Constructing Zero-deficiency Parallel Prefix Adder of Minimum Depth

Haikun Zhu, Chung-Kuan Cheng, Ronald Graham
Department of Computer Science and Engineering
La Jolla, California 92093
{hazhu,kuan,rgraham}@cs.ucsd.edu

***Abstract*— Parallel prefix adder is a general technique for speeding up binary addition. In unit delay model, we denote the size and depth of an n-bit prefix adder $C(n)$ as $s_{C(n)}$ and $d_{C(n)}$ respectively. Snir proved that $s_{C(n)}+d_{C(n)} \geq 2n-2$ holds for arbitrary prefix adders. Hence, a prefix adder is said to be of zero-deficiency if $s_{C(n)}+d_{C(n)} = 2n-2$. In this paper, we first propose a new architecture of zero-deficiency prefix adder dubbed $Z(d)$, which provably has the minimal depth among all kinds of zero-deficiency prefix adders. We then design a 64-bit prefix adder Z64, which is derived from $Z(d)|_{d=8}$, and compare it against several classical prefix adders of the same bit width in terms of area and delay using logical effort method. The result shows that the proposed $Z(d)$ adder is also promising in practical VLSI design.**

I. Introduction

Binary adders are the most fundamental modules in computer arithmetic design, and consequently have been investigated extensively for decades. Quite a few classic fast adders, such as the carry-skip adder, the carry-select adder, and the carry-lookahead adder, were proposed in the past [1]. However, these fast adders are ad-hoc in structure, and each of them represents a unique area-time tradeoff in the design space. Parallel prefix adder, on the other hand, represents a class of general adder structure that exhibits flexible area-time tradeoffs for adder design. Therefore, identifying the exact area-delay tradeoff curve of the parallel prefix adder is an interesting problem that has received much research attention.

In designing parallel prefix adders, it has been popular to assume the unit delay timing model, in which the computation nodes are arranged in levels that represent the signal timing [1], [2], [4]∼ [8]. Denoting the size (i.e., the number of computation nodes) and depth of an n-bit prefix adder $C(n)$ as $s_{C(n)}$ and $d_{C(n)}$ respectively, Snir proved that $s_{C(n)}+d_{C(n)} \geq 2n-2$ holds for arbitrary prefix circuits [2]. He defined the deficiency of a prefix circuit as

$$def(C(n)) = s_{C(n)} + d_{C(n)} - 2n + 2 \tag{1}$$

Therefore, a prefix adder is said to be of **zero-deficiency** if $s_{C(n)} + d_{C(n)} = 2n-2$.

Snir's theorem indicates that the solution space for parallel prefix adders should look like Fig. 1. For loose depth constraints we can observe a linear tradeoff between the depth and size which is exhibited by zero-deficiency prefix adders. However, if the depth constraint is too tight, the size of the prefix adders will grow dramatically and zero-deficiency prefix adders no longer exist. It remains an open question to find the zero-deficiency prefix adders of minimum depth, i.e., to identify the curve $d = f(n)$ shown in Fig. 1.

Various zero-deficiency prefix circuits were proposed in the past. Among them the most notable ones are Snir's design [2], $LYD(n)$ circuit [5] and $LS(n)$ circuit [4]. The main purpose of these work has been tightening the depth of zero-deficiency prefix circuits as small as possible for a given width. In [6], Zimmermann proposed an heuristic for prefix adder optimization using depth-controlled compression and expansion. His approach in many cases produces depth-size optimal or near optimal prefix adders. However, the optimality of his results is not guaranteed.

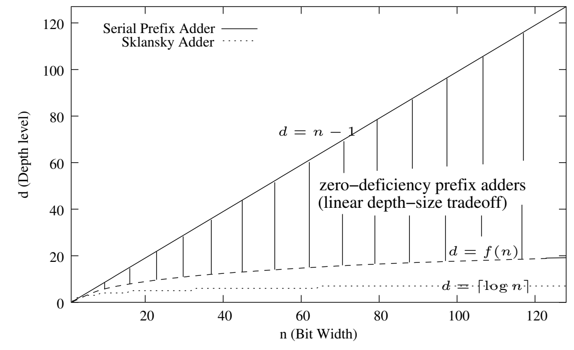


Fig. 1. Optimal Depth-Size tradeoffs of Parallel Prefix Adders. For a given width n, the maximal depth of the prefix adders is $n-1$ (serial prefix adder) while the minimal depth is $\lceil \log n \rceil$ (Sklansky adder [3]).

In this paper, we propose a new kind of zero-deficiency prefix adder called $Z(d)$ which provably has the minimal depth for a given width. As in previous work, we adopt the unit delay timing model since it is simple enough but also easy for extension. We design our structure from an alternative point of view, that is, by constructing a zero-deficiency prefix adder of maximal width for a given depth. We then design a 64-bit prefix adder derived from $Z(d)|_{d=8}$, and compare it against several classical prefix adders in terms of area and delay using logical effort method [16], [17]. The result shows that $Z(d)$ adder is promising in practical VLSI design.

