國立交通大學

電子工程學系 電子研究所

碩 士 論 文

高速及面積最小化之可組態加法器設計

High-Speed Area-Minimized Reconfigurable Adder Design

研 究 生：馮翊展

指導教授：黃俊達 博士

中 華 民 國 九 十 五 年 七 月

高速及面積最小化之可組態加法器設計

High-Speed Area-Minimized Reconfigurable Adder Design

研 究 生：馮翊展　　Student: Yi-Zeng Fong
指導教授：黃俊達 博士　　Advisor: Dr. Juinn-Dar Huang

國立交通大學

電子工程學系　電子研究所

碩士論文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical & Computer Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Electronics Engineering & Institute of Electronics

July 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年七月

高速及面積最小化之可組態加法器設計

研究生：馮翊展　　　　　　指導教授：黃俊達　博士

國立交通大學

電子工程學系 電子研究所碩士班

摘　　要

在數位電路系統設計中，加法是最基本的算術運算之一。因此過去有多種演算法及架構被提出來用以符合不同的設計需求。採用不同的加法器架構像是進位選擇(carry-select)式、平行前綴(parallel-prefix)式、和進位提前預知(carry-lookahead)式會有不同的面積、速度、還有功耗表現。一般而言，如果想要得到較佳的速度及對後段製程實現較容易的電路結構，則Kogge-Stone平行前綴加法器架構是一套不錯的解決方案。在進位的產生方面，我們提出的架構利用了Ling加法器去縮減一個邏輯閘的延遲時間。而且我們用混合式平行前綴(hybrid parallel-prefix)/進位選擇(carry-select)架構及一些特殊的功能元件用以縮減整個加法器的面積。從實驗的結果可看出我們的新架構比傳統Kogge-Stone平行前綴加法器面積縮減了25%。近來, 多媒體已在我們生活當中扮演了一個重要角色。在處理多媒體訊號方面需要一個可即時調整成處理不同精準度運算的高速可組態加法器。然而，為了達到可組態所使用的切割架構(partition scheme)是需要一些額外的負擔。因此，我們也提出了經由修改本來的架構但卻不需付出太多額外面積及延遲時間的新可組態式架構。從實驗數據可看出我們只需要多增加5.12%延遲時間及3.98%面積就可完成新的可組態加法器。簡而言之，我們所提出的加法器能在不影響速度下又盡可能的去縮減面積，且又容易拓展成可組態架構。

High-Speed Area-Minimized Reconfigurable Adder Design

Student:Yi-Zeng Fong Advisor: Dr. Juinn-Dar Huang

Department of Electronics Engineering & Institute of Electronics
National Chiao Tung University

Abstract

Binary addition is one of the fundamental arithmetic operations in digital system design. Consequently, several adder architectures have been proposed to meet different design requirements in the past. Various architectures like carry-select, parallel-prefix, and carry-lookahead lead to different performance among area, delay, and power. In general, Kogge-Stone parallel-prefix adders provide a good solution to optimize delay and regular structure for VLSI implementation. The proposed architecture uses Ling addition to reduce one logic level delay in parallel-prefix structure for the carry generation. Furthermore, using hybrid parallel-prefix/carry-select architecture and some special function blocks can reduce overall area. Experimental results reveal that the proposed architecture achieves 25% area reduction when compared to traditional Kogge-Stone parallel-prefix adders. Recently, the multimedia plays an important role in our life. Multimedia signal processing usually needs a fast reconfigure adder, which can be run-time reconfigured to handle the operations with different precisions. However, the extra overhead of partition scheme for the purpose of reconfigurability is unavoidable. Therefore, we present a new reconfigurable approach by modifying our original architecture without introducing significant extra area and timing penalty. Finally, experimental results show that the new reconfigurable adder needs only 5.12% delay penalty and 3.98% area penalty. In brief, the proposed adders do our utmost to reduce area without affecting speed and extent to reconfigurable scheme easily.

誌　　謝

首先, 我要感謝我的指導教授-黃俊達博士, 在碩士兩年當中給我的支持和鼓勵, 讓我能有良好的研究環境, 在自由的學習風氣之下, 培養出獨立研究的能力, 又能隨時給予寶貴的意見及指導, 對老師的感激之情, 並非以簡短的文句可以表達。

當然也要感謝養育我長大的父母對我的栽培, 沒有他們, 就沒有今日的我。接著要感謝的是所有來參與口試的所有教授們, 張添烜教授和黃稚存教授,百忙當中抽空前來指導我, 讓我受益匪淺, 也讓我得到了寶貴的經驗, 謝謝你們。

最後也要感謝所有實驗室的同仁們, 阿光、維聖、士祐、孝恩、詠翔學長、丁之暉、許哲霖、李南興學弟們, 跟大家一起修課、做實驗、和討論及分析研究成果更是我人生旅途中一段最值得珍惜的回憶, 希望未來在讀書或工作還有機會一起努力。

希望這篇論文能對人類社會有小小的貢獻, 如此一來在辛苦也就值得了, 再次謝謝大家的幫忙。

Contents

List of Tables

List of Figures

Chapter 1 Introduction

Binary addition is the primitive operation in computer arithmetic. A high-speed and area-minimized architecture for binary addition is the critical element for designing a high-performance digital IC. Therefore, a systematic design methodology for the new architecture which can meet the requirements of high performance and small area cost has been proposed in the thesis.

1.1 Motivation

Various architectures for the binary addition have been proposed in [1-13]. In general, the carry-select [5], carry-lookahead [6], parallel-prefix [7-11], Ling [12] are the most common adders used to meet different design requirements. Parallel-prefix adders provide a highly efficient solution to the fast binary addition and regular VLSI implementation from the use of simple logic cells and low complexity of wiring. Meanwhile, the Ling adder offers a simplified carry-computation to gain the improvement of speed. In [13], a new architecture which combines the Ling and the parallel-prefix algorithms has been proposed. The parallel-prefix Ling adders [13] preserve the benefits over the traditional parallel-prefix carry-computation and offer reduced delay and fanout requirements. However, this architecture needs more area than the traditional parallel-prefix adder for achieving the purpose of high-speed. Even though the hybrid parallel-prefix / carry-select Ling architecture is used, the extra area overhead is still significant when compared to the hybrid structure which uses the traditional parallel-prefix algorithm. To solve the problem of significant area penalty, a new architecture has been proposed in this thesis. The proposed architecture preserves the property of high-speed by using Ling addition, while, at the same time, minimizes the overall area as far as possible.

In our daily life, multimedia devices require real-time audio and video signals processing. The algorithms used in the digital signal processing usually need large computations of multiple addition or multiplication. Therefore, SIMD (single instruction, multiple data) architecture is useful for these computation-hungry functions by calculating data in parallelism. To achieve these requirements, efficient reconfigurable computational elements such as reconfigurable adders are needed. For example, a reconfigurable 32-bit adder can execute one 32-bit, two 16-bit, or four 8-bit additions depending on user requirements. Several structures [14-19] have been proposed to achieve real-time processing of media signals. Therefore, a high-speed area-minimized reconfigurable adder is also presented in this thesis. In the proposed architecture, the additional area cost and timing penalty for the partition scheme are very small. In other words, the new reconfigurable adders can be generated from the original architecture with minor changes. Finally, a high-speed area-minimized hybrid Ling adder with/without reconfigurability can be implemented by the systematic methodology presented in the later chapters.

1.2 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 gives a brief description of the parallel-prefix adders, basic definitions of Ling addition, parallel-prefix Ling adders, and the methods of handling the carry-in signals. Chapter 3 introduces the previous approaches of the hybrid architecture, the detailed description of the proposed adder design, and the related experimental results. Chapter 4 describes the design methodology to build the proposed architecture with reconfigurability and corresponding experimental results. Chapter 5 presents the RTL code generator derived from the design steps described in Chapter 3 and Chapter 4. In the end, Conclusions and future works are drawn in Chapter 6.

Chapter 2 Preliminaries

In this chapter, we give a brief description of equations and notation about our proposed architecture. This background knowledge contains the basic concepts required to understand our proposed design.

2.1 Parallel-Prefix (P-P) Addition

To consider the addition of two N-bit binary numbers, A = $a_{N-1}a_{N-2}\ldots a_0$ and B = $b_{N-1}b_{N-2}\ldots b_0$. S = $S_{N-1}S_{N-2}\ldots S_0$ denotes the sum of the two binary number. A parallel-prefix addition can be partitioned into three stages. The first stage computes bit-generate (g_n) and bit-propagate (p_n), and half-sum bit (d_n). For every n ($0 \le n \le N-1$), these signals can be computed by the following equations :

$$g_n = a_n \,\&\, b_n,\ p_n = a_n \mid b_n,\ d_n = a_n \wedge b_n \tag{1}$$

The second stage is the parallel-prefix tree for computing the carry signals (C_n) by using the associative operator ○. The associative operator ○ is defined as :

$$(g_n, p_n) \circ (g_m, p_m) = (g_n \mid (p_n \,\&\, g_m), p_n \,\&\, p_m) \tag{2}$$

We can group these pairs of generate and propagate bits (g_n , p_n) to generate group term ($G_{m:n}$, $P_{m:n}$) by using associative operator ○ consecutively :

$$(G_{m:n}, P_{m:n}) = (g_m, p_m) \circ (g_{m-1}, p_{m-1}) \circ (g_{m-2}, p_{m-2}) \circ .. \circ (g_{n-1}, p_{n-1}) \tag{3}$$

Follow the above definitions, each carry C_n is equal to $G_{n:0}$. At the final stage, the sum bits (S_n) can be computed according to the following equation:

$$S_n = d_n \wedge C_{n-1} \tag{4}$$

Several architectures have been proposed to compute the group term ($G_{m:n}$, $P_{m:n}$) in the second stage for different goals of design. Fig. 1 presents the 8-bit parallel-prefix adder which is proposed by Kogge and Stone. Fig. 2 presents the 8-bit parallel-prefix adder which is proposed by Lander and Fisher. There are three basic cells to construct the parallel-prefix. The diamond-type node ◇ represents the logic cell for the generation of g_n, p_n, and d_n. The black node ● represents the logic cell for the associative operator. The white node ○ represents the buffer node. The ▨ node represents the logic cell for the generation of sum bit. All of the function cells are shown in Fig. 3.

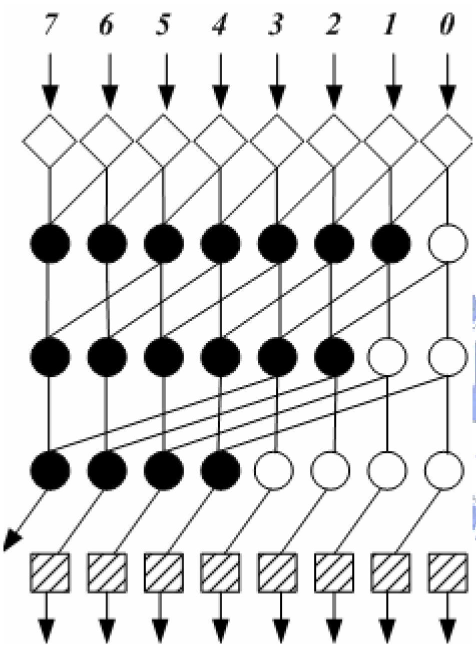


Fig. 1. An 8-bit Kogge-Stone adder.

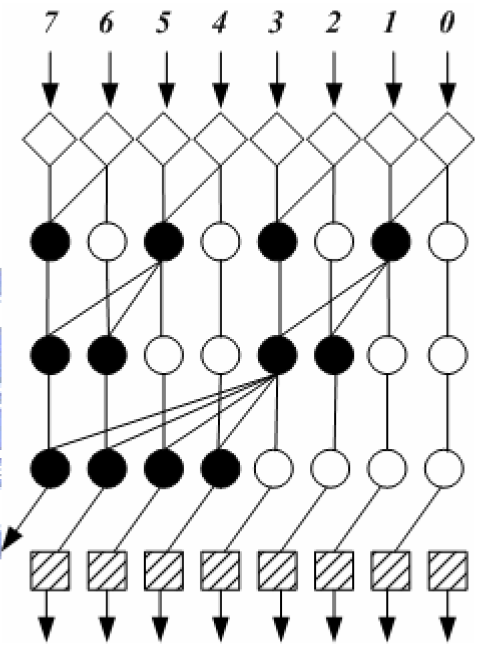


Fig. 2. An 8-bit Lander-Fisher adder.

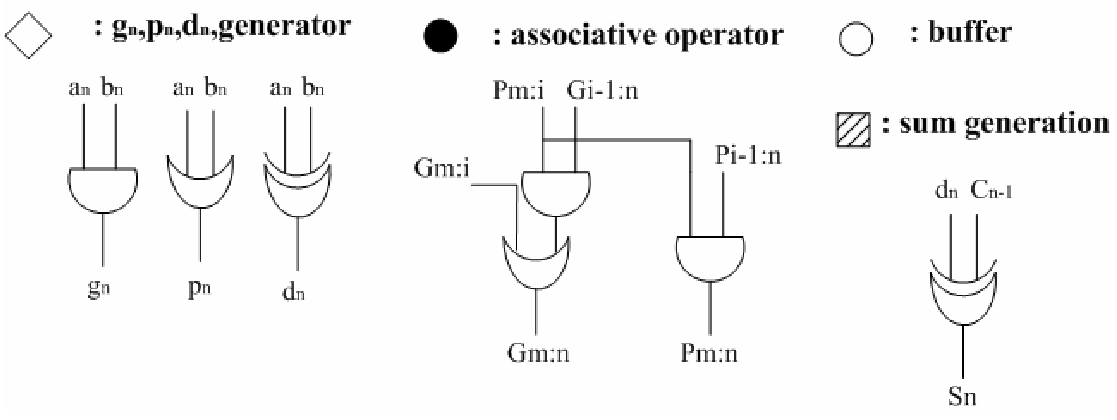


Fig. 3. Overview of function cells.

2.2 Ling Addition

According to the architecture proposed by Ling [12], a pseudo-carry signal (H_n) can be expressed as :

$$
\begin{aligned}
H_n &= C_n + C_{n-1} \\
&= G_{n:0} + G_{n-1:0} \\
&= (g_n + p_n \& g_{n-1} + p_n \& p_{n-1} \& g_{n-2} + \ldots + p_n \& p_{n-1} \& p_{n-2} \& \ldots \& p_1 \& g_0) + \\
&\quad (g_{n-1} + p_{n-1} \& g_{n-2} + \ldots + p_{n-1} \& p_{n-2} \& \ldots \& p_1 \& g_0) \\
&= g_n + g_{n-1} + p_{n-1} \& g_{n-2} + \ldots + p_{n-1} \& p_{n-2} \& \ldots \& p_1 \& g_0
\end{aligned}
$$

(5)

A carry signal (C_n) also can be expressed as :

$$
\begin{aligned}
C_n &= G_{n:0} \\
&= (\quad g_n \quad + p_n \& g_{n-1} + p_n \& p_{n-1} \& g_{n-2} + \ldots + p_n \& p_{n-1} \& p_{n-2} \& \ldots \& p_1 \& g_0) \\
&= (p_n \& g_n + p_n \& g_{n-1} + p_n \& p_{n-1} \& g_{n-2} + \ldots + p_n \& p_{n-1} \& p_{n-2} \& \ldots \& p_1 \& g_0) \\
&= p_n \& (g_n + g_{n-1} + p_{n-1} \& g_{n-2} + \ldots + p_{n-1} \& p_{n-2} \& \ldots \& p_1 \& g_0) \\
&(based\ \ on\ p_n \& g_n = g_n)
\end{aligned}
\tag{6}
$$

From Equation(5) and Equation(6), the relationship between H_n and C_n is :

$$
\begin{aligned}
C_n &= p_n \& (g_n + g_{n-1} + p_{n-1} \& g_{n-2} + \ldots + p_{n-1} \& p_{n-2} \& \ldots \& p_1 \& g_0) \\
&= p_n \& H_n
\end{aligned}
\tag{7}
$$

Therefore, a parallel-prefix adder using Ling addition, one row of carry-merge gates ($G_{n:n-1}$) can be replaced with OR gates, which produce the pseudo-carry signal (H_n). As a result, the calculation of H_n can save one logic level delay compared to the traditional generation of carry signal (C_n) in parallel-prefix tree. Finally, the generation of sum bit (S_n) should be modified to meet the pseudo-carry signal (H_n). The sum bit (S_n) derived from H_n can be computed as :

$$
\begin{aligned}
S_n &= d_n \wedge C_{n-1} \\
&= d_n \wedge (p_{n-1} \& H_{n-1}) \\
&= \overline{H}_{n-1} \& d_n + H_{n-1} \& (d_n \wedge p_{n-1})
\end{aligned}
\tag{8}
$$

From Equation(8), we replace XOR gates with 2-1 multiplexers for the generation of sum bits. A systematic methodology that allows the parallel-prefix computation of Ling carries (H_n) is presented in [13]. First, we define two symbols G_n* and P_n* as the following equations :

$$G_n{*} = g_n + g_{n-1} \tag{9}$$

$$P_n{*} = p_{n-1} \,\&\, p_{n-2} \tag{10}$$

If the addition has no carry-in, we define $g_{-1} = p_{-1} = 0$ and $g_n = p_n = 0$, for $n < -1$. In the following, we use an 8-bit adder as an example. According to Equation(5) and g_n & $p_n = g_n$, the Ling carries at the fourth bit position are expressed as :

$$\begin{aligned} H_4 &= g_4 + g_3 + p_3 \,\&\, g_2 + p_3 \,\&\, p_2 \,\&\, g_1 + p_3 \,\&\, p_2 \,\&\, p_1 \,\&\, g_0 \\ &= (g_4 + g_3) + (p_3 \,\&\, p_2) \,\&\, (g_2 + g_1) + (p_3 \,\&\, p_2) \,\&\, (p_1 \,\&\, p_0) \,\&\, g_0 \end{aligned} \tag{11}$$

From Equation(9) and Equation(10), Equation(11) can be rewritten as :

$$\begin{aligned} H_4 &= (g_4 + g_3) + (p_3 \,\&\, p_2) \,\&\, (g_2 + g_1) + (p_3 \,\&\, p_2) \,\&\, (p_1 \,\&\, p_0) \,\&\, g_0 \\ &= G_4{*} + P_4{*} \,\&\, G_2{*} + P_4{*} \,\&\, P_2{*} \,\&\, G_0{*} \\ &(\textit{based on } G_0{*} = g_0 + g_{-1} = g_0 + 0 = g_0) \end{aligned} \tag{12}$$

Using the associative operator ○, Equation(12) can be expressed as :

$$\begin{aligned} H_4 &= G_4{*} + P_4{*} \,\&\, G_2{*} + P_4{*} \,\&\, P_2{*} \,\&\, G_0{*} \\ &= (G_4{*}, P_4{*}) \circ (G_2{*}, P_2{*}) \circ (G_0{*}, P_0{*}) \end{aligned} \tag{13}$$

As a result, each H_n in an 8-bit adder can be derived using associative operator ○ and (G_n*,P_n*) pairs as follows :

$$\begin{aligned} H0 &= (G_0{*}, P_0{*}) \\ H2 &= (G_2{*}, P_2{*}) \circ (G_0{*}, P_0{*}) \\ H4 &= (G_4{*}, P_4{*}) \circ (G_2{*}, P_2{*}) \circ (G_0{*}, P_0{*}) \\ H6 &= (G_6{*}, P_6{*}) \circ (G_4{*}, P_4{*}) \circ (G_2{*}, P_2{*}) \circ (G_0{*}, P_0{*}) \\ H1 &= (G_1{*}, P_1{*}) \\ H3 &= (G_3{*}, P_3{*}) \circ (G_1{*}, P_1{*}) \\ H5 &= (G_5{*}, P_5{*}) \circ (G_3{*}, P_3{*}) \circ (G_1{*}, P_1{*}) \\ H7 &= (G_7{*}, P_7{*}) \circ (G_5{*}, P_5{*}) \circ (G_3{*}, P_3{*}) \circ (G_1{*}, P_1{*}) \end{aligned}$$

From the example, the parallel-prefix Ling adder needs an additional logic level to generate (G_n^*,P_n^*) . But, the computation of the pseudo carry (H_n) can reduce one level of associative operation compared to the traditional carry (C_n). In other words, we replace one level of associative operation (2 logic levels) with the new one logic level to generate (G_n^*,P_n^*). Therefore, the modified architecture of the parallel-prefix Ling adder can save one logic level for the computations of carries. The n-bit parallel-prefix Ling addition can be easily derived from the example of the 8-bit adder and Equation(13). The pseudo carry (H_n) can be expressed as :

$$\begin{aligned}&\textit{when } n \textit{ is even}\\&H_n = (G_n^*, P_n^*)\circ(G_{n-2}^*, P_{n-2}^*)\circ\ldots\circ(G_0^*, P_0^*)\\&\textit{when } n \textit{ is odd}\\&H_n = (G_n^*, P_n^*)\circ(G_{n-2}^*, P_{n-2}^*)\circ\ldots\circ(G_1^*, P_1^*)\end{aligned} \tag{14}$$

According to Equation(8), the multiplexers should be used to generate the sum bits (S_n) at the final stage to meet the change from traditional carry out (C_n) to pseudo carry out (H_n). Follow the above steps, the design of parallel-prefix Ling adders can be systematically constructed. In Fig. 4 and Fig. 5, the architecture of the parallel-prefix Ling adders is presented. In Fig. 6, it shows the implementation of new function cells.

