# Area-Efficient Parallel-Prefix Ling Adders

Tso-Bing Juang[#1], Pramod Kumar Meher[*2], Chung-Chun Kuan[#3]

[# 1,3]*Department of Computer Science and Information Engineering, National Pingtung Institute of Commerce*
*No. 51, Ming-Sheng E. Road, Pingtung City, Taiwan 900*
[1]tsobing@npic.edu.tw
[3]nibi771688@hotmail.com
[*2]*Institute for Infocomm Research*
*No. 21-01 Connexis (South Tower) 1 Fusionopolis Way, Singapore 138632*
[2]pkmeher@i2r.a-star.edu.sg

***Abstract*—Efficient addition of binary numbers plays a very important role in the design of dedicated as well as general purpose processors for the implementation of arithmetic and logic units, branch decision, and floating-point operations, address generations, etc. Several methods have been reported in the literature for the fast and hardware-efficient realization of binary additions. Among these methods, parallel-prefix addition schemes have received much of attentions, since they provide many design choices for delay/area–efficient implementations and optimization of tradeoffs. In this paper, we have proposed area-efficient approach for the design of parallel-prefix Ling adders. We have achieved the area efficiency by computing the real carries, based on the Ling carries produced by the lower bit positions. Using the proposed method, the number of logic levels can be reduced by one, which leads to reduction of delay as well as significant saving of area complexity of the adder. We have implemented the proposed adders using 0.18μm CMOS technology; and from the synthesis results, we find that our proposed adders could achieve up to 35% saving of area over the previously reported parallel-prefix Ling adders under the same delay constraints.**

***Keywords*—Computer arithmetic, VLSI design, Additions, Parallel-prefix computations, Ling adders.**

## I. Introduction

Addition is one of the most frequently used operations in processors, including: arithmetic and logic units (ALU), branch decision, and floating-point operations, address generations, etc. Several methods have been reported in the literature for the fast and hardware-efficient realization of binary additions. Among these methods, parallel-prefix based adders [1-8, 10-14] have received much of attentions since they can provide many varieties of delay/area-efficient implementations and optimization of tradeoffs. The Ling adders [9] can relax the computation of carries compared to the carry lookahead adders (CLA) design. This feature of Ling adders has been used to improve the speed of conventional CLA.

In [10-11], the authors have proposed a parallel-prefix Ling adder design with some modifications of Ling adders and parallel-prefix based computation units. The work in [10-11] can achieve higher delay/area efficiency over the conventional CLA design. Afterwards, some techniques for carry select addition, by using hybrid Ling adders or mixed-radix Ling adder have been suggested in [12-14] to achieve faster addition at the cost of more area due to the multiplexors.

In this work, we have proposed area-efficient parallel-prefix adders by computing the real carries, based on the Ling carries produced by the lower bit positions. By the proposed scheme, the number of levels of logic gates can be reduced by one, which leads to significant savings of area with lower delay. We have implemented the proposed adders using 0.18μm CMOS technology; and find that our proposed adders could outperform the previously reported parallel-prefix Ling adders in terms of up to 35% area savings under the same delay constraint.

The rest of this paper is organized as follows. In Section II, we will introduce the designs of conventional parallel-prefix adders and Ling adders with/without parallel-prefix computation units. Then our proposed area-efficient parallel-prefix Ling adders are given in Section III. The synthesis results and comparisons will be presented in Section IV, and Section V draws the conclusions.

## II. Previous Ling Adders and Parallel-Prefix Ling Adders Implementations

### A. Parallel-prefix adders

Given two $n$-bit binary numbers $A$ and $B$, where $A=a_{n-1}a_{n-2}\ldots a_0$ and $B=b_{n-1}b_{n-2}\ldots b_0$, the sum $S=s_{n-1}s_{n-2}\ldots s_0$ of $A$ and $B$ can be obtained by the following three steps.