The remainder of the paper is organized as follows. Section II explains how binary addition is formulated as a parallel prefix problem. In Section III, we first give a revised proof for Snir's lower bound theorem $s_{C(n)} + d_{C(n)} \geq 2n-2$, and then discuss the properties of zero-deficiency adders. Our main contribution lies in Section IV, where a new class of zero-deficiency prefix circuits $Z(d)$ is proposed. Section V focuses on area and delay analysis using logical effort, as well as comparisons. Section VI concludes the paper.

II. Background

The generalized prefix problem is to compute $y_i = x_i \bullet x_{i-1} \bullet \cdots x_1$ for $1 \leq i \leq n$, given n inputs $x_1, x_2, \ldots, x_n$ and an arbitrary associative operator $\bullet$.

Binary addition is usually expressed in terms of carry generation signal g_i, carry propagation signal p_i, carry signal

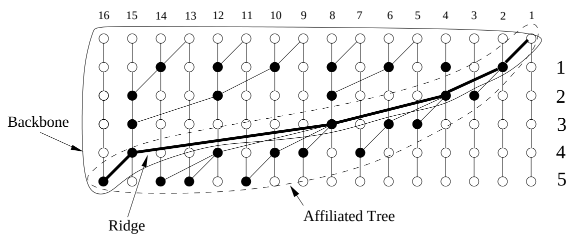


Fig. 2. An examples of parallel prefix adder: $n = 16$, $d = 4$, $s = 32$

c_i and sum signal s_i at each bit position ($1 \le i \le n$):

$$g_i = a_i b_i \tag{2}$$

$$p_i = a_i \oplus b_i \tag{3}$$

$$c_i = \begin{cases} g_i & \text{if } i = 1 \\ g_i + p_i c_{i-1} & \text{if } 2 \le i \le n \end{cases} \tag{4}$$

$$s_i = p_i \oplus c_{i-1} \tag{5}$$

where $A = a_n a_{n-1} \cdots a_1$ and $B = b_n b_{n-1} \cdots b_1$ are the two binary numbers. The concepts of carry generation and carry propagation can be easily extended to a block of consecutive bits:

$$G_{[i:j]} = \begin{cases} g_i & \text{if } i = j; \\ G_{[i:k]} + P_{[i:k]} G_{[k-1:j]} & \text{if } n \ge i > j \ge 1 \end{cases} \tag{6}$$

$$P_{[i:j]} = \begin{cases} p_i & \text{if } i = j; \\ P_{[i:k]} P_{[k-1:j]} & \text{if } n \ge i > j \ge 1 \end{cases} \tag{7}$$

By introducing an associative operator $\bullet$, the computation of a pair of (G, P) signals and carry signals can be rewritten as:

$$(G, P)_{[i:i]} = (g, p)_i \tag{8}$$

$$\begin{aligned}(G, P)_{[i:j]} &= (G, P)_{[i:k]} \bullet (G, P)_{[k-1:j]} \\ &= (G_{[i:k]} + P_{[i:k]} G_{[k-1:j]}, P_{[i:k]} P_{[k-1:j]})\end{aligned} \tag{9}$$

$$c_i = G_{[i:1]} \tag{10}$$

Therefore, the carry signal generation, namely, the $(G, P)_{[i:1]}$ signal generation in terms of equation (9) is exactly a prefix computation problem. Since the generation of g_i, p_i signals and sum bits s_i are just local operations, the performance of the adder is determined by the prefix circuit of generating (G, P) signals. In the sequel, we will only show the intermediate prefix circuit of the prefix adders. As an example, Fig. 2 shows a 16-bit prefix circuit of depth 5. Each computation node (i.e., black node) in the figure is a (G, P) generator. Generally, for an n-bit prefix circuit $C(n)$, its size $s_{C(n)}$ is defined as the number of computation nodes while its depth $d_{C(n)}$ is defined as the level of the latest prefix output. The white nodes in the figure are duplication nodes since they do nothing but pass the signals. In the rest of the paper when we say “a node”, we always refer to a computation (or black) node.

III. Properties of the zero-deficiency prefix adder

Snir’s original proof for $s_{C(n)} + d_{C(n)} \ge 2n - 2$ is by mathematical induction and does not reveal the structural information of zero-deficiency circuits. We notice that there are some nice ideas in [7] and [8] which can be used to devise an elegant proof for Snir’s theorem. In this section, we polish these ideas and prove an enhanced version of Snir’s lower bound theorem.