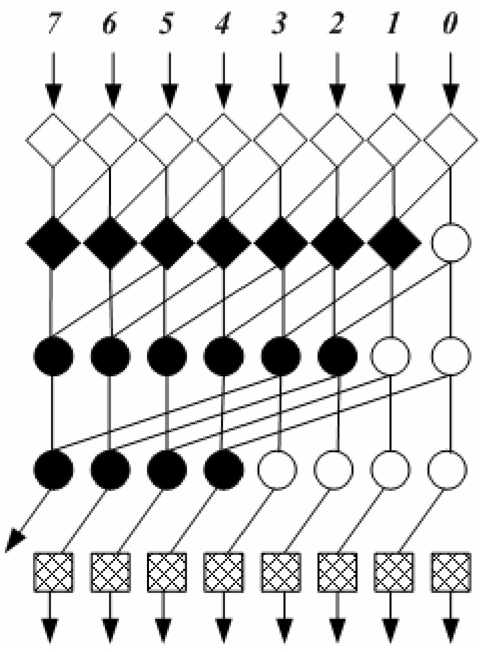


Fig. 4. A 8-bit Kogge-Stone Ling adder.

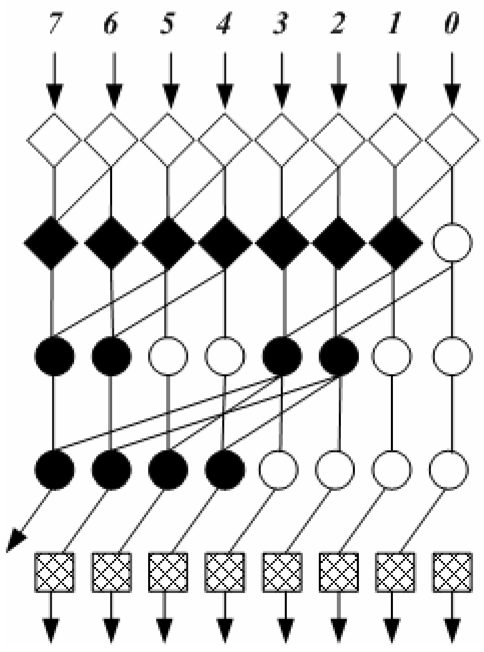


Fig. 5. A 8-bit Lander-Fisher Ling adder.

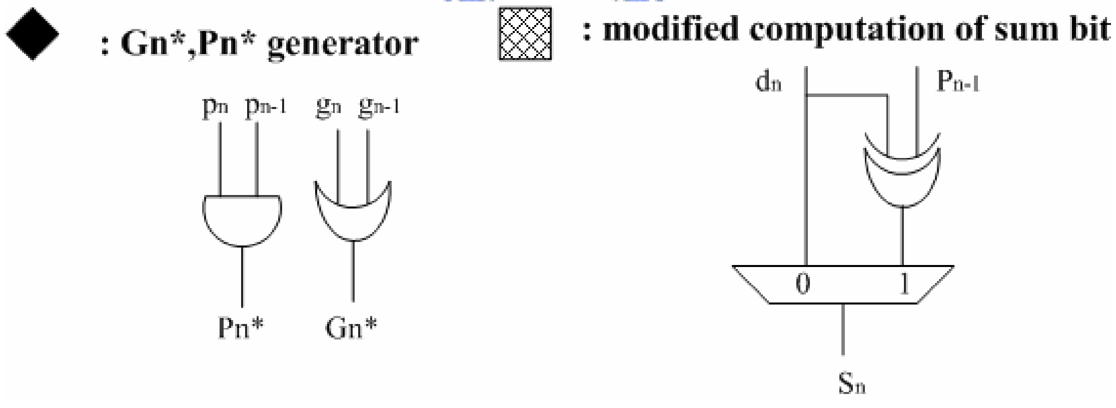


Fig. 6. New function cells for parallel-prefix Ling adders.

2.3 Carry-In Operation

In general, most binary adders need to consider the external carry-in. From [20], there are three ways to incorporate a carry-in into a Ling parallel-prefix structure. The first method needs an additional stage to generate the sum bit at each bit position from the p_n signal and the carry from the previous bit position according the following equation :

$$H_{n\_cin} = H_n \circ (C_{in}, 1) \tag{15}$$

According to the Equation(15), there is an extra associative operation at the last level to generate correct H_n. The architecture of the first method is illustrated in Fig. 7 for an 8-bit parallel-prefix Ling adder.

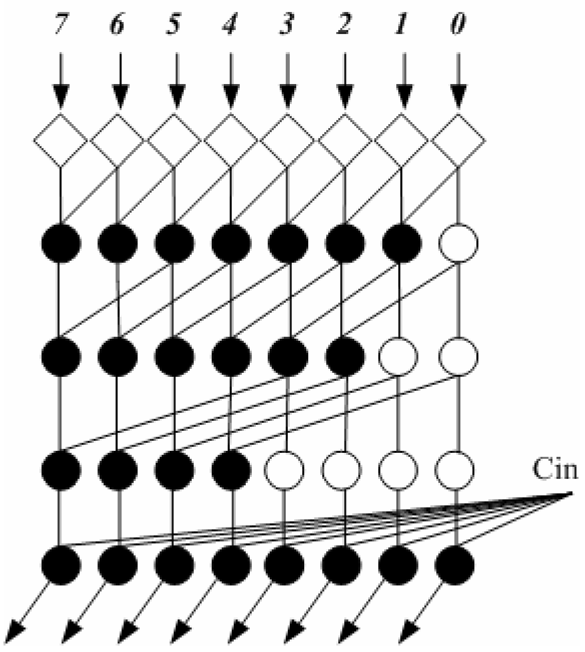


Fig. 7. An 8-bit parallel-prefix Ling adder with a carry-in using the first method.

The second method allows the first generate signal without modification and changes some white nodes (buffer nodes) to black nodes (associative operation) for incorporating the carry-in signal in parallel. This approach needs an additional stage to generate the final carry-out (C_{N-1}), but the computation of sum bits can be executed in parallel. An 8-bit parallel-prefix Ling adder is illustrated in Fig. 8.

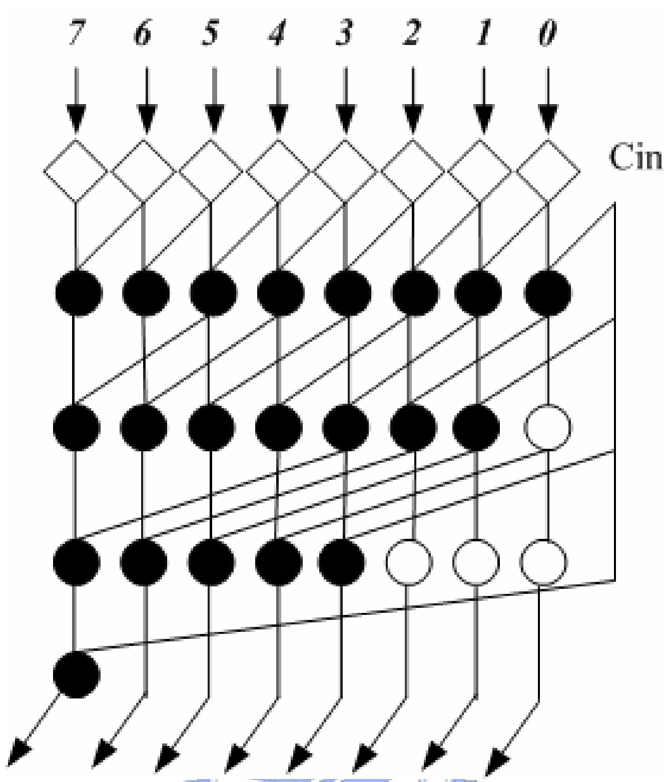


Fig. 8. An 8-bit parallel-prefix Ling adder with a carry-in using the second method.

In the end, the third method can incorporate the carry-in by redefining the first generate signal (g_0). The carry-in (C_{in}) can be considered an (g_{-1}, p_{-1}) pair. Then, the new g_0 can be derived by setting (g_{-1}, p_{-1}) = (C_{in}, 1) and using Equation(2). The modified first generate signal (g_{0_m}) can be expressed as :

$$g_{0\_m} = g_0 \mid (p_0 \,\&\, C_{in}) \tag{16}$$

Therefore, the architecture of the third method is similar to the original adder without carry-in except the additional logic for the newly modified first generate signal (g_{0_m}). An 8-bit parallel-prefix Ling adder is presented in Fig. 9.

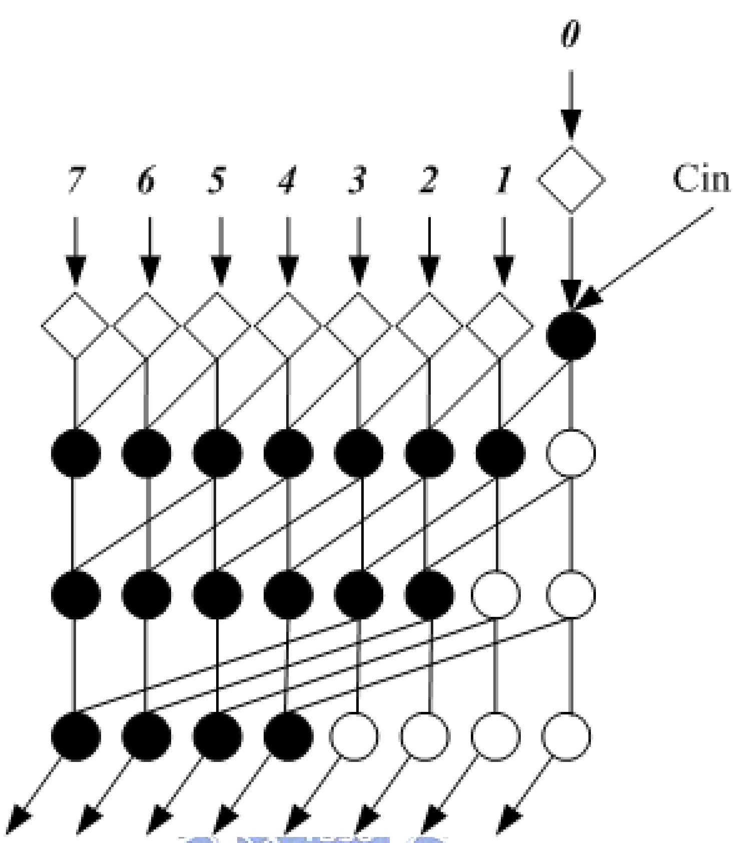


Fig. 9. An 8-bit parallel-prefix Ling adder with a carry-in using the third method.

In general, fanout issue, logic depth, area, and complexity of wiring are the major issues to construct adders. In terms of fanout, the first method has the large amount of fanout on the carry-in (C_{in}) from the additional stage to generate correct carries. But, the fanout of other nodes has the max fanout of 2 for all methods. The third method has the smallest amount of fanout on C_{in} from the only one extra function cell to modify g0. In terms of logic depth, the second method has the smallest logic depth to generate sum bits, but the effect of fanout on C_{in} may affect the overall delay when the size of the adder is large. Under the consideration of the fanout effect, the constant fanout 1 of the third method may have the same or better performance on speed. By the way, the logic depth to generate final carry-out (C_{N-1}) of each method is the same.

Because an additional stage is needed in the first method, there are N black nodes (associative operation) should be inserted into the original structure without the carry-in. The second method has the intermediate penalty of area from the additional $\log_2 N + 1$ black nodes. There is the only one function cell needed to incorporate with carry-in to the architecture of the third method. So the third method has the smallest area penalty for the computation of carry-in. The complexity of wiring for the carry-in is like the fanout issue on C_{in}. Therefore, the third method is the best choice for small complexity of wiring. Consequently, the third method is a better way for parallel-prefix adders under the considerations of area, speed, and fanout effect.

Chapter 3 Proposed Adder Architecture

In this chapter, the previous works of parallel-prefix Ling adders are introduced in the first section. The remaining sections give the detailed description of the proposed architecture.

3.1 Previous Approaches

In Chapter 2, the fundamental architecture of parallel-prefix Ling adders has been introduced. But Kogge-Stone parallel-prefix adders comparing to other adders like carry-ripple adder, carry-select adders, and carry-skip adders has the critical problem on cost of area. Consequently, the modern architecture of adders utilizes a hybrid scheme. A hybrid scheme like the parallel-prefix/carry-select adder leads to the reduction of area and the requirement for high-speed. Fig. 10 mentioned in [13] illustrates a hybrid 32-bit adder which combines the Kogge-Stone parallel-prefix tree for the generation of carries and carry-select blocks for the sum-bits.

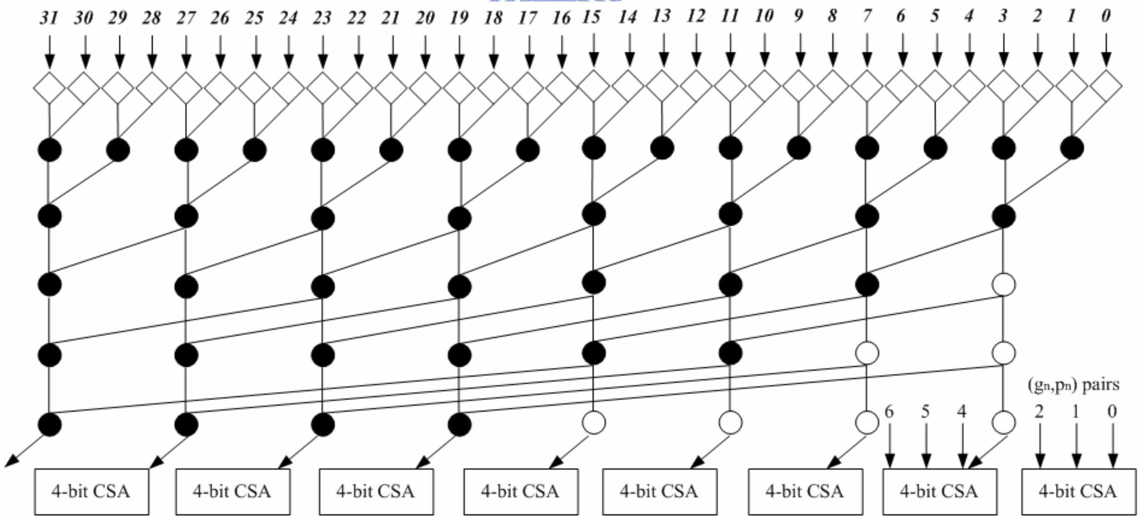


Fig. 10. A 32-bit hybrid parallel-prefix adder.

The following architecture in [21] is a hybrid carry-lookahead/carry-select adder. In Fig. 11, the correct sum-bits are selected from the carries generated by the carry-lookahead unit.

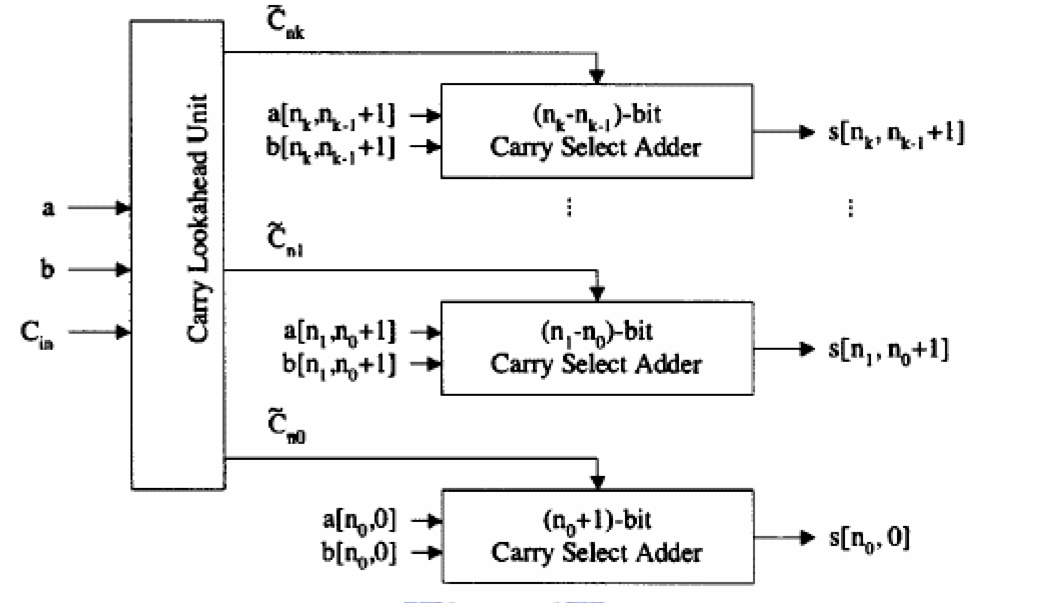


Fig. 11. A general architecture of hybrid carry-lookahead/carry-select adder.

The goal of hybrid structures is to overlap the time of computation for carries at the boundaries of the carry-select blocks with the time needed to calculate the sum bits. Base on the previous hybrid adders and parallel-prefix Ling adders described in Chapter 2, the hybrid parallel-prefix/carry-select Ling adders using Kogge-Stone algorithm (hybrid K-S Ling adders) are proposed in [13]. The new approach employs a Kogge-Stone parallel-prefix Ling structure to generate the partial pseudo- carries (H_n) for carry-select blocks. However, the traditional carry-select blocks need some modification for using pseudo-carries (H_n) instead of the normal carries (C_n). It also separates the generation of carries and sum bits into even and odd bit positions. The modified carry-select adders (MCSA) [13] use the pairs (G_n*, P_n*) as inputs instead of traditional (g_n, p_n) pairs. A 32-bit hybrid Ling adder and related function cells defined in [13] are shown in Fig. 12, Fig. 13, and Fig. 14.