Step 1: Computing the signals propagation bits $p_i$, generation bits $g_i$, and half sum bits $d_i$, for $i$=0 to $n$-1, where

$$p_i = a_i + b_i \quad g_i = a_i \cdot b_i \quad d_i = a_i \oplus b_i \tag{1}$$

Step 2: Using any type of parallel-prefix unit to compute the carry signals $c_i$ for each bit position $i$, where $i$=0 to $n$-1.

$$c_i = g_i + p_i \cdot c_{i-1} \tag{2}$$

In order to compute the recursive carry signals efficiently, the carry node is defined as an operator o that can accept two pairs of inputs ($g$, $p$) and ($g$', $p$') and perform the logic operations [11]:

$$(g, p) \circ (g', p') = (g + p \cdot g', p \cdot p') \tag{3}$$

Step 3: Producing the output of sum $s_i$ for each bit $i$=0 to $n$-1, where

$$s_i = d_i \oplus c_{i-1} \tag{4}$$

Fig. 1 (a) is an 8-bit Kogge-Stone parallel-prefix adder. The fucntions of different node are defined in Fig. 1 (b). The black circlular node is defined as the carry node which performs the prefix computations given in Eq. (3), the white circular node is defined as the buffer.

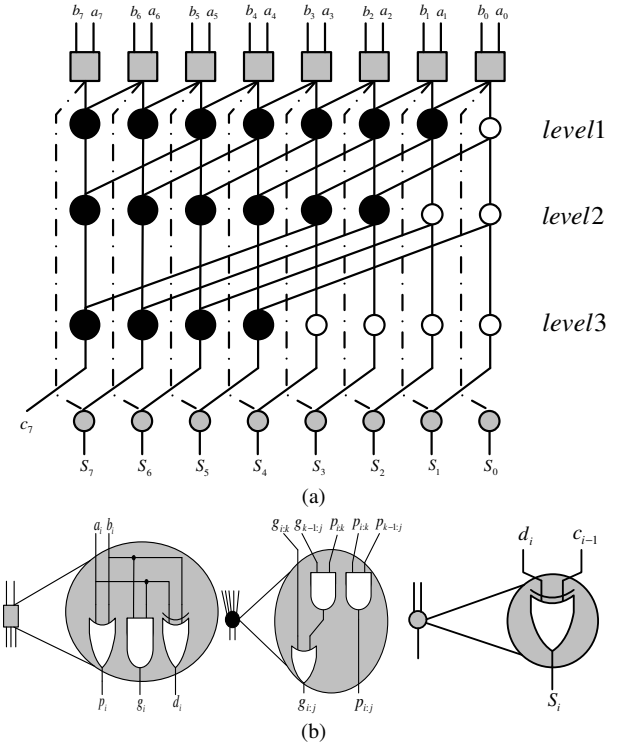


Fig. 1: (a) The architecture of an 8-bit Kogge-Sone parallel-prefix adder, and (b) the corresponding nodes defined in (a).

### *B. Ling adders with/without parallel-prefix computation units*

In [9], Ling adders were proposed to speed up the computation of conventional CLAs. This is achieved by introducing one signal $H_i$, where $H_i$ (also can be denoted as *Ling carries*) is obtained by the logical OR operations of the two consecutive carry signals, i.e.,

$$H_i = c_i + c_{i-1} \tag{5}$$

Using (2) in (5), we can have:

$$H_i = g_i + g_{i-1} + p_{i-1} \cdot g_{i-2} + .... + p_{i-1} \cdot p_{i-2} \cdot .... \cdot p_1 \cdot g_0 \tag{6}$$

As for example we can have $c_3$ as

$$c_3 = g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 \tag{7}$$

and $H_3$ can be represented as

$$H_3 = g_3 + g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 \tag{8}$$

Comparing (7) with (8), we can observe that the number of logic levels for generating $H_3$ is shorter by one for generating $c_3$, which leads to faster computation of carries for each bit position, and the sum for Ling adders can be obtained according to (9), using a 2-to-1 multiplexor as shown in Fig. 2(b).

$$s_i = \overline{H}_{i-1} \cdot d_i + H_{i-1} \cdot (d_i \oplus p_{i-1}) \tag{9}$$

In [10-11], the authors found that the operations of Ling adders can be executed in a parallel-prefix way, take $H_4$ and $H_5$ for example, the values of $H_4$ and $H_5$ can be computed by Ling adders which are given in Eq. (10).