Theorem 1: *Let $C(n)$ be an n-bit prefix circuit, with its size and depth being denoted by $s_{C(n)}$ and $d_{C(n)}$, respectively. Denote the depth of its most significant bit (MSB) output by $d^M_{C(n)}$. Then*

$$s_{C(n)} + d_{C(n)} \ge s_{C(n)} + d^M_{C(n)} \ge 2n - 2 \tag{11}$$

Proof: Consider the MSB output node in $C(n)$. This output node is actually the root of an alphabetical tree which is upside-down with all the input nodes being its leaves. The size of this tree is exactly $n - 1$, and its depth is $d^M_{C(n)}$. Including the LSB bit, at most $d^M_{C(n)} + 1$ prefix outputs can be obtained from this tree. For each of the columns where the prefix results are not ready, at least one extra node is needed to generate the output. Thus, besides the tree, we need at least $n - (d^M_{C(n)} + 1)$ nodes for the outputs. Consequently, we have the inequality

$$s_{C(n)} \ge (n - 1) + (n - (d^M_{C(n)} + 1)) = 2n - 2 - d^M_{C(n)}$$

which leads to

$$s_{C(n)} + d^M_{C(n)} \ge 2n - 2$$

■

We now define a few new concepts about the prefix circuits as follows.

Definition 1: *For a prefix circuit $C(n)$, the binary alphabetical tree generating the MSB prefix output is called the* ***backbone*** *of $C(n)$. In addition, there is another tree whose nodes are exactly all the prefix output nodes, with the first input node being its root. This tree is called the the* ***affiliated tree*** *of $C(n)$. The common part of the backbone and the affiliated tree, that is, the path from the first input to the MSB output, is called the* ***ridge*** *of $C(n)$.*

For illustration, the backbone of the prefix circuit in Fig. 2 is enclosed by a solid line loop, while the affiliated tree is enclosed by a dashed line loop. Their common part, i.e., the ridge, is highlighted using heavy line.

According to the proof of Theorem 1, it is straightforward to derive the following corollary:

Corollary 1: *A prefix circuit $C(n)$ of depth d is of zero-deficiency if and only if*

1) *The backbone of $C(n)$ has depth d, and its size is $n-1$.*
2) *The affiliated tree of $C(n)$ has size $n - 1$, and it has exactly one node per column (excluding the LSB column). Each node of the affiliated tree is a prefix output.*
3) *The ridge has d nodes, one node per level.*

IV. Proposed $Z(d)$ circuit

In this section, we propose a new class of zero-deficiency prefix circuits, called $Z(d)$, which have the minimum depth among all zero-deficiency prefix circuits.

We will first construct a class of parameterized trees called $T^k(t)$ trees which will be used to form the backbone of the $Z(d)$ circuit. We then define the $A^k(t)$ trees which will be used to form the affiliated tree of $Z(d)$. $Z(d)$ circuit is constructed by assembling $T^k(t)$ trees and $A^k(t)$ trees together.

The $T^k(t)$ trees are defined by a recursive way as shown in Fig. 3. The parameter $t (\ge 1)$ is the depth of $T^k(t)$ tree while k represents the maximum number of black nodes in a single column. As an example, Fig. 5(a) shows the $T^3(5)$ tree whose

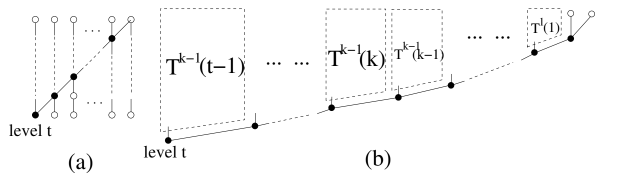


Fig. 3. The recursive definition of $T^k(t)$ tree: (a) $T^1(t)$; (b) $T^k(t)$, $1 \leq k \leq t$.

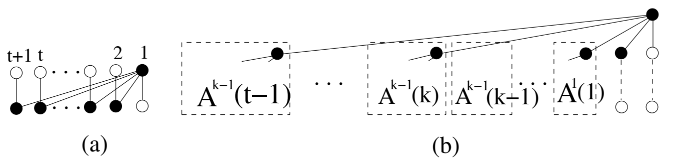


Fig. 4. The recursive definition of $A^k(t)$ tree: (a) $A^1(t)$; (b) $A^k(t)$, $1 \leq k \leq t$.

Algorithm 1 Generation of the $T^k(t)$ tree

```
T_tree_generation(int $t$, int $k$)
1: for $i = 1$ to $k - 1$ do
2:     for $j = i$ to $t - k + i$ do
3:         Construct $T^i(j)$ according to Fig. 3
4:     end for
5: end for
6: Construct $T^k(t)$ according to Fig. 3
```

Algorithm 2 Generation of the $A^k(t)$ tree

```
A_tree_generation(int $t$)
1: for $i = 1$ to $k - 1$ do
2:     for $j = i$ to $t - k + i$ do
3:         Construct $A^i(j)$ according to Fig. 4
4:     end for
5: end for
6: Construct $A^k(t)$ according to Fig. 4
```

depth is 5 and maximum number of nodes per column is 3, and also shows how it is composed of $T^2(t)$, $T^2(3)$, $T^2(2)$ and $T^1(1)$. Algorithm 1 formally presents how we generate $T^k(t)$ trees.

