Cell-Based Multilevel Carry-Increment Adders with Minimal AT- and PT-Products

Reto Zimmermann and Hubert Kaeslin

Abstract—**Carry-select addition techniques imply the computation of double sum and carry bits with subsequent selection of the correct values, resulting in significant area overheads. This overhead increases massively when the selection scheme is applied to multiple levels in order to further reduce computation time. A recently proposed reduced-area scheme for carry-select adders lowers this overhead by computing the carry and sum bits for a block-carry-in value of 0 only and by incrementing them afterwards depending on the final block-carry-in. The resulting *carry-increment* adder cuts circuit size down by 23% with no change in performance. This paper extends this increment scheme hierarchically to build multilevel carry-increment adders. It is shown that such adders are considerably faster while maintaining the same area complexity. The implemented 2-level carry-increment adder has roughly the same size as the 1-level version, but is up to 29% faster. For large word lengths (up to 128 bits), its area-delay product as well as its power-delay product is the smallest of all known adder architectures for cell-based design technologies. Comparisons are carried out on the basis of unit gate-delay and gate-count models as well as CMOS standard-cell implementations.**

Index Terms— **Computer arithmetic, fast cell-based addition techniques, multilevel carry-select/carry-increment adders, VLSI CMOS standard-cell design, power consumption.**

I. Introduction

WITH respect to asymptotic time and area complexity, binary adder architectures can be divided into three primary classes [1]. Table I lists these classes with their complexity measures and their associated adder schemes. The first class consists of the very slow ripple-carry adder with the smallest area. The carry-skip and carry-select adders with variable block sizes form the second adder class and give a good compromise combining relatively small area requirements with substantially shortened computation times. For further delay reduction, groups of skip/select-blocks can in turn be skipped or selected by additional levels of skipping/selection circuitry. The delays of the resulting multilevel carry-skip and carry-select schemes decrease while their area costs increase with the number of hierarchy levels l. The same as for the carry-select adder holds true for the derived carry-increment adder described in this paper. Carry-lookahead, parallel-prefix, and conditional-sum adders from the third class represent the fastest possible addition techniques at the expense of the largest circuit sizes. Circuit speed can be traded off against area even further within each adder class by choosing the appropriate adder type as well as an optimum number of levels for multilevel schemes.

TABLE I

Classification of adder architectures.

area	delay	AT-product	adder schemes
$O(n)$	$O(n)$	$O(n^2)$	ripple-carry
$O(n)$	$O(n^{\frac{1}{l+1}})$	$O(n^{\frac{l+2}{l+1}})$	carry-skip, carry-select, carry-increment
$O(n \log n)$	$O(\log n)$	$O(n \log^2 n)$	carry-lookahead, parallel-prefix, conditional-sum

The *carry-skip* adder (CSKA) [1], [2], [3] has an area increase of only 15 - 30% over the *ripple-carry* adder (RCA) caused by the propagate signal generation and the skip multiplexers (Fig. 1a). Multilevel skipping techniques as well as flexible block sizing [4], [5] allow for further speed improvement. The fact that carry generation starts and carry redistribution ends in slow ripple-carry blocks constrains all skip blocks to be relatively small and thus limits the speed gain substantially. Testability as well as automatic circuit optimization is limited by the inherent hardware redundancy in carry-skip adders. Existing redundancy removal techniques [6] solve this problem, but sacrifice the advantage of small area.

Carry-select adders (CSLA) [1], [7] are about 25% faster, but also up to 50% larger than comparable carry-skip adders. Speed improvement here is achieved by distributing the block-carry-in to all sum bits of a block in parallel (Fig. 1b). This results in fewer but larger blocks and a shorter delay. The carry-select adder can also be made faster by applying multiple levels of select-circuitry at considerably increased area costs.

The *conditional-sum* adder (COSA) [2], [8] is a hierarchical carry-select adder with a maximum number ($\log n$) of selection levels (Fig. 1d). It represents the fastest possible adder scheme on the basis of a unit gate-delay model, but the large number of multiplexers makes it very costly.

Instead of computing both possible sum bits in advance and selecting the correct one via multiple levels of multiplexers, the same intermediate information can be coded in pairs of generate/propagate signals which eventually determine the correct sum bits through a final XOR. Since the computation of new generate/propagate pairs in each level is done with much simpler gates than multiplexers, the overall gate-count is almost cut in half. This simplified structure is known as *parallel-prefix* (or

Manuscript received July 25, 1995; revised March 11, 1996.

The authors are with the Integrated Systems Laboratory, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.

IEEE Log Number 9507076.

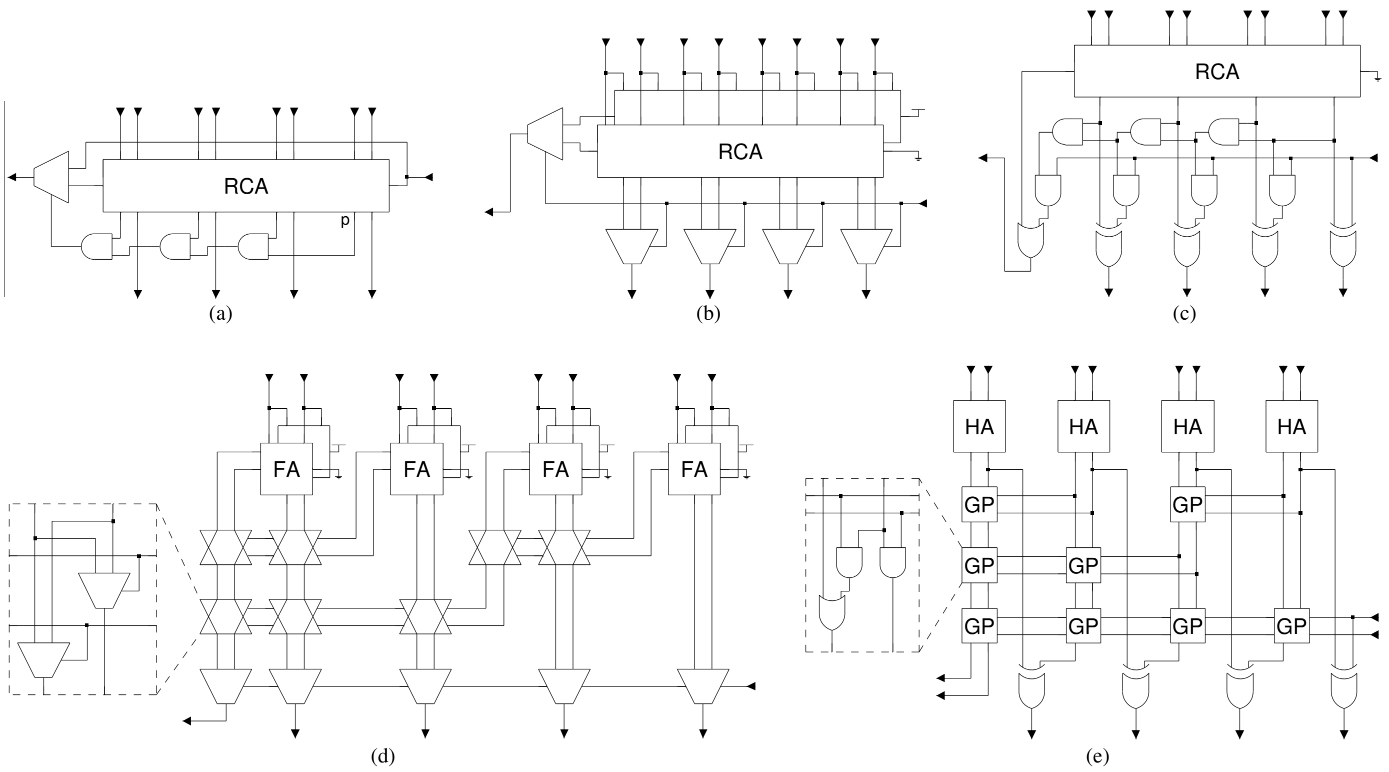


Fig. 1. Schemes of (a) carry-skip, (b) carry-select, (c) carry-increment, (d) conditional-sum, and (e) parallel-prefix adders.