$$\begin{aligned} H_4 &= g_4 + g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 \\ H_5 &= g_5 + g_4 + p_4 \cdot g_3 + p_4 \cdot p_3 \cdot g_2 + p_4 \cdot p_3 \cdot p_2 \cdot g_1 + p_4 \cdot p_3 \cdot p_2 \cdot p_1 \cdot g_0 \end{aligned} \tag{10}$$

By using the property of $g_i \bullet p_i = g_i$, $H_4$ and $H_5$ can be re-written as:

$$\begin{aligned} H_4 &= (g_4 + g_3) + ((p_3 \cdot p_2) \cdot (g_2 + g_1)) + (p_3 \cdot p_2 \cdot p_1 \cdot p_0) \cdot g_0 \\ H_5 &= (g_5 + g_4) + ((p_4 \cdot p_3) \cdot (g_3 + g_2)) + (p_4 \cdot p_3 \cdot p_2 \cdot p_1) \cdot (g_1 + g_0) \end{aligned} \tag{11}$$

if we define $G_{i:i-1} = g_i + g_{i-1}$ and $P_{i:i-1} = p_i \cdot p_{i-1}$,

Equation (11) can be further expressed:

$$\begin{aligned} H_4 &= G_{4:3} + P_{3:2} \cdot G_{2:1} + P_{3:2} \cdot P_{1:0} \cdot G_{0:-1} \\ H_5 &= G_{5:4} + P_{4:3} \cdot G_{3:2} + P_{4:3} \cdot P_{2:1} \cdot G_{1:0} \end{aligned} \tag{12}$$

By adopting the prefix carry nodes (explained in (3)), the values of $H_4$ and $H_5$ can be obtained in a parallel-prefix way, i.e.,

$$\begin{aligned} H_4 &= (G_{4:3}, P_{3:2}) \circ (G_{2:1}, P_{1:0}) \circ (G_{0:-1}, P_{-1:-2}) \\ H_5 &= (G_{5:4}, P_{4:3}) \circ (G_{3:2}, P_{2:1}) \circ (G_{1:0}, P_{0:-1}) \end{aligned} \tag{13}$$

Fig. 2 is an 8-bit Kogge-Stone parallel-prefix Ling adder.

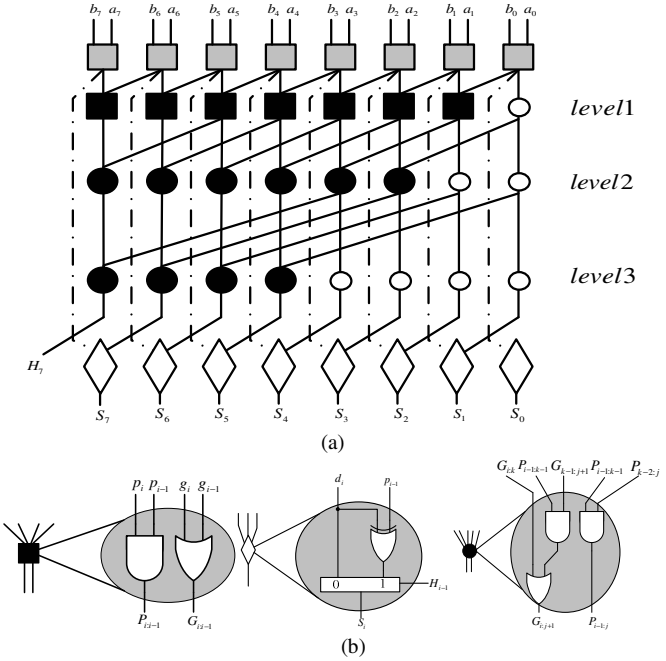


Fig. 2: (a) The architecture of an 8-bit Kogge-Stone parallel-prefix Ling adder, (b) (from left to right) the black square nodes for computing the values of $P_i$ and $G_i$, the diamonds nodes for computing the values of sum using $H_{i-1}$ as the selection signal, and the black circle nodes for computing group generate and propagation signals.