Following nearly the same recursive way of construction as $T^k(t)$ trees, we define the $A^k(t)$ trees as shown in Fig. 4. For $A^k(t)$ tree, the parameter t is the lateral fan-out of the root, while $k+1$ is its depth. We also give an example of $A^3(5)$ tree shown in Fig. 5(b). Similar to the structure of $T^3(5)$, $A^3(5)$ comprises $A^2(4)$, $A^2(3)$, $A^2(2)$ and $A^1(1)$. The algorithmic description of $A^k(t)$ trees is presented in Algorithm 2.

It is interesting to note that, $T^3(5)$ and $A^3(5)$ can be assembled together to form a partial prefix adder, as shown in Fig. 5(c). If the root of $A^3(5)$ is the prefix output of bit i, then essentially we have the prefix outputs from bit $i+1$ to $i+26$. Generally, we can always combine a pair of $T^k(t)$ tree and $A^k(t)$ tree to form a partial prefix adder of depth $k+t$, as shown in Fig. 6. This is feasible because $T^k(t)$ tree and $A^k(t)$ tree have the same width, which is due to the fact that they follow the same recursive way of definition.

Theorem 2 gives the width of $T^k(t)$ tree and $A^k(t)$ tree. Actually since T-tree and A-tree are defined recursively, their width is a two dimensional integer recurrence. Equation (12) is obtained by deriving a closed form formula of that recurrence. The detailed mathematical derivation can be found in [10], and is omitted here due to lack of space.

Theorem 2: *$T^k(t)$ tree and $A^k(t)$ tree has the same width, which is*

$$N(k,t) = \sum_{i=0}^{k} \binom{t}{i} \text{ for } 1 \leq k \leq t \tag{12}$$

Furthermore, the size of $T^k(t)$ tree is $N(k,t)-1$, while the size of $A^k(t)$ tree is $N(k,t)$ with one node per column.

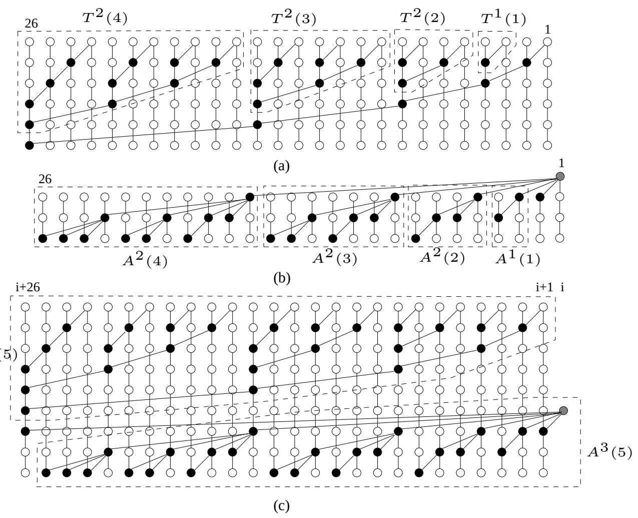


Fig. 5. Examples of $T^k(t)$ and $A^k(t)$ trees: (a) $T^3(5)$; (b) $A^3(5)$ (c) Assembling of $T^3(5)$ and $A^3(5)$.

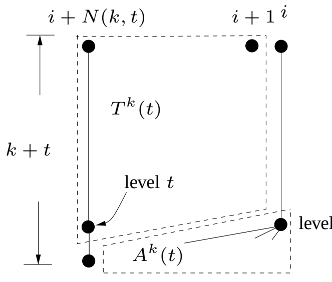


Fig. 6. Assembling of $T^k(t)$ tree and $A^k(t)$ tree

Now we are ready to introduce our new zero-deficiency circuit $Z(d)$. Algorithm 3 along with Fig. 7 defines the $Z(d)$ circuit. It can be seen that essentially $Z(d)$ is defined over its depth d. The width of $Z(d)$ is given in Theorem 3. Again, the derivation is omitted and can be found in [10].

Theorem 3: *The width of the $Z(d)$ circuit, which we denote by $N_Z(d)$, is*

$$N_Z(d) = F(d+3) - 1 \text{ for } d \geq 1 \tag{13}$$

where $F(k)$ are the natural Fibonacci numbers.