binary/bitwise carry-lookahead) adder (PPA) [9], [10] and represents the simplest and most regular carry-lookahead scheme (Fig. 1e). Bounded fan-out parallel-prefix algorithms have been proposed which are based on tree-like generation and redistribution of carries to avoid additional delays caused by large fan-outs[1]. These prefix techniques range from small and slow [11] to large and fast [12] circuit solutions with hybrid solutions in between [13] (see also [14]). On the other hand, unbounded fan-out structures for prefix computation [15] perform redistribution of carries in parallel, resulting in fewer gate delays but larger fan-outs[2]. Note that the well-known *carry-lookahead* adder[3] (CLA) [1], [2], [16] is in fact a bounded fan-out parallel-prefix adder with block sizes larger than two bits (i.e. usually four bits). The various prefix algorithms are also applicable to the conditional-sum adder scheme, resulting in diverse multiplexer trees differing in size and speed [14].

The above principle of using generate/propagate signal pairs can also be applied to the carry-select adder in order to reduce its circuit size. The resulting *carry-increment* adder scheme (CIA) was proposed and published by Tyagi [17].

In this paper we introduce a new adder structure that combines the carry-increment approach with the processing of carry bits on multiple levels of hierarchy. 2-level carry-increment adders are discussed as a special case of this *multilevel carry-increment adder* scheme. It will be shown that computation times are substantially reduced while the extra area required is minimal which, together, results in extremely low AT-product and PT-product values. It is demonstrated that multilevel speed-up schemes do not necessarily lead to higher area complexities.

[1] In bounded fan-out circuits, all gate fan-outs are limited by a constant factor (e.g. 2).

[2] In unbounded fan-out circuits, gate fan-outs are usually dependent on word length and can be very large, making additional buffering necessary.

[3] The name *carry-lookahead adder* is sometimes also used for discrete 4-bit adder components with linear delay, such as the SN 74181.

This work is substantiated by performance comparisons of the proposed adder schemes with a variety of well-known addition techniques and focuses on cell-based VLSI technologies, such as standard-cells, sea-of-gates, and FPGAs. The cell-based approach is justified by its wide-spread use in the ASIC design community and its compatibility with hardware synthesis, which in turn satisfies the demand for ever higher productivity.

The paper is organized as follows. The previously published carry-increment scheme is described in Section II. Our new 2-level carry-increment adder architecture is then introduced in Section III. Section IV outlines the properties of carry-increment schemes with more than two levels. Section V summarizes the adder comparisons carried out and discusses the results with respect to circuit area, delay, and power requirements. Finally, some conclusions based on the performed investigations are drawn in Section VI.

In the following sections circuit size and delay estimations are done on the basis of a simple gate-count and gate-delay model, which accounts for fan-in but not for fan-out and wire length figures. Therefore each two-input gate is counted as one elementary gate except for the XOR, which has a gate-count of two. Likewise the delay through an elementary gate is counted as one gate-delay, while the XOR contributes two gate-delays. Note that buffers and inverters are not counted within this

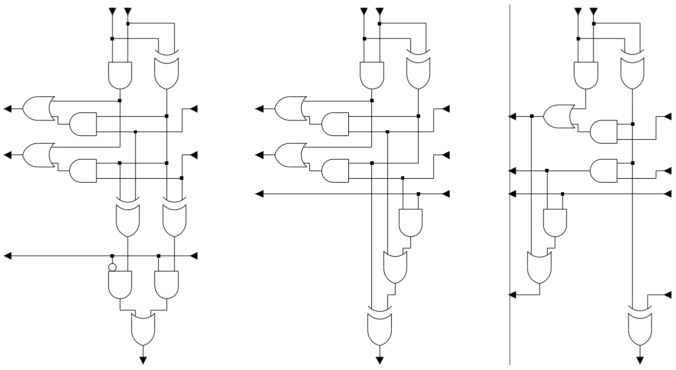


Fig. 2. (a), (b) Select and (c) increment adder cells.

TABLE II
SIGNAL DESCRIPTIONS.

signal	description
a_i, b_i, s_i	ith primary adder input/output bit
c_{in}, c_{out}, c_i	carry-in, carry-out, ith carry bit
$c_i^0, c_i^1, s_i^0, s_i^1$	ith carry/sum bits for block-carry-in = 0/1
g_i, p_i	ith generate/propagate bit
$c_{pb}, c_{pb}^1, c_{pb}^2$	carry-out of previous (level-1/2) block
$c_{tb}, c_{tb}^1, c_{tb}^2$	carry-out of this (level-1/2) block
P_i, P_i^1, P_i^2	(level-1/2) block propagate up to ith bit
$P_{pb}, P_{pb}^1, P_{pb}^2$	propagate of previous (level-1/2) block
$P_{tb}, P_{tb}^1, P_{tb}^2$	propagate of this (level-1/2) block

unit-gate model. Additionally, in order to obtain more realistic comparison results for the different adder structures, exact delay and area measures are taken from placed and routed VLSI CMOS standard-cell implementations.

II. THE 1-LEVEL CARRY-INCREMENT ADDER

The carry-increment principle used in the proposed adders can be explained using different approaches. As already mentioned, the same technique which transforms the area-intensive conditional-sum adder into an area-efficient parallel-prefix structure can be applied to the carry-select adder.

A more intuitive approach is shown in Fig. 1b/c. Instead of computing two results for each block and selecting the correct one, only one sum is calculated and incremented afterwards if necessary, according to the carry input. Thus the second adder and the multiplexers in the carry-select scheme (b) can be replaced by a much smaller incrementer structure (c). Put differently, the computation of a second sum and carry bit is reduced to the generation of a propagate signal per bit position. Furthermore, this increment logic can be incorporated into the adder logic, thus reducing the circuit size even further. The delay from the block-carry-in to the block-carry-out and the sum bits is still small and independent of the block size. Note that the propagate signals do not add any delay because they are generated simultaneously and with fewer gates than the (slower) carry signals.

In his reduced-area scheme for carry-select adders [17] Tyagi shows how the two ripple-chains for both possible block-carry-in values in a typical carry-select adder can be replaced by one ripple-chain and some additional increment logic. Let us start with the logic equations of a carry-select adder bit-slice (Fig. 2a) with a gate-count of 14:

$$
\begin{aligned}
&g_i = a_i\, b_i \quad\; p_i = a_i \oplus b_i \\
&c_i^0 = g_i + p_i\, c_{i-1}^0 \quad\; c_i^1 = g_i + p_i\, c_{i-1}^1 \\
&s_i^0 = p_i \oplus c_{i-1}^0 \quad\; s_i^1 = p_i \oplus c_{i-1}^1 \quad\; s_i = \overline{c_{pb}}\, s_i^0 + c_{pb}\, s_i^1,
\end{aligned}
$$

where $c_i^0(c_i^1)$ denotes the carry at the ith bit position with block-carry-in 0(1) and c_{pb} is the carry output of the previous block. Table II gives a summary of all used signal names and their meanings.