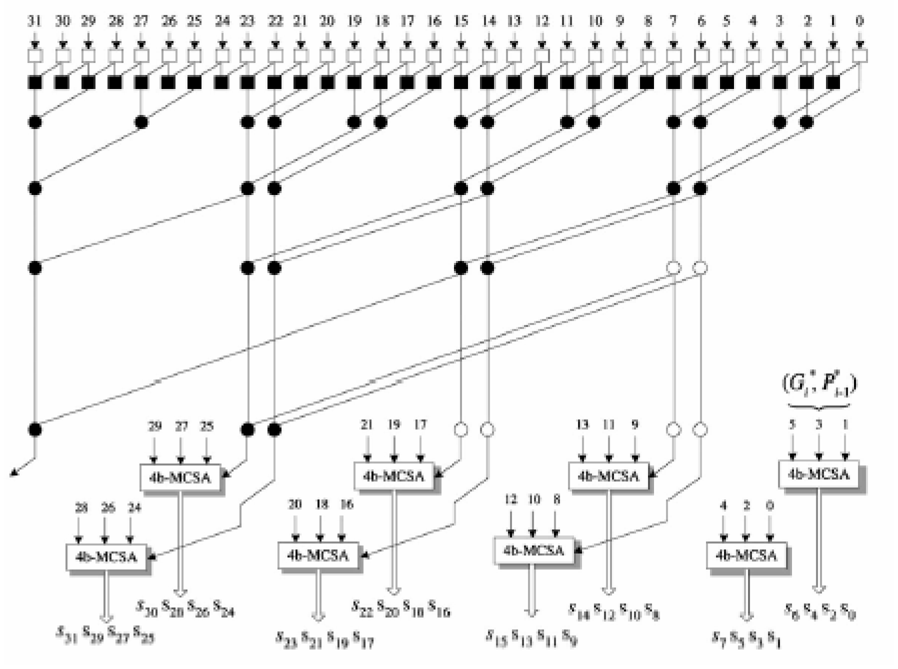

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
(G_i^*, P_{i-1}^*)
29 27 25
28 26 24
21 19 17
20 18 16
13 11 9
12 10 8
5 3 1
4 2 0
4b-MCSA
$s_{30}\,s_{28}\,s_{26}\,s_{24}$
$s_{31}\,s_{29}\,s_{27}\,s_{25}$
$s_{22}\,s_{20}\,s_{18}\,s_{16}$
$s_{23}\,s_{21}\,s_{19}\,s_{17}$
$s_{14}\,s_{12}\,s_{10}\,s_{8}$
$s_{15}\,s_{13}\,s_{11}\,s_{9}$
$s_{6}\,s_{4}\,s_{2}\,s_{0}$
$s_{7}\,s_{5}\,s_{3}\,s_{1}$

Fig. 12. A 32-bit hybrid parallel-prefix/carry-select Ling adder.

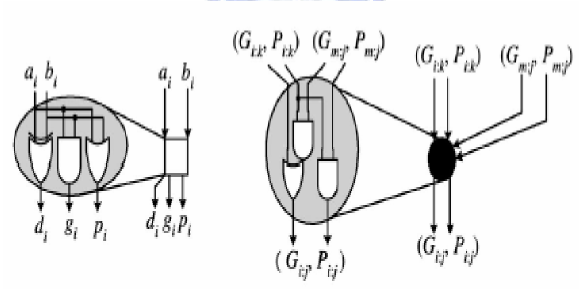


Fig. 13. Logic cells – 1.

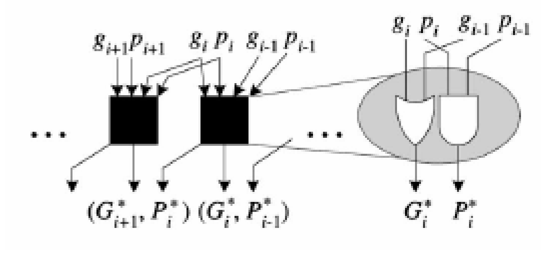

$g_{i+1}p_{i+1}$ $g_i\ p_i$ $g_{i-1}\ p_{i-1}$
(G_{i+1}^*, P_i^*) (G_i^*, P_{i-1}^*)
G_i^* P_i^*

Fig. 14. Logic cells – 2.

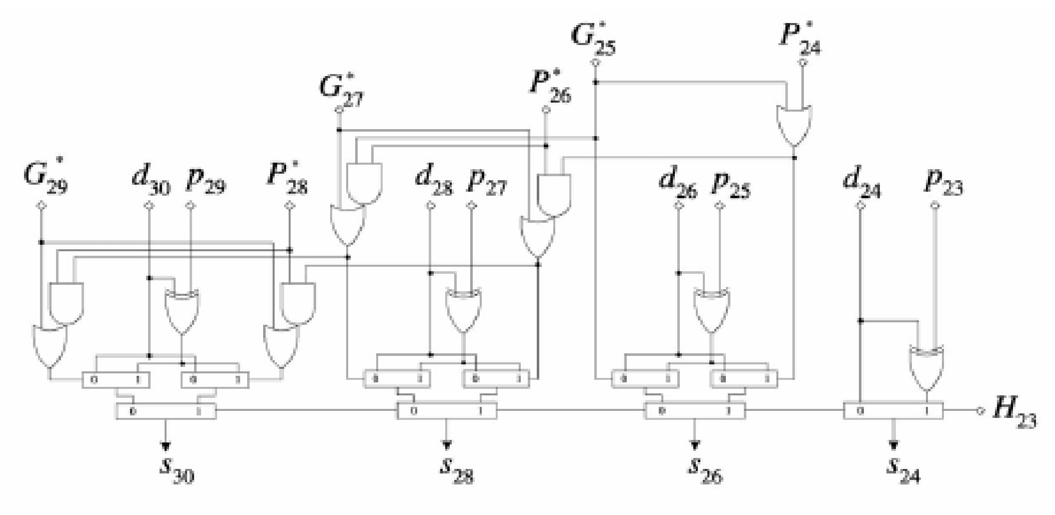
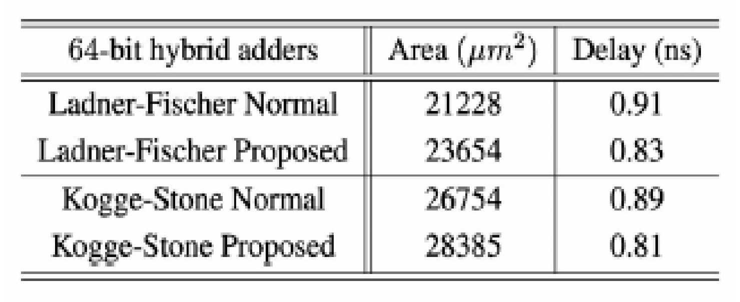


Fig. 15. A modified 4-bit carry-select block (MCSA).

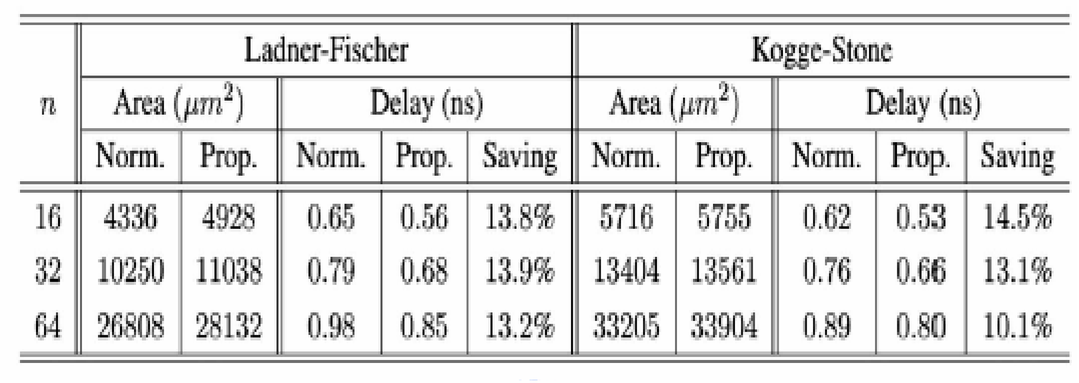
However, the area of the MCSA is obviously larger than traditional carry-select block which is shown in Fig. 15. The Table 1 in [13] shows that the speed improvement of the hybrid parallel-prefix Ling adders induces the additional cost of area compared to the traditional hybrid parallel-prefix adders. However, the hybrid Ling adders can save area when compared to traditional parallel-prefix adders and parallel-prefix Ling adders from the simulation results in Table 2 presented by [13]. According to these results, a hybrid K-S Ling adder can improve speed and reduce the overall area when compared to the parallel-prefix adders. But, the area of hybrid K-S Ling adder is still large than hybrid K-S adder from the use of MCSA blocks. Based on the analysis of previous works, the goal of our proposed adder is to further reduce area without additional timing penalty by a systematic methodology in the following sections. Using the proposed architecture can solve the area problem of using MCSA blocks, reduce more area in parallel-prefix tree, and preserve the benefit of Ling addition.

Table 1. The timing and area results for hybrid adders.



64-bit hybrid adders	Area (μm^2)	Delay (ns)
Ladner-Fischer Normal	21228	0.91
Ladner-Fischer Proposed	23654	0.83
Kogge-Stone Normal	26754	0.89
Kogge-Stone Proposed	28385	0.81

Table 2. The timing and area results for Ling adders.



n	Ladner-Fischer					Kogge-Stone				
	Area (μm^2)		Delay (ns)			Area (μm^2)		Delay (ns)		
	Norm.	Prop.	Norm.	Prop.	Saving	Norm.	Prop.	Norm.	Prop.	Saving
16	4336	4928	0.65	0.56	13.8%	5716	5755	0.62	0.53	14.5%
32	10250	11038	0.79	0.68	13.9%	13404	13561	0.76	0.66	13.1%
64	26808	28132	0.98	0.85	13.2%	33205	33904	0.89	0.80	10.1%

3.2 Proposed Architecture

The following sections introduce the proposed architecture with detailed block diagram and the systematic methodology to design the high-speed area-minimized hybrid Ling adder. From the experimental results, the proposed scheme can achieve area reductions of up to 25 percent when compared to the traditional Kogge-Stone parallel-prefix architectures. And further, it also can reduce nearly one-fifth area than the hybrid K-S Ling adders.

3.2.1 Area-Minimized Parallel-Prefix Tree

There are many algorithms to build the parallel-prefix tree. Various architectures result in different area, delay, complexity of wiring, and fanout issue. These different architectures from [22] are shown in Fig. 16, Fig. 17, Fig. 18, and Fig. 19. In these figures, all the (p_n, g_n) pairs are generated in the first level and other function nodes are shown in Fig. 3. Table 3 is a collection of area, delay, and fanout according to [22]. From these data in Table 3, the parallel-prefix architecture of Kogge-Stone type has the advantage of fast speed, the fixed number of fanout, and the regularity of wiring. Consequently, the Kogge-Stone parallel-prefix tree is chosen in our proposed approach.

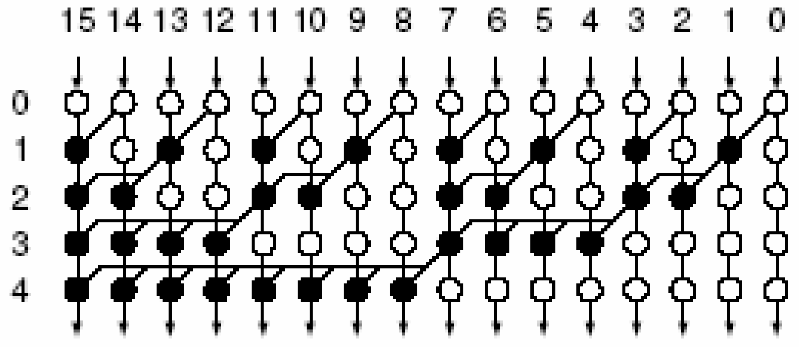


Fig. 16. A 16-bit Sklansky parallel-prefix adder.

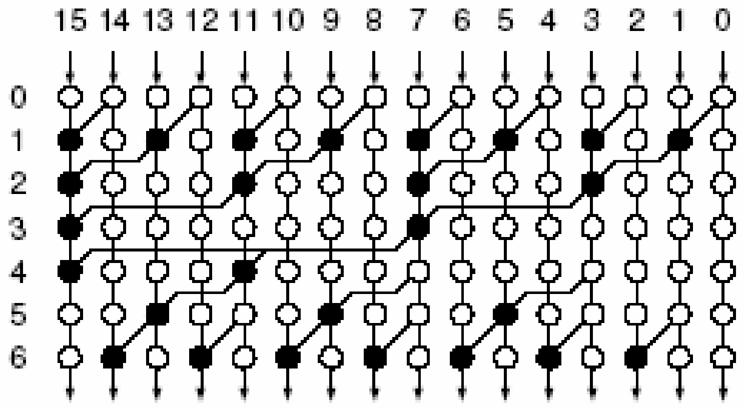


Fig. 17. A 16-bit Brent-Kung parallel-prefix adder.

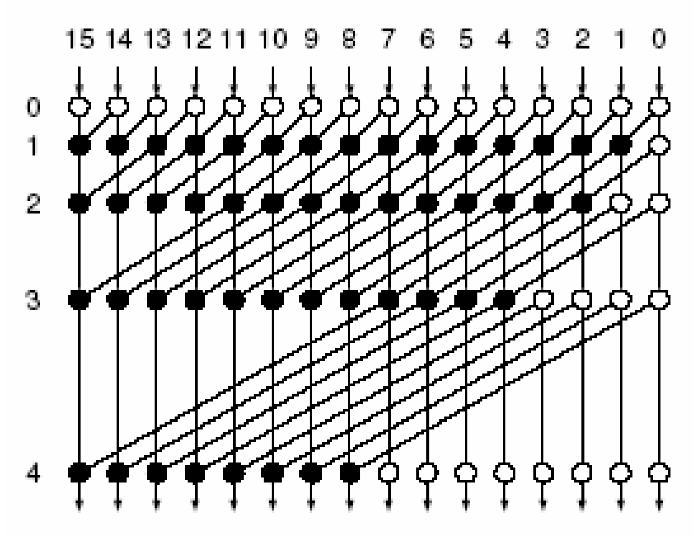


Fig. 18. A 16-bit Kogge-Stone parallel-prefix adder.

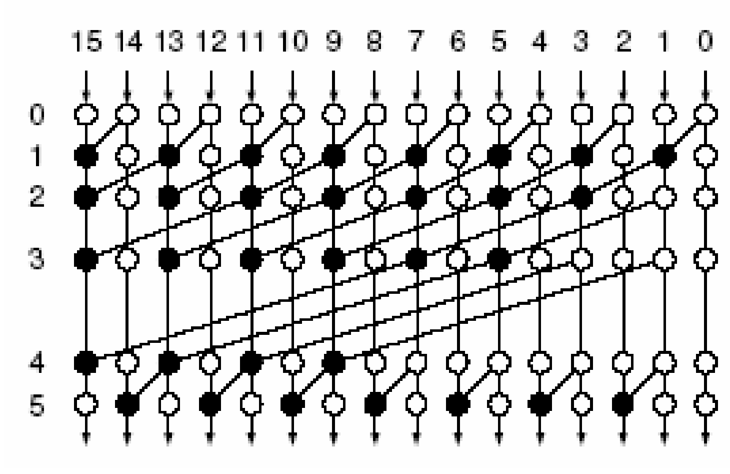


Fig. 19. A 16-bit Han-Carson parallel-prefix adder.

Table 3. Comparison of different parallel-prefix algorithms.

Algorithm Type	Max. FanOut	Delay (logic depth)	Area (black node)
Sklansky	1/2 N	$\log_2 N$	1/2 $N*\log_2 N$
Brent-Kung	$\log_2 N$	$2*\log_2 N - 2$	$2N - \log_2 N - 2$
Kogge-Stone	2	$\log_2 N$	$N*\log_2 N - N + 1$
Han-Carson	2	$\log_2 N + 1$	1/2 $N*\log_2 N$
N : the length of addition			

However, all the data in Table 3 are produced by mathematics. We need more simulation results by physical implementation to get more confidence in Kogge-Stone parallel-prefix structure. From Table 4 by arranging the data in [23], it shows that the Kogge-Stone parallel-prefix adder (K-S adder) is a good solution to design a high-speed adder. The unit to estimate delay is FO4 (fanout-of-four inverter delay). But, the Kogge-Stone architecture achieves high performance by using a large amount of associative operation. More area may cause more power consumption and so the way to reduce overall area without loss of performance is important. A hybrid structure is used to achieve delay reductions in our design. And further, we use the parallel-prefix Ling structure introduced in Chapter 2 to be the main backbone. A 32-bit Kogge-Stone parallel-prefix tree for the proposed hybrid Ling adder is shown in Fig. 20.

Table 4. Delay estimation for different parallel-prefix algorithms.

Algorithm Type	Delay (FO4) N = 32	Delay (FO4) N = 64
Ripple	54.6	107.7
Sklansky	16.3	23.4
Brent-Kung	16.8	21.8
Lander-Fisher	15.6	20.2
Han-Carson	13.3	16.4
Kogge-Stone	13.4	18.0
N : the number of addition 1. Non-inverting CMOS 2. uniform cell size		

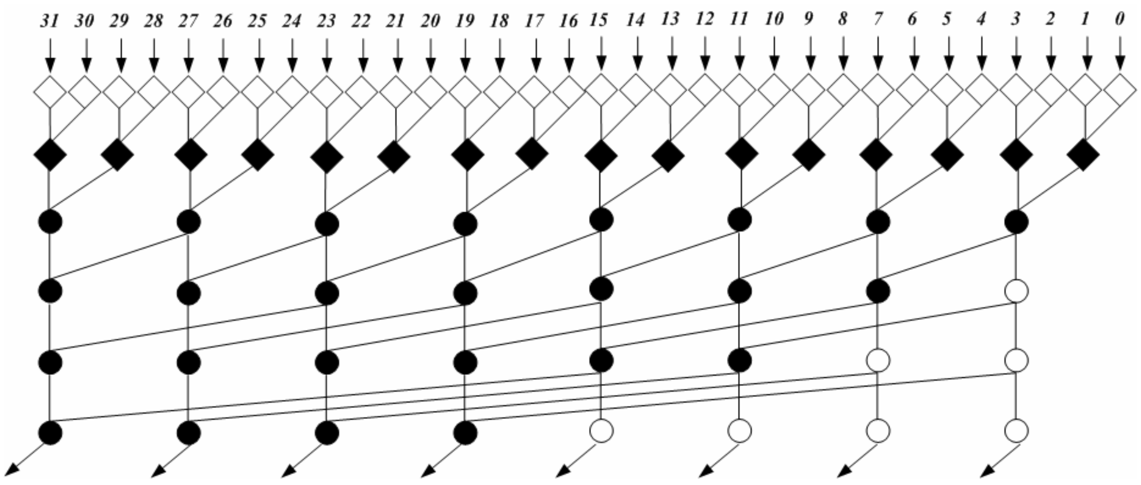
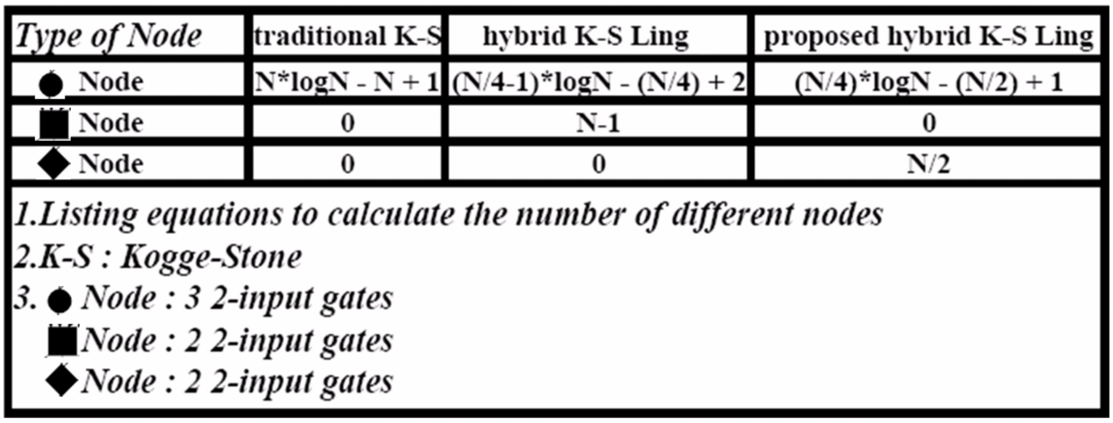


Fig. 20. A 32-bit Kogge-Stone parallel-prefix tree for the hybrid Ling adder.