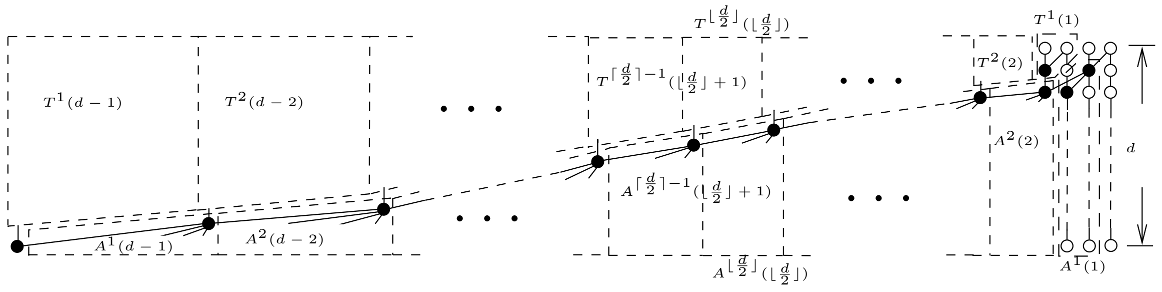
In order to prove the optimality of $Z(d)$ circuit, we shall first show that the $Z(d)$ circuit is indeed of zero-deficiency, and then prove that it does have the minimal depth among all possible zero-deficiency circuits. These two facts are presented in Theorem 4 and Theorem 5 respectively. For Theorem 4, the proof is relatively simple. We just count the number of computation nodes in $Z(d)$ and verify it satisfies the definition of zero-deficiency. For Theorem 5, the proof is by contradiction. The basic idea is to show that, given a prefix circuit of depth d, if its ridge spans wider than that of $Z(d)$ does, it can not be of zero-deficiency indeed. The detailed proofs are presented in [10].

Theorem 4: *The parallel prefix circuit $Z(d)$ shown in Fig. 7 is of zero-deficiency.*

Theorem 5: *$Z(d)$ has the maximum width for a given depth d among all zero-deficiency prefix circuits.*

As an example, Fig. 8 shows the $Z(d)$ circuit for $d = 8$.

It is now clear that the curve of function $d = f(n)$ is just the reverse function of $N_Z(d) = F(d+3) - 1$ given in Theorem 3. Table I shows the widths of $Z(d)$ circuits for



$T^1(d-1)$
$T^2(d-2)$
$T^{\lceil \frac{d}{2} \rceil -1}(\lfloor \frac{d}{2} \rfloor + 1)$
$T^{\lfloor \frac{d}{2} \rfloor}(\lfloor \frac{d}{2} \rfloor)$
$T^2(2)$
$T^1(1)$
$A^1(d-1)$
$A^2(d-2)$
$A^{\lceil \frac{d}{2} \rceil -1}(\lfloor \frac{d}{2} \rfloor + 1)$
$A^{\lfloor \frac{d}{2} \rfloor}(\lfloor \frac{d}{2} \rfloor)$
$A^2(2)$
$A^1(1)$
d

Fig. 7. A new class of zero-deficiency prefix circuits $Z(d)$.

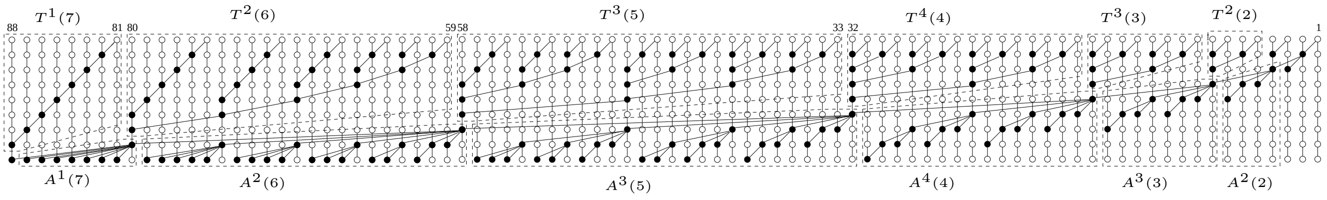


Fig. 8. Example of $Z(d)$ circuits: $Z(d)|_{d=8}$.

Algorithm 3 Generation of the $Z(d)$ circuit

```
Z_circuit_generation(int $d$) // $d$ is the depth
1: for $i = 1$ to $\lceil \frac{d}{2} \rceil - 1$ do
2:     T_tree_generation($d - i$,$i$); // call algorithm 1
3:     A_tree_generation($d - i$,$i$); // call algorithm 2
4: end for
5: for $i = \lfloor \frac{d}{2} \rfloor$ to 1 do
6:     T_tree_generation($i$,$i$);
7:     A_tree_generation($i$,$i$);
8: end for
9: Stitch all the T, A trees together to form $Z(d)$ as shown in Fig. 7.
```

$3 \leq d \leq 18$, with the results of Lin's design [4] and the $LYD(n)$ circuit [5] listed for comparison. The numbers are read as the maximal widths up to which zero-deficiency prefix circuits of the specified type and depth can be constructed. Clearly, our design dominates the other two, especially when the width is large.

Since $Z(d)$ adder has provable optimality, it is also truthfully better than the result produced by Zimmermann's algorithm [6]. For example, given width 54 and depth 7, our design has 99 nodes, while Zimmermann's algorithm gives a design of 104 nodes [9].

V. Analysis and Comparison

In this section, we compare a $Z(d)$-derived zero-deficiency adder with several classical prefix adders in terms of both delay and area, for 64-bit width. The selected prefix adders include Sklansky adder [3], Brent-Kung (BK) adder [13], Kogge-Stone (KS) adder [14] and Han-Carlson (HC) adder [15]. These adders are well known to be fast because of small logic depth or regular layout.