In a first step, the order of the XOR and multiplexer operation for the sum bit computation can be reversed, resulting in $c_{i-1} = \overline{c_{pb}}\, c_{i-1}^0 + c_{pb}\, c_{i-1}^1$ and $s_i = p_i \oplus c_{i-1}$. Since $c_i^1 = c_i^0 + c_i^1$ holds for the two carries (i.e. $c_i^0 = 1 \;\Rightarrow\; c_i^1 = 1$), the first equation can be reduced to $c_{i-1} = c_{i-1}^0 + c_{pb}\, c_{i-1}^1$. Thus, the simplified carry-select adder bit-slice (Fig. 2b) counts 11 gates and computes

$$
\begin{aligned}
&g_i = a_i\, b_i \quad\; p_i = a_i \oplus b_i \\
&c_i^0 = g_i + p_i\, c_{i-1}^0 \quad\; c_i^1 = g_i + p_i\, c_{i-1}^1 \\
&c_{i-1} = c_{i-1}^0 + c_{pb}\, c_{i-1}^1 \quad\; s_i = p_i \oplus c_{i-1}.
\end{aligned}
$$

The following transformations show that $c_i^1 = g_i + p_i\, c_{i-1}^1$ can be reduced to $c_i^{1'} = p_i\, c_{i-1}^1$:

$$
\begin{aligned}
c_i &= c_i^0 + c_{pb}\, c_i^1 \\
&= g_i + p_i\, c_{i-1}^0 + c_{pb}\,(g_i + p_i\, c_{i-1}^1) \\
&= g_i + p_i\, c_{i-1}^0 + c_{pb}\, g_i + c_{pb}\, p_i\, c_{i-1}^1 \\
&= g_i + p_i\, c_{i-1}^0 + c_{pb}\, p_i\, c_{i-1}^1 \\
&= c_i^0 + c_{pb}\, c_i^{1'}.
\end{aligned}
$$

Here $c_i^{1'}$ becomes a block propagate for bit positions k through i and is renamed P_i (k is the first bit of the block and $P_{k-1} = 1$). Also, the principle of sum bit selection has changed to an incrementer structure, where each sum bit is toggled depending on the carry of the previous bit position, the carry of the previous block, and the actual block propagate. Therefore this adder type is referred to as *1-level carry-increment* adder throughout this paper. Its basic full-adder bit-slice counts 10 gates and contains the following logic (Fig. 2c):

$$
\begin{aligned}
&g_i = a_i\, b_i \quad\; p_i = a_i \oplus b_i \quad\; P_i = p_i\, P_{i-1} \\
&c_i^0 = g_i + p_i\, c_{i-1}^0 \quad\; c_i = c_i^0 + P_i\, c_{pb} \quad\; s_i = p_i \oplus c_{i-1}.
\end{aligned}
$$

The carry-out c_i of the last slice in a block is the block-carry-out c_{tb}. The and/or gates determining c_i have now been rearranged (i.e. moved into the previous slice) in order to get a more regular block structure (see Fig. 3). Note that the delay from c_{pb} to s_i increased while the delay from c_{i-1}^0 to s_i decreased compared with the original carry-select cell (Fig. 2a), which, however, has no effect on block sizes or addition speed.

The entire adder structure is depicted in Fig. 3. As demonstrated, only three logically different slices are required, and their arrangement is linear and straightforward. The gate-delay model used yields the following optimal block and maximum

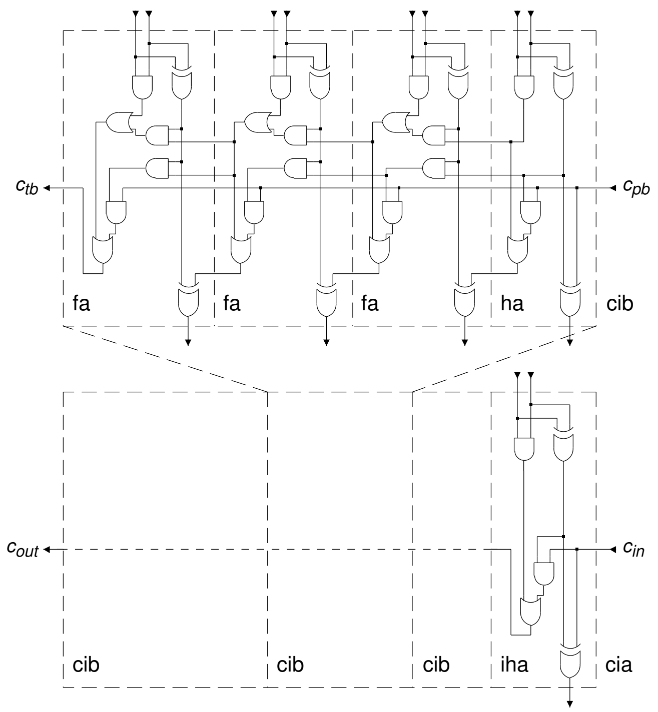
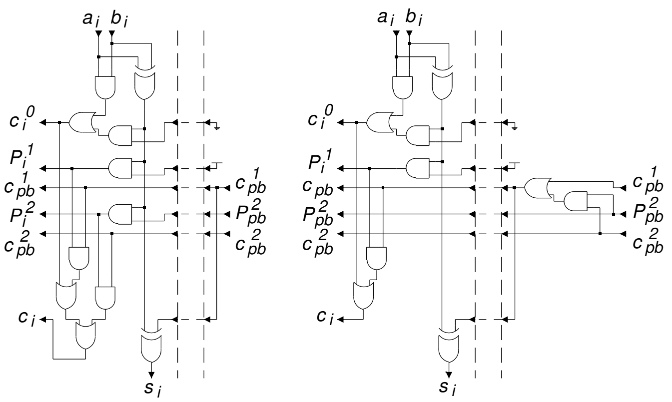


Fig. 3. 1-level carry-increment adder structure.



(a) (b)

Fig. 4. 2-level increment adder cell with (a) 13 and (b) 10 gates.

adder sizes for a given delay:

gate-delays	4	6	8	10	12	14	16	18	20	22	24	...	34
block sizes		2	3	4	5	6	7	8	9	10	11	...	16
max. adder size	2	4	7	11	16	22	29	37	46	56	67	...	137

As an example, an adder with a delay of 24 gates can have a maximum of 67 bits with block sizes of 2, 3, ..., 11. Note that each increment block counts one more bit than its predecessor (same as in Tyagi [17]), and that each additional block adds two gate-delays to the adder. The carry computation in the first slice is sped up using a fast majority gate (i.e. $c_0 = a_0\, b_0 + a_0\, c_{in} + b_0\, c_{in}$) which is counted here as two gate-delays.

III. The 2-Level Carry-Increment Adder

For further speed improvement, Tyagi proposes a select-prefix adder scheme where the ripple-carry blocks of his carry-select adder are replaced by parallel-prefix blocks. The resulting delay reduction is considerable, but at the expense of a massively increased area consumption.

The basic idea of the new adder scheme introduced in this paper is to exchange the ripple-carry blocks of the carry-increment adder for a second level of carry-increment adders. This section shows how the two levels of increment logic can be merged in a structure which makes use of the same basic full-adder cell as the 1-level adder. Thus the resulting 2-level carry-increment adder has nearly the same size as the 1-level version. Yet, the size of the largest ripple-carry block and thus the delay grows only with $O(n^{1/3})$ instead of $O(n^{1/2})$, which increases speed substantially for medium and large word lengths.

Let us now derive the structure and the bit-slice logic of the 2-level carry-increment adder. A first-level increment block consists of full-adder slices with ripple-carry propagation, whereas the second-level increment blocks are composed of first-level blocks. Finally, the whole adder is composed of several second-level increment blocks. Each second-level block gets c^2_{pb} as block-carry-in and advances its carry-out c^2_{tb} to the next block. The inputs to a first-level block are c^1_{pb} and c^2_{pb} as block-carry-ins of levels 1 and 2, and P^2_{pb} as propagate signal from all previous first-level blocks within the same second-level block.