From Fig. 2(a), it can be seen that, the signals $H_i$ are computed by the parallel-prefix units, and the sums are produced by using 2-to-1 multiplexors for each bit position. If the last level in the parallel-prefix units can be further replaced to compute the carries instead of $H_i$, the values of sum can be computed by Exclusive-OR (XOR) gates direclty without multiplexors. This will lead to an area-efficient implementatoin, which we have discussed in Section III.

## III. Proposed Area Efficient Parallel-Prefix Ling Adders

As can be seen in Fig. 2(a), there are three levels for computing the Ling carries $H_0$ to $H_7$, and one level for producing the correct sums $s_0$ to $s_7$. There are two drawbacks for this kind of implementation: first, $H_7$ can not be used as the carry-in for the addition with longer bit-width of inputs; and the second, the diamond nodes for producing the sum bits in Fig. 2(b) demand more area than the conventional parallel-prefix adders (shown in Fig. 1(a)) since there is one additional multiplexor for producing each correct bit of sum using the Ling carries for the selection signal.

Our proposed architecture is shown in Fig. 3(a). Suppose the carry-in to the adder is set to zero. We expect to obtain the real carries $c_0$ to $c_7$ directly instead of using Ling carries for producing the sums, thus the diamond nodes in Fig. 2(a) can then be replaced by the grey circle nodes ( i.e., XOR gates) for producing the sums.

Now we partition the Ling carries obtained by Fig. 2(a) into two groups, namely, the lower group and the higher group. For the lower group, i.e., the bit positions 0 to 3, since they already contain the values of $H_0$ to $H_3$ after level 2, which can be expressed as:

$$\begin{aligned} &G_0 = H_0 = g_0 \\ &G_{1:0} = H_1 = g_0 + g_1 \\ &G_{2:0} = H_2 = (G_{2:1}, P_{1:0}) \circ (G_{0:-1}, P_{-1:-2}) = G_{2:1} + P_{1:0} \cdot G_{0:-1} = (g_2 + g_1) + ((p_1 \cdot p_0) \cdot (g_0 + g_{-1})) \\ &G_{3:0} = H_3 = (G_{3:2}, P_{2:1}) \circ (G_{1:0}, P_{0:-1}) = G_{3:2} + P_{2:1} \cdot G_{1:0} = (g_3 + g_2) + ((p_2 \cdot p_1) \cdot (g_1 + g_0)) \end{aligned} \quad (14)$$

If we apply $p_i$ to be logic AND-ed by $H_i$ ($i$=0 to 3), then we can have: (note: $p_i \cdot g_i = g_i$)

$$\begin{aligned} &p_0 \cdot H_0 = p_0 \cdot g_0 = c_0 \\ &p_1 \cdot (H_1) = p_1 \cdot (g_1 + g_0) = g_1 + p_1 g_0 = c_1 \\ &p_2 \cdot (H_2) = p_2 \cdot (g_2 + g_1 + p_1 \cdot p_0 \cdot g_0) = c_2 \\ &p_3 \cdot (H_3) = p_3 \cdot ((g_3 + g_2) + p_2 \cdot p_1 \cdot (g_1 + g_0)) = c_3 \end{aligned} \quad (15)$$

Thus, the real carries for $c_i$ in the lower group can be directly obtained by four logic AND gates (circular nodes) as shown in Fig. 3(b).

For the higher group, i.e., for the bit positions 4 to 7, after level 2, the outputs for producing $H_i$ ($i$ = 4 to 7) can be expressed as:

$$\begin{aligned} &G_{4:0} = H_4 = (G_{4:1}, P_{3:0}) \circ (G_0, P_{-1}) = G_{4:1} + P_{3:0} \cdot H_0 \\ &G_{5:0} = H_5 = (G_{5:2}, P_{4:1}) \circ (G_{1:0}, P_{0:-1}) = G_{5:2} + P_{4:1} \cdot H_1 \\ &G_{6:0} = H_6 = (G_{6:3}, P_{5:2}) \circ (G_{2:0}, P_{1:-2}) = G_{6:3} + P_{5:2} \cdot G_{2:0} = G_{6:3} + P_{5:2} \cdot H_2 \\ &G_{7:0} = H_7 = (G_{7:4}, P_{6:3}) \circ (G_{3:0}, P_{2:-1}) = G_{7:4} + P_{6:3} \cdot G_{3:0} = G_{7:4} + P_{6:3} \cdot H_3 \end{aligned} \quad (16)$$