We use logical effort method for fast estimation of the adder delay [16]. Logical effort method is a shorthand for RC delay model yet provides reasonable accuracy. For a single stage gate, logical effort method measures its delay in units τ, which is the intrinsic delay of an ideal inverter:

$$D = D_{abs}/\tau = gh + p \tag{14}$$

where g is the logical effort of the gate, which is the ratio of the input capacitance of the gate to the input capacitance of an inverter with the same unit effective resistance; h is the electrical effort of the gate, which is the ratio of load capacitance to input capacitance. p characterizes the parasitic delay of the gate. Note that by incorporating wire capacitance into h the interconnect delay, as well as fan-out effect, can be easily considered. The overall path delay, is simply the sum of the gate delay

$$D_{path} = \sum_i D_i = \sum_i f_i g_i + \sum_i p_i \tag{15}$$

The first item is called path effort delay while the second item is called path parasitic delay.

In this study, we exactly follow the experimental settings in [17]. We assume that the (P, G) generators are implemented using inverting static CMOS, as shown in Fig. 9. The transistors are sized such that each pull down stack has unit effective resistance. Note that there are two kinds of (P, G) generators. The black cells in Fig. 9(a) calculate both P and G signals while the gray cells in Fig. 9(b) only calculate P signals. Both black cells and gray cells have two versions of opposite polarities.

Table II lists the logical efforts, parasitic delays and circuit area of black and gray cells. The parasitic delay is estimated by counting the total transistor width on the output node. The cell area is calculated by summing up all the transistor area in the cell in unit squares. Since in the prefix network alternating stages uses alternating polarities of inputs and outputs, all the values in Table II are the average of the two polarities.

In analyzing the delay, two basic assumptions are made as they were in [17].

- The wires are short enough so that distributive RC delay can be neglected. Thus the wires are only considered as capacitive load. This assumption is supported by [18].

TABLE I
Widths of $LS(n)$, $LYD(n)$ and $Z(d)$ circuits.

d	LS	LYD	Z	d	LS	LYD	Z
3	7	7	7	11	131	169	376
4	11	12	12	12	191	242	609
5	16	20	20	13	260	308	986
6	23	33	33	14	383	446	1596
7	33	54	54	15	517	576	2583
8	47	77	88	16	575	843	4180
9	66	95	143	17	1030	1101	6764
10	95	135	232	18	1535	1625	10945

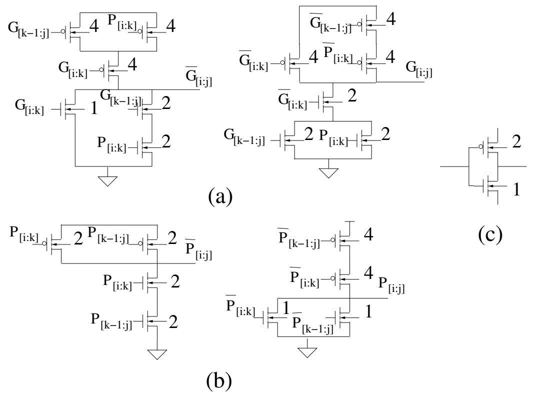


Fig. 9. Inverting CMOS Logic: (a) black cells; (b) gray cell; (c) inverter

TABLE II

LOGICAL EFFORT, PARASITIC DELAY AND AREA OF ADDER CIRCUIT BLOCKS

Cell	Term	Value	Cell	Term	Value
Black Cell	$LE_{blackgu}$	4.5/3	Gray Cell	LE_{graygu}	4.5/3
	$LE_{blackgl}$	6/3		LE_{graygl}	6/3
	$LE_{blackpu}$	10.5/3		LE_{graypu}	6/3
	$LE_{blackpl}$	4.5/3		PD_{gray}	7.5/3
	PD_{blackg}	7.5/3		A_{gray}	9
	PD_{blackp}	6/3	buffer	LE_{buf}	1*1/2
	A_{black}	16.5		A_{buf}	1/2

LE=Logical Effort, PD=Parasitic Delay, A=Area

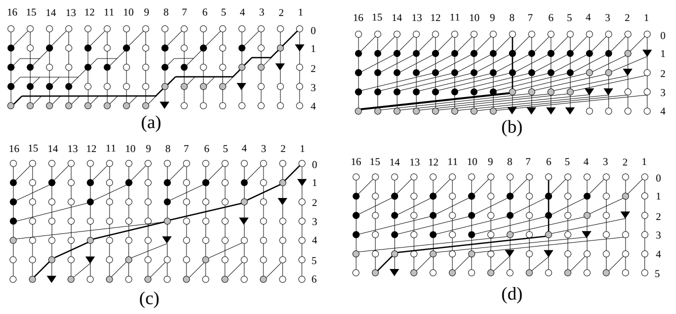


Fig. 10. Parallel prefix adders: (a) Sklansky; (b) Kogge-Stone; (c) Brent-Kung; (d) Han-Carlson.