By adding the second-level increment logic to the formulae of the 1-level carry-increment bit-slice, we obtain:

$$
\begin{aligned}
& g_i = a_i\, b_i \quad p_i = a_i \oplus b_i \\
& P^1_i = p_i\, P^1_{i-1} \quad P^2_i = p_i\, P^2_{i-1} \\
& c^0_i = g_i + p_i\, c^0_{i-1} \quad c_i = c^0_i + P^1_i\, c^1_{pb} + P^2_i\, c^2_{pb} \\
& s_i = p_i \oplus c_{i-1}.
\end{aligned}
$$

Additionally, each first-level block has to compute its propagate and carry-out signal,

$$P^2_{tb} = P^1_{tb}\, P^2_{pb} \qquad c^1_{tb} = c^0_{tb} + P^1_{tb}\, c^1_{pb},$$

and each second-level block its carry-out,

$$c^2_{tb} = c^1_{tb} + P^2_{tb}\, c^2_{pb}.$$

P^1_{tb} and c^0_{tb} denote P^1_i and c^0_i of the last first-level block slice, whereas P^2_{tb} is used as P^2_{i-1} of the first slice in the next first-level block.

However, this adder slice has an increased gate-count of 13 (Fig. 4a). Since P^2_i can be expressed by $P^2_i = P^1_i\, P^2_{pb}$, the following transformations are possible:

$$
\begin{aligned}
c_i &= c^0_i + P^1_i\, c^1_{pb} + P^2_i\, c^2_{pb} \\
&= c^0_i + P^1_i\, c^1_{pb} + P^1_i\, P^2_{pb}\, c^2_{pb} \\
&= c^0_i + P^1_i\, (c^1_{pb} + P^2_{pb}\, c^2_{pb}) \\
&= c^0_i + P^1_i\, c_{pb},
\end{aligned}
$$

where $c_{pb} = c^1_{pb} + P^2_{pb}\, c^2_{pb}$ is constant within each first-level increment block and can be precomputed once (Fig. 4b right). Thus, the simplified full-adder bit-slice has again the same structure as the 1-level adder slice with a gate-count of 10 (Fig. 4b left):

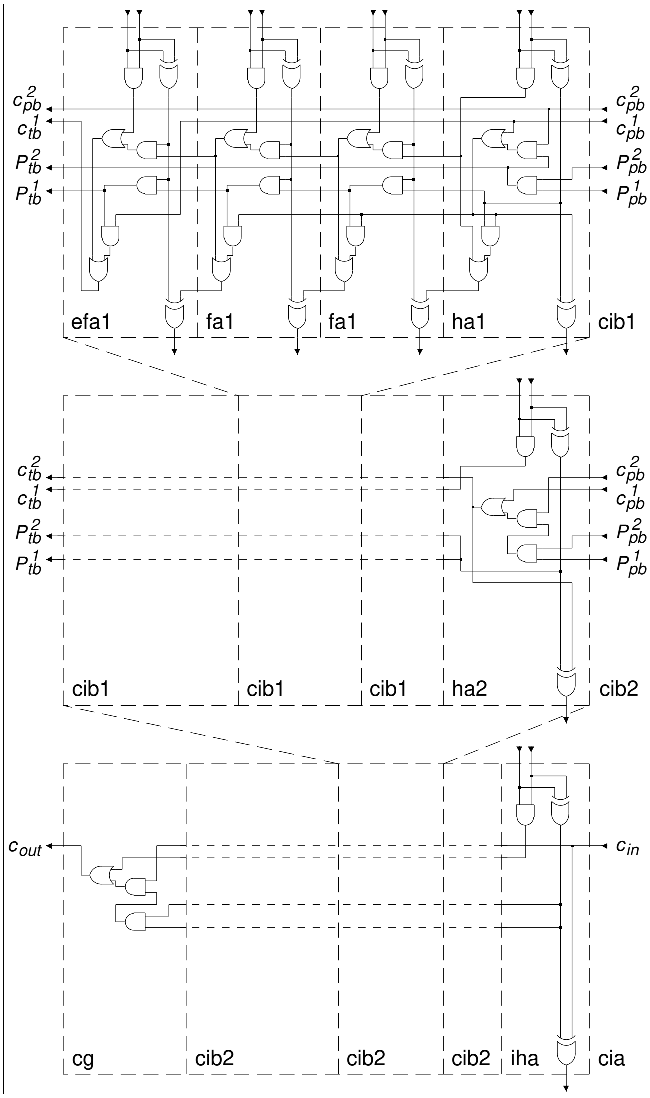


Fig. 5. 2-level carry-increment adder structure.

$$g_i = a_i\, b_i \quad p_i = a_i \oplus b_i \quad P_i^1 = p_i\, P_{i-1}^1$$
$$c_i^0 = g_i + p_i\, c_{i-1}^0 \quad c_i = c_i^0 + P_i^1\, c_{pb} \quad s_i = p_i \oplus c_{i-1}.$$

Furthermore, every first-level block has to compute

$$P_{tb}^2 = P_{tb}^1\, P_{pb}^2$$
$$c_{tb}^1 = c_{tb}^0 + P_{tb}^1\, c_{pb}^1 \quad c_{tb} = c_{tb}^1 + P_{tb}^2\, c_{pb}^2$$

once, while the block-carry-out c_{tb}^2 of a second-level block corresponds to the carry-out c_{tb} of its last first-level block.

The 2-level carry-increment adder structure is depicted in Fig. 5. As can be seen, all the needed gates can be arranged so that every bit-slice contains the same 10 gates, with the exception of some smaller slices. However, some slices differ in routing, and one additional small slice is required for the final carry-out generation, thus resulting in 6 logically different slices. The linear arrangement of the slices is again straightforward and thus also suited for tiled-layout and automated layout generation.

The block size computation for the 2-level carry-increment adder is still quite simple and can be expressed by exact formulae. The optimal block and maximum adder sizes for a given number of gate-delays are:

gate-delays	4	6	8	10	12	14	16	18	20	22
first-level block sizes			1	2	3	4	5	6	7	8
second-level block sizes		1	2	4	7	11	16	22	29	37
max. adder size	2	3	5	9	16	27	43	65	94	131

Note again that each first-level block counts one more bit than its predecessor, and that each second-level block counts one more first-level block than its predecessor. Thus an increase of the overall delay by two gates allows the adder to be expanded by an additional (larger) second-level block.

As was demonstrated the 2-level carry-increment adder consists of the same basic cell as the 1-level version and has only slightly larger cells at the beginning of each increment block. Thus the massive speed improvement by the second increment level comes at negligible additional area costs.

IV. Multilevel Carry-Increment Adders

Carry-increment adders with more than two increment levels are built by applying the same scheme as for the 2-level adder repeatedly. The example of a 3-level carry-increment adder shows that the gate-count increase remains small, whereas a gate-delay reduction is achieved only for adder sizes larger than 64 bits (see results section). This holds true for a larger number of levels as well. Also, the circuit structure becomes more complex, and the upper limit of 10 gates is exceeded for some bit-slices. Therefore, the 2-level carry-increment adder seems to be the best compromise between high area-time performance and low circuit complexity for adder sizes of up to 128 bits.

Note that, at the other extreme, a carry-increment adder scheme with a maximum number of levels ($= \log n$) leads to an unbounded fan-out parallel-prefix algorithm. This special case differs from all other multilevel carry-increment schemes in being slightly faster at considerably higher area costs.

V. Comparisons and Results

A. *Adder Architectures Compared*

All main adder architectures were compared for word lengths of 8, 16, 32, 64, and 128 bits with carry input and output. The realization of the ripple-carry adder (RCA[4]) is straightforward. The 1/2-level carry-skip adders (CSKA-1L/-2L), the 1-level carry-select adders (CSLA-1L), and the 1/2/3-level carry-increment adders (CIA-1L/-2L/-3L) were implemented using variable block sizes. The optimal block sizes were determined by minimizing the overall circuit delay and equalizing all parallel signal paths under the given unit gate-delay model. This was achieved by constructing adders with maximum block sizes and numbers of bits for some given delays and cutting them down to the required adder sizes [4]. The block sizes for the carry-select adders are the same as for the carry-increment adders which are given in the tables from the previous sections.