In (16), $G_{i:j}$ and $P_{i:j}$ are denoted as the group generate and propagation signals from bit positions $j$ to $i$, respectively. Now we modify the black circular nodes (defined in Fig. 2(b)) to be logic wire AND-ed by $p_i$ to produce the value of $P_{i:j}$, which is defined in Fig. 3(c). While in level 3, we modify the black circular nodes to perform the following operations:

$$c_i = p_i \cdot G_{i:i-(n/2-1)} + (P_{i:i-(n/2)}) \cdot H_{i-n/2} \quad (17)$$

Taking $c_7$ for example,

$$\begin{aligned} &p_7 \cdot G_{7:4} + (P_{7:3}) \cdot H_3 = \\ &p_7 \cdot ((g_7 + g_6) + ((p_6 \cdot p_5) \cdot (g_5 + g_4))) + (p_7 \cdot (p_6 \cdot p_5 \cdot p_4 \cdot p_3)) \cdot ((g_3 + g_2) + ((p_2 \cdot p_1) \cdot (g_1 + g_0))) \\ &= c_7 \end{aligned} \quad (18)$$

Equation (18) is equivalent to the real carries produced by parallel-prefix adders.

Similarly, we can obtain the values of $c_4$ to $c_6$ by:

$$\begin{aligned} &p_4 \cdot (G_{4:1}) + (P_{4:0}) \cdot H_0 \\ &= p_4 \cdot ((g_4 + g_3) + p_3 \cdot p_2 \cdot (g_2 + g_1)) + (p_4 \cdot p_3 \cdot p_2 \cdot p_1 \cdot p_0) \cdot H_0 = c_4 \\ &p_5 \cdot (G_{5:2}) + (P_{5:1}) \cdot H_1 \\ &= p_5 \cdot ((g_5 + g_4) + p_4 \cdot p_3 \cdot (g_3 + g_2)) + (p_5 \cdot p_4 \cdot p_3 \cdot p_2 \cdot p_1) \cdot H_1 = c_5 \\ &p_6 \cdot (G_{6:3}) + (P_{6:2}) \cdot H_2 \\ &= p_6 \cdot ((g_6 + g_5) + p_5 \cdot p_4 \cdot (g_4 + g_3)) + (p_6 \cdot p_5 \cdot p_4 \cdot p_3 \cdot p_2) \cdot H_2 = c_6 \end{aligned} \quad (19)$$

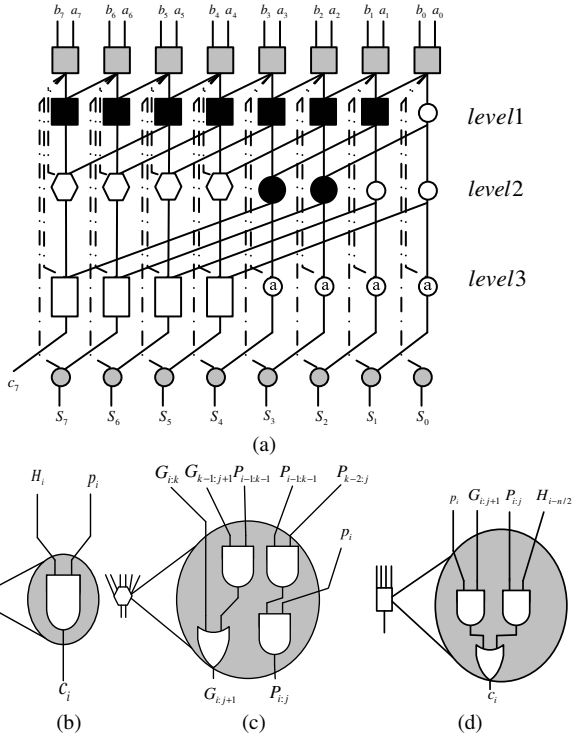


Fig. 3 (a): Our proposed 8-bit parallel-prefix Ling adder, (b) the nodes for computing the real carries for bit 0 to $n$/2-1, (c) the hexagon nodes for computing the group generation from bit $i$ to $j$+1 and group propagation from bit $i$ $j$, and (d) the white rectangle nodes for computing the real carries for bit $n$/2 to $n$-1.