According to the Table 3, Fig. 12, Fig. 20, a 32-bit traditional Kogge-Stone parallel-prefix adder needs 129 ● nodes, a 32-bit architecture proposed in [13] it needs 29 ● nodes and 31 ■ nodes, and the parallel-prefix tree in our design needs 25 ● nodes and 16 ◆ nodes. Assuming the use of two-input logic gate, the ● node requires 3 logic gates, the ■ node or the ● node requires 2 logic gates. The area cost of the proposed scheme is 107 gates, the architecture in [13] needs 137 gates, and the traditional architecture requires 387 gates. Consequently, the type of parallel-prefix tree in Fig. 20 can achieve the minimum area when compared to other architectures. The equations to calculate the cost of area for the three architectures are listed in Table 5. Table 6 and Table 7 explain area reductions of the parallel-prefix tree clearly via the following cases of various tree structures. From these cases, using hybrid parallel-prefix Ling structure can easily achieve the goal of area-reduction on the part of parallel-prefix tree.

Table 5. Equations to calculate number of different nodes.



Type of Node	traditional K-S	hybrid K-S Ling	proposed hybrid K-S Ling
● Node	N*logN - N + 1	(N/4-1)*logN - (N/4) + 2	(N/4)*logN - (N/2) + 1
■ Node	0	N-1	0
◆ Node	0	0	N/2

1.Listing equations to calculate the number of different nodes
2.K-S : Kogge-Stone
3. ● Node : 3 2-input gates
■ Node : 2 2-input gates
◆ Node : 2 2-input gates

Table 6. Area comparison between K-S adders and proposed architecture.

N (bit number)	traditional K-S	proposed hybrid K-S Ling	saving percentage
16	147	43	70.75%
32	387	107	72.35%
64	963	259	73.10%
Using the number of 2-input gates to estimate the area			

Table 7. Area comparison between hybrid K-S Ling and proposed architecture.

N (bit number)	hybrid K-S Ling	proposed hybrid K-S Ling	saving percentage
16	60	43	28.33%
32	149	107	28.19%
64	354	259	26.83%
Using the number of 2-input gates to estimate the area			

3.2.2 Carry-In Handling

In general, most adders need the carry-in to support the multi-word addition and subtraction. Therefore, it is important to find the efficient approaches for incorporating the carry-in to a parallel-prefix structure. In Chapter 2, there are three ways for solving the carry-in problem. To achieve the minimum fanout of Cin, the smallest area penalty, and the regularity of parallel-prefix structures, we choose the third method to be the solution of our proposed adders with carry-in. First, we can see the carry-in as a (g-1,p-1) pair. Because the carry-in must propagate to the bit position 0, setting g-1= Cin and p-1 = 1 represents the Carry-in. Finally, the original (g0, p0) needs to be modified as (g0_m, p0_m) by the following equation. The logic-level implementation for the (g0_m, p0_m) pair is shown in Fig. 21.

$$\begin{aligned}(g_{0\_m}, p_{0\_m}) &= (g_0, p_0) \circ (g_{-1}, p_{-1}) \\ &= (g_0, p_0) \circ (C_{in}, 1) \\ &= (g_0 \mid (p_0 \,\&\, C_{in}), p0 \,\&\, 1) \\ &= (g_0 \mid (p_0 \,\&\, C_{in}), p0)\end{aligned} \tag{17}$$

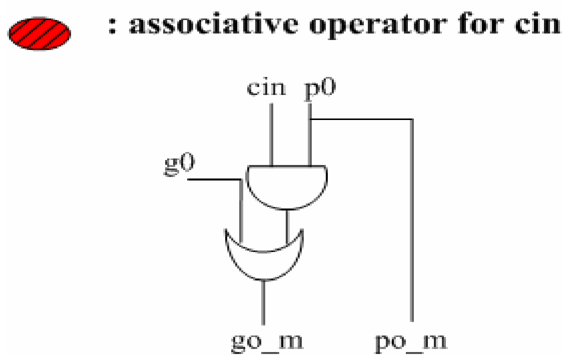


Fig. 21. The logic cell to incorporate carry-in to the (g0, p0) pair.

3.2.3 The block size of carry-select adders (CSA)

In a hybrid parallel-prefix/carry-select adder, the size of the carry-select block should be chosen carefully. The timing of addition may become worse due to the un-optimized block size. Assuming the 2-input AND gate be the basic unit to estimate the delay. In Table 8, the timing information of all logic gates used to calculate delay of the carry-select block are listed. The formula to estimate delay of K-bit carry-select block can be derived form the general 4-bit block presented in Fig. 22. The equation is expressed as :

$$\begin{aligned}
&T(longest\ delay\ of\ Sum_{\mathrm{N}}^{1}) \\
&= 1 * T_{OR\ gate} + (K-2) * T_{AND-OR\ gate} + 1 * T_{XOR} \\
&= 1*1 + (K-2)*2 + 1*2 \\
&= 2K-1 \\
&(Sum_{\mathrm{K}}^{1} : \text{temporal sum bit K when assuming Cin is 1, K>1})
\end{aligned} \tag{18}$$

When the time to generate the carries for selecting sum bits produced by carry-select blocks is equal to the delay of Sum_{K}^{1}, the block-size is optimized. However, the regularity of structure is also important for designing an adder. For the convenience of transferring our proposed architecture to the reconfigurable version, the block size should be the factor of the minimum partition-size. However, the small size like 2-bit, 3-bit cannot get the benefit of area-reduction from the hybrid structure. Therefore, the nearly optimum block size K should meet the following Equations(19).

Table 8. Normalized logic delay table.

Logic Type	Logic Delay
2-input AND	1
2-input OR	1
2-inpu MUX	2
An associative operator : 2 level AND-OR gate	2
the unit to estimate delay is based on the delay of 2-input AND gate	

$a.$ K is integer

$b.$

$Without\ Carry\text{-}In$

$$T(SUM_K^1) <= T(parallel\text{-}prefix\ tree\ with\ Ling\ addition)$$

$$=> \quad (2K-1) <= T_{associaive\ opearor} * (\text{Floor}[\log_2 N]) - 1$$

$$=> \quad K <= \text{Floor}[\log_2 N]$$

$With\ Carry\text{-}In$

$$T(SUM_K^1) <= T(parallel\text{-}prefix\ tree\ with\ Ling\ addition)$$

$$=> \quad (2K-1) <= T_{associaive\ opearor} * (\text{Floor}[\log_2 N] + 1) - 1$$

$$=> \quad K <= \text{Floor}[\log_2 N] + 1$$

$c.$ The length of addition (N) is divisible by K

(to balance the fan-out of carries generated from the parallel-prefix tree)

$d.$ Choose the biggest integer under the conditions of a, b, c

Note: The timing information in Table 8 is used to estimating delay (19)

Finally, using the cases of 16/32/64/128-bit adders to explain the way of choosing the nearly optimum size of the carry-select block are shown the Table 9. In general, 4-bit block is a good solution for designing the proposed adder. Fig. 22 presents a 4-bit simple (traditional) carry-select adder (SCSA).

Table 9. Examples of selecting nearly optimum block size.

N	*optimum block size (K) with/without Cin*	*nearly optimum block size (K) with/without Cin*
16	4/5	4/4
32	5/6	4/4
64	6/7	4/4
128	7/8	4/8
256	8/9	8/8
N* : *the length of addition *K* : *the block-size of the carry-select block*		

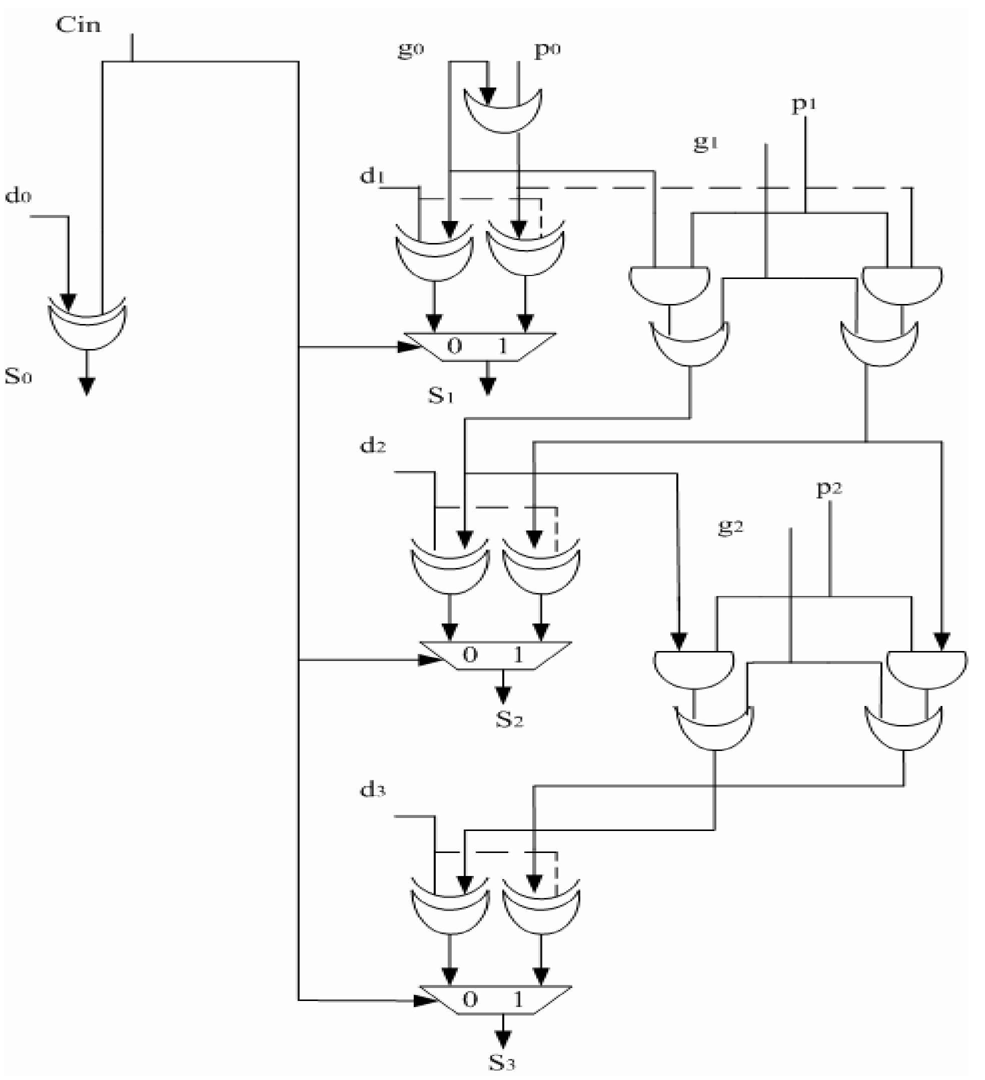


Fig. 22. A 4-bit carry-select adder.

3.2.4 CSA Optimization

The following content, the special function blocks for reducing the overall area are introduced. In section 3.2.2, the way for incorporating C_{in} to parallel-prefix Ling structure is presented. According to the skills mentioned in 3.2.1 and 3.2.2, the proposed architecture without CSA optimization is shown in Fig. 23. Although the parallel-prefix tree has been minimized, the area-problem of MCSA is still alive. The method to minimize the area of using MCSA is to insert some logic cells into the additional stage for C_{in}. There are two new logic cells in the proposed architecture. First, using the "&1" cells to replace some buffer nodes can transfer the H_n into C_n, for n <= N/2. The principle of the new cell can be derived from Equation(7). Fig. 24 presents the logic implementation of the "&1" cell. However, there are not enough buffer nodes to insert the "&1" cells into the parallel-prefix tree. The pseudo carries Hn, n > N/2, has no significant buffer nodes in their path. Therefore, the new logic cell ("&2") is proposed. Inserting the new cells into the stage for Cin can translate the pseudo carries H_n (n > N/2) into normal carries C_n (n > N/2). The following equation can explain the principle of the "&2" cell. Fig. 25 also displays the logic implementation of the "&2" cell. Finally, the new architecture can avoid the generation of H_n. Consequently, the proposed hybrid Ling adder can fully use simple (traditional) carry-select adders to eliminate the additional area cost on MCSAs. A 16-bit proposed adder is shown in Fig. 26.

$$\begin{aligned}
Cn &= pn \,\&\, Hn \\
&= pn \,\&\, [(Gn^*,Pn^*) \bigcirc Hn\text{-}2] \\
&= [pn \,\&\, (Gn^*,Pn^*)] \bigcirc Hn\text{-}2 \\
&= (\,(Gn^*\&pn)\,,\,(Pn^*\&pn)\,) \bigcirc Hn\text{-}2 \\
&= (\,pn\&(gn+gn\text{-}1)\,,\,pn\&(pn\text{-}1\&pn\text{-}2)\,) \bigcirc Hn\text{-}2 \\
&= (\,(gn+pn\&gn\text{-}1)\,,\,(pn\&pn\text{-}1)\&pn\text{-}2) \bigcirc Hn\text{-}2 \\
&= (\text{modified } Gn^*,\ \text{modified } Pn^*) \bigcirc Hn\text{-}2
\end{aligned} \qquad (20)$$

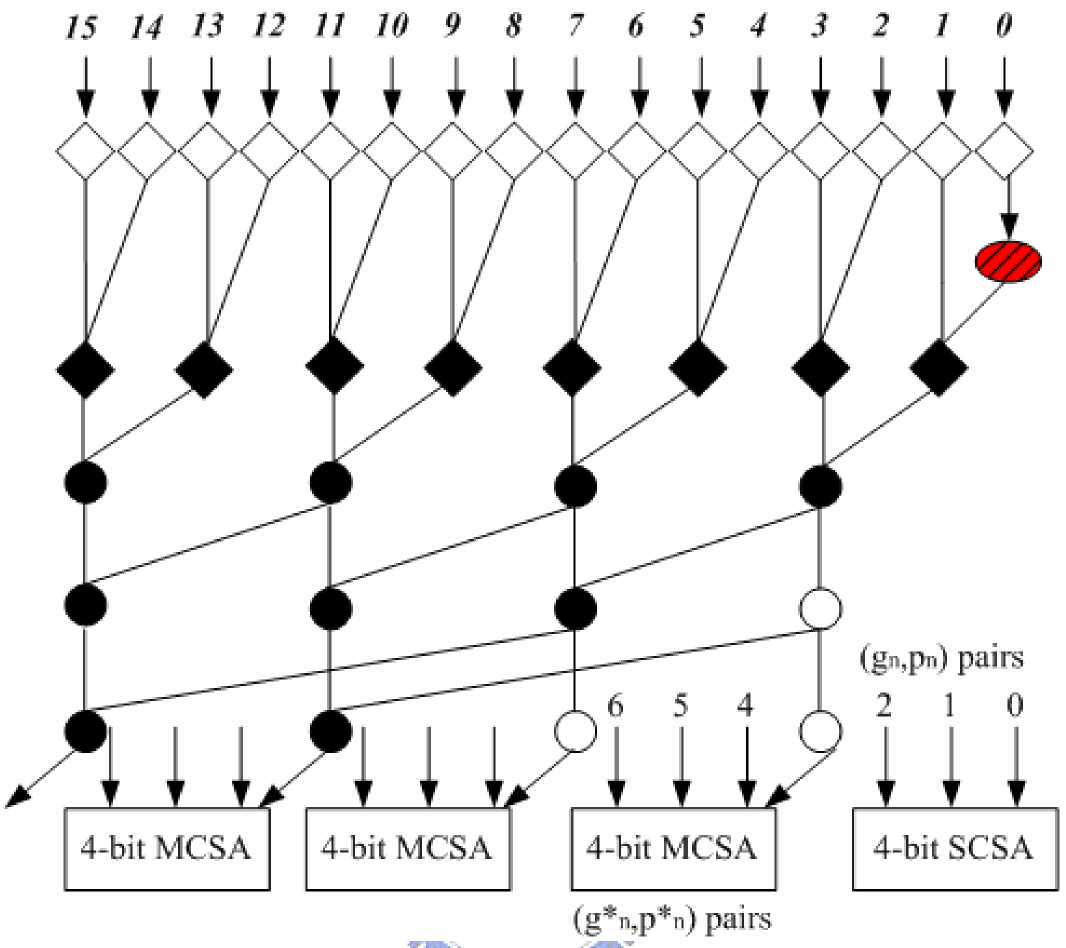


Fig. 23. A 16-bit proposed adder without CSA optimization.

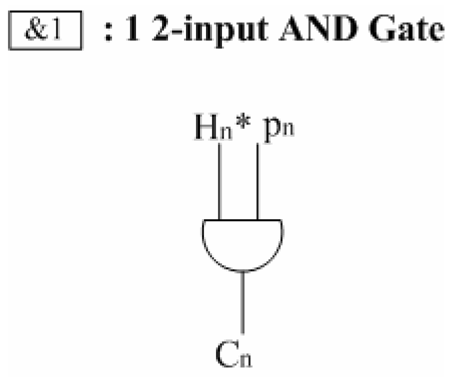


Fig. 24. Logic implementation of the &1 cell.

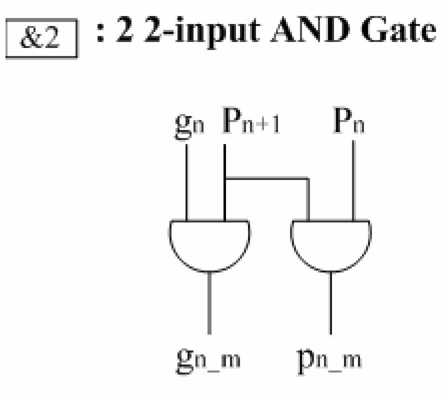


Fig. 25. Logic implementation of the &2 cell.

