation in VLSI technology", Proceedings of 7th Symposium on Computer Arithmetic (Cat. No. 85CH2146-9). IEEE Comput. Soc. Press., pp.2-8. Silver Spring, MD, USA, 1985.

[8] O. Bedrij, "Carry select adder", IRE Transactions on Electronic Computers, vol. 11, pp. 340C346, 1962.

[9] J. Sklansky, "Conditional-sum addition logic", IRE Transactions on Electronic Computers, vol. 9, pp. 226C231, 1960.

[10] A. Weinberger, J. Smith, "A One-Microsecond Adder Using One Megacycle Circuitry", IRE Transactions on Electronic Computers, pp. 65-73, 1956.

[11] R. Ladner, M. Fischer, "Parallel prefix computation", Journal of the Association for Computing Machinery, vol.27, no.4, pp.831-8. Oct. 1980.

[12] P. Kogge, H. Stone, "A parallel algorithms for the efficient solution of a general class of recurrence equations", IEEE Transactions on Computers, vol.C22, no.8, pp.786-93, Aug. 1973.

[13] R. Brent, H. Kung, "A regular layout for parallel adders", IEEE Transactions on Computers, vol.C-31, no.3, pp.260-4, March 1982.

[14] B. Sugla, D. Carlson, "Extreme area-time tradeoffs in VLSI", IEEE Transactions on Computers, vol.39, no.2, pp.251-7, Feb. 1990.

[15] J. Fishburn, "A depth-decreasing heuristic for combinational logic; or how to convert a ripple-carry adder into a carry-lookahead adder or anything in-between", 27th ACM/IEEE Design Automation Conference Proceedings, pp.361-4, 1990.

[16] R. Zimmermann R, "Non-Heuristic Optimization and Synthesis of Parallel-Prefix Adders", Proc. Int. Workshop on Logic and Architecture Synthesis (IWLAS'96), pp. 123-132, Dec. 1996.

[17] I. Sutherland, B. Sproull, D. Harris, "Logical Effort: Designing Fast CMOS Circuits", Morgan Kaufmann Publishers, 1999.

[18] Vanichayobon S, Dhall S, Lakshmivarahan S, Antonio J, "Power-speed Trade-off in Parallel Prefix Circuits", Proceeding of SPIE Vol. 4863, pp.109 - 120, 2002.

[19] S. Mathew, M. Anders, R.K. Krishnamurthy, S. Borkar, "A 4-GHz 130-nm Address Generation Unit With 32-bit Sparse-Tree Adder Core", IEEE Journal of Solid-State circuits, Vol38, No.5, May 2003.

[20] J. Robertson "A New Class of Digital Division Methods", IRE Trans. on Electronic Computers, vol. 7, pp. 218-222, 1958.

[21] T. Tocher, "Techniques of Multiplication and Division for Automatic Binary Computers", Quarterly J. Mech. App. Math., vol. 2, pt. 3, pp. 364-384, 1958

[22] A. Svoboda, "An Algorithm for Division", Information Processing Machines, vol. 9, pp. 183-190, 1963.

[23] C. Tung, "A Division Algorithm for Signed-Digit Arithmetic", IEEE Trans. on Computers, vol.17, pp. 887-889, 1968.

[24] T. Coe, P. Tang, "It takes six ones to reach a flaw [Pentium processor]", Computer Arithmetic, 1995., Proceedings of the 12th Symposium on 19-21, pp. 140-146, July 1995.

[25] P. Hung, H. Fahmy, O. Mencer, M. Flynn, "Fast Division Algorithm with a Small Lookup Table", Conference Record of the 33rd Asilomar conference on Signals, Systems, and Computers, IEEE. Part vol. 2, pp. 1465-1468, 1999.

[26] D. Wong, M. Flynn, "Fast Division Using Accurate Quotient Approximations to Reduce the number of Iterations", IEEE Trans. on Computers, vol. 41, NO. 8, pp. 981-995, 1992.

[27] S. Oberman, M. Flynn, "Division Algorithms and Implementations", IEEE Trans. on Computers, vol. 46, NO. 8, pp. 833-854, 1997.

[28] R. Goldschmidt, "Applications of Division by Convergence", MS thesis, Dept. of Electrical Eng., Massachusetts Inst. of Technology, Cambridge, Mass., June 1964.

[29] S. Oberman, "Floating Point Division and Square Root Algorithms and Implementation in the AMD-K7 Microprocessor", Proc. 14th IEEE Symp. Computer Arithmetic, I. Koren and P. Kornerup, eds., pp. 106- 115, Apr. 1999.

[30] P. Markstein, "Computation of Elementary Functions on the IBM RISC System/6000 Processor", IBM J. Research and Development, vol. 34, no. 1, pp.111-119, Jan. 1990.

[31] P. Montuschi, T. Lang, "Boosting Very-High Radix Division with Prescaling and selection by Rounding", IEEE Trans. on Computers, vol 50, NO. 1, pp. 13-27, 2001.

[32] http://www.synopsys.com/products/logic/design compiler.html

[33] http://www.synopsys.com/products/solutions/galaxy/power/power.html

[34] http://www.altera.com/products/devices/stratix2/st2-index.jsp

[35] http://www.altera.com/products/software/products/quartus2/qts-index.html