In Fig. 3(c) and (d) we have depicted two of our proposed parallel-prefix units. To derive these units, we have modified the original black circular nodes of Fig. 2(b) to obtain the values of group generate and propagation signals and the real carries for bit $n$/2 to $n$-1. The level of logic gate for these two units is only two, which is the same as the black circle nodes in Fig. 2(b). The overall delay for our proposed architecture for the parallel-prefix computation is only five logic gates, which is equivalent to the original parallel-prefix Ling adder [11], but it is one fewer than that of the Kogge-Stone adder [1]. In addition, since we avoid the use of multiplexors for computing the sums, the area cost for our proposed method is fewer than that of the methods proposed in [11], and is comparable to the parallel-prefix Kogge-Stone adder. In Table I we have compared the number of logic gates for our proposed adder with the parallel-prefix Ling adder

[11]. Our proposed adders can save more than 11% logic gates over that of [11]. We have extended our methods to design the 16-bit parallel-prefix Ling adder as shown in Fig. 4, where eight hexagon nodes in level 3 and eight rectangle nodes in level 4 in the higher group (i.e., bit-positions 8 to 15), and eight AND gates in level 4 in the lower group (i.e., bit-positions 0 to 7) produce the real carries $c_0$ to $c_{15}$.

Table I: comparisons of the number of logic gates for our proposed adder with the parallel-prefix Ling adder [11].

| *n* | [11] | **Ours** | Savings |
|---|---|---|---|
| 8-bit | 112 | 92 | 17% |
| 16-bit | 268 | 228 | 14% |
| 32-bit | 628 | 548 | 12% |
| 64-bit | 1444 | 1284 | 11% |

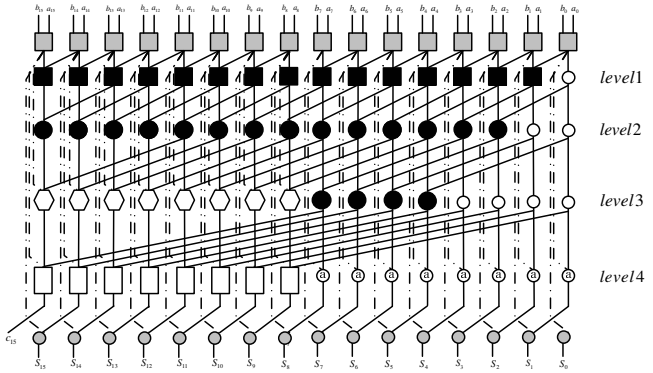

Fig. 4: Our proposed 16-bit parallel-prefix Ling adder.

## IV. Synthesis Results and Comparisons

We have coded our proposed adders using Verilog structuring HDL and synthesized using TSMC 0.18 µm CMOS technology. Table II shows the synthesis result for our adders and the previously reported parallel-prefix adders in [1] and parallel-prefix Ling adders in [11] based on Kogge-Stone style implementations.

Table II: area comparisons for our proposed adder with the parallel-prefix Ling adder [11] and parallel-prefix adder [1] based on Kogge-Stone style implementations. (Unit: $\mu m^2$)

| *n* | [11] | [1] | Ours | Savings | |
|---|---|---|---|---|---|
| | Area | Area | Area | vs.[11] | vs. [1] |
| 8 (delay=1.03ns) | 2341 | 1872 | 1922 | 17% | -2% |
| 16 (delay=1.42ns) | 5651 | 5069 | 4151 | 26% | 18% |
| 32 (delay=1.87ns) | 12357 | 9427 | 8262 | 33% | 22% |
| 64 (delay=2.11ns) | 29864 | 27133 | 19349 | 35% | 28% |

In Table II, we can find that our proposed adder can save up to 28% and 35% of area over that of [1] and [11], respectively, for higher bit-widths. The delay for our proposed adder is shorter than [1]. Therefore, under the same delay constraint, the area required by [1] becomes more than ours. In Table III we show the area×delay savings for our proposed method over those of [1] and [11] with minimum delay. Our method could achieve up to 28.35% and 35.21% of saving in area×delay over those in [1] and [11], respectively.