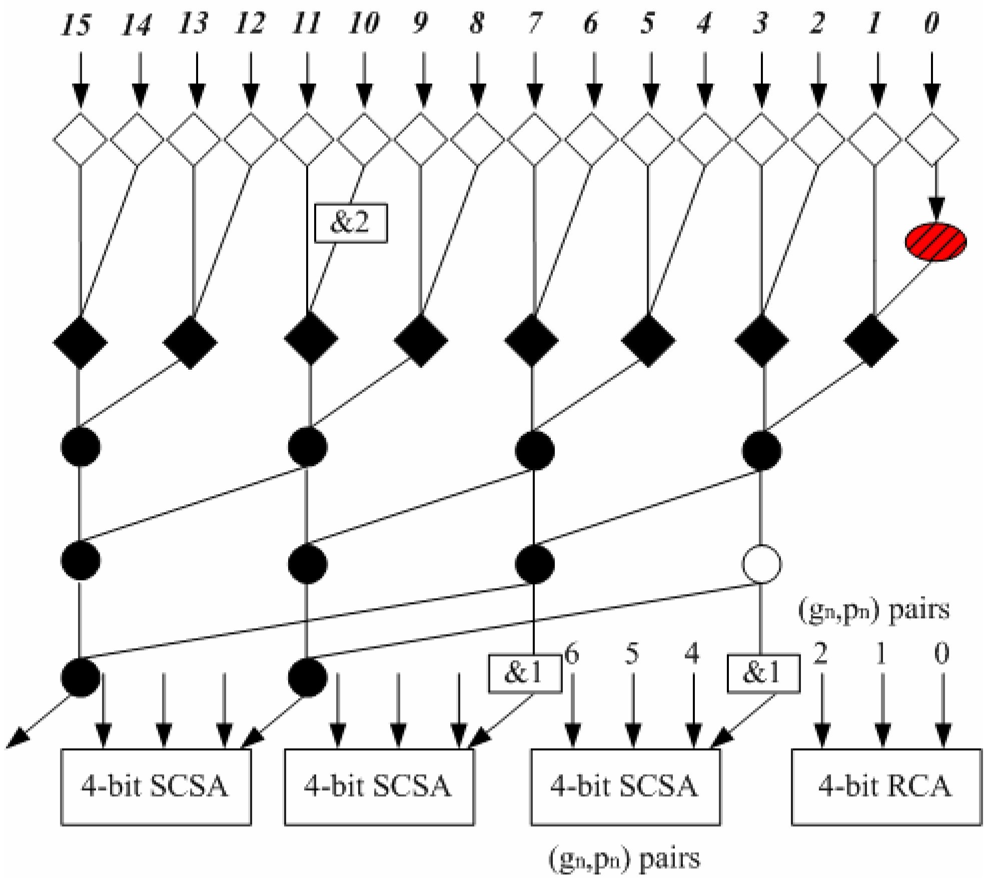


Fig. 26. A 16-bit proposed adder without CSA optimization.

3.2.5 Fanout Issue

For achieving minimum delay, we should consider the fanout. In general, the calculation of delay consists of intrinsic delay and delay of fanout loading. Therefore, the goal of high-speed can be achieved by minimizing the loading of the logic-cell. The Kogge-Stone parallel-prefix adder has the advantage of fixed fanout whether the length of addition grows or not. Fig. 27 shows that each associative operator in the parallel-prefix tree maintains the fanout of 2. In our architecture, the fanout also preserves the property of constant number of fanout except the last stage. But, the effect of the last stage is small from the small growth of fanout (2 -> 4) in general case. Compared to the hybrid architecture in [13], our proposed adder has the same fanout in the critical-path. From Fig. 28 and Fig. 29, the fanout issue of my proposed architecture and the hybrid architecture in [13] can be understood clearly. Consequently, the proposed hybrid architecture can minimize area without significant penalty on the number of fanout.

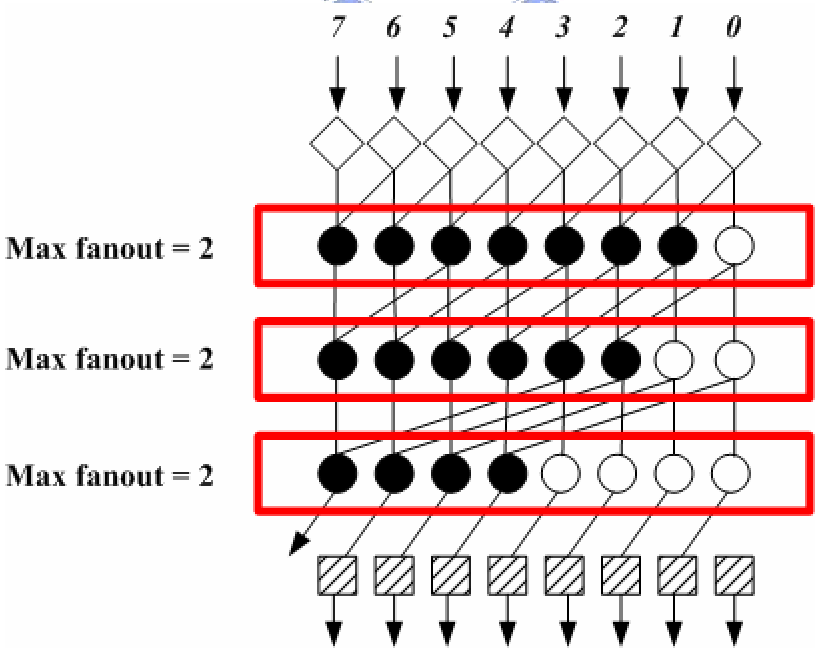


Fig. 27. Fanout issue of a 8-bit K-S adder.

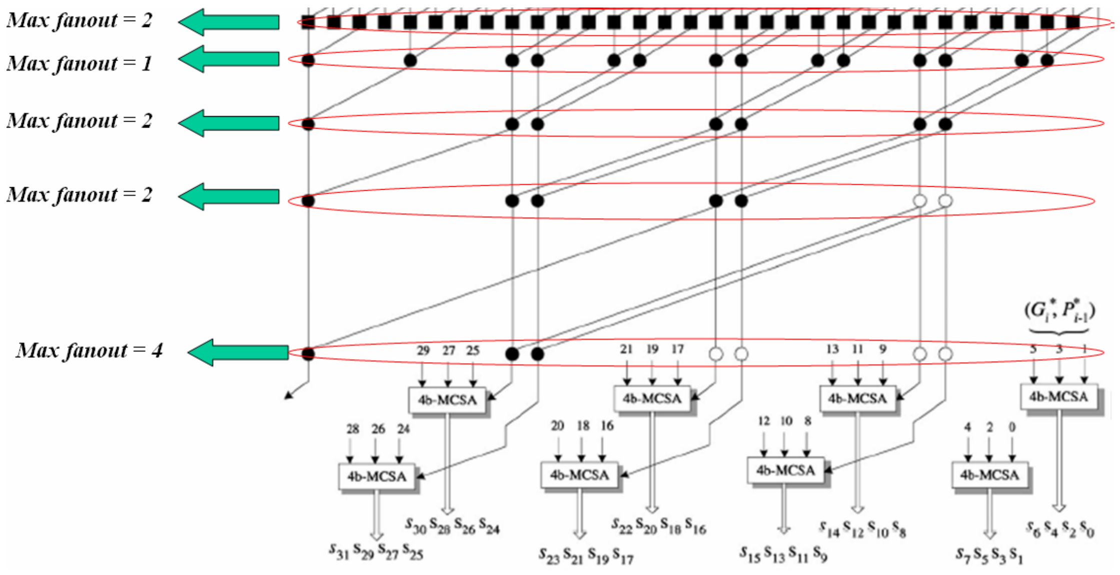

Max fanout = 2
Max fanout = 1
Max fanout = 2
Max fanout = 2
Max fanout = 4
4b-MCSA
(G_i^*, P_{i-1}^*)
$s_{31}\ s_{29}\ s_{27}\ s_{25}$
$s_{30}\ s_{28}\ s_{26}\ s_{24}$
$s_{23}\ s_{21}\ s_{19}\ s_{17}$
$s_{22}\ s_{20}\ s_{18}\ s_{16}$
$s_{15}\ s_{13}\ s_{11}\ s_9$
$s_{14}\ s_{12}\ s_{10}\ s_8$
$s_7\ s_5\ s_3\ s_1$
$s_6\ s_4\ s_2\ s_0$

Fig. 28. Fanout issue of a 32-bit hybrid K-S Ling adder.

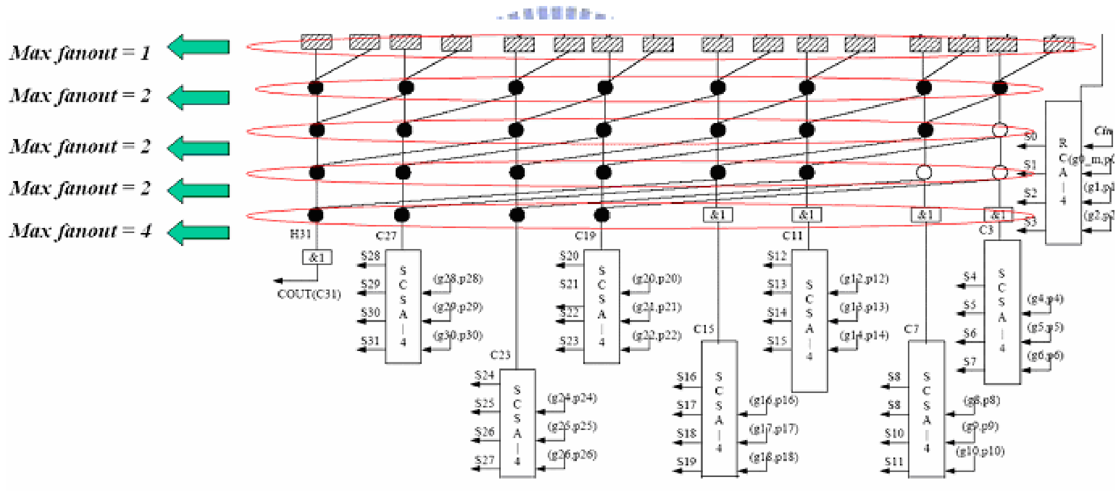


Fig. 29. Fanout issue of a 32-bit proposed adder.

3.3 Summary of Proposed Architecture

According to the basic components and main idea introduced in previous sections, the design of proposed architecture can be summarized in the following steps :

- Generate the bit-generate (g_n) and bit-propagate (p_n), and half-sum bit (d_n) at the first stage.
- Handle the carry-in signal by using the proposed method and insert the "&2" cells into the specific locations of generate and propagate pairs (gn, pn), with n = 4k-2 and k = N/8+1, N/8+2, …, N/4-1 (N>=16).
- Use Equation(9) and Equation(10) to generate the intermediate generate and propagate pairs (G_n*, P_n*) for Ling addition.
- Use the pairs (G_n*, P_n*) to build the parallel-prefix tree mentioned in section 3.2.1. The parallel-prefix structure can be employed for the generation of the pseudo carries H_{4k-1}, k = 1, 2, …, N/4.
- Replace the specific buffer nodes by the "&1" cells in parallel-prefix trees. These buffer nodes pass pseudo carries H_{4k-1}, k = 1, 2, …, N/8. After the replacement, these pseudo carries are transferred to normal carries C_{4k-1}.
- Select correct sum-bits produced from SCSA blocks by using normal carries C_{4k-1}, k = 1, 2, …, N/4-1. Finally, combining H_{N-1} and p_{N-1} can derive the final carry-out from using Equation(7).

In Fig. 30, the design steps of proposed architecture are illustrated. The order of steps conforms to the design methodology mentioned above. Therefore, we can design a high-speed and area-minimized adder from our proposed approaches. These approaches include area-minimized parallel-prefix tree, the better way to handling carry-in, and CSA optimization for Ling addition. In the end, a systematic methodology has been introduced for designing the proposed architecture in this section.

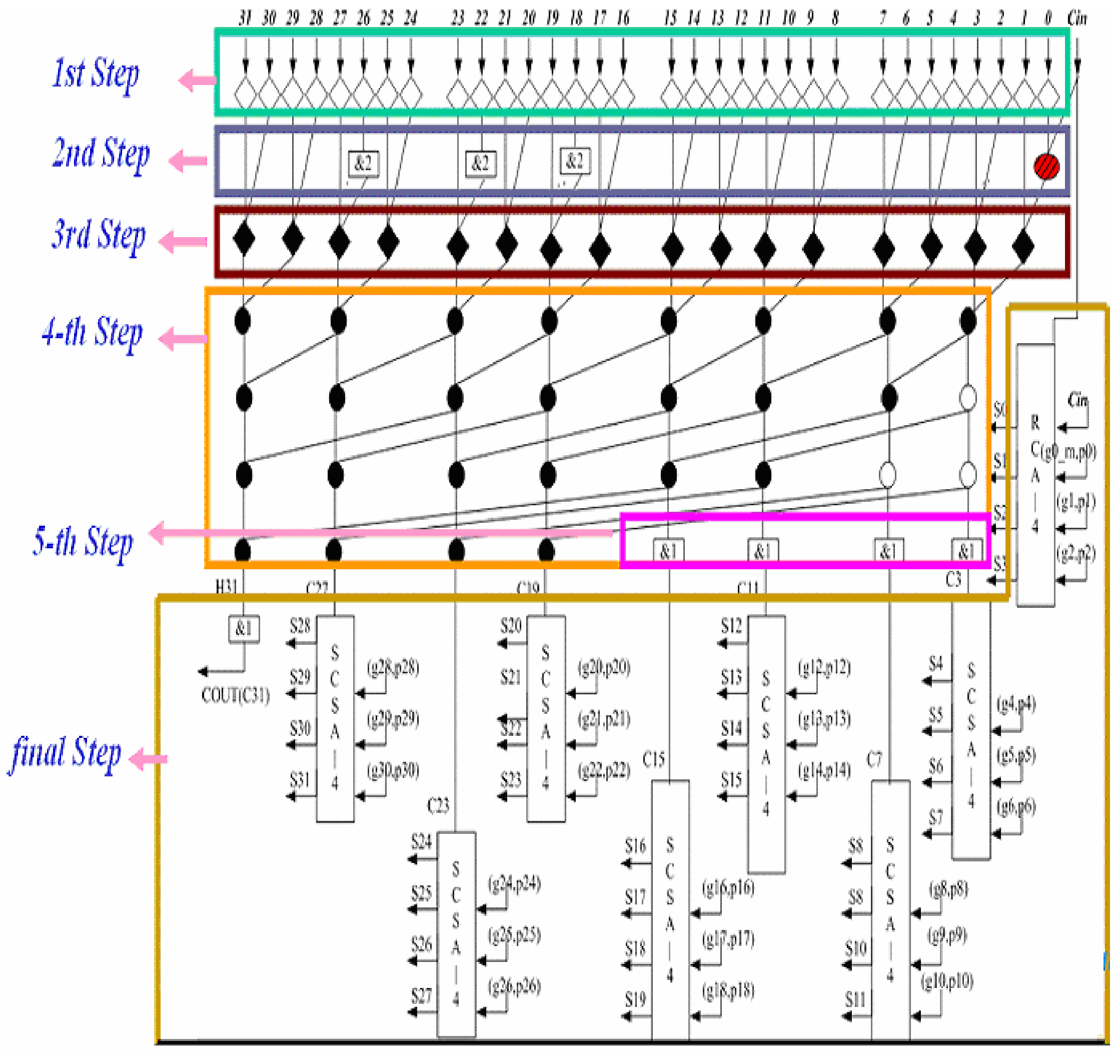


Fig. 30. Design steps of the proposed architecture.

3.4 Experimental Results and Analysis

After introducing the main idea and basic components in previous sections, the advantages of proposed architecture can be proved by the following experimental results. First, Table 10 gives the simple description of the experimental environment.

Table 10. Experimental environment 1.

Experimnetal Enviroment	
HDL	Verilog
Process	UMC 0.18um (under TT corner)
Synthesis Tool	Synopsys Design Compiler (Version : W-2004.12-SP2-2)
Power Analysis	Power Compiler (Version : W-2004.12-SP2-2)

The proposed architecture compares with two objects, one for the traditional K-S adder and one for the hybrid K-S Ling adder. Because the carry-in handling in K-S Ling adder [13] uses method 1 or method 2 mentioned in the Chapter 2, the two objects of comparison use these two methods. The proposed adder uses only method 3. Table 11 shows the comparison between the K-S adder which uses method1 (K-S_1) and the proposed adder (proposed). From the result of the experiment, our proposed architecture can save 31.56% area on average and get a speed-improvement.

Table 11. Experimental results of K-S_1 and proposed adders.

Timing & Area Analysis (UMC: 0.18um/TT corner)					
Data Width	K-S_1 (Delay)	Proposed (Delay)	K-S_1 (Area)	Proposed (Area)	Area Saving
16	0.72	0.69	6630	4121	37.84%
32	0.85	0.82	11636	8549	26.53%
64	1.02	0.95	24485	17061	30.32%
average area saving: 31.56%					
Note : Delay(ns) / Area(um^2)					

Table 12 shows the comparison between the K-S adder which uses method2 (K-S_2) and the proposed adder. From the result of the experiment, our proposed architecture can save 26.46% area on average and get a speed-improvement.

Table 12.Experimental results of K-S_2 and proposed adders.

Timing & Area Analysis (UMC: 0.18um/TT corner)					
Data Width	K-S_2 (Delay)	Proposed (Delay)	K-S_2 (Area)	Proposed (Area)	Area Saving
16	0.71	0.69	5888	4121	30.01%
32	0.84	0.82	11213	8549	23.76%
64	1.05	0.95	22939	17061	25.62%
average area saving: 26.46%					
Note : Delay(ns) / Area(um^2)					

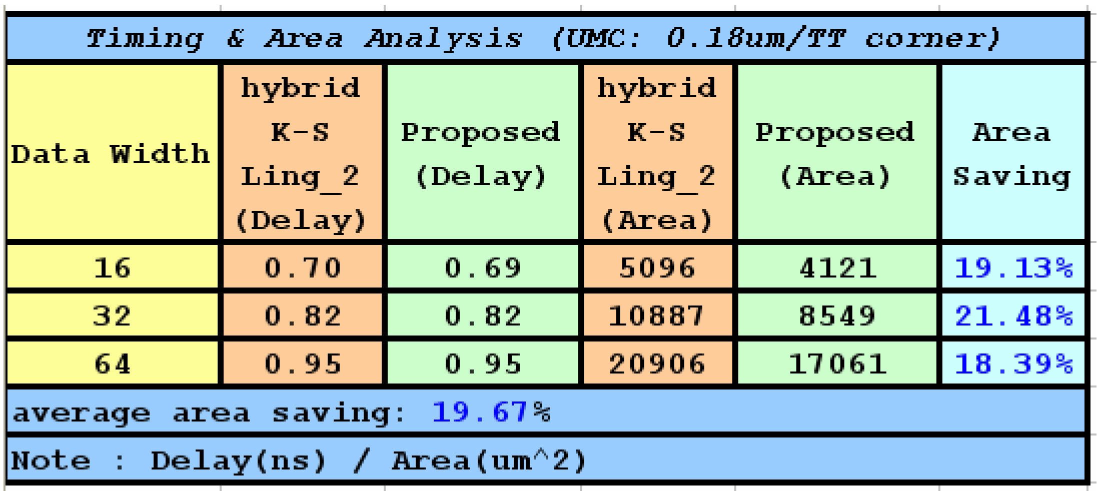
Table 13 shows the comparison between the hybrid K-S Ling adder which uses method1 (hybrid K-S Ling_1) and the proposed adder. From the result of the experiment, our proposed architecture can save 22.33% area on average and preserve the advantage of high-speed by using Ling addition.

Table 13. Experimental results of hybrid K-S Ling_1 and proposed adders.

Timing & Area Analysis (UMC: 0.18um/TT corner)					
Data Width	hybrid K-S Ling_1 (Delay)	Proposed (Delay)	hybrid K-S Ling_1 (Area)	Proposed (Area)	Area Saving
16	0.69	0.69	5575	4121	26.08%
32	0.83	0.82	10418	8549	17.94%
64	0.96	0.95	22150	17061	22.98%
average area saving: 22.33%					
Note : Delay(ns) / Area(um^2)					

Table 14 shows the comparison between the hybrid K-S Ling adder which uses method2 (hybrid K-S Ling_2) and the proposed adder. From the result of the experiment, our proposed architecture can save 19.67% area on average and preserve the advantage of high-speed by using Ling addition.

Table 14.Experimental results of hybrid K-S Ling_2 and proposed adders.