TABLE III

CRITICAL PATHS OF VARIOUS 64-BIT ADDERS

Sklansky	(0,1)input→(1,2)g→(2,4)g→(3,8)g→(4,16)g→(5,32)g→(6,64)g
KS	(0,32)input→(1,32)b→(2,32)b→(3,32)b→(4,32)b→(5,32)g→(6,64)g
HC	(0,30)input→(1,30)b→(2,30)b→(3,30)b→(4,30)b→(5,30)g→(6,62)g →(7,63)g
BK	(0,1)input→(2,4)g→(3,8)g→(4,16)g→(5,32)g→(6,48)g→(7,56)g →(8,60)g→(9,62)g→(10,63)g
Z64	(0,1)input→(1,2)g→(2,4)→(3,8)g→(4,16)g→(5,32)g→(6,57)g →(7,61)g→(8,64)g

$(i, j)b$ denotes a black cell at level i, column j, similar for $(i, j)g$.

TABLE IV

DELAY AND AREA OF VARIOUS ADDERS

Adder	Delay	Area	Logic Depth
Z64	81.5(w=0.5)	1474	8
BK	81(w=0.5)	1507.05	10
Sklansky	175.5(w=0.5)	2695.5	6
KS	100(w=1)	4824	6
HC	105.5(w=1)	2695.9	7

- Vertical wires are short enough to be neglected. The horizontal wire capacitance is measured as w units per column spanned. It is estimated that $w \approx 0.5$ for a 0.18um technology. For KS and HC adders, there are a lot of parallel wires which have significant coupling capacitance [18]. Therefore, for these two adders $w = 0.1$ is used.

For illustration purpose, Fig. 10 shows 16-bit Sklansky, BK, KS and HC adders with critical paths identified. A few buffers are inserted to decouple the capacitance load from the critical path. These buffers have half the drive of an ordinary gate and hence half the input capacitance. Note that in [17] a fixed critical path is specified for a given adder structure of various bit widths. This is barely true because when bit width increases, the critical paths vary due to increased wire capacitance. Instead, we analyze the critical paths of 64-bit adders by hand, and list them in Table III.

The 64-bit $Z(d)$-derived prefix adder, denoted as $Z64$ for short, is generated as follows:

1) Generate $Z(d)|_{d=8}$, whose width is 88;
2) Scan the nodes of $Z(d)|_{d=8}$ one by one from level 0 to level 8, and column 1 to column 88. Do the following recursively for a selected node:
   (a) If the fan-out of the node is larger than 4, slide down it's branch of least lateral connection by one level; Otherwise skip to the next node;
   (b) If some nodes on the sliding down branch exceeds level 8, the entire columns where the exceeding nodes reside are deleted. Do local connection adjustment if needed. After this step, a 72-bit prefix network whose largest fan-out is limited by 4 is generated;
3) Discard the eight MSB columns, yielding a 64-bit prefix network;
4) Further optimize the prefix adder by inserting buffers to decouple load capacitance from the critical path.

Fig. 11 shows Z64 with critical path highlighted in heavy line. As an example, we calculate its delay and area as follows:

$$\begin{aligned}
D_F &= 4*(LE_{graygl}+LE_{buf}) + (2*LE_{graygl}+LE_{buf}) \\
&\quad +(3*LE_{graygl}+LE_{buf}) + (LE_{graygl}+LE_{buf}) \\
&\quad +3*LE_{graygl}+LE_{buf}) \\
&= 13*LE_{graygl}+8*LE_buf = 30 \\
D_P &= 8*PD_gray = 20 \\
D_{wire} &= 63*w = 63*0.5 = 31.5 \\
D_{total} &= D_F + D_P + D_{wire} = 81.5 \\
Area &= (\#black\ cells)*A_{black} + (\#gray\ cells)*A_{gray} \\
&= 1474
\end{aligned}$$

The other four adder structures are evaluated in the same way, and the results are listed in Table IV.

Clearly Z64 and BK adders are the best two of the five in terms of both delay and area. Sklansky's problem is that the fan-out grows exponentially as logic depth increases, resulting in huge delay. KS and HC adders, on the other hand, suffer from high coupling capacitance (w=1). The area of Sklansky, KS and HC is way larger than that of Z64 and BK.

Compared to BK adder, Z64 has smaller logic depth but larger fan-out. These two factors effectively cancel out so that BK and Z64 adders have nearly the same delay. However, we conjecture that Z64 adder be more power efficient than BK adder. The reason is that circuit cells of deep logic depth tend to have high activity rate, hence consume more power. A detailed power analysis of the proposed $Z(d)$ circuit is projected as a future work.