Three types of parallel-prefix architectures were chosen: the unbounded fan-out structure used by Sklansky [8], [14] (PPA-SK) and the two bounded fan-out prefix algorithms by Brent

[4]All adder acronyms are summarized in Table III with a short architecture description.

and Kung [11] (PPA-BK) and Kogge and Stone [12] (PPA-KS). The conditional-sum adders (COSA) use the unbounded fan-out prefix structure by Sklansky. Finally, a carry-lookahead adder (CLA) with 4-bit blocks was chosen as a representative of the ordinary carry-lookahead scheme [16]. As already mentioned, this scheme corresponds to PPA-BK with a blocking factor of four rather than two. For adder sizes not being a power of four, CLA uses 2-bit blocks in the first level.

Other adder architectures were also studied but not taken into consideration for the comparisons here because they do not provide better performing structures for cell-based design techniques than the above architectures they are derived from. They include various parallel-prefix [10], [13], [18], conditional-sum [14], and carry-skip [19], [20] adders as well as some hybrid architectures [21]–[26] which partly make use of advanced (i.e. not standard-cell conform) circuit techniques.

B. Unit-Gate Model

First comparisons were performed under a simple unit-gate model. Each two-input gate has a gate-count and a gate-delay of one, except for the XOR/XNOR having gate-counts and gate-delays of two. Gates with more inputs are composed of two-input gates using their fastest possible arrangement and summing up their gate-counts and gate-delays (e.g. 4-input AND: gate-count 3, gate-delay 2). Inverters and buffers are ignored. Note that this unit-gate model is very similar to the fan-in model where the fan-in of a gate is taken as its gate-count. The drawbacks of these models are that fan-out and wire length contributions are not accounted for.

Table III lists the gate-count, gate-delay, and gate-count$\times$gate-delay product (gc$\times$gd-product) complexities for all investigated adder architectures as a function of the word length (all adders with carry-in). Because some of the exact formulae are quite complex, only their asymptotic order is given by the highest exponent of the word length n and its multiplication factor. Note that PPA-BK and CLA are the only adder architectures within the group of $O(\log n)$ gate-delays that have linear gate-count complexity. However, their custom layout solutions usually require $O(n \log n)$ area because of regularity reasons (i.e. n bits $\times$ $O(\log n)$ prefix levels).

The exact gate-count and gate-delay numbers for all adder architectures and sizes are given in Tables IV and V. Table VI gives their relative gate-count$\times$gate-delay products with the proposed 2-level carry-increment adder as a reference.

C. Standard-Cell Implementation

After schematic entry, automatic timing and area minimization was performed on all circuits in order to optimize performance under standard-cell library specifications. Circuit size and static timing (i.e. critical path) information was extracted from the compiled layout by taking into account the actual wiring contributions as well as ramp delays at the primary inputs and outputs based on typical external driving strength and capacitive load values (fan-out = 1). Finally, the average power consumption was calculated based on standard-cell power and node capacitance and transition information obtained from gate-level simulation with a set of 1000 random input patterns [27], [28]. All examinations were done with the CMN8A 0.8μm 5 V double-metal CMOS high-density standard-cell library from VLSI Technology Inc. using the design tools by COMPASS Design Automation.

The cell primitives used by the circuit optimizer include multi-input AND/NAND/OR/NOR-gates, various AOI/OAI-gates, two-input XOR/XNOR-gates and two-input multiplexers. Since the usage of full-adder cells for the ripple-carry and carry-select adders showed better area, but worse speed and AT/PT-product performances, no such cells were used for the comparisons.

Tables VIII–X list the area, delay, and relative area-delay (AT) product measures for the standard-cell implementations after placement and routing. The corresponding average power dissipation and relative power-delay (PT) product numbers are given in Tables VII and XI with the proposed CIA-2L acting as reference. Note that the delays are given for worst-case PTV conditions (slow process, 70^{o} C, 4.8 V), whereas typical-case numbers (typical process, 27^{o} C, 5.0 V) are obtained by multiplication with a derating factor of 0.56. Worst-case conditions are also assumed for power estimation (fast transistors, 0^{o} C, 5.5 V).

Figs. 6 and 7 give a graphical representation of the comparison results obtained from the standard-cell implementations. Area vs delay resp. power vs delay measures are drawn on a logarithmic scale visualizing the area-delay and power-delay trade-offs for some important adder architectures.

D. Results and Discussion

The results from the standard-cell implementation comparisons are subsequently discussed. Compared to the ripple-carry adder (RCA) which has the smallest area and longest delay, the carry-skip adders (CSKA) are much faster with a moderate area increase. The carry-increment adders (CIA) achieve further speed improvement at small additional area costs. 2-level implementations of both carry-skip and carry-increment adders (CSKA-2L, CIA-2L) are considerably faster but only slightly larger than their 1-level counterparts (CSKA-1L, CIA-1L). Because the carry-increment adder is a simplification of the carry-select adder (CSLA), it outperforms the latter in all respects.

The various parallel-prefix addition schemes open a wide range of solutions with different area and time performances. The unbounded fan-out parallel-prefix scheme (PPA-SK) represents the fastest adder architecture for large word lengths at the price of quite large area requirements. The bounded fan-out structure by Brent and Kung (PPA-BK) is more area efficient but has a longer computation time. It has similar performance values as the proposed CIA-2L. The bounded fan-out parallel-prefix adder by Kogge and Stone (PPA-KS) suffers from the very large circuit size and routing overhead. The advantage of bounded fan-out is therefore destroyed by the large capacitive load of long wires, which degrades circuit speed and overall performance significantly. The carry-lookahead adder (CLA) being a 4-bit block version of PPA-BK is somewhat faster but considerably larger than the latter. Note that the 8-, 32-, and 128-bit versions of CLA have better area performance because their first lookahead level consist of 2-bit blocks (as in PPA-BK). Finally, the conditional-sum adder (COSA) has a very high area cost which annihilates its speed advantage for large word lengths.

Regarding the area-delay product, the two carry-increment

TABLE III

ADDER COMPLEXITIES (UNIT-GATE MODEL).

adder type	gate-count	gate-delays	gc×gd-product	architecture description
RCA	$7\,n$	$2\,n$	$14\,n^2$	ripple-carry
CSKA-1L	$8\,n$	$4\,n^{1/2}$	$32\,n^{3/2}$	1-level carry-skip
CSKA-2L	$8\,n$	$x\,n^{1/3}$ *	$x\,n^{4/3}$ *	2-level carry-skip
CSLA-1L	$14\,n$	$2.8\,n^{1/2}$	$39\,n^{3/2}$	1-level carry-select
CIA-1L	$10\,n$	$2.8\,n^{1/2}$	$28\,n^{3/2}$	1-level carry-increment
CIA-2L	$10\,n$	$3.6\,n^{1/3}$	$36\,n^{4/3}$	2-level carry-increment
CIA-3L	$10\,n$	$4.4\,n^{1/4}$	$44\,n^{5/4}$	3-level carry-increment
PPA-SK	$3/2\,n\,\log n$	$2\,\log n$	$3\,n\,\log^2 n$	unbounded fan-out parallel-prefix (Sklansky)
PPA-BK	$10\,n$	$4\,\log n$	$40\,n\,\log n$	bounded fan-out parallel-prefix (Brent-Kung)
PPA-KS	$3\,n\,\log n$	$2\,\log n$	$6\,n\,\log^2 n$	bounded fan-out parallel-prefix (Kogge-Stone)
CLA	$14\,n$	$4\,\log n$	$56\,n\,\log n$	"standard" carry-lookahead (i.e. 4-bit PPA-BK)
COSA	$3\,n\,\log n$	$2\,\log n$	$6\,n\,\log^2 n$	unbounded fan-out conditional-sum (Sklansky)

* The exact factors for CSKA-2L have not been computed due to the highly irregular optimal block sizes.