Timing & Area Analysis (UMC: 0.18um/TT corner)					
Data Width	hybrid K-S Ling_2 (Delay)	Proposed (Delay)	hybrid K-S Ling_2 (Area)	Proposed (Area)	Area Saving
16	0.70	0.69	5096	4121	19.13%
32	0.82	0.82	10887	8549	21.48%
64	0.95	0.95	20906	17061	18.39%
average area saving: 19.67%					
Note : Delay(ns) / Area(um^2)					

From the above results, it is easily to see that the proposed architecture preserve the high-speed of Ling addition and do everything possible to minimize the overall area. However, the low-power is also an important factor to design a good adder or IC. In general, the area and power are in direct proportion under the same frequency. Therefore, the proposed architecture is expected to get an improvement on reducing power dissipation. All designs in the following table are all under their fastest speed. According to the results of Table 15, the proposed adders can achieve power reductions of up to 21% when compared to K-S adders. From the results of Table 16, the proposed adders can achieve power reductions of up to 17% when compared to hybrid K-S Ling adders.

Table 15.Power estimation of K-S adders and proposed adders.

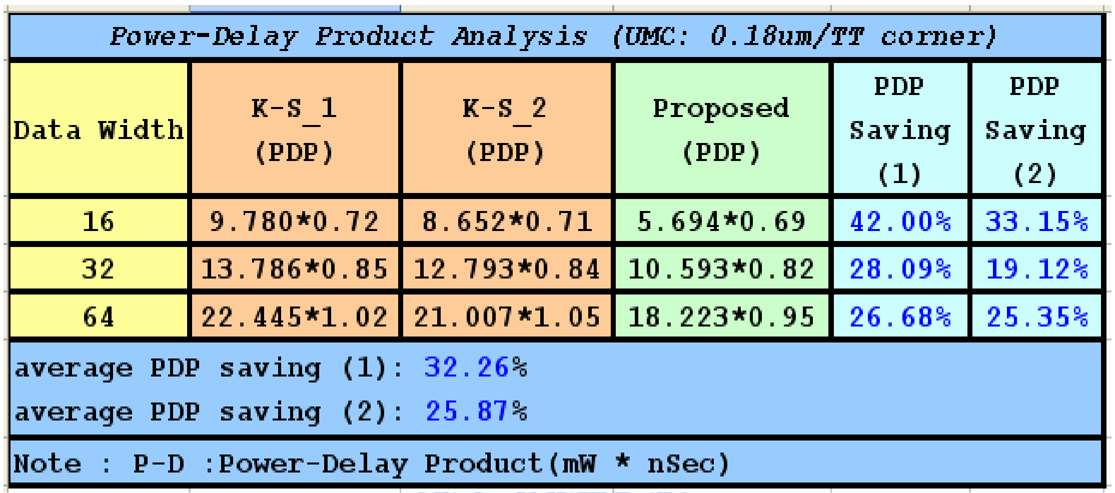
Power Analysis (UMC: 0.18um/TT corner)					
Data Width	K-S_1 (Power)	K-S_2 (Power)	Proposed (Power)	Power Saving (1)	Power Saving (2)
16	9.780	8.652	5.694	41.78%	34.19%
32	13.786	12.793	10.593	23.16%	17.20%
64	22.445	21.007	18.223	18.81%	13.25%
average power saving (1): 27.91% average power saving (2): 21.55%					
Note : Power(mW)					

Table 16.Power estimation of hybrid K-S Ling adders and proposed adders.

Power Analysis (UMC: 0.18um/TT corner)					
Data Width	hybrid K-S Ling_1 (Power)	hybrid K-S Ling_2 (Power)	Proposed (Power)	Power Saving (1)	Power Saving (2)
16	8.100	7.268	5.694	29.70%	21.66%
32	12.515	12.853	10.593	15.36%	17.58%
64	22.159	21.347	18.223	17.76%	14.63%
average power saving (1): 20.94% average power saving (2): 17.96%					
Note : Power(mW)					

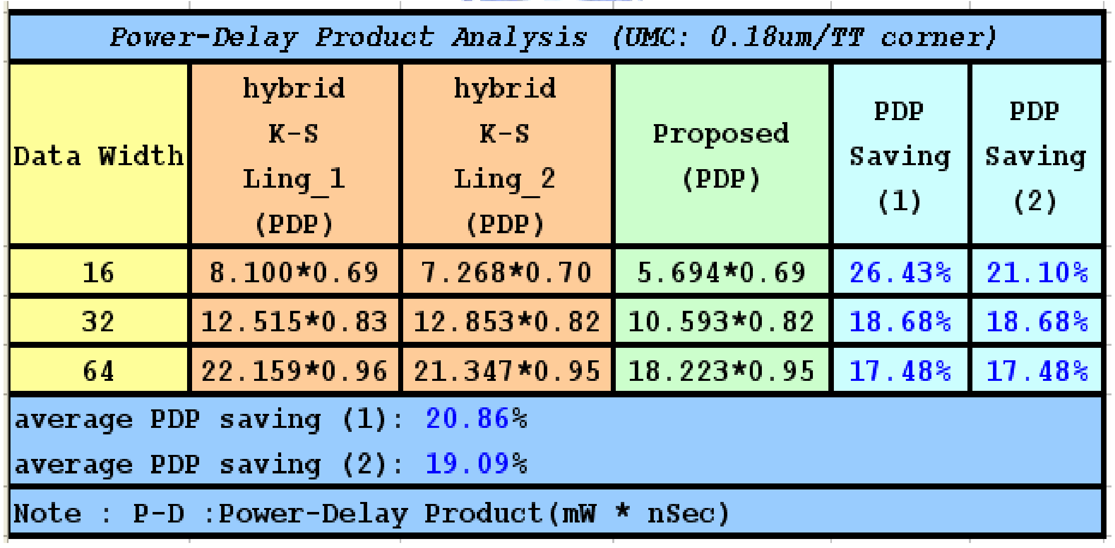
However, the frequency (or delay) affects the simulation results of power greatly. The faster adder may cause the higher power results form its high frequency (or low delay). The more accurate analysis of energy dissipation is to compare the new measure: Power-Delay Product (PDP). Using PDP to analyze the designs is the same as measuring the power of each design under the same frequency. From Table 17 and Table 18, the energy-saving percentage is closed to the area-saving percentage. Compared to K-S adders, the PDP saving percentage is up to 25%. Even though the proposed adders compare with hybrid K-S Ling adders, we still can save nearly 20% PDP. So, the proposed high-speed architecture meets the goal of low-power and area reduction.

Table 17.PDP estimation of K-S adders and proposed adders.



Power-Delay Product Analysis (UMC: 0.18um/TT corner)					
Data Width	K-S_1 (PDP)	K-S_2 (PDP)	Proposed (PDP)	PDP Saving (1)	PDP Saving (2)
16	9.780*0.72	8.652*0.71	5.694*0.69	42.00%	33.15%
32	13.786*0.85	12.793*0.84	10.593*0.82	28.09%	19.12%
64	22.445*1.02	21.007*1.05	18.223*0.95	26.68%	25.35%
average PDP saving (1): 32.26%					
average PDP saving (2): 25.87%					
Note : P-D :Power-Delay Product(mW * nSec)					

Table 18.PDP estimation of hybrid K-S Ling adders and proposed adders.



Power-Delay Product Analysis (UMC: 0.18um/TT corner)					
Data Width	hybrid K-S Ling_1 (PDP)	hybrid K-S Ling_2 (PDP)	Proposed (PDP)	PDP Saving (1)	PDP Saving (2)
16	8.100*0.69	7.268*0.70	5.694*0.69	26.43%	21.10%
32	12.515*0.83	12.853*0.82	10.593*0.82	18.68%	18.68%
64	22.159*0.96	21.347*0.95	18.223*0.95	17.48%	17.48%
average PDP saving (1): 20.86%					
average PDP saving (2): 19.09%					
Note : P-D :Power-Delay Product(mW * nSec)					

Chapter 4 Reconfigurable Adder Architecture

Recently, a high-speed reconfigurable adder plays an important role for achieving real-time processing of media signals. Thus a fast and reconfigurable architecture for addition is needed. This chapter presents a methodology to design a high-speed area-minimized reconfigurable hybrid Ling adder. The proposed approach is based on the architecture presented in Chapter 3. The delay-penalty or area cost for the partition scheme in the proposed architecture is small. In other words, there are few modifications in the original architecture. Finally, a systematic design methodology of the reconfigurable adder is presented in the later section.

4.1 Review of Previous Approach

Several reconfigurable adders have been proposed in [14-19], A reconfigurable ripple carry adder in [16] uses additional bits for partition. Each partition bit determines the propagation of the carry signal generated from previous segment of addition. Fig. 31 shows the 32-bit reconfigurable adder using additional 4 bits to support partition. Obviously, this approach causes large delay penalty and area cost. In [19], a reconfigurable carry-skip adder has been proposed which minimizes the energy-delay product by using non-uniform linearly increasing block sizes. A 64-bit reconfigurable adder in displayed in Fig. 32. In [24], the reconfigurable carry-select adder is proposed. It is faster than the reconfigurable ripple carry adder in [16]. But, the architecture still locates partition scheme in the critical path. Fig. 33 shows a 12-bit reconfigurable CSA. The reconfigurable hybrid carry-lookahead/carry-select adder has been proposed in [25]. The partition approaches are located in carry-lookahead blocks and carry-select blocks. The partition scheme incurs no additional delay in the critical path regardless of the size of adders. The additional area cost is also small. A 16-bit reconfigurable hybrid carry-lookahead/carry-select adder (CLSA) is presented in Fig. 34. However, it achieves high-speed by reducing the variety of reconfigurability. Compared with other approaches, the proposed reconfigurable adders can achieve the goal of reducing timing penalty avoiding inserting the partition scheme into the critical path. Moreover, the high reconfigurability and small area cost are the features of our approach.

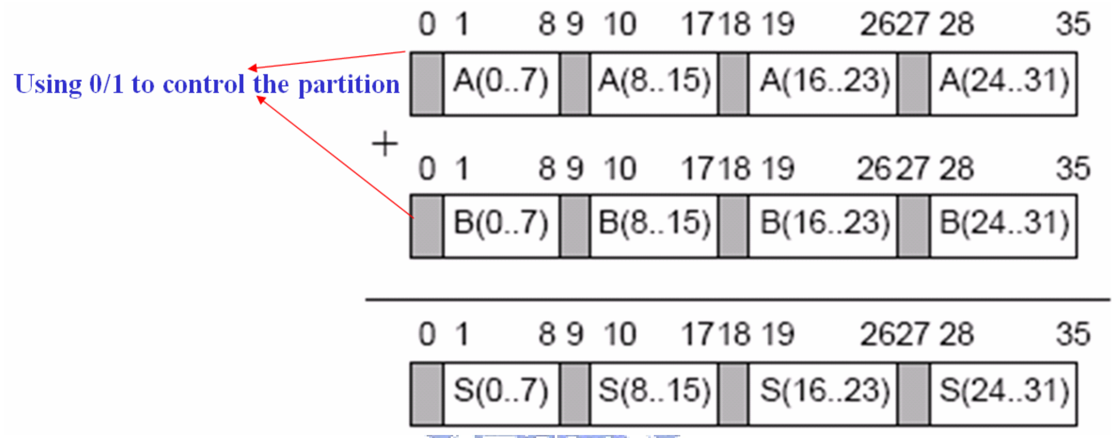


Fig. 31. A 32-bit reconfigurable carry ripple adder.

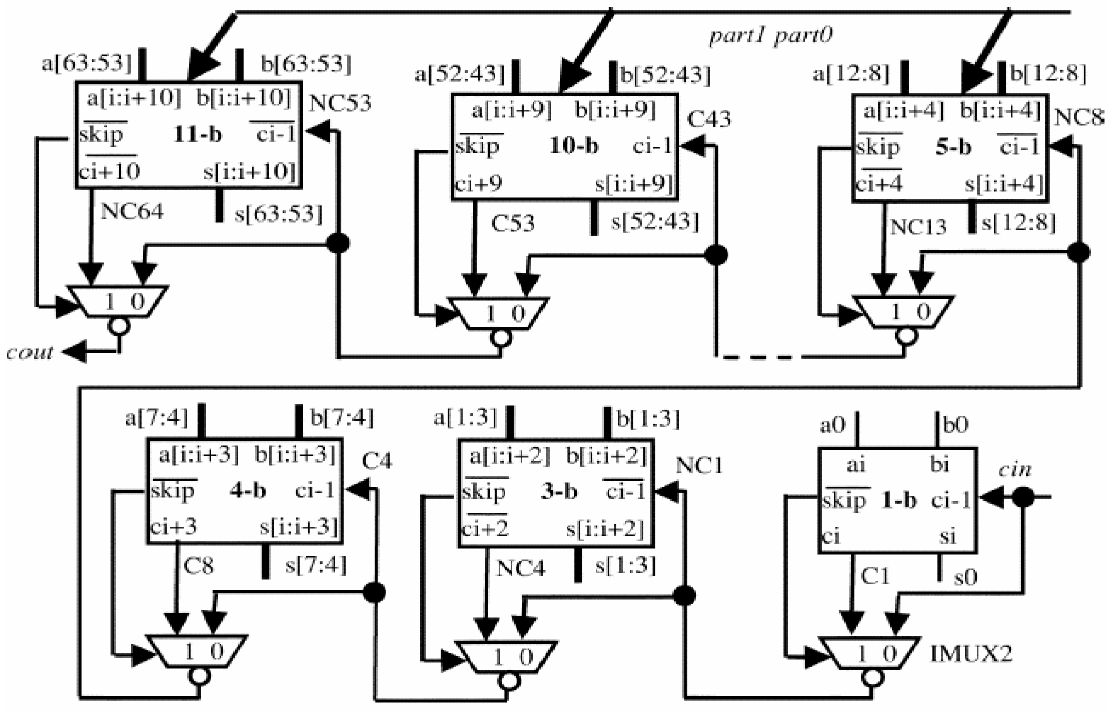


Fig. 32. A 64-bit reconfigurable carry-skip adder.

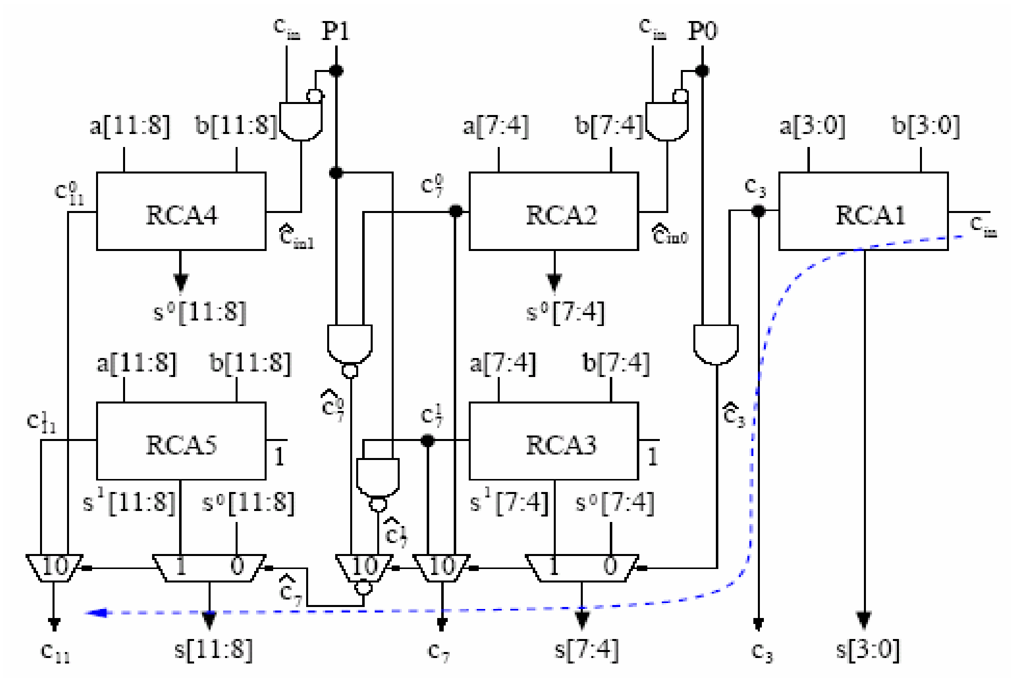


Fig. 33. A 12-bit reconfigurable carry-select adder.

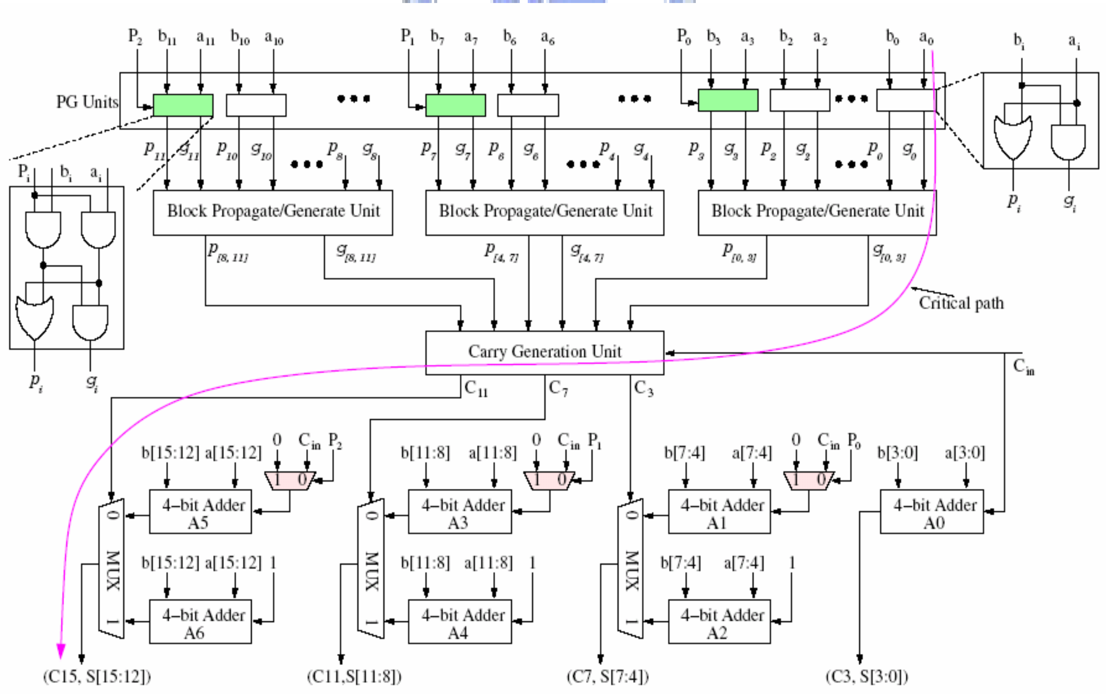


Fig. 34. A 16-bit reconfigurable CLSA adder.

4.2 Proposed Architecture

The new reconfigurable blocks need to be inserted into the original architecture. The ☒ cell uses the signal break_i to break the propagate signal and generate signal from previous block. The ■ cell also uses the signal break_i to select the partial carry-out from previous block or the external carry-in for partition. The ⊞ cell combine the "&2" cell and ☒ cell. The function of the ⊞ cell is the same as the ☒ cell. The Fig. 35 shows the logic implementation of the three reconfigurable blocks. The delay of each reconfigurable block is 2-level logic. Then, these blocks are used in the stage of handling carry-in. Because the logic depth of these blocks is the same as the logic cell of carry-in operation, the partition approaches don't incur delay-penalty on the critical path and only need small additional area cost.