VI. CONCLUSIONS

In this paper, we have proposed a new class of zero-deficiency prefix adder $Z(d)$ which has the minimal depth

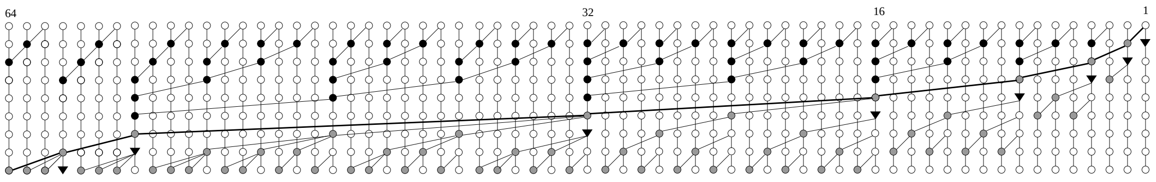


Fig. 11. A 64-bit $Z(d)$ derived prefix adder: Z64.

for a given width. The $Z(d)$ adder helps us understand the extreme of linear depth-size tradeoff of prefix adders. To appreciate the effectiveness of $Z(d)$ adder, we designed a 64-bit prefix adder Z64 based on $Z(d)|_{d=8}$. We analyzed the delay and area of Z64 using logical effort method, and compared it with several classical prefix adders. The result shows that Z64 is fast because it achieves a reasonable tradeoff between logic depth, fan-out and lateral wire connection. It also has the smallest area because of zero-deficiency.

For future work we plan to analyze the power efficiency of $Z(d)$ adder. Another interesting direction is to consider best depth-size tradeoff under non-uniform arrival times.

REFERENCES

[1] B. Parhami, *Computer Arithmetic–Algorithm and Hardware Designs*, Oxford University Press, 2000.

[2] M. Snir, "Depth-size trade-offs for parallel prefix computation," in *Journal of Algorithms* 7, pp.185–201, 1986.

[3] J. Sklansky, "Conditional sum addition logic", in *IRE Tran. Electron. Comput.*, vol. EC-9, no. 6, pp.226-231, June 1960

[4] Y-C Lin and C-C Shih, "A new class of depth-size optimal parallel prefix circuits," in *The Journal of Supercomputing*, 14, pp.39–52, 1999.

[5] S. Lakshmivarahan, C. M. Yang, and S. K. Dhall, "On a new class of optimal parallel prefix circuits with (size+depth)=$2n-2$ and $\lceil \log n \rceil \leq depth \leq (2\lceil \log n \rceil - 3)$," in *Proc. of the 1987 International Conference on Parallel Processing"*, pp.58–65, 1987.

[6] Reto Zimmermann, "Non-heuristic optimization and synthesis of parallel-prefix adders," in *International Workshop on Logic and Architecture Synthesis (IWLAS'96)*, Grenoble, December 1996.

[7] F. E. Fich, "New bounds for parallel prefix circuits," in *Proc. of the 15th Annu. ACM Sympos. Theory of Comput.*, 1983, pp.100–109.

[8] S. Lakshmivarahan, S. K. Dhall, *Parallel Computing: Using the Prefix Problem*, Oxford University Press, 1994.

[9] Reto Zimmermann, Java applet for adder synthesis, http://www.iis.ee.ethz.ch/ zimmi/applets/prefix.html

[10] Haikun Zhu, Chung-Kuan Cheng, Ronald Graham, "Constructing Zero-deficiency Parallel Prefix Adder of Minimum Depth," ACM Transactions on Design Automation of Electronic Systems, submitted.

[11] Yen-Chun Lin and Jun-Wei Hsiao, "A new Approach to construct optimal prefix circuits with small depth," in *Procedding of the Int. Symp. on Parallel Architectures, Algorithms and Networks , ISPAN'02*

[12] Y.-C. Lin and Y.-H. Hsu, "Constructing H4, a fast depth-size optimal prefix circuit," in *The Journal of Supercomputing*, 24, 279-304, 2003.

[13] R. Brent and H. Kung, "A regular layout for parallel adders," in *IEEE Trans. Computers*, vol. C-31, no. 3, pp.260-264, March 1982.

[14] P. Kogge and H. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence relations," in *IEEE Trans. Computers*, vol. C-22, no .8, pp. 786-793, Aug. 1973.

[15] T. Han and D. Carlson, "Fast area-efficient VLSI adders," in *Proc. 8-th Symp. Arith.*, pp.49-56, Sep. 1987.

[16] M. Smith, *Application Specific Integrated Circuits*, Chap. 3.3, Addison-Wesley, 1997.

[17] D. Harris and I. Sutherland, "Logical effort of carry propagate adders," in *Proc. 37th Asilomar Conf. Signals, Systems, and Computers*, vol. 1, pp.873-878, 2003

[18] Z. Huang and M. Ercegovac, "Effect of wire delay on the design of prefix adders in deep submicron technology," in *Proc. 35th Asilomar Conf. Signals, Systems, and Computers*, vol. 2, pp.1713-1717, 2000.