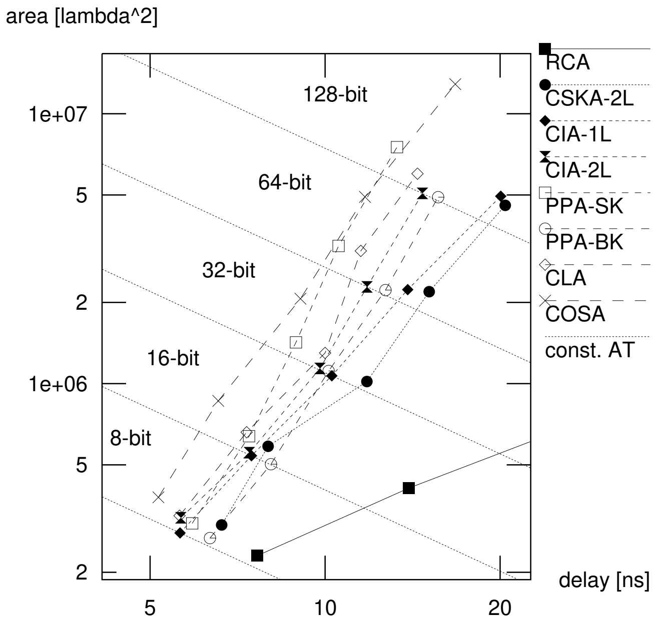


Fig. 6. Area vs delay (logarithmic scale).

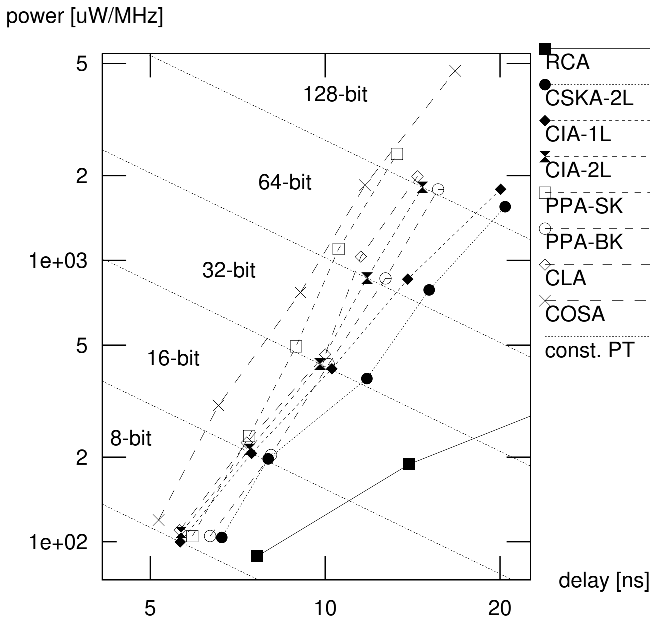


Fig. 7. Power vs delay (logarithmic scale).

and the PPA-BK architectures perform best for all adder sizes with the proposed CIA-2L having the lowest AT-product values for large word lengths. The least area-time efficient structures are RCA, CSLA, PPA-KS, and COSA.

A high correlation can be observed between area and power requirements, which is clearly documented by the similarity of the two graphs in Figs. 6 and 7. This fact is not surprising, however, because dynamic power dissipation mainly originates from charging node capacitances and thus is linearly dependent on the number of circuit nodes and on wire lengths. Therefore, all area-efficient structures like RCA, CSKA, CIA, and PPA-BK are also the most power-efficient ones. The carry-increment adders perform best regarding the power-delay product for all adder sizes with the 2-level version (CIA-2L) having the lowest PT-product for large word lengths. On the other hand, RCA, CSLA, PPA-KS, and COSA show weak power-delay performance. Note that structures with heavily loaded nodes (PPA-KS, COSA) show a significantly higher power/area ratio.

The power dissipated in glitching transitions (i.e. transitions that are followed by an inverse transition within the same computation cycle) is of special interest and was investigated, too. Generally, sources of glitches are gates where an output transition caused by a first input change is undone by a second, delayed transition on another input. This potentially occurs in every

TABLE IV
GATE-COUNT.

adder type	word length [bits] 8	16	32	64	128
RCA	**58**	**114**	**226**	**450**	**898**
CSKA-1L	76	146	286	554	1090
CSKA-2L	71	158	323	633	1248
CSLA-1L	87	194	403	836	1707
CIA-1L	78	157	314	631	1266
CIA-2L	79	158	316	635	1273
CIA-3L	80	159	324	639	1280
PPA-SK	73	165	373	837	1861
PPA-BK	70	147	304	621	1258
PPA-KS	88	216	520	1224	2824
CLA	92	204	428	876	1772
COSA	115	289	687	1581	3563

TABLE V
GATE-DELAY.

adder type	word length [bits] 8	16	32	64	128
RCA	16	32	64	128	256
CSKA-1L	12	16	24	32	48
CSKA-2L	12	16	20	24	32
CSLA-1L	10	12	18	24	34
CIA-1L	10	12	18	24	34
CIA-2L	10	12	16	18	22
CIA-3L	10	12	16	18	20
PPA-SK	10	12	14	16	18
PPA-BK	12	16	20	24	28
PPA-KS	10	12	14	16	18
CLA	12	16	20	24	28
COSA	**8**	**10**	**12**	**14**	**16**

TABLE VI
GATE-COUNT × GATE-DELAY PRODUCT (NORMALIZED).

adder type	word length [bits] 8	16	32	64	128
RCA	1.17	1.92	2.86	5.04	8.21
CSKA-1L	1.15	1.23	1.36	1.55	1.87
CSKA-2L	1.08	1.33	1.28	1.33	1.43
CSLA-1L	1.10	1.23	1.43	1.76	2.07
CIA-1L	0.99	**0.99**	1.12	1.32	1.54
CIA-2L	1.00	1.00	**1.00**	**1.00**	1.00
CIA-3L	1.01	1.01	1.03	1.01	**0.91**
PPA-SK	**0.92**	1.04	1.03	1.17	1.20
PPA-BK	1.06	1.24	1.20	1.30	1.26
PPA-KS	1.11	1.37	1.44	1.71	1.82
CLA	1.40	1.72	1.69	1.84	1.77
COSA	1.16	1.52	1.63	1.94	2.04

TABLE VII
POST-LAYOUT POWER ($\mu W/MHz$).

adder type	word length [bits] 8	16	32	64	128
RCA	**89**	**188**	**331**	**703**	**1440**
CSKA-1L	103	199	373	731	1493
CSKA-2L	104	197	380	785	1551
CSLA-1L	110	253	568	1238	2679
CIA-1L	100	206	412	857	1789
CIA-2L	108	213	428	865	1817
PPA-SK	105	238	495	1098	2389
PPA-BK	105	204	428	863	1786
PPA-KS	113	294	616	1458	3534
CLA	110	225	463	1030	1987
COSA	120	305	769	1848	4709

TABLE VIII
POST-LAYOUT AREA ($k\,\lambda^2$).

adder type	word length [bits] 8	16	32	64	128
RCA	**230**	**410**	**723**	**1626**	**3643**
CSKA-1L	282	596	943	1944	4134
CSKA-2L	299	586	1018	2191	4575
CSLA-1L	306	669	1302	2751	6282
CIA-1L	280	540	1070	2232	4950
CIA-2L	318	555	1139	2284	5069
PPA-SK	303	638	1421	3239	7522
PPA-BK	268	503	1114	2222	4908
PPA-KS	348	950	1914	4676	13626
CLA	324	661	1299	3104	5997
COSA	379	863	2065	4920	12879