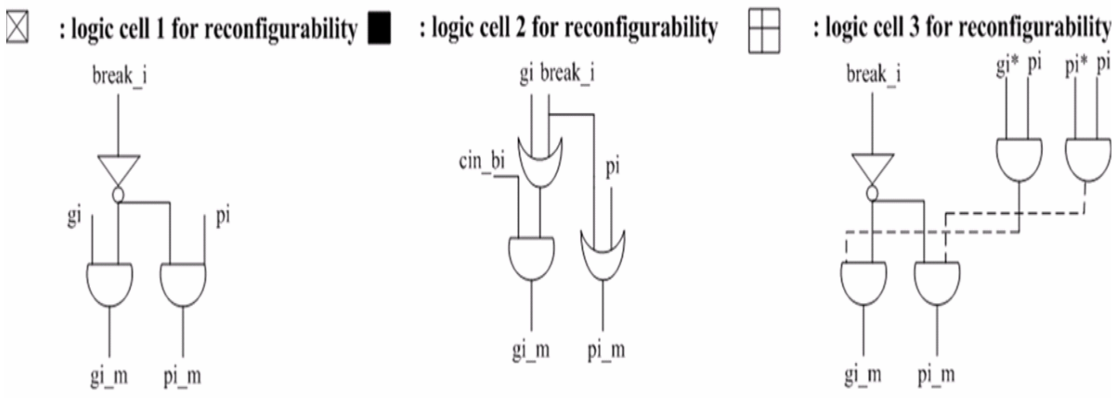

☒ : logic cell 1 for reconfigurability
■ : logic cell 2 for reconfigurability
⊞ : logic cell 3 for reconfigurability
break_i
gi
pi
gi_m
pi_m
gi break_i
cin_bi
pi
gi_m
pi_m
break_i
gi* pi
pi* pi
gi_m
pi_m

Fig. 35. Reconfigurable blocks.

The CSA blocks at the boundary of partition also need some modification. The simple carry-select adder for reconfigurability (SCSA-R) is used in the proposed architecture. The SCSA-R adds some logic for the generation of partial carry-out. The Fig. 36 presents the architecture of a 4-bit SCSA-R.

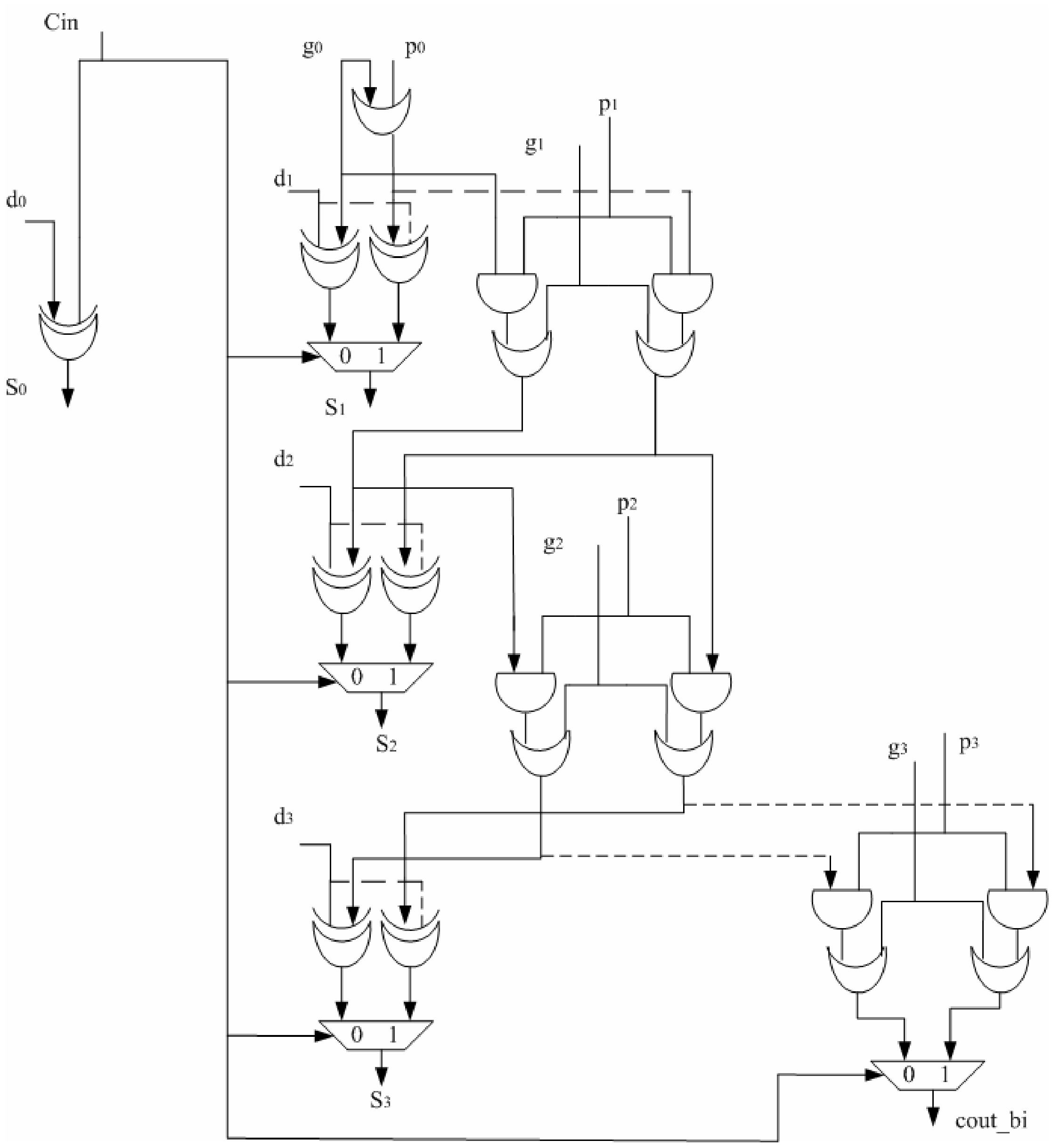


Fig. 36. A 4-bit SCSA-R.

For the purpose of balancing the fanout of carries generated from the parallel-prefix tree, the partial parallel-prefix tree in the left-hand side is used to generate final carry-out instead of using carry-select block. Although using the SCSA-R block to be the last CSA block can produce the final carry-out, the fanout of the last carry to select sum bits and carry-out is larger than other carries. However, the reconfigurable architecture needs to use the SCSA-R blocks. Consequently, the part of parallel-prefix tree for final carry-out can be saved to reduce area. By this way, the area cost for partition also can be minimized. This unbalanced fanout may cause some timing penalty, but the additional delay is small. According to the experimental results in the later section, the timing penalty is only 5.12%. A 32-bit proposed reconfigurable adder is presented in Fig. 37. The different parts are also illustrated in the graph. Finally, we introduce the method to control the partition by using the break_i signal. For example, four individual 8-bit additions can be derived by setting the partition signals (break_2, break_1, break_0) as (1, 1, 1). When all break_i signals are zeros, an entire 32-bit addition can be achieved. The part needed to pay attention is that the cin_bi relative to break_i should be setted by one when the partition is unnecessary. Other cases of a 32-bit proposed reconfigurable adder are presented in Table 19.

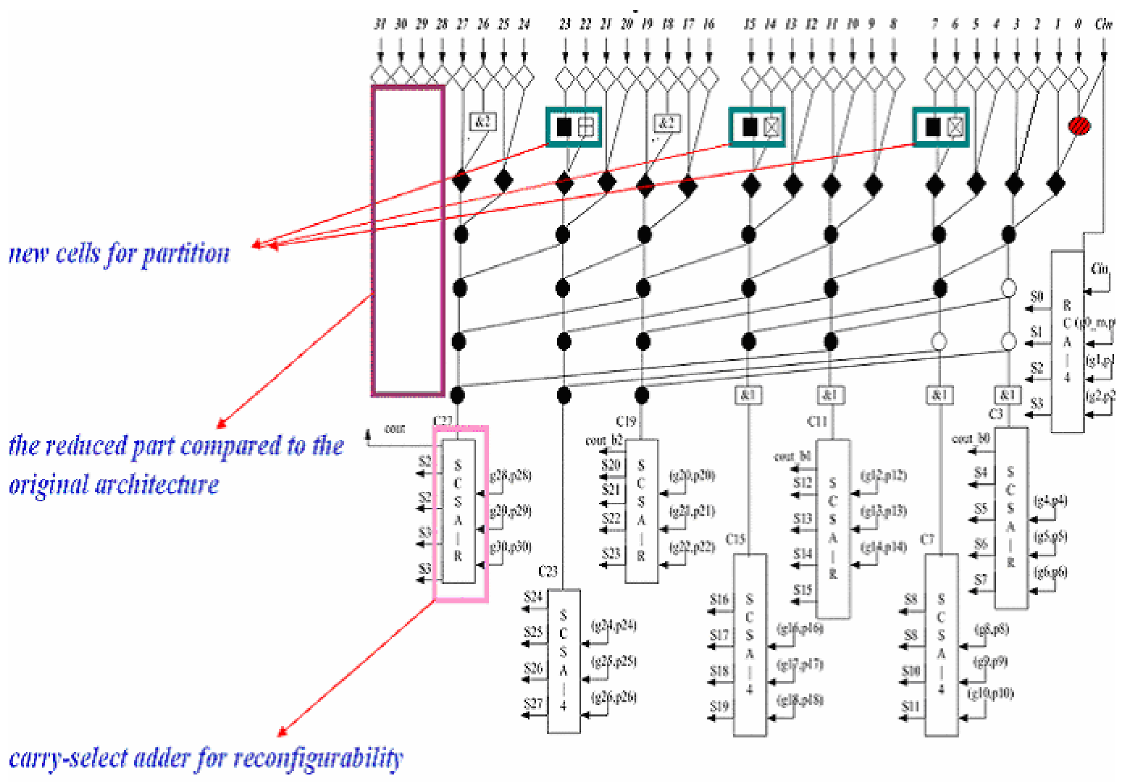

new cells for partition
the reduced part compared to the original architecture
carry-select adder for reconfigurability
Cin
cout
cout_b2
cout_b1
cout_b0
C27
C23
C19
C15
C11
C7
C3
SCSA-R
SCSA-4
RCA-4

Fig. 37. A 32-bit proposed reconfigurable adder.

Table 19. Partition Methods.

break_2	break_1	break_0	(cin_b0,cin_b1,cin_b2)	partition scheme
0	0	0	(1 , 1 , 1)	a 32-bit addition
0	0	1	(1 , 1 , DC)	(24 , 8)
0	1	0	(1 , DC , 1)	(16 , 16)
0	1	1	(1 , DC , DC)	(16 , 8 , 8)
1	0	0	(DC , 1 , 1)	(8 , 24)
1	0	1	(DC , 1 , DC)	(8 , 16, 8)
1	1	0	(DC , DC , 1)	(8 , 8 , 16)
1	1	1	(DC , DC , DC)	(8 , 8 , 8 , 8)
DC : Don't Care ; Controlled by User				

4.3 Summary of Proposed Architecture

According to the basic components and main idea introduced in previous sections, the design of proposed reconfigurable architecture can be summarized in the following steps.

- Generate the bit-generate (g_n) and bit-propagate (p_n), and half-sum bit (d_n) at the first stage.
- Handle the carry-in signal by using the proposed method. To insert the "&2" cells into the specific locations of generate and propagate pairs (g_n, p_n), with n = 4k-2 and k = N/8+1, N/8+2, …, N/4-1 (N>=16). If the position of "&2" cell is at the partition-boundary should be replaced by the ⊞ cell. The ⊠ cell is used in the location of (g_n, p_n) pairs, with n = m-2 and m = k, 2k, …, N/2 (k = the minimum block-size for partition). In the end, the ■ cell is inserted in the position of (g_n, p_n) pairs, with n = m-1 and m = k, 2k, …, N-k.
- Use Equation(9) and Equation(10) to generate the intermediate generate and propagate pairs (G_n*, P_n*) for Ling addition.
- Use the pairs (G_n*, P_n*) to build the parallel-prefix tree mentioned in section 3.2.1. The parallel-prefix structure can be employed for the generation of the pseudo carries H_{4k-1}, k = 1, 2, …, N/4-1.
- Replace the specific buffer nodes by the "&1" cells in parallel-prefix trees. These buffer nodes pass pseudo carries H_{4k-1}, k = 1, 2, …, N/8. After the replacement, these pseudo carries are transferred to normal carries C_{4k-1}.
- Select correct sum-bits and partial carry-out signals produced from SCSA-R blocks by using normal carries C_{4k-1}, k = 1, 2, …, N/4-1.

In Fig. 38, the design steps of proposed architecture are illustrated. The order of steps conforms to the design methodology mentioned above. Therefore, we can design a high-speed and area-minimized reconfigurable adder from our proposed approaches. These approaches preserve the good properties of original architecture. In the end, a systematic methodology has been introduced for designing the proposed reconfigurable architecture in this section.

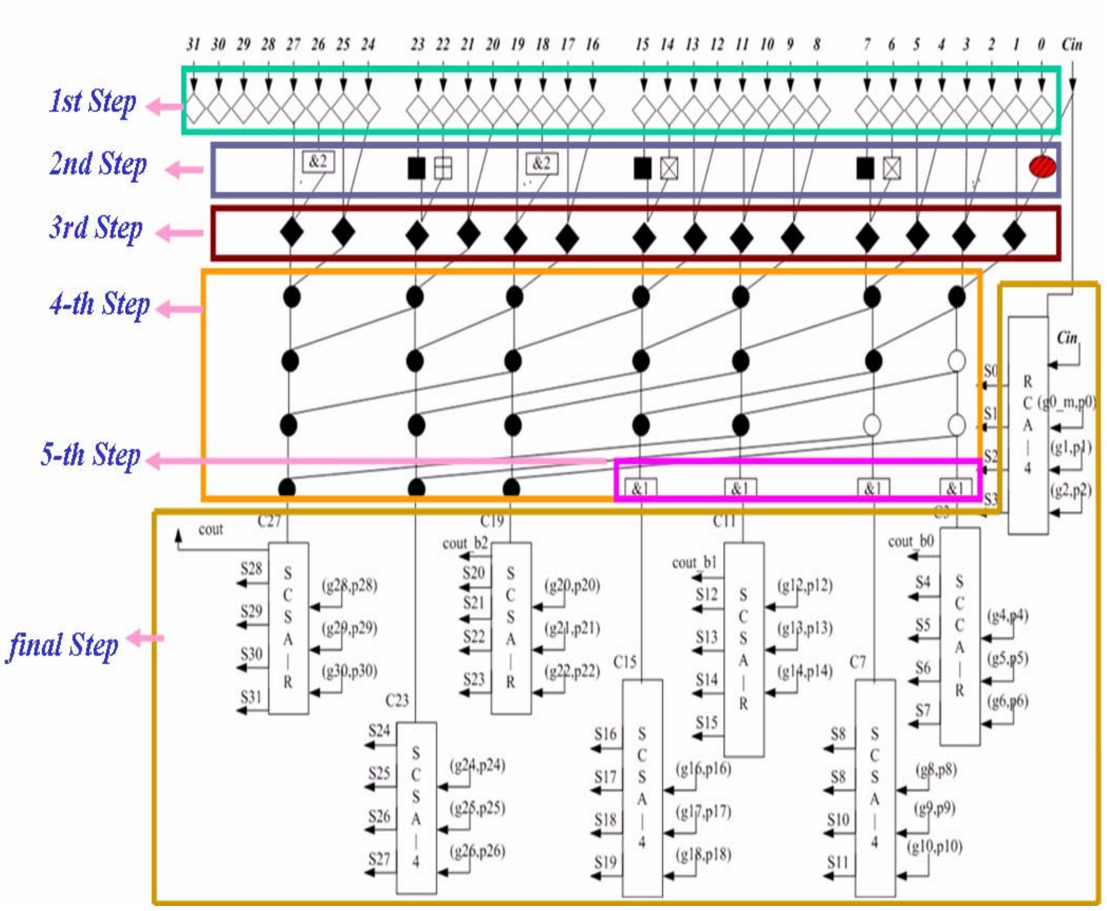


Fig. 38. Design steps of the proposed reconfigurable architecture.

4.4 Experimental Results and Analysis

After introducing the main idea and basic components in previous sections, the advantages of proposed architecture can be proved by the following experimental results. First, Table 20 gives the simple description of the experimental environment.

Table 20.Experimental environment 2.

Experimnetal Enviroment	
HDL	Verilog
Process	UMC 0.18um (under SS corner) TSMC 0.18um (under TT corner)
Synthesis Tool	Synopsys Design Compiler (Version : W-2004.12-SP2-2)
Partition Size	8-bit block

Normally, the key of designing a reconfigurable adder successfully is to minimize the area overhead and timing penalty. According to the results in Table 21, the proposed reconfigurable architecture (Prop.-R) can achieve the goal of small penalty on area and timing indeed. The additional area cost of the partition scheme is only about 3.98% and the timing penalty is about 5.12% when compared to the proposed architecture without reconfigurability (Prop.).

Table 21.Area and Timing penalty of proposed reconfigurable adders.

Timing & Area Analysis (UMC: 0.18um / SS coner)						
Data Width	Prop.-R (Delay)	Prop. (Delay)	Prop.-R (Area)	Prop. (Area)	Timing Penalty	Area Penalty
16	1.72	1.6	5528	6167	7.50%	-10.36%
32	2.06	1.95	11849	11230	5.64%	5.51%
64	2.3	2.25	24333	23754	2.22%	2.44%
average timing penalty: 5.12% /average area penalty: 3.98%						
Note : Delay(ns) / Area(um^2)						

There are two reference reconfigurable adders for comparison. One is like the reconfigurable ripple-carry adder mentioned in previous work. The multiplexer which is used to be the partition scheme which selects the carry-out signal form previous block or the external carry-in signal. All blocks of addition are described in RTL (+). Fig. 39 shows the first type of the reconfigurable adder (adder-R type1). Another reconfigurable adder is the reconfigurable carry-select adder (adder-R type2). The architecture also uses the multiplexer to be the partition scheme. The critical path of the adder is on the multiplexer-chain except the first block of addition. Therefore, it is fast than the first reconfigurable adder. All blocks of addition are also described in RTL (+). The architecture view is presented in Fig. 40.

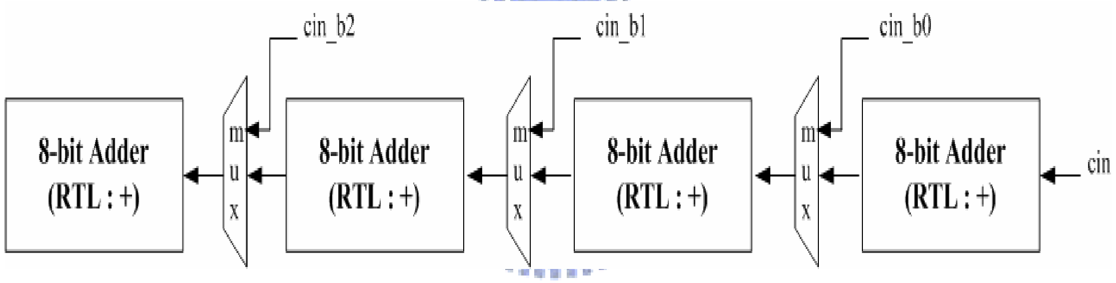


Fig. 39. The first type of the reconfigurable adder.