Table III: area×delay savings for our proposed adders. (Delay: ns, Area: $\mu m^2$)

| *n* | [1] | | [11] | | Ours | | **Area×delay Savings** | |
|---|---|---|---|---|---|---|---|---|
| | Area | Delay | Area | Delay | Area | Delay | **vs. [1]** | **vs. [11]** |
| 8-bit | 1872 | 1.03 | 2341 | 1.03 | 1962 | 0.98 | **0.28%** | **20.26%** |
| 16-bit | 5069 | 1.42 | 5728 | 1.4 | 4826 | 1.27 | **14.85%** | **23.57%** |
| 32-bit | 10395 | 1.69 | 12357 | 1.87 | 9826 | 1.67 | **6.59%** | **28.89%** |
| 64-bit | 27133 | 2.1 | 29864 | 2.11 | 19349 | 2.11 | **28.35%** | **35.21%** |

## V. Conclusions

In this paper, we have proposed a technique for area-efficient realization of parallel-prefix Ling adders using efficient parallel-prefix units. Synthesis results show significant saving of area by our method over the previous methods. The proposed method not only could be applied to the Kogge-Stone-style Ling adders, but also other parallel-prefix style Ling adders.


### Acknowledgment

This work was supported partly by NSC-99-2221-E-251-008, Taiwan.


## References


[1] 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.

[2] R.E. Ladner and M.J. Fisher, "Parallel Prefix Computation," *J. ACM,* vol. 27, no. 4, pp. 831-838, Oct. 1980.

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

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

[5] S. Knowles, "A Family of Adders," *Proc. 14th Symp. Computer Arithmetic,* pp. 30-34, Apr. 1999. Reprinted in ARITH-15, pp. 277-281.

[6] A. Beaumont-Smith and C.C. Lim, "Parallel-Prefix Adder Design," *Proc. 15th Symp. Computer Arithmetic,* pp. 218-225, June 2001.

[7] V.G. Oklobdzija et al., "Energy-Delay Estimation Technique for High-Performance Microprocessor VLSI Adders," *Proc. 16th Symp. Computer Arithmetic,* pp. 15-22, June 2003.

[8] S.Das, and S.P.Khatri,, "A Novel Hybrid Parallel-Prefix Adder Architecture With Efficient Timing-Area Characteristic ," *IEEE Trans. VLSI Systems*, Vol. 16, no. 3, pp. 326-331, March 2008.

[9] H. Ling, "High-Speed Binary Adder," *IBM J. R&D,* vol. 25, pp. 156-166, May 1981.

[10] C. Efstathiou, H.T. Vergos, and D. Nikolos, "Ling Adders in Standard CMOS Technologies," *Proc. IEEE Int'l Conf. Electronics, Circuits, and Systems* (*ICECS*), vol. 2, pp. 485-488, Sept. 2002.

[11] D. Nikolos and C. Efstathiou, "High-Speed Parallel-Prefix Ling Adders," *IEEE Trans. Computers*, Vol. 54, No.2, pp. 225-231, Feb. 2005.

[12] Lakshmanan, Ali Meaamar and Masuri Othman, "High-Speed Hybrid Parallel-Prefix Carry-Select Adder Using Ling's Algorithm," *Proc. ISCE*, pp. 598-603, 2006.

[13] Alvaro V´azquez and Elisardo Antelo, "New Insights on Ling Adders," *Proc. 18th Symp. Computer Arithmetic*, pp. 227-232, June 2007.

[14] Youngmoon Choi and Earl E. Swartzlander, Jr., "Speculative Carry Generation with Prefix Adder," *IEEE Trans. VLSI Systems,* vol. 16, no. 3, pp. 321-326, Mar. 2008.