TABLE IX
POST-LAYOUT DELAY (ns).

adder type	word length [bits] 8	16	32	64	128
RCA	7.7	13.9	27.7	53.8	103.8
CSKA-1L	6.4	8.5	12.4	17.1	25.4
CSKA-2L	6.6	8.0	11.8	15.1	20.4
CSLA-1L	5.9	7.3	10.0	13.8	20.8
CIA-1L	5.6	7.5	10.3	13.9	20.0
CIA-2L	5.7	7.4	9.8	11.8	14.7
PPA-SK	5.9	7.4	**8.9**	**10.6**	**13.3**
PPA-BK	6.3	8.1	10.1	12.7	15.6
PPA-KS	5.8	7.2	9.1	12.1	16.6
CLA	5.6	7.3	10.0	11.5	14.4
COSA	**5.2**	**6.5**	9.1	11.7	16.7

TABLE X
POST-LAYOUT AT-PRODUCT (NORMALIZED).

adder type	word length [bits] 8	16	32	64	128
RCA	0.97	1.39	1.79	3.24	5.07
CSKA-1L	1.00	1.23	1.04	1.23	1.41
CSKA-2L	1.10	1.13	1.07	1.22	1.25
CSLA-1L	1.00	1.19	1.17	1.40	1.75
CIA-1L	**0.87**	**0.98**	**0.98**	1.14	1.33
CIA-2L	1.00	1.00	1.00	**1.00**	**1.00**
PPA-SK	0.99	1.14	1.13	1.26	1.34
PPA-BK	0.94	**0.98**	1.01	1.04	1.03
PPA-KS	1.12	1.67	1.56	2.10	3.03
CLA	1.01	1.17	1.16	1.32	1.16
COSA	1.08	1.37	1.67	2.13	2.89

TABLE XI
POST-LAYOUT PT-PRODUCT (NORMALIZED).

adder type	word length [bits] 8	16	32	64	128
RCA	1.11	1.66	2.18	3.70	5.59
CSKA-1L	1.08	1.07	1.10	1.22	1.42
CSKA-2L	1.12	0.99	1.06	1.16	1.18
CSLA-1L	1.06	1.17	1.35	1.67	2.08
CIA-1L	**0.92**	**0.97**	**1.00**	1.16	1.34
CIA-2L	1.00	1.00	**1.00**	**1.00**	**1.00**
PPA-SK	1.01	1.11	1.05	1.13	1.18
PPA-BK	1.09	1.04	1.03	1.07	1.04
PPA-KS	1.07	1.34	1.33	1.73	2.19
CLA	1.01	1.04	1.10	1.16	1.07
COSA	1.01	1.26	1.66	2.11	2.95

circuit with unequal signal path delays and multiple, relatively independent input signals. Sources of glitches within adder circuits are the XOR/multiplexer gates used for sum bit generation and the carry propagation circuits, which are subject to race conditions under certain input patterns. The average glitching power contributions range from 10% for 8-bit up to 20% for 128-bit adders, whereas the glitching activity of the primary outputs can be as high as 50% in the worst case. No significant systematic variations between different adder architectures can be observed regarding glitching power. Furthermore, the potential for power savings by suppressing glitching transitions (e.g. balancing of path delays by buffer insertion to avoid race conditions) is very limited due to the large signal delay differences and the additional buffer node activity.

The comparison results obtained allow the conclusion that RCA and CSKA are the choices for small area and moderate speed requirements, whereas CIA-2L and PPA-SK are the best performing candidates for high-speed demands. It is interesting to observe that the area and speed numbers of CIA-2L lie between the two parallel-prefix algorithms PPA-BK and PPA-SK. Thus, the proposed 2-level carry-increment scheme proves to be a high-performing adder architecture which strongly conserves low area and short delay properties also for large adder sizes and under consideration of actual wiring contributions.

In addition to the investigations described so far, the same adder circuits were also optimized and compared using a 0.5μm standard-cell library and tools by Synopsys Inc. with, however, only predictive capacitance information. The results highly match the ones given above and show again the best performance figures for the CIA-2L, PPA-BK, and PPA-SK architectures.

E. More General Observations

It can be observed that the gate-delay and gate-count measures are quite inaccurate when compared to the numbers obtained from actual layout realizations, especially for the area intensive adder architectures. This is because the unit-gate model used disregards basic aspects such as fan-out and wiring contributions. The fan-in model mentioned earlier has not shown better results. In order to get more reliable pre-route information on circuit complexity and speed, the model has to be refined by incorporating fan-out and interconnection aspects. This becomes even more important for deep submicron technologies, where RC delays from wiring become dominant. On the other hand, the unit-gate models are good enough for indicating some general tendencies and for allowing a rough architecture classification with respect to circuit area and delay complexity.

Another obvious observation is that area-intensive structures (like PPA-KS, COSA) suffer from considerable speed degradations caused by long wires and interconnection delays, whereas circuits with smaller area demands preserve their performance during the layout phase. In other words, efficient speed-up is not always achieved by using exhaustive parallelization and hardware duplication techniques. The conclusion is that architectures resulting in small circuit sizes will get increasing performance advantages in area, delay, and power respects as process feature sizes shrink.

Note that all adder architectures can also be classified regarding their fan-out properties. Bounded fan-out structures are found in RCA, CSKA, PPA-BK, and CLA, whereas CSLA, CIA, PPA-SK, PPA-KS, and COSA have unbounded fan-outs. Unbounded fan-out circuits are usually faster due to a higher parallelism but also larger which, together with the higher fan-out loads, slows down computation again. Both classes of fan-out schemes contain area and time efficient adder structures.

The unit-gate model based examinations demonstrate that CIA-2L is slower than PPA-SK by only two gate-delays with the exception of the 128-bit adder. It can be shown that this holds for all multilevel carry-increment adders except for the one with the maximum ($\log n$) number of levels, which actually is equivalent to PPA-SK. Thus, all carry-increment adders with an intermediate number of levels offer no speed advantage over the 2-level implementation but have higher area costs. Therefore, the two extremes with two (CIA-2L) resp. $\log n$ (PPA-SK) increment levels represent the best performing multilevel carry-increment adder schemes.

Further investigations on 4-bit block versions of different parallel-prefix adder architectures have shown no advantages of such structures over their 2-bit block counterparts, whereas solutions with a blocking factor of eight have turned out to become considerably larger and slower.

Hybrid adder architectures, which are mainly used in full-custom implementations of ALUs and multipliers [21]–[26], were marginally considered in this study. The advantages of these architectures seem to lie in the efficient implementation of specific subblocks using advanced circuit techniques, such as pass-transistor logic or dynamic logic (e.g. manchester-chain adders [2]), which are not compatible with cell-based technologies. Investigations based on the unit-gate model have shown no performance advantages of such hybrid adder architectures. Put differently, most addition speed-up techniques seem to reveal their full potential when consistently applied to as large blocks as possible instead of mixing them up.

Full-custom implementations and layout generators ask for adder architectures with high circuit regularity, like e.g. CSKA, CIA, and PPA. Because the layout size of fast parallel-prefix and conditional-sum adders ($O(n \log n)$) grows in both dimensions depending on the word length ($1st$ dimension: number of bits, $2nd$ dimension: number of levels), the 2-level carry-increment adder is the fastest adder structure with linear layout arrangement and area demands ($O(n)$).

AT- and PT-product minimization are, of course, not the only optimization criteria for adder circuits. Tight speed, power, or area constraints cut down possibilities of choice from circuit architectures with diverse performance trade-offs. However, AT- and PT-product measures help finding the most efficient solution from a set of possible circuit candidates.