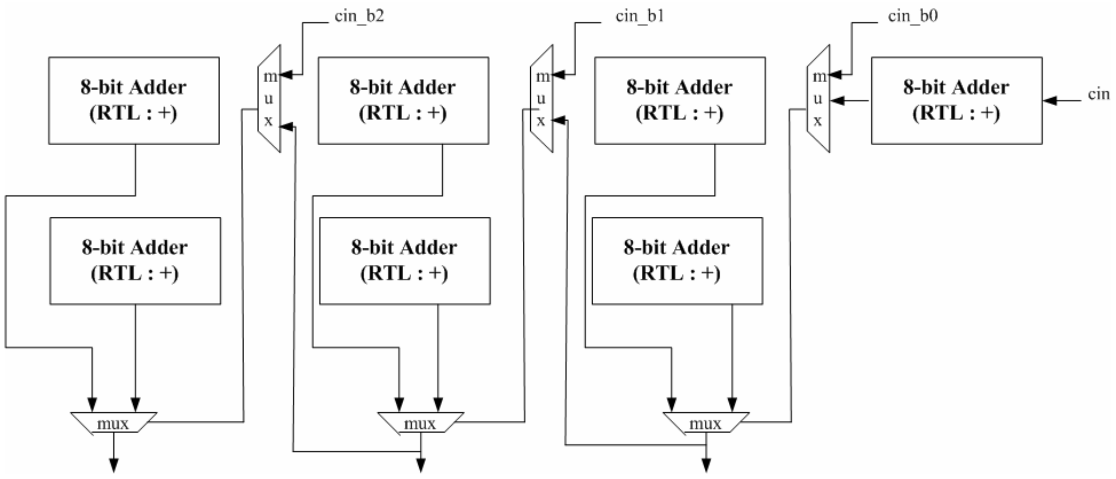
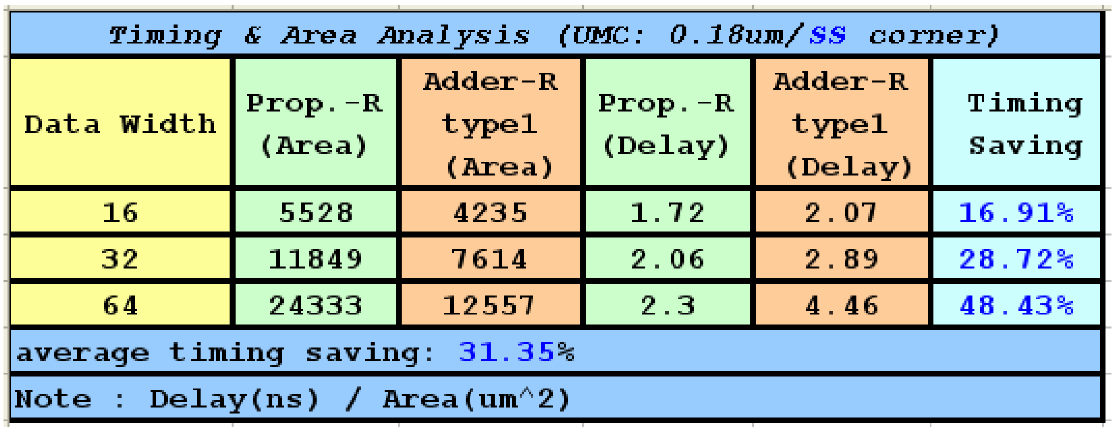


Fig. 40. The second type of the reconfigurable adder.

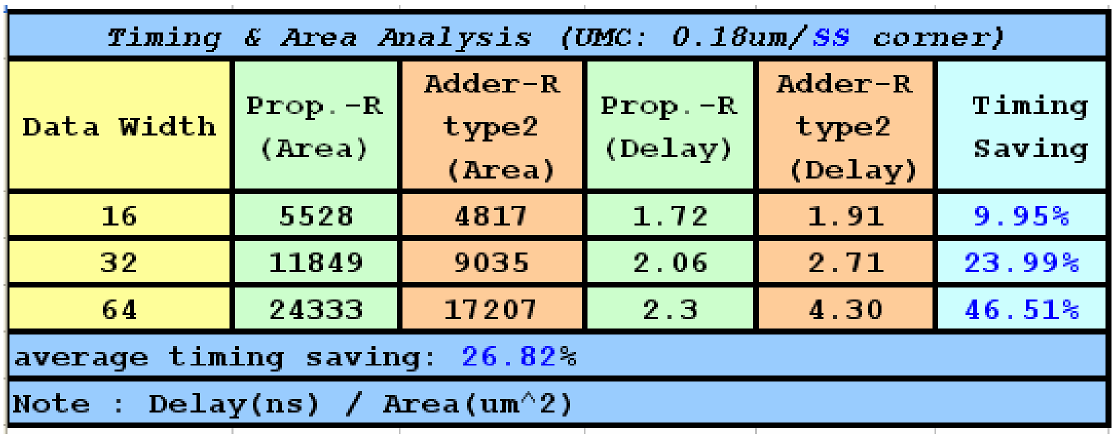
The proposed reconfigurable architecture compares with the two reconfigurable adders mentioned above. From Table 22, the proposed reconfigurable adders (Prop.-R) achieve delay reductions of up to 31 percent on average when compared to the first type of reconfigurable adder. Because the partition scheme of adder-R type1 is located in the carry-chain, the delay penalty grows greatly when the size becomes large. From Table 23, the proposed adders achieve delay reductions of up to 26 percent on average when compared to the second type of reconfigurable adder. From the utilization of CSA blocks, the critical path is not on the generation of sum bits or carries. But, it still uses the multiplexer to select the boundary carry form previous block or the individual carry for partition. Unfortunately, these multiplexers are still on the critical path. So, this kind of reconfigurable adder also has the opportunity to gain the more improvement on speed. Finally, Table 24 presents the comparison between our 64-bit reconfigurable adder and the adder in [24]. The architecture in [24] and the second type reconfigurable adder have the same problem in critical path. Both of them insert the partition scheme on their critical path. According to these results, it obviously shows that the performance-improvement and data-width are in the direct ratio. The reconfigurable architecture designed by the proposed methodology can meet the target of high-speed and small penalty of partition scheme. Moreover, the penalty for partition would not increase greatly when the data width becomes large in our proposed adder design.

Table 22.Timing analysis of adder-R type1 and proposed-R.



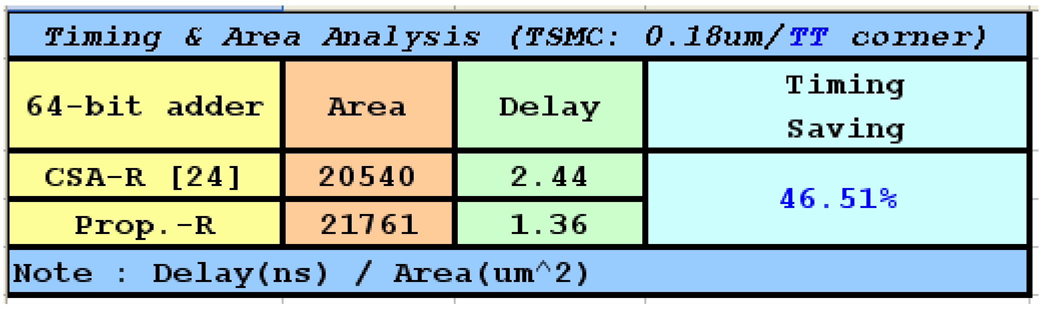
Timing & Area Analysis (UMC: 0.18um/SS corner)					
Data Width	Prop.-R (Area)	Adder-R type1 (Area)	Prop.-R (Delay)	Adder-R type1 (Delay)	Timing Saving
16	5528	4235	1.72	2.07	16.91%
32	11849	7614	2.06	2.89	28.72%
64	24333	12557	2.3	4.46	48.43%
average timing saving: 31.35%					
Note : Delay(ns) / Area(um^2)					

Table 23.Timing analysis of adder-R type2 and proposed-R.



Timing & Area Analysis (UMC: 0.18um/SS corner)					
Data Width	Prop.-R (Area)	Adder-R type2 (Area)	Prop.-R (Delay)	Adder-R type2 (Delay)	Timing Saving
16	5528	4817	1.72	1.91	9.95%
32	11849	9035	2.06	2.71	23.99%
64	24333	17207	2.3	4.30	46.51%
average timing saving: 26.82%					
Note : Delay(ns) / Area(um^2)					

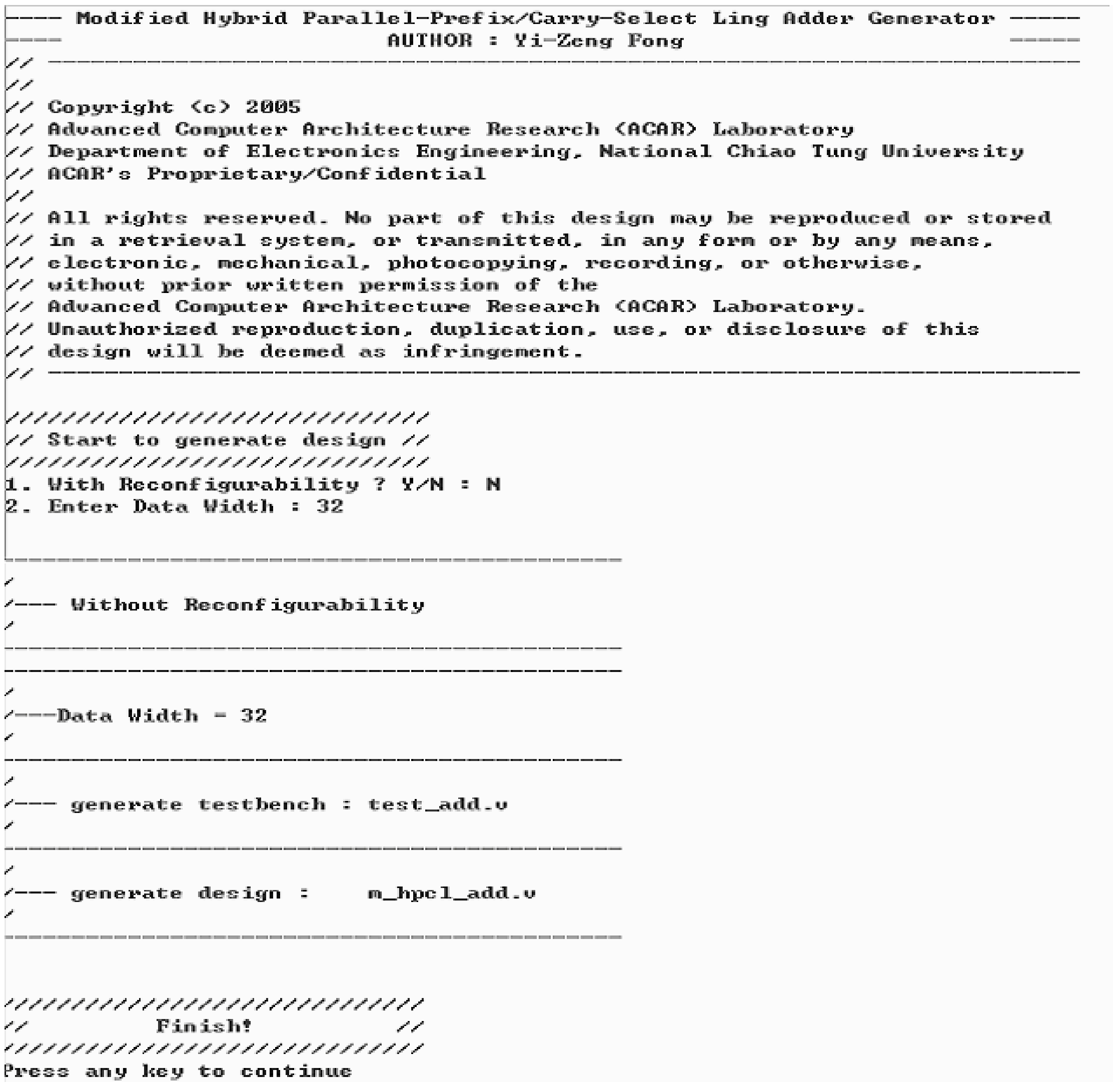
Table 24.Timing analysis of CSA-R [24] and proposed-R.



Timing & Area Analysis (TSMC: 0.18um/TT corner)			
64-bit adder	Area	Delay	Timing Saving
CSA-R [24]	20540	2.44	46.51%
Prop.-R	21761	1.36	
Note : Delay(ns) / Area(um^2)			

Chapter 5 RTL Code Generator

This chapter introduces the RTL code generator of our proposed architecture. Section 3.3 and Section 4.3 describe the systematic design methodology of the proposed adders. According to the methodology, we can derive the RTL code generator. In the first version, the coding environment is the Visual C++R 6.0. Then, an execution file (.exe) of MS-DOS version has been generated to be the RTL code generator. The graphical user interface (GUI) is the main target for our next version. Fig. 41 shows the RTL code generator of MS-DOS version. Our proposed generator provides not only RTL code, but also the testbench code.



```
---- Modified Hybrid Parallel-Prefix/Carry-Select Ling Adder Generator -----
----                      AUTHOR : Yi-Zeng Fong                         -----
// -------------------------------------------------------------------------
//
// Copyright (c) 2005
// Advanced Computer Architecture Research (ACAR) Laboratory
// Department of Electronics Engineering, National Chiao Tung University
// ACAR's Proprietary/Confidential
//
// All rights reserved. No part of this design may be reproduced or stored
// in a retrieval system, or transmitted, in any form or by any means,
// electronic, mechanical, photocopying, recording, or otherwise,
// without prior written permission of the
// Advanced Computer Architecture Research (ACAR) Laboratory.
// Unauthorized reproduction, duplication, use, or disclosure of this
// design will be deemed as infringement.
// -------------------------------------------------------------------------

/////////////////////////////////
// Start to generate design //
/////////////////////////////////
1. With Reconfigurability ? Y/N : N
2. Enter Data Width : 32


-----------------------------------------------
/
/--- Without Reconfigurability
/
-----------------------------------------------
-----------------------------------------------
/
/---Data Width = 32
/
-----------------------------------------------
/
/--- generate testbench : test_add.v
/
-----------------------------------------------
/
/--- generate design :     m_hpcl_add.v
/
-----------------------------------------------



////////////////////////////////
//         Finish!          //
////////////////////////////////
Press any key to continue
```

Fig. 41. RTL code generator.

Chapter 6 Conclusions and Future Works

A high-speed area-minimized adder design has been presented. A systematic methodology is also introduced in previous chapters. The proposed architecture preserves the benefits of hybrid K-S Ling adders. Moreover, the methods for parallel-prefix tree minimization and CSA optimization are used to further reduce area. According the experimental results, the area saving percentage is up to 26% when compared to traditional K-S adders. Compared with hybrid K-S adder, the saving percentage is about 20%. Meanwhile, the power dissipation is also reduced from the reduction of area. For achieving real-time media signals processing, the proposed reconfigurable adder is also presented. To design an efficient reconfigurable adder, the proposed partition scheme for reconfigurability causes only small delay-penalty and area cost. According to experimental results, the delay penalty is closed to 5% and additional area overhead is lower than 4%. Compared to other reconfigurable adders, our architecture has a great improvement on speed from the low penalty of timing. From these results, we can draw that the adder here presented exhibits the high-speed, the minimized area, the more power-saving, and low overhead for high reconfigurability. However, the Ling addition can only save one logic-level delay. Consequently, the new algorithm [26] may be applied to our architecture for reducing more delay than Ling addition. A more user-friendly RTL code generator is also needed.

References

[1] K. Hwang, Computer Arithmetic. New York: Wiley, 1979.

[2] D, J, Kuck, The Structure of Computers and Computations. New York: Wiley, 1978.

[3] I. Koren, Computer Arithmetic Algorithms. A.K. Peters, Ltd., 2002.

[4] B. Parhami, Computer Arithmetic-Algorithms and Hardware Designs. Oxford Univ. Press, 2000.

[5] O.J. Bedrij, “Carry Select Adder”, IRE Trans., EC-11, pp.340-346, June 1962.

[6] A. Weinberger and J.L Smith, “A Logic for High-Speed Addition”, Nat. Bur. Stand. Circ., 591, pp.3-12, 1958.

[7] P.M. Kogge and H.S. Stone, “A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations,” IEEE Trans. Computers, vol.22, no. 8, pp. 786-792, Aug. 1973.

[8] R.E. Lander and M.J. Fisher, “Parallel Prefix Computation,” J. ACM, vol. 27, no. 4, pp. 831-838, Oct. 1980.

[9] R.P. Bent and H.T.Kung, “A Regular Layout for Parallel Adders,” IEEE Trans. Computers, vol. 31, no. 3, pp. 260-264, Mar. 1982.

[10] T. Han and D. Carlson, “Fast Area-Efficient VLSI Adders,” Proc. Symp. Computer Arithmetic, pp. 49-56, May 1987.

[11] S. Knowles, “A Family of Adders,” Proc. 14th Symp. Computer Arithmetic, pp. 30-32, Apr. 1999. Reprinted in ARITH-15, pp. 277-281.

[12] H. Ling, “High-Speed Binary Adder,” IBM J.R&D, vol. 25, pp. 156-166, May 1981.

[13] Dimitrakopoulos, G.; Nikolos, D.," High-speed parallel-prefix VLSI Ling adders", IEEE Trans. Computers, vol. 54, Issue 2, pp. 225-231, Feb. 2005.

[14] R.-B. Lee, "Subword parallelism with MAX-2", IEEE Micro, vol. 16, no. 4, pp. 51-59, Aug. 1996.

[15] M. Tremblay, J.,-M. O'Connor, V. Narayanan, and H. Liang, "VIS speeds new media processing", IEEE Micro, vol. 16, no. 4, pp. 10-20, Aug. 1996.

[16] M. S. Schmookler, M. Putrino, A. Mather, J. Tyler, and H. V. Nguyen, "A low-power, high-speed implementation of a PowerPC^{TM} Microprocessor vector extension", IEEE Symp. Computer Arithmetic, pp. 12-19, 1999.

[17] A. A. Farooqui, V. G. Oklobdzija, and F. Chechrazi, "64-bit media adder," in Proc. IEEE Int. Symp. Circuits and Systems (ISCAS), Orlando, May 1999.

[18] S. Perri, P. Corsonello, and G. Cocorullo, "A 64-bit reconfigurable adder for low power media processing", Electronics Letters, vol. 38, no. 9, pp. 397-399, Apr. 2002.

[19] S. Perri, P. Corsonello, and G. Cocorullo, "A high-speed energy-efficient 64-bit reconfigurable binary adder," IEEE Trans. VLSI Systems, vol. 11, no. 5, pp. 939-943, Oct. 2003.

[20] A. Goldovsky et al., "A 1.0-nsec 32-bit Prefix Tree Adder in 0.25-um static CMOS," Proc. Midwest Symp. Circuits and Systems, vol. 2, pp. 608-612, Aug. 1999.

[21] Y. Wang , C. Pai Xiaoyu Song, "The Design of Hybrid Carry-Lookahead/Carry Select Adders", IEEE Trans. circuits and systems-II: analog and digital signal processing, vol.49, no. 1, Jan. 2002.

[22] R. Zimmermann, "Binary Adder Architectures for Cell-Based VLSI and Their Synthesis", Ph.D. dissertation, Swiss Federal Institute of Technology (ETH), Zurich, 1998.

[23] Harris, D. Sutherland, I. Harvey Mudd Coll., Claremont, "Logical effort of carry propagate adders", Proceedings of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers, vol. 1, pp. 873-878, Nov. 2003.

[24] Jin-Fu Li, Yao-Chang Kuo, Chao-Da Huang, Tsu-Wei Tseng, Chin-Long Wey,"Design of Reconfigurable Carry select Adders", IEEE Asia-Pacific Conference on Circuits and Systems, Mar. 2004.

[25] Jin-Fu Li, Jiunn-Der Yu, Yu-Jen Huang, "A design methodology for hybrid carry-lookahead/carry-select adders with reconfigurability", IEEE Int. Symp. Circuits and Systems, vol. 1, pp. 77-80, May 2005.

[26] Jackson, R., Talwar, S., "High speed binary addition", Thirty-Seventh Asilomar Conference on Signals, Systems and Computers, vol. 2, pp. 1350-1353, Nov. 2004.