The presented standard-cell investigation results can by no means be applied to transistor-level design techniques, which open many more circuit alternatives and leave room for further AT- and PT-product optimizations. There, usage of dedicated logic styles and circuit techniques – such as pass-gate/pass-transistor logic or dynamic logic – potentially increases circuit efficiency of multiplexer structures and linear carry-chains. As a result, custom conditional-sum or hybrid adder implementations, for instance, are documented to be highly competitive [21]–[26], resulting in significantly different architecture comparison out-comes.

VI. Conclusions

The proposed multilevel carry-increment addition scheme achieves addition speeds comparable to carry-lookahead and parallel-prefix adders with area requirements close to that of the carry-skip adders. Through its lowest AT- and PT-products, high regularity, and simple blocking scheme, the 2-level carry-increment adder becomes a good choice for a wide range of applications, including layout generators and cell-based arithmetic unit synthesis. The proposed adder structure shows that efficient multilevel speed-up techniques can be realized with a negligible area overhead, as opposed to e.g. multilevel carry-select schemes.

The adder architecture comparisons for cell-based VLSI technologies – such as standard-cells, sea-of-gates, and FPGA – presented in this paper have shown that the only well-performing alternatives to the proposed multilevel carry-increment adder are the ripple-carry and the carry-skip adders for lower area and some parallel-prefix adders for higher speed requirements. The well-known carry-select scheme is outperformed by the simplified carry-increment structure in all respects and must therefore be considered obsolete. The same holds true for any multilevel carry-select structure and, as a special case, the conditional-sum adder, which compares poorly to the related parallel-prefix scheme.

In addition, the increment scheme can be used more universally. In his paper [17] Tyagi replaced the ripple-carry blocks of the carry-increment adder by parallel-prefix blocks, resulting in a hybrid select-prefix adder with a reduction in area and delay compared to a full parallel-prefix adder. Other possible applications are adders with late carry-in which has to be processed very quickly, as used in *modulo* $(2^n - 1)$ or 1's-complement adders with end-around carry [29]. This can be done efficiently by applying an additional increment level for the late carry-in to the entire adder. The resulting adder circuit is only slightly larger and slower than a normal (2's-complement) carry-increment adder.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, 1990.
[2] I. Koren, *Computer Arithmetic Algorithms*, Prentice Hall, 1993.
[3] M. Lehman and N. Burla, "Skip techniques for high-speed carry propagation in binary arithmetic units", *IRE Trans. Electron. Comput.*, vol. EC-10, no. 12, pp. 691–698, Dec. 1961.
[4] S. Turrini, "Optimal group distribution in carry-skip adders", in *Proc. 9th Computer Arithmetic Symp.*, Santa Monica, CA, Sept. 1989, pp. 96–103.
[5] V. Kantabutra, "Designing optimum one-level carry-skip adders", *IEEE Trans. Comput.*, vol. 42, no. 6, pp. 759–764, June 1993.
[6] A. Saldanha, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Circuit structure relations to redundancy and delay", *IEEE Trans. Computer-Aided Design*, vol. 13, no. 7, pp. 875–883, July 1994.
[7] O. J. Bedrij, "Carry-select adder", *IRE Trans. Electron. Comput.*, vol. EC-11, no. 6, pp. 340–346, June 1962.
[8] J. Sklansky, "Conditional sum addition logic", *IRE Trans. Electron. Comput.*, vol. EC-9, no. 6, pp. 226–231, June 1960.
[9] R. E. Ladner and M. J. Fischer, "Parallel prefix computation", *J. ACM*, vol. 27, no. 4, pp. 831–838, Oct. 1980.
[10] B. W. Y. Wei and C. D. Thompson, "Area-time optimal adder design", *IEEE Trans. Comput.*, vol. 39, no. 5, pp. 666–675, May 1990.
[11] R. P. Brent and H. T. Kung, "A regular layout for parallel adders", *IEEE Trans. Comput.*, vol. 31, no. 3, pp. 260–264, Mar. 1982.
[12] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations", *IEEE Trans. Comput.*, vol. 22, no. 8, pp. 783–791, Aug. 1973.
[13] T. Han and D. A. Carlson, "Fast area-efficient VLSI adders", in *Proc. 8th Computer Arithmetic Symp.*, Como, May 1987, pp. 49–56.
[14] H. Lindkvist and P. Andersson, "Techniques for fast CMOS-based conditional sum adders", in *Proc. IEEE Int. Conf. Comput. Design: VLSI in Computers and Processors*, Cambridge, USA, Oct. 1994, pp. 626–635.
[15] F. E. Fich, "New bounds for parallel prefix circuits", in *Proc. 15th ACM Symp. Theory Comput.*, Apr. 1983, pp. 100–109.
[16] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*, John Wiley & Sons, 1979.
[17] A. Tyagi, "A reduced-area scheme for carry-select adders", *IEEE Trans. Comput.*, vol. 42, no. 10, pp. 1162–1170, Oct. 1993.
[18] T. P. Kelliher, R. M. Owens, M. J. Irwin, and T.-T. Hwang, "ELM – a fast addition algorithm discovered by a program", *IEEE Trans. Comput.*, vol. 41, no. 9, pp. 1181–1184, Sept. 1992.
[19] A. Guyot, B. Hochet, and J. M. Muller, "A way to build efficient carry-skip adders", *IEEE Trans. Comput.*, vol. 36, no. 10, Oct. 1987.
[20] P. K. Chan, M. D. F. Schlag, C. D. Thomborson, and V. G. Oklobdzija, "Delay optimization of carry-skip adders and block carry-lookahead adders", in *Proc. 10th Computer Arithmetic Symp.*, Grenoble, June 1991, pp. 154–164.
[21] D. W. Dobberpuhl et al., "A 200-MHz 64-b dual-issue CMOS microprocessor", *IEEE J. Solid-State Circuits*, vol. 27, no. 11, pp. 1555–1564, Nov. 1992.
[22] G. Gerosa et al., "A 2.2 W, 80 MHz superscalar RISC microprocessor", *IEEE J. Solid-State Circuits*, vol. 29, no. 12, pp. 1440–1454, Dec. 1994.
[23] M. Matsui et al., "A 200 MHz 13 mm^2 2-D DCT macrocell using sense-amplifying pipeline flip-flop scheme", *IEEE J. Solid-State Circuits*, vol. 29, no. 12, pp. 1482–1490, Dec. 1994.
[24] V. G. Oklobdzija and D. Villeger, "Improving multiplier design by using improved column compression tree and optimized final adder in CMOS technology", *IEEE Trans. VLSI Syst.*, vol. 3, no. 2, pp. 292–301, June 1995.
[25] N. Ohkubo et al., "A 4.4 ns CMOS 54 $\times$ 54-b multiplier using pass-transistor multiplexer", *IEEE J. Solid-State Circuits*, vol. 30, no. 3, pp. 251–257, Mar. 1995.
[26] J. Mori et al., "A 10-ns 54$\times$54-b parallel structured full array multiplier with 0.5-mum CMOS technology", *IEEE J. Solid-State Circuits*, vol. 26, no. 4, pp. 600–606, Apr. 1991.
[27] F. N. Najm, "A survey of power estimation techniques in VLSI circuits", *IEEE Trans. VLSI Syst.*, vol. 2, no. 4, pp. 446–455, Dec. 1994.
[28] T. K. Callaway and E. E. Swartzlander Jr., "Estimating the power consumption of CMOS adders", in *Proc. 11th Computer Arithmetic Symp.*, Windsor, Ontario, June 1993, pp. 210–219.
[29] C. Efstathiou, D. Nikolos, and J. Kalamatianos, "Area-time efficient modulo $2^n - 1$ adder design", *IEEE Trans. Circuits and Syst.*, vol. 41, no. 7, pp. 463–467, July 1994.